MySQL 8.0 for Developers

What's new for developers in next major version of "the world's most popular open source database"

Georgi D. Sotirov

Who am I?

- Software Development Manager & Team Leader @ Codix
- Using MySQL for more than 18 years (since 3.23.x)
- Building MySQL Server and related products for Slackware Linux for more than 13 years (check SlackPack)
- Formula 1 fan (Go Ferrari!)

- gdsotirov @

Disclaimer

The opinions expressed in this presentation are **mine** and do not necessarily reflect those of the company I'm working for.

I'm not affiliated with Oracle.

Agenda

- MySQL's time line and history with focus on development
- New features for developers in 8.0
- Other new features and improvements in 8.0
- What is still missing in MySQL (for designers and developers)

MySQL version 3.23 (Jul 1999 – Sep 2003)

- **Version 3.23** (*first alpha on 1999-07-05, production with 3.23.31 from 2001-01-17, last 3.23.58 from 2003-09-11*):
  - **MyISAM storage engine** with large file support (engine included since first alpha, but large file support is since 3.23.44 from 2001-10-31)
    - **Full-text indexing and searching** with `MATCH (col1, col2,…) AGAINST (expr)` (*since 3.23.23 from 2000-09-01*)
  - **InnoDB transactional storage engine** (*since 3.23.34 from 2001-03-10*) with "Oracle-like features". It was originally named InnoBase
    - **Foreign key** checks (*since 3.23.44 from 2001-10-31*)
    - Support for **NULL values in keys** (*since 3.23.47 from 2001-12-27*)
  - Berkeley DB (BDB) storage engine for transaction-safe tables (*since 3.23.34 from 2001-03-10*). Not supported as of 5.1
  - MERGE (or MRG_MyISAM) storage engine – collection of identical MyISAM tables - i.e. a kind of partitioning (*since 3.23.25 from 2000-09-29*)
  - Replication support for a master with many slaves

MySQL version 4.0 (Oct 2003 – Feb 2007)

- **Version 4.0** (*first alpha in October 2001, production with 4.0.12 from 2003-03-15, last 4.0.30 from 2007-02-12*):
  - InnoDB storage engine becomes standard feature
  - **Query cache** (*since first alpha*)
  - **Unions** in `SELECT` statements (*since first alpha*)
  - **Multi-table `DELETE` and `UPDATE`** statements (*since first alpha*)
  - `ORDER BY` clause on `UPDATE` and `DELETE` statements (*since first alpha*)
  - Logical and bitwise `XOR` operator (*since 4.0.2 from 2002-07-01*)
  - `SQL_CALC_FOUND_ROWS` option and `FOUND_ROWS` function (*since first alpha*)

MySQL version 4.1 (Apr 2003 – Dec 2008)

- **Version 4.1** (*first alpha 2003-04-03, production with 4.1.7 from 2004-10-23, last 4.1.25 from 2008-12-01*):
  - **Subqueries** and **derived tables** or unnamed views (*since first alpha*)
  - **Prepared statements** (*since first beta 4.1.3 from 2004-06-28*)
  - `CREATE TABLE t2 LIKE t1` statement (*since first alpha*)
  - Support for OpenGIS spatial types (geographical data) for MyISAM storage engine (*since first alpha*)
  - Insert or update with `INSERT ... ON DUPLICATE KEY UPDATE ...` Syntax (since first alpha)
  - **New storage engines**: EXAMPLE (*since 4.1.2 from 2004-05-28*), NDBCLUSTER (*since 4.1.2 from 2004-05-28*), ARCHIVE (*since 4.1.3 from 2004-06-28*), CSV (*since 4.1.4 from 2004-08-26*), BLACKHOLE (*since 4.1.11 from 2005-04-01*)
  - Function `GROUP_CONCAT` added
  - `WITH ROLLUP` modifier for `GROUP BY` clause (*since 4.1.1 from 2003-12-01*)
  - Support for **character sets and collations**
  - `HELP` statement based on help on the server (*since first alpha*)

MySQL version 5.0 (Dec 2003 – Mar 2012)

- **Version 5.0** (*first alpha 2003-12-22, production with 5.0.15 from 2005-10-19, last 5.0.96 from 2012-03-21, see release notes*):
  - **INFORMATION_SCHEMA** for standard access to MySQL metadata (*since 5.0.2 from 2004-12-01*);
  - **Stored routines** - functions and procedures (*since first alpha*):
    - no prepared statements inside procedures (*lifted with 5.0.13 from 2005-09-22*)
  - Limited support for **triggers** (*since 5.0.2 from 2004-12-01*):
    - row level only;
    - only one per event and action time (*lifted with 5.7.2 from 2013-09-21*)
    - not activated by foreign keys actions
  - **Views**, including updateable (*since 5.0.1 from 2004-07-27*)
    - No triggers on views, which limits to views not result of joins; and views not having subqueries other than in the WHERE clause
  - Elementary support for server-side **cursors**
  - **XA (distributed) transactions** for InnoDB (*since first beta 5.0.3 from 2005-03-23*)
  - BIT data type - up to 64 bits (*since first beta 5.0.3 from 2005-03-23*)
  - Federated storage engine (*since first beta 5.0.3 from 2005-03-23*)

MySQL version 5.1 (Nov 2005 – Dec 2013)

- **Version 5.1** (*first is 5.1.3 from 2005-11-29, GA with 5.1.30 from 2008-11-14, last 5.1.73 from 2013-12-03, see release notes*):
  - User-defined **table partitioning** (*since 5.1.3 from 2005-11-29*):
    - partition pruning (*since 5.1.6 from 2006-02-01*)
    - limited to tables without foreign keys and full-text indexes!
  - **XML** functions with Xpath support (*since 5.1.5 from 2006-01-10*)
  - Plugin API – e.g. full-text parsers for PDF, UDF, etc. (*since 5.1.3 from 2005-11-29*)
  - **Event scheduler** (*since 5.1.6 from 2006-02-01*)
  - *Row-based* (*since 5.1.5 from 2006-01-10*) and *mixed* (*since 5.1.8*) replication
  - Server log tables - `general_log` and `slow_log` in system schema (*since 5.1.6 from 2006-02-01*)

MySQL version 5.5 (Dec 2009 – 2018)

- **Version 5.5** (*first release on 2009-12-07, GA with 5.5.8 from 2010-12-03, last 5.5.60 from 2018-04-19, see release notes*):
  - **InnoDB** becomes **default storage engine**…
  - Improved **Unicode** support (character sets utf16, utf32 and utf8mb4 added)
  - More partitioning options - RANGE and LIST types (*since first release*)
  - **SIGNAL** and **RESIGNAL** statements - for condition (error) handling in stored routines (*since first release*)
  - XML enhancements (LOAD XML INFILE)
  - **PERFORMANCE_SCHEMA** (*since 5.5.3 from 2010-03-24*)

MySQL version 5.6 (Apr 2011 – 2018)

