IT workshop v Plzni

Co je nového v MySQL®

1. 2. 2023 v 18:00

Restaurace U Salzmannů

Salonek v 1. patře

Vstup a občerstvení zdarma

PeoplePath

Focus of this talk

- Interesting features from MySQL 5.7 and 8.0.
- Mainly from developer perspective.
- Some features will contain an example or possible use cases.

MySQL versioning

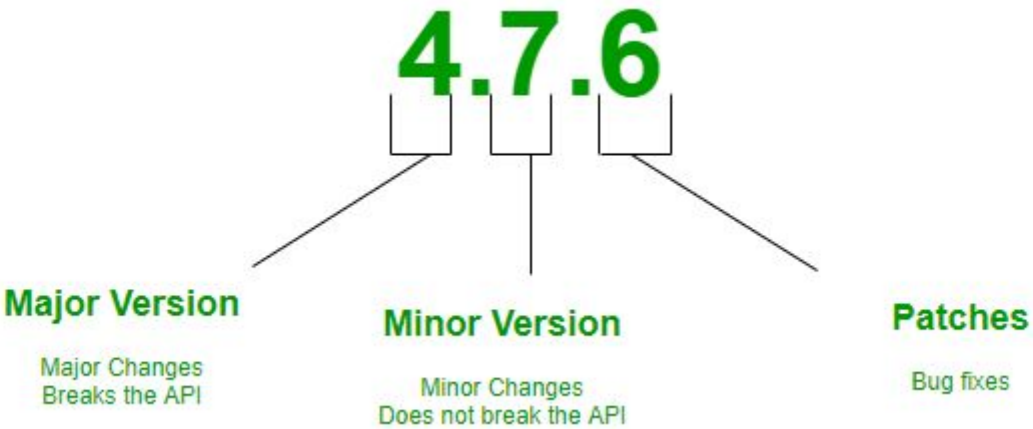
- Since 8.0 release quarterly.

- GA version since 8.0.11 (2018-04-19),

- latest version, 8.0.32 (2023-01-17).

- Does NOT FOLLOW semantic versioning.

Semantic versioning



4.7.6

Major Version

Major Changes
Breaks the API

Minor Version

Minor Changes
Does not break the API

Patches

Bug fixes

MySQL versioning

- Does NOT FOLLOW semantic versioning.
- Breaking changes even in patch releases:
  - Removal of TLSv1 and TLSv1.1 in 8.0.28
  - MySQL Protocol changes in 8.0.24

But also a lot of new exciting features!

Exciting features

- Generated columns,
- JSON support,
- instant DDL,
- various index improvements,
- common table expression and
- window functions.

Generated columns

Generated columns

- since 5.7.5 (5.7.9 GA)
- Allows to store automatically generated data in a table.
- Column value is computed by predefined expression.
- Value cannot be changed manually and can be indexed.
- `GENERATED ALWAYS AS (expression) [STORED|VIRTUAL]`

Generated columns

```
CREATE TABLE users (
   id        INT AUTO_INCREMENT PRIMARY KEY,
   name      VARCHAR(60) NOT NULL,
   surname   VARCHAR(60) NOT NULL,
   full_name VARCHAR(120)
   GENERATED ALWAYS AS (CONCAT(name, ' ', surname))
);
```

Generated columns

```
SELECT * FROM users;
```

```
+----+-----------+----------+---------------+
| id | name      | surname  | full_name     |
+----+-----------+----------+---------------+
|  1 | Jane      | Doe      | Jane Doe      |
|  2 | Janie     | Stiles   | Janie Stiles  |
|  3 | Richard   | Miles    | Richard Miles |
+----+-----------+----------+---------------+
```

Generated columns - few rules

- Used expression cannot:
    - reference another generated column,
    - use columns outside table,
    - contains non-deterministic functions (eg.: NOW()).

Generated columns - types

- VIRTUAL - evaluates its value on the fly (default).

- STORED - evaluates its value and store it on the disk.

```
CREATE TABLE users_alter (
   id          INT AUTO_INCREMENT PRIMARY KEY,
   name        VARCHAR(60) NOT NULL,
   surname     VARCHAR(60) NOT NULL,
   full_name   VARCHAR(120)
   GENERATED ALWAYS AS (CONCAT(name , ' ', surname )) VIRTUAL,
   hash varchar(32)
   GENERATED ALWAYS AS (MD5(CONCAT(name , ' ', surname ))) STORED
);
```

Generated columns - types

```
SELECT * FROM users_alter;
```

```
+----+------------+-----------+----------------+------------+
| id | name       | surname   | full_name      | hash       |
+----+------------+-----------+----------------+------------+
|  1 | Jane       | Doe       | Jane Doe       | f001124... |
|  2 | Janie      | Stiles    | Janie Stiles   | 55bac62... |
|  3 | Richard    | Miles     | Richard Miles  | f32f3cd... |
+----+------------+-----------+----------------+------------+
```

Generated columns - virtual

✅ They do not require disk space.

✅ INSERT and UPDATE queries come with no overhead.

❌ MySQL has to evaluate them during read operations.

Generated columns - stored

✅ No performance penalty during SELECT.

❌ INSERT or UPDATE comes with an overhead.

❌ They require disk space.

Use cases

- To simplify and unify queries.

- To cache a complicated conditions.

- To index a complex value.

- To extract a value from JSON data column.

JSON support

JSON support

- Since 5.7.9, more feature complete from 8.0
- Native JSON column data type.
- Cannot be indexed directly (functional indexes are way).
- Query language JSONPath.

JSONPath

- dot notation:

```
$.tool.jsonpath.creator.location[2]
```

- bracket notation:

```
$value['tool']['jsonpath']['creator']['location'][2]
```

JSON data type, advantages

- Advantages over text column:
  - automatic validation,
  - optimized storage format,
  - supports common operations (where condition, etc),
  - since 8.0.2 (8.0.11 GA) in-place update.

JSON data type

```
CREATE TABLE activity (
   id         int auto_increment primary key,
   event_name ENUM('page-view', 'user-login'),
   user_id    int,
   properties json,
   browser    json
);
```

JSON data type

```sql
INSERT INTO activity(event_name, user_id, properties, browser)
VALUES
(
  'page-view',
  1,
  '{ "page": 1 }',
  '{ "name": "Safari", "os": "Mac", "resolution": { "x": 1920, "y": 1080 } }'
),
(
   'page-view',
   2,
   '{ "page": 2 }',
   '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 2560, "y": 1600 }
}'
);
```

JSON data type, path operator

- column path operator (->) - shortcut to JSON_EXTRACT

```
SELECT id, browser->'$.name' browser
FROM activity;
```

```
+----+-----------+
| id | browser   |
+----+-----------+
|  1 | "Safari"  |
|  2 | "Firefox" |
+----+-----------+
```

JSON data type, inline path operator

- column inline path operator (->>), since 8.0

```
SELECT id, browser->>'$.name' browser
FROM activity;
```

```
+----+-----------+
| id | browser   |
+----+-----------+
|  1 |  Safari   |
|  2 |  Firefox  |
+----+-----------+
```

JSON data type, inline path operator, where

- inline path operator (->>) in where condition

```
SELECT id, browser->>'$.os' os
FROM activity
WHERE browser->>'$.name'='Firefox';
```

```
+----+---------+
| id | os      |
+----+---------+
|  2 | Windows |
+----+---------+
```

Partial update

- Since 8.0.2 (8.0.11 GA), JSON_SET(column, path, value)

```
UPDATE activity
SET `browser` = JSON_SET(
    `browser`,
    '$.name',
    'Phoenix'
)
WHERE browser->>'$.name'='Firefox';
```

```
[2022-10-02 22:18:41] 1 row affected in 4 ms
```

Partial update - rest of functions

- **`JSON_SET()`** replaces existing values and adds non existing values.
- **`JSON_INSERT()`** inserts values without replacing values.
- **`JSON_REPLACE()`** replaces only existing values.
- **`JSON_REMOVE()`** removes data from a JSON document.

Generated column + JSON = 😍

```
CREATE TABLE activity (
   id             int auto_increment primary key,
   event_name     ENUM('page-view', 'user-login'),
   user_id        int,
   properties     json,
   browser        json,
   browser_name varchar(20)
   GENERATED ALWAYS AS (`browser` ->> '$.name')
);
```

Use cases

- Allows to mix document database with relation database.
- This can be a tricky!
- Possible use cases:
    - Error logging,
    - application event logging and
    - piloting ideas.

Instant DDL

Instant “Data definition language” - schema changing command

Instant DDL

- Partial support since 8.0.12, extended in 8.0.29
- Allows schema changes without making data unavailable.
- No need to do anything special to enable online DDL.

But you have to understand what is happening under the hood!

DDL Algorithms

- Algorithms InnoDB supports:
    - COPY
    - INPLACE
    - INSTANT

COPY Algorithm

- MySQL internally:
    - Create a new table with the altered schema.
    - Migrate data into new table.
    - Swaps the table names.
    - Drops the old table.

COPY Algorithm drawbacks

- Rollback of operation can be an expensive process.

- Concurrent DML’s are not allowed during the ALTER table.

- Causes replication lag.