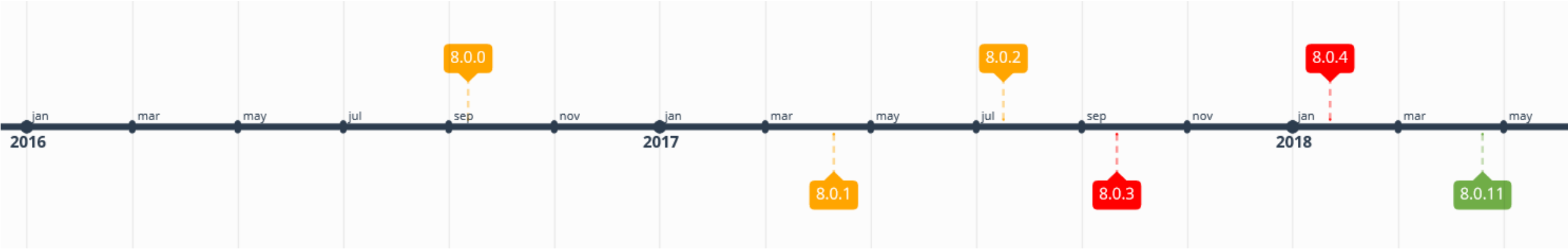
- **Version 5.6** (*first release is 5.6.2 from 2011-04-11, GA with 5.6.10 from 2013-02-05, last 5.6.40 from 2018-04-19, see release notes*):
  - **Full-text** search and indexes **on InnoDB tables** (*since 5.6.4 from 2011-12-20*)
  - New NoSQL-style **memcached API** for access to InnoDB tables (*since 5.6.6 from 2012-08-07*)
  - Explicit partition selection in `SELECT` / `INSERT` / `UPDATE` / `DELETE` / `REPLACE` and `LOAD_DATA`/`LOAD_XML`) queries with `PARTITION` (*since 5.6.2 from 2011-04-11*)
  - **Better handling of error conditions** with `GET DIAGNOSTICS` (*since 5.6.3 from 2011-10-03*)
  - Explain for `INSERT` / `UPDATE` / `DELETE` statements
  - Optimizer traces enabled with `optimizer_trace` variable and found in `OPTIMIZER_TRACE` in `INFORMATION_SCHEMA`
  - Fractional seconds now possible for `TIME`, `DATETIME` and `TIMESTAMP` values, with up to microseconds (6 digits) precision (*since 5.6.4 from 2011-12-20*)
  - OpenGIS improvements and geoJSON support
  - IPv6 manipulation functions `INET6_ATON` and `INET6_NTOA`

MySQL version 5.7 (Apr 2013 – 2018)

- **Version 5.7** (*first is 5.7.1 from 2013-04-23, GA with 5.7.9 from 2015-10-21, last 5.7.22 from 2018-04-19, see release notes*):
  - **SQL mode** now defaults to `ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ENGINE_SUBSTITUTION`
  - Spatial (geo) data types also for InnoDB (*since 5.7.1 from 2013-04-23*)
  - **JSON support** - JSON datatype and families of functions for creation, searching and modifying JSON values and attributes (*since 5.7.8 from 2015-08-03*)
  - Stacked diagnostics for condition handling (*since first official release*)
  - **Named triggers**, but still only row-level (*since 5.7.2 from 2013-09-21*)
  - **Generated columns** (*since 5.7.6 from 2015-03-09*)
  - Optimizer hints like in Oracle (*since 5.7.7 from 2015-04-08*)

MySQL 8.0's Timeline

- MySQL 8.0.0 (2016-09-12, DMR1)
- MySQL 8.0.1 (2017-04-10, DMR2)
- MySQL 8.0.2 (2017-07-17, DMR3)
- MySQL 8.0.3 (2017-09-21, RC1)
- MySQL 8.0.4 (2018-01-23, RC2)
- MySQL 8.0.5-8.0.10 (Skipped)
- **MySQL 8.0.11 (2018-04-19, GA)**



Preparatory work: dept_emp schema

```
CREATE DATABASE dept_emp;
USE dept_emp;

CREATE TABLE dept (
  deptno INTEGER,
  dname  VARCHAR(14),
  loc    VARCHAR(13),
  CONSTRAINT pk_dept PRIMARY KEY (deptno)
);
```

```
CREATE TABLE emp (
  empno    INTEGER,
  ename    VARCHAR(10),
  job      VARCHAR(9),
  mgr      INTEGER,
  hiredate DATE,
  sal      DECIMAL(7,2),
  comm     DECIMAL(7,2),
  deptno   INTEGER,
  CONSTRAINT pk_emp PRIMARY KEY (empno),
  CONSTRAINT fk_deptno FOREIGN KEY (deptno)
    REFERENCES dept (deptno)
);
```

Preparatory work: dept_emp data

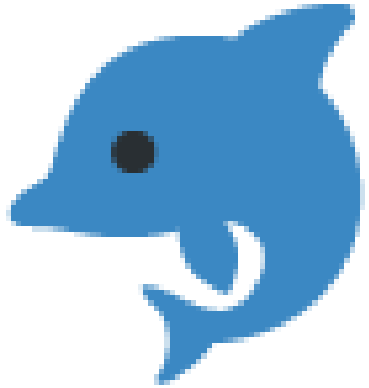
```
INSERT INTO dept VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO dept VALUES (20, 'RESEARCH'  , 'DALLAS');
INSERT INTO dept VALUES (30, 'SALES'     , 'CHICAGO');
INSERT INTO dept VALUES (40, 'OPERATIONS', 'BOSTON');

INSERT INTO emp VALUES (7839, 'KING'  , 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10);
INSERT INTO emp VALUES (7698, 'BLAKE' , 'MANAGER'  , 7839, '1981-05-01', 2850, NULL, 30);
INSERT INTO emp VALUES (7782, 'CLARK' , 'MANAGER'  , 7839, '1981-06-09', 2450, NULL, 10);
INSERT INTO emp VALUES (7566, 'JONES' , 'MANAGER'  , 7839, '1981-04-02', 2975, NULL, 20);
INSERT INTO emp VALUES (7788, 'SCOTT' , 'ANALYST'  , 7566, '1987-06-13', 3000, NULL, 20);
INSERT INTO emp VALUES (7902, 'FORD'  , 'ANALYST'  , 7566, '1981-12-03', 3000, NULL, 20);
INSERT INTO emp VALUES (7369, 'SMITH' , 'CLERK'    , 7902, '1980-12-17',  800, NULL, 20);
INSERT INTO emp VALUES (7499, 'ALLEN' , 'SALESMAN' , 7698, '1981-02-20', 1600,  300, 30);
INSERT INTO emp VALUES (7521, 'WARD'  , 'SALESMAN' , 7698, '1981-02-22', 1250,  500, 30);
INSERT INTO emp VALUES (7654, 'MARTIN', 'SALESMAN' , 7698, '1981-09-28', 1250, 1400, 30);
INSERT INTO emp VALUES (7844, 'TURNER', 'SALESMAN' , 7698, '1981-09-08', 1500,    0, 30);
INSERT INTO emp VALUES (7876, 'ADAMS' , 'CLERK'    , 7788, '1987-06-13', 1100, NULL, 20);
INSERT INTO emp VALUES (7900, 'JAMES' , 'CLERK'    , 7698, '1981-12-03',  950, NULL, 30);
INSERT INTO emp VALUES (7934, 'MILLER', 'CLERK'    , 7782, '1982-01-23', 1300, NULL, 10);
```