INPLACE Algorithm

- Operations are done in-place in the original table.
- Uses a temporary log file to track data changes by DML queries during the change.
- After in-place operation finishes the log will be applied.

INPLACE Algorithm drawbacks

- Large number of concurrent DML’s can fail.

- Rollback of operation can be an expensive process.

- Causes replication lag.

INSTANT Algorithm

Performs only metadata changes => Doesn’t touch the data file of the table.

✖

…is not supported for all DDL operations.

INSTANT Algorithm, support

- Adding a column
- Dropping a column
- Renaming a column
- Modifying a column default value
- Renaming table

INSTANT Algorithm drawbacks

- Best works on latest patch version of MySQL, e.g:
  - Prior to 8.0.29, a column can only be added as a last one.
- You can do only 64 INSTANT operations on one table.

INSTANT Algorithm drawbacks

- Is not supported for:
  - tables that use ROW_FORMAT=COMPRESSED,
  - tables with a FULLTEXT index,
  - temporary tables and
  - stored columns.

INSTANT Algorithm recommendations

- Can be forced by ALGORITHM=INSTANT;
- Check patch version of MySQL:
    - at least 8.0.29 is recommended.
- Be aware of combinations:
    - column drop is instant, **index drop** not!

INSTANT Algorithm recommendations

- Always consult with documentation

- Part: “*15.12.1 Online DDL Operations*”:

https://dev.mysql.com/doc/refman/8.0/en/innodb-online-ddl-operations.html

Indexes

Indexes

- In MySQL 8, new types of index:
    - Multi-Valued,
    - functional,
    - descending and
    - invisible.

Multi-Valued Index

- Since 8.0.17

- Index defined on a column that stores an array of values.

- Use case - index JSON arrays.

- CAST(... AS ... ARRAY) in the index definition.

Multi-Valued Indexes, creation

```
CREATE TABLE customers (
  id       BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  modified DATETIME DEFAULT CURRENT_TIMESTAMP,
  custinfo JSON,
  INDEX zips((CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY)))
);
```

Multi-Valued Indexes, usage

- Condition functions:
  - MEMBER OF(json_array)
  - JSON_CONTAINS(target, candidate[, path])
  - JSON_OVERLAPS(json_doc1, json_doc2)

Multi-Valued Indexes, MEMBER OF

Succeed whether a given value is an element of json_array.

```
SELECT * FROM customers
WHERE 123 MEMBER OF(custinfo->'$.zipcode');
```

```
+----+---------------------+-------------------------------------------------------------+
| id | modified            | custinfo                                                    |
+----+---------------------+-------------------------------------------------------------+
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 123, 94582]} |
|  3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 123]}     |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [123, 94582]}     |
+----+---------------------+-------------------------------------------------------------+
```

Multi-Valued Indexes, JSON_CONTAINS

Succeeds whether a given JSON document is contained within a target JSON document.

```
SELECT * FROM customers WHERE
JSON_CONTAINS(custinfo->'$.zipcode', CAST('[123,456]' AS JSON));
```

```
+----+---------------------+-----------------------------------------------------+
| id | modified            | custinfo                                            |
+----+---------------------+-----------------------------------------------------+
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 123, 456]}  |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [123, 456]}  |
+----+---------------------+-----------------------------------------------------+
```

Multi-Valued Indexes, JSON_OVERLAPS

Succeed if the two document have a common array elements.

```
SELECT * FROM customers WHERE
JSON_OVERLAPS(custinfo->'$.zipcode', CAST('[123,456]' AS JSON));
```

```
+----+---------------------+-------------------------------------------------------------------+
| id | modified            | custinfo                                                          |
+----+---------------------+-------------------------------------------------------------------+
|  1 | 2019-06-29 22:23:12 | {"user": "Jack", "user_id": 37, "zipcode": [456, 94536]}          |
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 123, 456]}     |
|  3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 123]}           |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [123, 456]}             |
+----+---------------------+-------------------------------------------------------------------+
```

Functional indexes

- Since 8.0.13

- Function can be used as base for index.

- Usage - a filter condition against functional expression.

```sql
SELECT AVG(price) FROM products WHERE MONTH(create_time)=10;
```

Functional indexes

```
CREATE TABLE `products` (
   `id`          int unsigned NOT NULL PRIMARY KEY AUTO_INCREMENT,
   `price`       integer DEFAULT NULL,
   `create_time` timestamp NULL DEFAULT NULL,
   KEY `functional_index` ((month(`create_time`)))
) ENGINE=InnoDB;
```

Functional indexes

```
EXPLAIN SELECT AVG(price) FROM products WHERE MONTH(create_time)=10;
```

without index

```
           id: 1
  select_type: SIMPLE
        table: products
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where
```

with index

```
           id: 1
  select_type: SIMPLE
        table: products
   partitions: NULL
         type: ref
possible_keys: functional_index
          key: functional_index
      key_len: 5
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
```

Functional indexes, implementation

- They are implemented as hidden virtual generated columns:
    - They counts against the total limit of table columns.
    - Use only functions permitted for generated columns.
    - Subqueries, parameters, variables, stored functions, and loadable functions are not permitted.