New features for developers

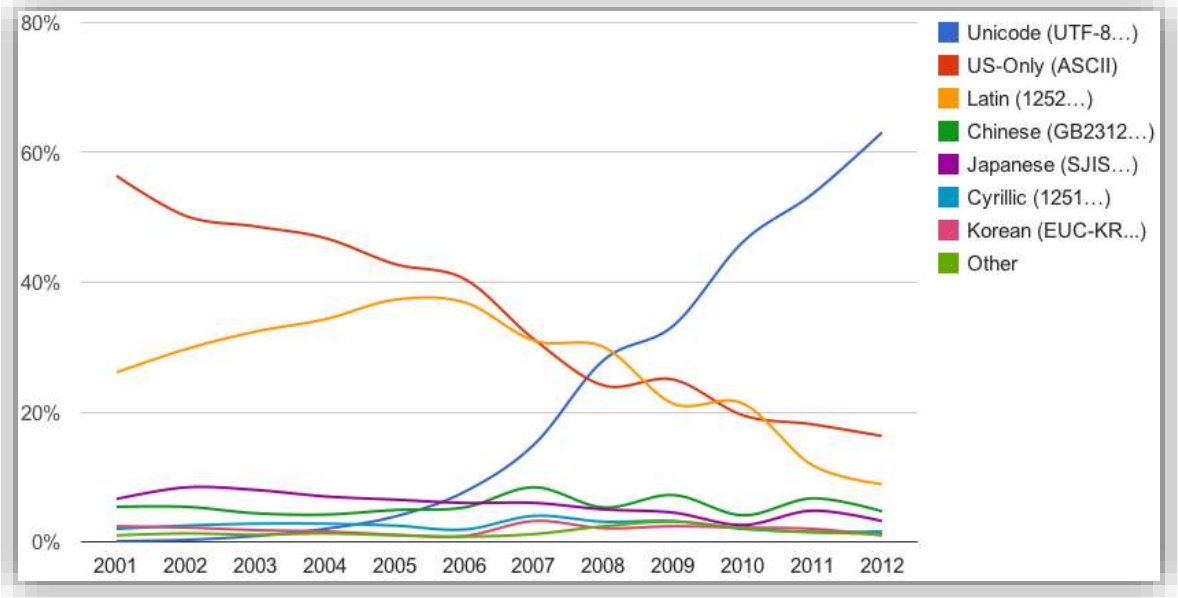
- **New default character set**
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes

Default character set

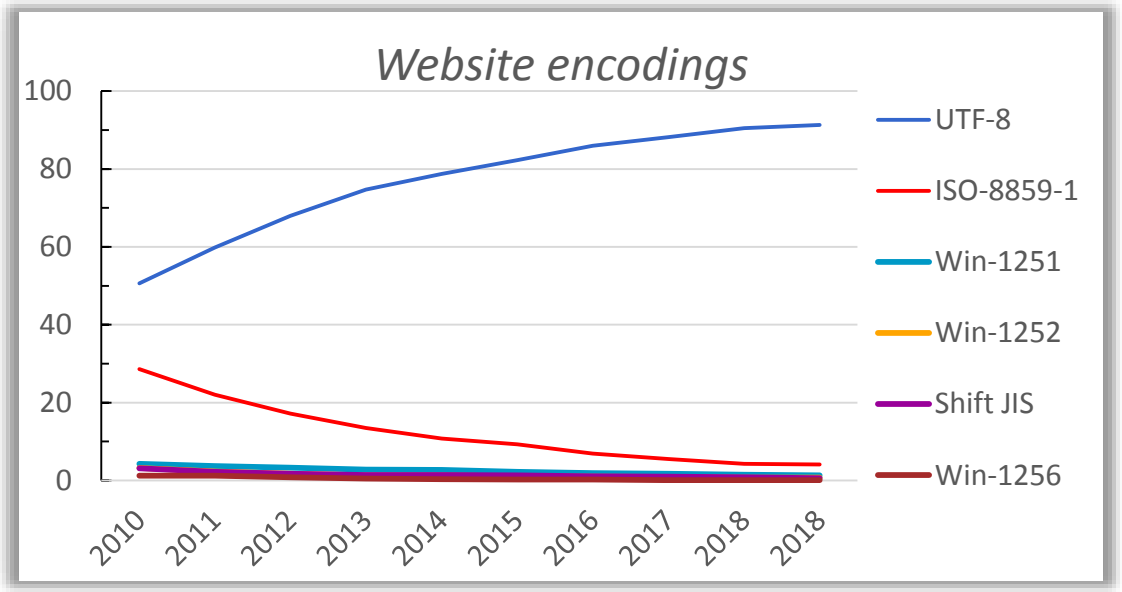
- Default character set in MySQL 8.0 is **utf8mb4**
- New accent and case sensitive collations (%_as_cs)
- Unicode 9 support
- Focus on modern web and mobile applications (and of course emoji)



U+1F42C



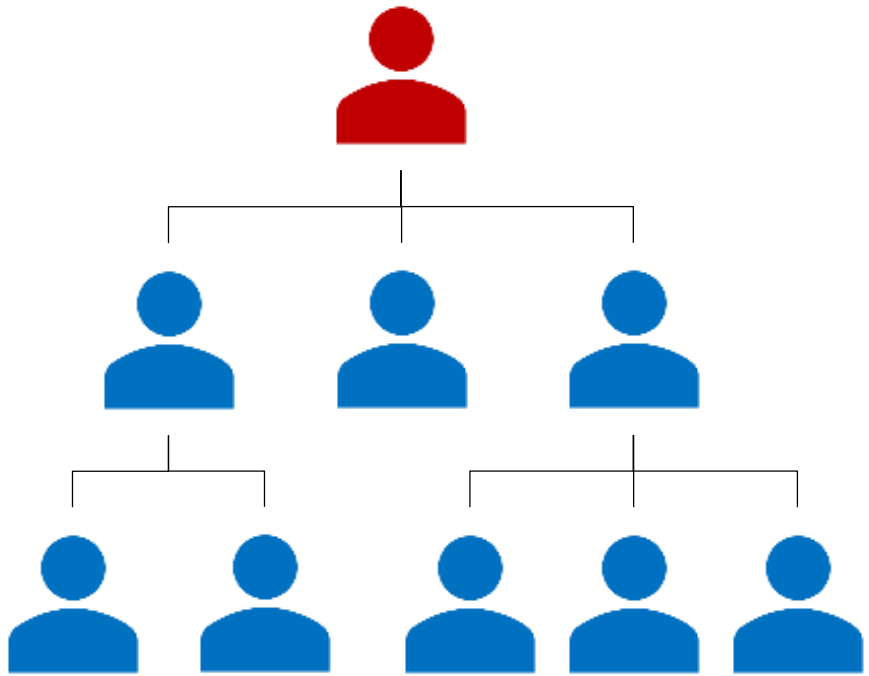
Source: Google



Source: W3Tech

New features for developers

- New default character set
- **Common Table Expressions**
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes



CTE (1/7): Definition and syntax

SQL:1999

- Common table expressions (CTE) allow a temporary result set to be referred within another SQL statement

- Basic syntax (for simplification of complex queries)
- Recursive syntax (for generation of data or traversing hierarchies)

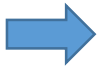
```
WITH cte1 AS (subquery), [cte2 AS (subquery)] ...
```

```
WITH RECURSIVE cte AS
(SELECT ... /* initial row set or "seed" */
 UNION ALL
 SELECT ... /* additional row sets */
)
...
```

CTE (2/7): Example 1

- Sequence years since 1989

```
SELECT SEQ.Year
  FROM (SELECT (THOUSANDS.SeqValue + HUNDREDS.SeqValue
                + TENS.SeqValue + ONES.SeqValue) Year
          FROM (SELECT 0 SeqValue
                  UNION ALL SELECT 1 SeqValue UNION ALL SELECT 2 SeqValue
                  UNION ALL SELECT 3 SeqValue UNION ALL SELECT 4 SeqValue
                  UNION ALL SELECT 5 SeqValue UNION ALL SELECT 6 SeqValue
                  UNION ALL SELECT 7 SeqValue UNION ALL SELECT 8 SeqValue
                  UNION ALL SELECT 9 SeqValue) ONES CROSS JOIN
                (SELECT 0 SeqValue
                  UNION ALL SELECT 10 SeqValue UNION ALL SELECT 20 SeqValue
                  UNION ALL SELECT 30 SeqValue UNION ALL SELECT 40 SeqValue
                  UNION ALL SELECT 50 SeqValue UNION ALL SELECT 60 SeqValue
                  UNION ALL SELECT 70 SeqValue UNION ALL SELECT 80 SeqValue
                  UNION ALL SELECT 90 SeqValue) TENS CROSS JOIN
                (SELECT 0 SeqValue
                  UNION ALL SELECT 100 SeqValue UNION ALL SELECT 200 SeqValue
                  UNION ALL SELECT 300 SeqValue UNION ALL SELECT 400 SeqValue
                  UNION ALL SELECT 500 SeqValue UNION ALL SELECT 600 SeqValue
                  UNION ALL SELECT 700 SeqValue UNION ALL SELECT 800 SeqValue
                  UNION ALL SELECT 900 SeqValue) HUNDREDS CROSS JOIN
                (SELECT 0 SeqValue
                  UNION ALL SELECT 1000 SeqValue
                  UNION ALL SELECT 2000 SeqValue) THOUSANDS
       ) SEQ
 WHERE SEQ.Year BETWEEN 1989 AND YEAR(NOW())
 ORDER BY SEQ.Year;
```



MySQL 8.0.1 and newer!

```
WITH RECURSIVE years AS
(
   SELECT 1989 AS yr
   UNION ALL
   SELECT yr + 1 FROM years
    WHERE yr < YEAR(NOW())
)
SELECT yr FROM years;
```

CTE (3/7): Example 1 Explain plan

- Explain plan comparison

Cost: 343,79

```
+----+-------------+-------------++------+----------+-----------------------------------------------------+
| id | select_type | table       || rows | filtered | Extra                                               |
+----+-------------+-------------++------+----------+-----------------------------------------------------+
|  1 | PRIMARY     | <derived33> ||    3 |   100.00 | Using temporary; Using filesort                     |
|  1 | PRIMARY     | <derived3>  ||   10 |   100.00 | Using join buffer (Block Nested Loop)               |
|  1 | PRIMARY     | <derived13> ||   10 |   100.00 | Using join buffer (Block Nested Loop)               |
|  1 | PRIMARY     | <derived23> ||   10 |   100.00 | Using where; Using join buffer (Block Nested Loop)  |
| 33 | DERIVED     | NULL        || NULL |     NULL | No tables used                                      |
| 34 | UNION       | NULL        || NULL |     NULL | No tables used                                      |
| 35 | UNION       | NULL        || NULL |     NULL | No tables used                                      |
| 23 | DERIVED     | NULL        || NULL |     NULL | No tables used                                      |
...
| 11 | UNION       | NULL        || NULL |     NULL | No tables used                                      |
| 12 | UNION       | NULL        || NULL |     NULL | No tables used                                      |
+----+-------------+-------------++------+----------+-----------------------------------------------------+
37 rows in set, 1 warning (0.00 sec)
```

Cost: 2,84

```
+----+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------------+
| id | select_type | table      | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL | NULL    | NULL |    3 |   100.00 | NULL                  |
|  2 | DERIVED     | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | No tables used        |
|  3 | UNION       | years      | ALL  | NULL          | NULL | NULL    | NULL |    2 |    50.00 | Recursive; Using where |
+----+-------------+------------+------+---------------+------+---------+------+------+----------+-----------------------+
3 rows in set, 1 warning (0.00 sec)
```

CTE (4/7): Example 2

- Generate chain of command for the employees (traversing hierarchy top-down)
- Top ranked employee has no manager (i.e. mgr IS NULL), all others report to someone else

```
WITH RECURSIVE employee_chain (empno, ename, deptno, job, mgr, `rank`, `path`) AS
(
  SELECT empno, ename, deptno, job, empno, 1, CAST(ename AS CHAR(128))
    FROM emp
   WHERE mgr IS NULL
  UNION ALL
  SELECT E.empno, E.ename, E.deptno, E.job, E.mgr, `rank`+1, CONCAT(EC.`path`, ' <- ', E.ename)
    FROM employee_chain EC,
         emp            E
   WHERE E.mgr = EC.empno
)
SELECT ECTE.empno, ECTE.ename, D.dname, ECTE.job, ECTE.`rank`, ECTE.`path`
  FROM employee_chain ECTE,
       dept           D
 WHERE D.deptno = ECTE.deptno
 ORDER BY D.dname, ECTE.`rank`;
```

CTE (5/7): Example 2 Result

empno	ename	dname	job	rank	path
7839	KING	ACCOUNTING	PRESIDENT	1	KING
7782	CLARK	ACCOUNTING	MANAGER	2	KING <- CLARK
7934	MILLER	ACCOUNTING	CLERK	3	KING <- CLARK <- MILLER
7566	JONES	RESEARCH	MANAGER	2	KING <- JONES
7788	SCOTT	RESEARCH	ANALYST	3	KING <- JONES <- SCOTT
7902	FORD	RESEARCH	ANALYST	3	KING <- JONES <- FORD
7369	SMITH	RESEARCH	CLERK	4	KING <- JONES <- FORD <- SMITH
7876	ADAMS	RESEARCH	CLERK	4	KING <- JONES <- SCOTT <- ADAMS
7698	BLAKE	SALES	MANAGER	2	KING <- BLAKE
7499	ALLEN	SALES	SALESMAN	3	KING <- BLAKE <- ALLEN
7521	WARD	SALES	SALESMAN	3	KING <- BLAKE <- WARD
7654	MARTIN	SALES	SALESMAN	3	KING <- BLAKE <- MARTIN
7844	TURNER	SALES	SALESMAN	3	KING <- BLAKE <- TURNER
7900	JAMES	SALES	CLERK	3	KING <- BLAKE <- JAMES

14 rows in set (0.01 sec)

CTE (6/7): Example 3

- Get the chain of command for a single employee (traverse hierarchy bottom-up)

```
WITH RECURSIVE chain_cmd (empno, mgr, `path`) AS
(
SELECT empno, mgr, CAST(ename AS CHAR(200))
  FROM emp
 WHERE empno = 7788 /* SCOTT */
UNION ALL
SELECT EMP.empno, EMP.mgr, CONCAT(CCMD.`path`, ' -> ', EMP.ename)
  FROM chain_cmd CCMD,
       emp       EMP
 WHERE EMP.empno = CCMD.mgr
)
SELECT *
  FROM chain_cmd
 WHERE mgr IS NULL;
```

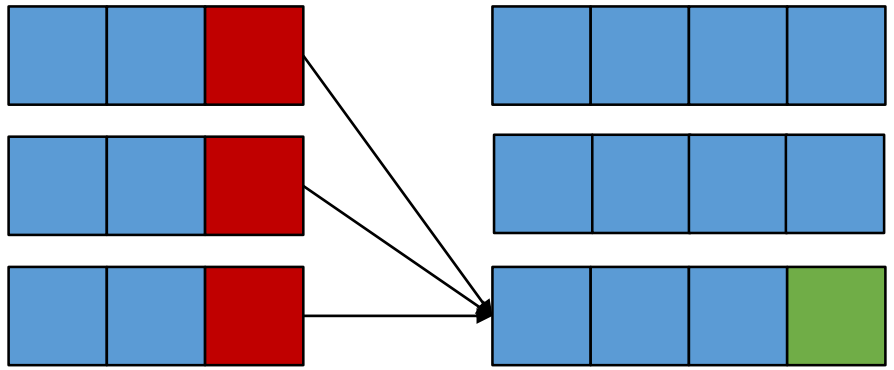
```
+-------+------+-----------------------+
| empno | mgr  | path                  |
+-------+------+-----------------------+
|  7839 | NULL | SCOTT -> JONES -> KING |
+-------+------+-----------------------+
1 row in set (0.00 sec)
```

CTE (7/7): Security

- What if a CTE omits a condition to terminate the recursion?
- MySQL provides:
  - New system variable `cte_max_recursion_depth` defines maximum recursion levels (default 1000)
  - System variable `max_execution_time` defines execution timeout for SELECT statements in the current session (default 0, so no limit)
  - Optimizer hint `MAX_EXECUTION_TIME` for a per-query execution limit
- Always control your CTEs recursion depth!
- Otherwise, Ctrl+C or `KILL QUERY` from another session

New features for developers

- New default character set
- Common Table Expressions
- **Window functions**
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes



Window functions (1/7)

SQL:2003

- Analytic functions that perform calculation across a set of rows related to the current row (e.g. aggregation)
- The calculation is performed for each row in the result set, so rows are not grouped into a single output row
- Two kinds:
  - SQL aggregate functions (e.g. `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, etc.)
  - Specialized window functions (e.g. `RANK`, `NTILE`, `ROW_NUMBER`, etc.)
- New keywords:
  - **`OVER`** to signal window function
  - **`PARTITION BY`** to limit the result set "seen" by window functions
  - **`ORDER BY`** with **`RANGE`** for *logical* and **`ROWS`** for *physical* frame boundaries
  - **`CURRENT ROW`**, **`UNBOUNDED PRECEDING`**, **`UNBOUNDED FOLLOWING`**, **`expr PRECEDING`** and **`expr FOLLOWING`** for specifying frame extents

Window functions (2/7): Example 1

- Employees report with total salary and commission per department

```
SELECT E.ename, E.job, D.dname, E.sal,
       dept_sal, E.comm, dept_comm
  FROM emp  E,
       dept D,
       (SELECT deptno, SUM(sal) dept_sal
          FROM emp
         GROUP BY deptno) DSAL,
       (SELECT deptno,
               SUM(COALESCE(comm, 0)) dept_comm
          FROM emp
         GROUP BY deptno) DCOM
 WHERE E.deptno    = D.deptno
   AND DSAL.deptno = D.deptno
   AND DCOM.deptno = D.deptno;
```

MySQL 8.0.2 and newer!

```
SELECT E.ename, E.job, D.dname,
       E.sal, SUM(E.sal)
         OVER (PARTITION BY E.deptno) AS dept_sal,
       E.comm, SUM(COALESCE(E.comm, 0))
         OVER (PARTITION BY E.deptno) AS dept_comm
  FROM emp  E,
       dept D
 WHERE E.deptno = D.deptno;
```

Window functions (3/7): Example 1 Result

- Same result with derived tables and window functions

ename	job	dname	sal	dept_sal	comm	dept_com
CLARK	MANAGER	ACCOUNTING	2450.00	8750.00	NULL	0.00
KING	PRESIDENT	ACCOUNTING	5000.00	8750.00	NULL	0.00
MILLER	CLERK	ACCOUNTING	1300.00	8750.00	NULL	0.00
SMITH	CLERK	RESEARCH	800.00	10875.00	NULL	0.00
JONES	MANAGER	RESEARCH	2975.00	10875.00	NULL	0.00
SCOTT	ANALYST	RESEARCH	3000.00	10875.00	NULL	0.00
ADAMS	CLERK	RESEARCH	1100.00	10875.00	NULL	0.00
FORD	ANALYST	RESEARCH	3000.00	10875.00	NULL	0.00
ALLEN	SALESMAN	SALES	1600.00	9400.00	300.00	2200.00
WARD	SALESMAN	SALES	1250.00	9400.00	500.00	2200.00
MARTIN	SALESMAN	SALES	1250.00	9400.00	1400.00	2200.00
BLAKE	MANAGER	SALES	2850.00	9400.00	NULL	2200.00
TURNER	SALESMAN	SALES	1500.00	9400.00	0.00	2200.00
JAMES	CLERK	SALES	950.00	9400.00	NULL	2200.00

14 rows in set (0.00 sec)