Functional indexes + JSON = 💖

- JSON_VALUE() introduced in MySQL 8.0.21 allows to transparently index properties.

```
SELECT JSON_VALUE('{"fname": "Joe", "lname": "Palmer"}', '$.fname');

+--------------------------------------------------------------+
| JSON_VALUE('{"fname": "Joe", "lname": "Palmer"}', '$.fname') |
+--------------------------------------------------------------+
| Joe                                                          |
+--------------------------------------------------------------+
```

Functional indexes + JSON = 💖

```
SELECT JSON_VALUE(json_doc, path RETURNING type);
```

Equivalent to:

```
SELECT CAST(
   JSON_UNQUOTE(JSON_EXTRACT(json_doc, path))
   AS type
);
```

Functional indexes + JSON = 💖

```
CREATE TABLE data(
   j JSON,
   INDEX i1 ( (JSON_VALUE(j, '$.id' RETURNING UNSIGNED)) )
);

EXPLAIN SELECT * FROM data WHERE JSON_VALUE(j, '$.id' RETURNING UNSIGNED) = 123;

           id: 1
  select_type: SIMPLE
        table: data
   partitions: NULL
         type: ref
possible_keys: i1
          key: i1
      key_len: 9
          ref: const
         rows: 1
     filtered: 100.00
        Extra: NULL
```

Descending Indexes

- Since 8.0.1, (8.0.11)
- Stores key values in descending order.
- Can be combined in multiple-column indexes.
- Can increase the performance of following pattern:

```
ORDER BY field1 DESC, field2 ASC LIMIT N;
```

Descending Indexes

```
CREATE TABLE `articles` (
   `id` int(11) NOT NULL AUTO_INCREMENT,
   `name` varchar(100) DEFAULT NULL,
   `created` datetime DEFAULT NULL,
   PRIMARY KEY (`id`),
   KEY `created_desc_name_asc` (`created` DESC,`name`)
) ENGINE=InnoDB;
```

```
SELECT * FROM articles ORDER BY created DESC, name ASC limit 10;
```

```
+----+------+---------------------+
| id | name | created             |
+----+------+---------------------+
|  1 | foo  | 2022-10-01 16:20:52 |
|  3 | quz  | 2022-06-18 16:21:27 |
|  2 | bar  | 2022-06-09 16:21:08 |
+----+------+---------------------+
```

Invisible indexes

- Maintained indexes that are not used by the optimizer.
- Allows to test the effect of removing an index, without making a destructive change.

```
ALTER TABLE t1 ALTER INDEX i_idx INVISIBLE;
ALTER TABLE t1 ALTER INDEX i_idx VISIBLE;
```

Indexes

- Other improvements:
    - histograms,
    - simultaneous index build (8.0.27) and
    - CHECK Constraints 😍.

Intersect and except

Intersect and except

- since 8.0.31

- **`INTERSECT`** limits the result from multiple SELECT statements to those rows which are common to all.

- **`EXCEPT`** limits the result from the first SELECT statement to those rows which are (also) not found in the second.

Intersect and except

- As with UNION, the operands must have the same number of columns.
- DISTINCT modifier can remove duplicates from either side of the intersection.

Intersect and except

```
SELECT * FROM ordered_food;
```

```
+----+-------------+-------+--------+
| id | name        | tacos | sushis |
+----+-------------+-------+--------+
|  1 | Kenny       |  NULL |     10 |
|  2 | Miguel      |     5 |      0 |
|  3 | Bohus       |     4 |      5 |
|  4 | Kajiyamasan |  NULL |     10 |
|  5 | Scott       |    10 |   NULL |
|  6 | Lenka       |  NULL |   NULL |
+----+-------------+-------+--------+
```

Intersect

```
SELECT * FROM ordered_food
WHERE tacos > 0;
```

```
+----+--------+-------+--------+
| id | name   | tacos | sushis |
+----+--------+-------+--------+
|  2 | Miguel |     5 |      0 |
|  3 | Bohus  |     4 |      5 |
|  5 | Scott  |    10 |   NULL |
+----+--------+-------+--------+
```

```
SELECT * FROM ordered_food
WHERE sushis > 0;
```

```
+----+-------------+-------+--------+
| id | name        | tacos | sushis |
+----+-------------+-------+--------+
|  1 | Kenny       |  NULL |     10 |
|  3 | Bohus       |     4 |      5 |
|  4 | Kajiyamasan |  NULL |     10 |
+----+-------------+-------+--------+
```

Intersect

```
SELECT * FROM ordered_food WHERE tacos>0
INTERSECT
SELECT * FROM ordered_food WHERE sushis>0;
```

```
+----+-------+-------+--------+
| id | name  | tacos | sushis |
+----+-------+-------+--------+
|  3 | Bohus |     4 |      5 |
+----+-------+-------+--------+
```

Except

- limits the result from the first statement to rows which are not found in the second.

```sql
SELECT * FROM ordered_food
WHERE tacos > 0;
```

```
+----+--------+-------+--------+
| id | name   | tacos | sushis |
+----+--------+-------+--------+
|  2 | Miguel |     5 |      0 |
|  3 | Bohus  |     4 |      5 |
|  5 | Scott  |    10 |   NULL |
+----+--------+-------+--------+
```

```sql
SELECT * FROM ordered_food
WHERE sushis > 0;
```

```
+----+-------------+-------+--------+
| id | name        | tacos | sushis |
+----+-------------+-------+--------+
|  1 | Kenny       |  NULL |     10 |
|  3 | Bohus       |     4 |      5 |
|  4 | Kajiyamasan |  NULL |     10 |
+----+-------------+-------+--------+
```

Except

```
SELECT * FROM ordered_food WHERE tacos>0
EXCEPT
SELECT * FROM ordered_food WHERE sushis>0;
```

```
+----+--------+-------+--------+
| id | name   | tacos | sushis |
+----+--------+-------+--------+
|  2 | Miguel |     5 |      0 |
|  5 | Scott  |    10 |   NULL |
+----+--------+-------+--------+
```

CTE - Common Table Expression

CTE - Common Table Expression

- since 8.0.1, (8.0.11)
- An alternative to a derived table and view.
- Simplifies complex joins and subqueries.
- syntax: `WITH <name> AS (<query>)`

CTE, motivation

- Better readability of the queries.

- Improved performance.

- A valid alternative to a VIEW and a temporary table.

- Possibility to create recursive queries.

CTE, example with EAV

- Entity–attribute–value model

id	name	surname
1	John	Doe

Entity (user_id)	Attribute	Value
1	name	John
1	surname	Doe

CTE, example with EAV

- Wordpress eg.: wp_usermeta

- Transform from EAV to regular table can be done by pivoting.

```
+----------+---------+------------+------------+
| umeta_id | user_id | meta_key   | meta_value |
+----------+---------+------------+------------+
|        1 |       1 | first_name | Emma       |
|        2 |       1 | last_name  | Obrien     |
|        3 |       1 | nickname   | admin      |
+----------+---------+------------+------------+
```

CTE, example with EAV

```
SELECT user_id,
   MAX(CASE WHEN meta_key='first_name' THEN meta_value END) as first,
   MAX(CASE WHEN meta_key='last_name' THEN meta_value END) as last
FROM `wp_usermeta`
GROUP BY user_id
```

```
+---------+--------+---------+
| user_id | first  | last    |
+---------+--------+---------+
|       1 | Emma   | Obrien  |
|       2 | Nial   | Casey   |
|       3 | Keeley | Brookes |
|       4 | Bert   | Mccoy   |
|       5 | Alyce  | Sheldon |
+---------+--------+---------+
```

CTE, example with EAV

- CTE with pivot query

```
WITH cte AS (
   SELECT user_id,
      MAX(CASE WHEN meta_key='first_name' THEN meta_value END) as first,
      MAX(CASE WHEN meta_key='last_name' THEN meta_value END) as last
   FROM `wp_usermeta`
   GROUP BY user_id
)
SELECT * FROM cte;
```

CTE, example with EAV

- CTE with pivot query

```
WITH cte AS (
   SELECT user_id,
      MAX(CASE WHEN meta_key='first_name' THEN meta_value END) as first,
      MAX(CASE WHEN meta_key='last_name' THEN meta_value END) as last
   FROM `wp_usermeta`
   GROUP BY user_id
)
SELECT * FROM cte
WHERE first = "Emma" OR last = "Sheldon"
ORDER BY first
```

CTE, recursive

- A recursive CTE with subquery that refers itself.
- Usage:
  - to generate series and
  - hierarchical or tree-structured data traversal.