Window functions (4/7): Explain plan

- Explain plan between derived tables and window functions

Cost: 25,27

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	D	ALL	PRIMARY	NULL	NULL	NULL	4	100.00	Using where
1	PRIMARY	<derived2>	ref	<auto_key0>	<auto_key0>	5	dept_emp.D.deptno	2	100.00	NULL
1	PRIMARY	<derived3>	ref	<auto_key0>	<auto_key0>	5	dept_emp.D.deptno	2	100.00	NULL
1	PRIMARY	E	ref	fk_deptno	fk_deptno	5	dept_emp.D.deptno	4	100.00	NULL
3	DERIVED	emp	index	fk_deptno	fk_deptno	5	NULL	14	100.00	NULL
2	DERIVED	emp	index	fk_deptno	fk_deptno	5	NULL	14	100.00	NULL

Cost: 22,25

id	select_type	table	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	E	ALL	fk_deptno	NULL	NULL	NULL	14	100.00	Using temporary; Using filesort
1	SIMPLE	D	ALL	PRIMARY	NULL	NULL	NULL	4	25.00	Using where; Using join buffer (Block Nested Loop)

Window functions (5/7): Example 2

```
SELECT E.ename, E.job, D.dname,
       E.sal, SUM(E.sal)
         OVER (PARTITION BY E.deptno ORDER BY E.ename) AS dept_sal,
       E.comm, SUM(COALESCE(E.comm, 0))
         OVER (PARTITION BY E.deptno ORDER BY E.ename) AS dept_com
  FROM emp  E,
       dept D
 WHERE E.deptno = D.deptno
 ORDER BY D.dname, E.ename;
```

Note: Just **ORDER BY** equals **RANGE BETWEEN** UNBOUNDED PRECEDING **AND** CURRENT ROW

```
+--------+----------+------------+---------+----------+---------+----------+
| ename  | job      | dname      | sal     | dept_sal | comm    | dept_com |
+--------+----------+------------+---------+----------+---------+----------+
| CLARK  | MANAGER  | ACCOUNTING | 2450.00 |  2450.00 |    NULL |     0.00 |
| KING   | PRESIDENT| ACCOUNTING | 5000.00 |  7450.00 |    NULL |     0.00 |
| MILLER | CLERK    | ACCOUNTING | 1300.00 |  8750.00 |    NULL |     0.00 |
| ADAMS  | CLERK    | RESEARCH   | 1100.00 |  1100.00 |    NULL |     0.00 |
| FORD   | ANALYST  | RESEARCH   | 3000.00 |  4100.00 |    NULL |     0.00 |
| JONES  | MANAGER  | RESEARCH   | 2975.00 |  7075.00 |    NULL |     0.00 |
| SCOTT  | ANALYST  | RESEARCH   | 3000.00 | 10075.00 |    NULL |     0.00 |
| SMITH  | CLERK    | RESEARCH   |  800.00 | 10875.00 |    NULL |     0.00 |
| ALLEN  | SALESMAN | SALES      | 1600.00 |  1600.00 |  300.00 |   300.00 |
| BLAKE  | MANAGER  | SALES      | 2850.00 |  4450.00 |    NULL |   300.00 |
| JAMES  | CLERK    | SALES      |  950.00 |  5400.00 |    NULL |   300.00 |
| MARTIN | SALESMAN | SALES      | 1250.00 |  6650.00 | 1400.00 |  1700.00 |
| TURNER | SALESMAN | SALES      | 1500.00 |  8150.00 |    0.00 |  1700.00 |
| WARD   | SALESMAN | SALES      | 1250.00 |  9400.00 |  500.00 |  2200.00 |
+--------+----------+------------+---------+----------+---------+----------+
14 rows in set (0.00 sec)
```

Window functions (6/7): Examples 3 & 4

- Named window, specialized window functions

```
SELECT sal,
       RANK()                OVER win     AS 'rank',
       DENSE_RANK()          OVER win     AS 'drank',
       NTILE(4)              OVER win     AS 'ntile',
       ROUND(CUME_DIST()     OVER win, 2) AS 'cdist',
       ROUND(PERCENT_RANK() OVER win, 2) AS 'prank'
  FROM emp
WINDOW win AS (ORDER BY sal DESC);
```

```
+---------+------+-------+-------+-------+-------+
| sal     | rank | drank | ntile | cdist | prank |
+---------+------+-------+-------+-------+-------+
| 5000.00 |    1 |     1 |     1 |  0.07 |  0.00 |
| 3000.00 |    2 |     2 |     1 |  0.21 |  0.08 |
| 3000.00 |    2 |     2 |     1 |  0.21 |  0.08 |
| 2975.00 |    4 |     3 |     1 |  0.29 |  0.23 |
| 2850.00 |    5 |     4 |     2 |  0.36 |  0.31 |
| 2450.00 |    6 |     5 |     2 |  0.43 |  0.38 |
| 1600.00 |    7 |     6 |     2 |  0.50 |  0.46 |
| 1500.00 |    8 |     7 |     2 |  0.57 |  0.54 |
| 1300.00 |    9 |     8 |     3 |  0.64 |  0.62 |
| 1250.00 |   10 |     9 |     3 |  0.79 |  0.69 |
| 1250.00 |   10 |     9 |     3 |  0.79 |  0.69 |
| 1100.00 |   12 |    10 |     4 |  0.86 |  0.85 |
|  950.00 |   13 |    11 |     4 |  0.93 |  0.92 |
|  800.00 |   14 |    12 |     4 |  1.00 |  1.00 |
+---------+------+-------+-------+-------+-------+
14 rows in set (0.00 sec)
```

- Generate row numbers (per partition!)

```
SELECT ROW_NUMBER() OVER() AS rnum,
       dname
  FROM dept
 LIMIT 2;
```

```
+------+------------+
| rnum | dname      |
+------+------------+
|    1 | ACCOUNTING |
|    2 | RESEARCH   |
+------+------------+
2 rows in set (0.00 sec)
```

```
SELECT ROW_NUMBER() OVER() AS rnum,
       dname
  FROM dept
 LIMIT 2 OFFSET 2;
```

```
+------+------------+
| rnum | dname      |
+------+------------+
|    3 | SALES      |
|    4 | OPERATIONS |
+------+------------+
2 rows in set (0.00 sec)
```