CTE, recursive

- A recursive CTE components:

```
WITH RECURSIVE cte AS (
   initial_query    -- "seed" member
   UNION ALL
   recursive_query  -- recursive member referring the same CTE
)
SELECT * FROM cte;  -- main query
```

CTE, recursive

- Seed member - initial query, executed in first iteration.
- Recursive member - contains the reference to the same CTE name.
- This second component will generate all the remaining items of the main query.

CTE, recursive, example

```
WITH RECURSIVE cte (n) AS
  (
      SELECT 1                           -- "seed" member
      UNION ALL
      SELECT n + 1 FROM cte WHERE n < 5  -- recursive member
  )
SELECT * FROM cte;                       -- main query
```

```
+-------+
| n     |
+-------+
|     1 |
|     2 |
|     3 |
|     4 |
|     5 |
+-------+
```

CTE, recursive, example

```sql
WITH RECURSIVE cte (n) AS
(
  SELECT '2013-01-01'
  UNION ALL
  SELECT n + INTERVAL 1 DAY FROM cte WHERE n < '2013-01-10'
)
SELECT * FROM cte;
```

```
+------------+
| n          |
+------------+
| 2013-01-01 |
| 2013-01-02 |
| 2013-01-03 |
| 2013-01-04 |
|     ...    |
```

CTE, recursive, example

```
CREATE TABLE orgchart(
    id         INT PRIMARY KEY AUTO_INCREMENT,
    name       VARCHAR(20),
    role       VARCHAR(20),
    manager_id INT,
    FOREIGN KEY (manager_id) REFERENCES orgchart(id)
);
```

CTE, recursive, example

```
+----+----------+--------------+------------+
| id | name     | role         | manager_id |
+----+----------+--------------+------------+
|  1 | Matthew  | CEO          |       NULL |
|  2 | Caroline | CFO          |          1 |
|  3 | Tom      | CTO          |          1 |
|  4 | Sam      | Treasurer    |          2 |
|  5 | Ann      | Controller   |          2 |
|  6 | Anthony  | Dev Director |          3 |
|  7 | Lousie   | Sys Admin    |          3 |
|  8 | Travis   | Senior DBA   |          3 |
|  9 | John     | Developer    |          6 |
| 10 | Jennifer | Developer    |          6 |
| 11 | Maria    | Junior DBA   |          8 |
+----+----------+--------------+------------+
```

CTE, recursive, example

```
+------+----------+---------------------------------------+-------+
| id   | name     | path                                  | level |
+------+----------+---------------------------------------+-------+
|  1   | Matthew  | Matthew                               |     1 |
|  2   | Caroline | Matthew -> Caroline                   |     2 |
|  3   | Tom      | Matthew -> Tom                        |     2 |
|  4   | Sam      | Matthew -> Caroline -> Sam            |     3 |
|  5   | Ann      | Matthew -> Caroline -> Ann            |     3 |
|  6   | Anthony  | Matthew -> Tom -> Anthony             |     3 |
|  7   | Lousie   | Matthew -> Tom -> Lousie              |     3 |
|  8   | Travis   | Matthew -> Tom -> Travis              |     3 |
|  9   | John     | Matthew -> Tom -> Anthony -> John     |     4 |
| 10   | Jennifer | Matthew -> Tom -> Anthony -> Jennifer |     4 |
| 11   | Maria    | Matthew -> Tom -> Travis -> Maria     |     4 |
+------+----------+---------------------------------------+-------+
```

CTE, recursive, example

```
WITH RECURSIVE reporting_chain(id, name, path, level) AS (
   SELECT id, name, CAST(name AS CHAR(100)), 1
   FROM orgchart
   WHERE manager_id IS NULL
   UNION ALL
   SELECT oc.id, oc.name, CONCAT(rc.path,' -> ',oc.name), rc.level+1
   FROM reporting_chain rc JOIN orgchart oc ON rc.id=oc.manager_id)
SELECT * FROM reporting_chain ORDER BY level;
```

CTE, recursive, example

```
+------+----------+---------------------------------------+-------+
| id   | name     | path                                  | level |
+------+----------+---------------------------------------+-------+
|  1   | Matthew  | Matthew                               |     1 |
|  2   | Caroline | Matthew -> Caroline                   |     2 |
|  3   | Tom      | Matthew -> Tom                        |     2 |
|  4   | Sam      | Matthew -> Caroline -> Sam            |     3 |
|  5   | Ann      | Matthew -> Caroline -> Ann            |     3 |
|  6   | Anthony  | Matthew -> Tom -> Anthony             |     3 |
|  7   | Lousie   | Matthew -> Tom -> Lousie              |     3 |
|  8   | Travis   | Matthew -> Tom -> Travis              |     3 |
|  9   | John     | Matthew -> Tom -> Anthony -> John     |     4 |
| 10   | Jennifer | Matthew -> Tom -> Anthony -> Jennifer |     4 |
| 11   | Maria    | Matthew -> Tom -> Travis -> Maria     |     4 |
+------+----------+---------------------------------------+-------+
```

CTE, recursive, limitation

- **cte_max_recursion_depth** - max recursion depth (1000).
- **max_execution_time** - execution timeout for SELECT statements.
- **MAX_EXECUTION_TIME** - optimizer hint enforces a per-query.