Window functions (7/7): Sliding average

- Generate sales data with a CTE

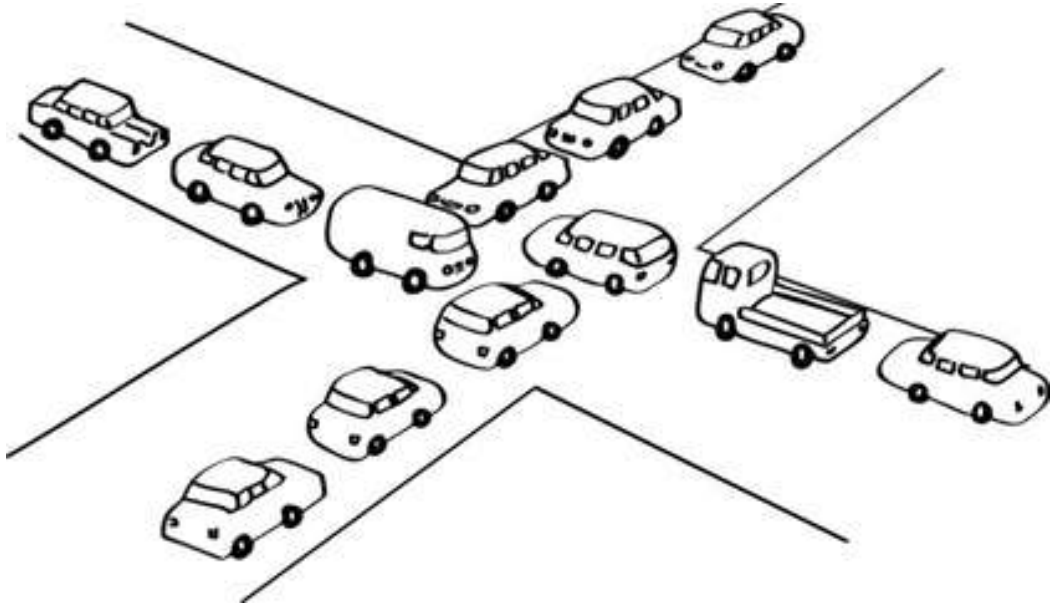
```
WITH RECURSIVE sales (dat, amt) AS
(SELECT CAST('2018-01-01' AS DATE) AS dat,
        CAST(100 AS DECIMAL(10,2)) AS amt
 UNION ALL
 SELECT DATE_ADD(dat, INTERVAL 15 DAY), amt + 100
   FROM sales
  WHERE MONTH(DATE_ADD(dat, INTERVAL 15 DAY)) < 6
)
SELECT * FROM sales;
+------------+---------+
| dat        | amt     |
+------------+---------+
| 2018-01-01 |  100.00 |
| 2018-01-16 |  200.00 |
| 2018-01-31 |  300.00 |
| 2018-02-15 |  400.00 |
| 2018-03-02 |  500.00 |
| 2018-03-17 |  600.00 |
| 2018-04-01 |  700.00 |
| 2018-04-16 |  800.00 |
| 2018-05-01 |  900.00 |
| 2018-05-16 | 1000.00 |
| 2018-05-31 | 1100.00 |
+------------+---------+
11 rows in set (0.00 sec)
```

- Calculate sliding average on sales per month

```
/* CTE here */
SELECT MONTH(dat) mnth, SUM(amt) tot_mnth,
       SUM(SUM(amt))
         OVER(ORDER BY MONTH(dat)) tot_cum,
       ROUND(AVG(SUM(amt))
             OVER(ORDER BY MONTH(dat)
                  RANGE BETWEEN 1 PRECEDING
                            AND 1 FOLLOWING
                 ),
             2) sliding_avg
  FROM sales
 GROUP BY MONTH(dat);
+------+----------+---------+-------------+
| mnth | tot_mnth | tot_cum | sliding_avg |
+------+----------+---------+-------------+
|    1 |   600.00 |  600.00 |      500.00 |
|    2 |   400.00 | 1000.00 |      700.00 |
|    3 |  1100.00 | 2100.00 |     1000.00 |
|    4 |  1500.00 | 3600.00 |     1866.67 |
|    5 |  3000.00 | 6600.00 |     2250.00 |
+------+----------+---------+-------------+
5 rows in set (0.00 sec)
```

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- **Locking Read Concurrency**
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes



Locking Read Concurrency (1/2)

- Locking reads exist in MySQL since long ago, but there were no ways for handling “hot rows”
- **`NOWAIT`** and **`SKIP LOCKED`** are now possible on **`SELECT ... FOR SHARE`** and **`SELECT ... FOR UPDATE`** statements
- A locking read with **`NOWAIT`** never waits for lock. Query executes immediately and fails if row is locked (otherwise it would wait until innodb_lock_wait_timeout, which defaults to 50s)
- A locking read with **`SKIP LOCKED`** never waits for lock. Query executes immediately, removing locked rows from the result set
- **`SELECT ... FOR SHARE`** replaces **`SELECT ... LOCK IN SHARE MODE`** (still available for backwards compatibility)

Locking Read Concurrency (2/2): Example

Session 1

```
START TRANSACTION;

SELECT * FROM dept WHERE
deptno = 20 FOR UPDATE;
```

Session 2

```
START TRANSACTION;

SELECT * FROM dept WHERE deptno = 20 FOR UPDATE NOWAIT;
ERROR 3572 (HY000): Statement aborted because lock(s)
could not be acquired immediately and NOWAIT is set.
SELECT deptno FROM dept FOR UPDATE SKIP LOCKED;
+--------+
| deptno |
+--------+
|     10 |
|     30 |
|     40 |
+--------+
3 rows in set (0.00 sec)
```

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- **Regular expression improvements**
- GROUPING function
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes

/^Regular expres{2}ion$/

Regular expr. improvements (1/2)

- Re-implemented using International Components for Unicode (ICU) for full Unicode support, multibyte safety and security
- New functions `REGEXP_LIKE` (`REGEXP` and `RLIKE` operators are now synonyms to this function), `REGEXP_INSTR`, `REGEXP_REPLACE`, and `REGEXP_SUBSTR`
- Possibility to specify *match type*: `c` – case sensitive, `i` – case insensitive, `m` – multi-line, `n` – . (any character) matches also line terminators, `u` – Unix-only line endings (only new line)
- New system variables (since 8.0.4 from 2018-01-23):
  - `regexp_stack_limit` – max memory in bytes for the internal stack used for regular expression matching operations
  - `regexp_time_limit` – the time limit for regular expression matching operations

Regular expr. improvements (2/2): Examples

MySQL 5.7

- *Multi-byte strings*

```
SELECT '周天月年历' RLIKE '^.{5}$';
-> 0
```

- *Match type*

```
SELECT 'abc' RLIKE 'ABC';
-> 1
```

MySQL 8.0

- *Multi-byte strings*

```
SELECT REGEXP_LIKE('周天月年历', '^.{5}$');
-> 1
```

- *Match type*

```
SELECT REGEXP_LIKE('abc', 'ABC', 'ic');
-> 0
```

Note: For contradictory options the rightmost takes precedence!