Window functions

Window functions

- since 8.0.2, (8.0.11)
- Offers aggregate-like functionality on a defined range of rows in a query.
- Window functions will return a value for every row in a query result.

Window functions

- **OVER** - indicates usage of window function
- **PARTITION BY** - marks how to divide the rows into groups

```
SELECT
   <agregation>(field) OVER() AS field_name,
   <agregation>(field) OVER(PARTITION BY field) AS field_name
FROM <table name>
```

Window functions, example

```
CREATE TABLE sales(
   id      INT PRIMARY KEY AUTO_INCREMENT,
   year    INT,
   country VARCHAR(20),
   product VARCHAR(32),
   profit  INT
);
```

Window functions, example

```
SELECT * FROM sales ORDER BY country, year, product;
```

```
+----+------+---------+------------+--------+
| id | year | country | product    | profit |
+----+------+---------+------------+--------+
|  1 | 2000 | Finland | Computer   |   1500 |
|  2 | 2000 | Finland | Phone      |    100 |
|  3 | 2001 | Finland | Phone      |     10 |
|  4 | 2000 | India   | Calculator |     75 |
|  5 | 2000 | India   | Calculator |     75 |
|  6 | 2000 | India   | Computer   |   1200 |
|  7 | 2000 | USA     | Calculator |     75 |
|  8 | 2000 | USA     | Computer   |   1500 |
|  9 | 2001 | USA     | Calculator |     50 |
| 10 | 2001 | USA     | Computer   |   1500 |
| 11 | 2001 | USA     | Computer   |   1200 |
| 12 | 2001 | USA     | TV         |    150 |
| 13 | 2001 | USA     | TV         |    100 |
+----+------+---------+------------+--------+
```

Window functions, regular aggregations

```
SELECT SUM(profit) AS total_profit FROM sales;
```

```
+--------------+
| total_profit |
+--------------+
|         7535 |
+--------------+
```

```
SELECT country, SUM(profit) AS country_profit

FROM sales GROUP BY country ORDER BY country;
```

```
+---------+----------------+
| country | country_profit |
+---------+----------------+
| Finland |           1610 |
| India   |           1350 |
| USA     |           4575 |
+---------+----------------+
```

Window functions, example

```
SELECT
   year, country, product, profit,
   SUM(profit) OVER() AS total_profit,
   SUM(profit) OVER(PARTITION BY country) AS country_profit
FROM sales
ORDER BY country, year, product, profit;
```

Window functions, example

```
+------+---------+------------+--------+--------------+----------------+
| year | country | product    | profit | total_profit | country_profit |
+------+---------+------------+--------+--------------+----------------+
| 2000 | Finland | Computer   |   1500 |         7535 |           1610 |
| 2000 | Finland | Phone      |    100 |         7535 |           1610 |
| 2001 | Finland | Phone      |     10 |         7535 |           1610 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Computer   |   1200 |         7535 |           1350 |
| 2000 | USA     | Calculator |     75 |         7535 |           4575 |
| 2000 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | Calculator |     50 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1200 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | TV         |    100 |         7535 |           4575 |
| 2001 | USA     | TV         |    150 |         7535 |           4575 |
+------+---------+------------+--------+--------------+----------------+
```

Window functions, example

```
SELECT
   year, country, product, profit,
   RANK() OVER(ORDER BY `profit` desc, `id`) total_profit_rank
FROM sales
ORDER BY total_profit_rank;
```

```
+------+---------+-------------+--------+-------------------+
| year | country | product     | profit | total_profit_rank |
+------+---------+-------------+--------+-------------------+
| 2000 | Finland | Computer    |   1500 |                 1 |
| 2000 | USA     | Computer    |   1500 |                 2 |
| 2001 | USA     | Computer    |   1500 |                 3 |
| 2000 | India   | Computer    |   1200 |                 4 |
| 2001 | USA     | Computer    |   1200 |                 5 |
| 2001 | USA     | TV          |    150 |                 6 |
| 2000 | Finland | Phone       |    100 |                 7 |
```

Window functions, example

```
SELECT
   year, country, product, profit,
   RANK() OVER(PARTITION BY `country` ORDER BY `profit` desc, id)
   total_profit_rank
FROM sales
ORDER BY country, total_profit_rank;
```

Window functions, example

```
+------+---------+------------+--------+-------------------+
| year | country | product    | profit | total_profit_rank |
+------+---------+------------+--------+-------------------+
| 2000 | Finland | Computer   |   1500 |                 1 |
| 2000 | Finland | Phone      |    100 |                 2 |
| 2001 | Finland | Phone      |     10 |                 3 |
| 2000 | India   | Computer   |   1200 |                 1 |
| 2000 | India   | Calculator |     75 |                 2 |
| 2000 | India   | Calculator |     75 |                 3 |
| 2000 | USA     | Computer   |   1500 |                 1 |
| 2001 | USA     | Computer   |   1500 |                 2 |
| 2001 | USA     | Computer   |   1200 |                 3 |
| 2001 | USA     | TV         |    150 |                 4 |
| 2001 | USA     | TV         |    100 |                 5 |
| 2000 | USA     | Calculator |     75 |                 6 |
| 2001 | USA     | Calculator |     50 |                 7 |
+------+---------+------------+--------+-------------------+
```

Window functions, example

```
SELECT
   year, country, product, profit,
   RANK() OVER(
      PARTITION BY `country`
      ORDER BY `profit` desc, id
   ) total_profit_rank,
   profit - FIRST_VALUE( profit ) OVER (
      PARTITION BY `country`
      ORDER BY `profit` desc, id
   ) profit_back_of_first
FROM sales
ORDER BY country, total_profit_rank;
```

Window functions, example

```
+------+---------+------------+--------+-------------------+----------------------+
| year | country | product    | profit | total_profit_rank | profit_back_of_first |
+------+---------+------------+--------+-------------------+----------------------+
| 2000 | Finland | Computer   |   1500 |                 1 |                    0 |
| 2000 | Finland | Phone      |    100 |                 2 |                -1400 |
| 2001 | Finland | Phone      |     10 |                 3 |                -1490 |
| 2000 | India   | Computer   |   1200 |                 1 |                    0 |
| 2000 | India   | Calculator |     75 |                 2 |                -1125 |
| 2000 | India   | Calculator |     75 |                 3 |                -1125 |
| 2000 | USA     | Computer   |   1500 |                 1 |                    0 |
| 2001 | USA     | Computer   |   1500 |                 2 |                    0 |
| 2001 | USA     | Computer   |   1200 |                 3 |                 -300 |
| 2001 | USA     | TV         |    150 |                 4 |                -1350 |
| 2001 | USA     | TV         |    100 |                 5 |                -1400 |
| 2000 | USA     | Calculator |     75 |                 6 |                -1425 |
| 2001 | USA     | Calculator |     50 |                 7 |                -1450 |
+------+---------+------------+--------+-------------------+----------------------+
```

Window functions, functions

CUME_DIST()

DENSE_RANK()

LEAD()

LAG()

LAST_VALUE()

NTH_VALUE()

NTILE()

PERCENT_RANK()

RANK()

ROW_NUMBER()

Other features

(also interesting…)

Other features

- Explain analyze,
- spatial Data Types,
- invisible columns and
- vastly improved replication.

Overview

Overview

- Even patch version can contains interesting features.

- Check which patch version you are running.

- You should consult documentation and release notes.

Interesting sources

- MySQL release notes - https://dev.mysql.com/doc/relnotes/mysql/8.0/en/preface.html
- Percona blog - https://www.percona.com/blog/

HOW YOU PLAN TO END TALKS:
AND THUS, THE CONCLUSION IS SELF-EVIDENT.
HOW YOU ALWAYS END TALKS:
UH. SO. YEAH.
SOLUTION: CONVERT IT TO LATIN
VAH. SIC. ETIAM.
smbc-comics.com

Questions?

Slides:

http://bohuslav.simek.si/pph-mysql-new-features/

RTsoft
Softwarová řešení na míru

Informační systémy na míru

Business analytika

Mobilní aplikace

Weby a E-shopy

Jak na čistý kód a škálovatelnou architekturu kódu

1. března 2023
18:00 hod

Tech Tower

Bonus slides

Maria DB compatibility

Feature	Status	Version	Note
Generated columns	✅	10.2.	syntax compatible
JSON support	✅	10.2.7	syntax compatible, basic funcionality
instant DDL	⚠️	10.4	to some degree
common table expression	✅	10.2.1	
window functions	✅	10.2	

Maria DB compatibility

Feature	Status	Version	Note
Intersect, except	✅	10.3	Maria DB has a longer support
Multi-Valued index	❌		
Functional index	❌		
Descending index	❌		
Invisible index	✅	10.6	Ignored indexes is pretty much same