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- **GROUPING function**
- JSON enhancements
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes

GROUPING function (1/2)

- `WITH ROLLUP` modifier was added with MySQL 4.1 in 2003
- For generation of extra output rows from higher-level (supper-aggregate) summary operations
- In `GROUP BY ... WITH ROLLUP` queries NULL represents all values
- So how to tell `NULL` in column data from `NULL` in the result set?
- The answer: `GROUPING` function
  - result is 1 for super-aggregate values; and
  - result is 0 for NULL in data
- Could be used in `SELECT` list and `HAVING` clause

GROUPING function (2/2): Example

- Find the total salary per department and organization together

```
SELECT COALESCE(D.dname, 'Grand Total') Dept,
       SUM(COALESCE(E.sal, 0)) Total
  FROM emp  E
       LEFT OUTER JOIN dept D
       ON E.deptno = D.deptno
GROUP BY D.dname WITH ROLLUP;
```

```
+-------------+----------+
| Dept        | Total    |
+-------------+----------+
| ACCOUNTING  |  8750.00 |
| RESEARCH    | 10875.00 |
| SALES       |  9400.00 |
| Grand Total | 29025.00 |
+-------------+----------+
4 rows in set (0.00 sec)
```

```
INSERT INTO emp
  (..., sal, deptno)
VALUES
  (..., 1600, NULL);
```

```
+-------------+----------+
| Dept        | Total    |
+-------------+----------+
| Grand Total |  1600.00 |
| ACCOUNTING  |  8750.00 |
| RESEARCH    | 10875.00 |
| SALES       |  9400.00 |
| Grand Total | 30625.00 |
+-------------+----------+
5 rows in set (0.00 sec)
```

```
CASE GROUPING(D.dname)
  WHEN 1 THEN
    'Grand Total'
  ELSE
    COALESCE(D.dname, 'No department')
END Dept
```

```
+---------------+----------+
| Dept          | Total    |
+---------------+----------+
| No department |  1600.00 |
| ACCOUNTING    |  8750.00 |
| RESEARCH      | 10875.00 |
| SALES         |  9400.00 |
| Grand Total   | 30625.00 |
+---------------+----------+
5 rows in set (0.00 sec)
```

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- **JSON enhancements**
- The Document store
- GIS improvements
- Datatype improvements
- Descending indexes

JSON (1/4): Enhancements summary

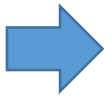
SQL:2016

- Extended syntax
  - New `column->>path` (inline path) operator equivalent to `JSON_UNQUOTE` after `JSON_EXTRACT`
  - Added support for ranges such as `$[1 to 3]`, indexes like `$[last]`, `$[last-1]` as well as `$[last-2 to last-1]` in XPath expressions
- New functions:
  - aggregation functions `JSON_ARRAYAGG` and `JSON_OBJECTAGG`
  - new merge function `JSON_MERGE_PATCH` (as union, so duplicate keys are removed), old `JSON_MERGE` renamed to `JSON_MERGE_PRESERVE`
  - new `JSON_TABLE` function for transforming JSON data as relational table
  - utility function `JSON_PRETTY`, `JSON_STORAGE_SIZE` and `JSON_STORAGE_FREE`
- Improved sorting - JSON values are now variable length in the sort key
- Better performance with partial (in-place) updates when using `JSON_SET`, `JSON_REPLACE` or `JSON_REMOVE` functions

JSON (2/4): Example 1 JSON_ARRAYAGG

- Read departments as array of JSON objects

```
SELECT JSON_PRETTY(
         JSON_ARRAYAGG(
           JSON_OBJECT("id"    , deptno,
                       "dname", dname,
                       "loc"   , loc
                      )
         )
       ) jsn
  FROM dept;
```



```
[
  {
    "id": 10,
    "loc": "NEW YORK",
    "dname": "ACCOUNTING"
  },
  {
    "id": 20,
    "loc": "DALLAS",
    "dname": "RESEARCH"
  },
  {
    "id": 30,
    "loc": "CHICAGO",
    "dname": "SALES"
  },
  {
    "id": 40,
    "loc": "BOSTON",
    "dname": "OPERATIONS"
  }
]
```

JSON (3/4): Example 2 JSON_TABLE

- Read departments table from JSON

```
SELECT *
  FROM JSON_TABLE('[{"id": 10,"loc": "NEW YORK","dname": "ACCOUNTING"},...',
         "$[*]" COLUMNS (
           id    INT          PATH "$.id",
           dname VARCHAR(32)  PATH "$.dname",
           loc   VARCHAR(32)  PATH "$.loc"
         )
       ) AS json_dept;
...
 WHERE id > 10;
```

```
+------+------------+----------+
| id   | dname      | loc      |
+------+------------+----------+
|   10 | ACCOUNTING | NEW YORK |
|   20 | RESEARCH   | DALLAS   |
|   30 | SALES      | CHICAGO  |
|   40 | OPERATIONS | BOSTON   |
+------+------------+----------+
4 rows in set (0.00 sec)
```

JSON (4/4): More on partial update

- Columns of JSON data type only
- Works only with JSON_SET, JSON_REPLACE or JSON_REMOVE functions on the same column, direct assignment doesn't work
- Only existing array or object values, no new elements
- New value must be no larger than the old one

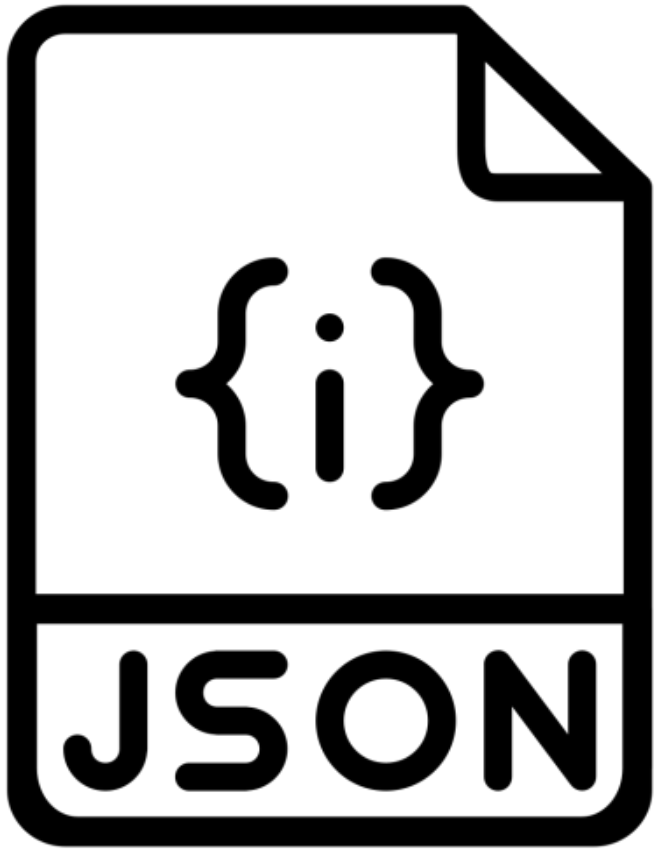
```
UPDATE json_table
   SET json_data = '{"id": 10, "loc": "NEW YORK", "dname": "ACCOUNTING"}'
 WHERE ...
```

```
UPDATE json_table
   SET json_data = JSON_SET(json_data, "$.name", 'ACCOUNTING & FINANCE')
 WHERE ...
```

```
UPDATE json_table
   SET json_data = JSON_SET(json_data, "$.name", 'REPAIRS')
 WHERE ...
```

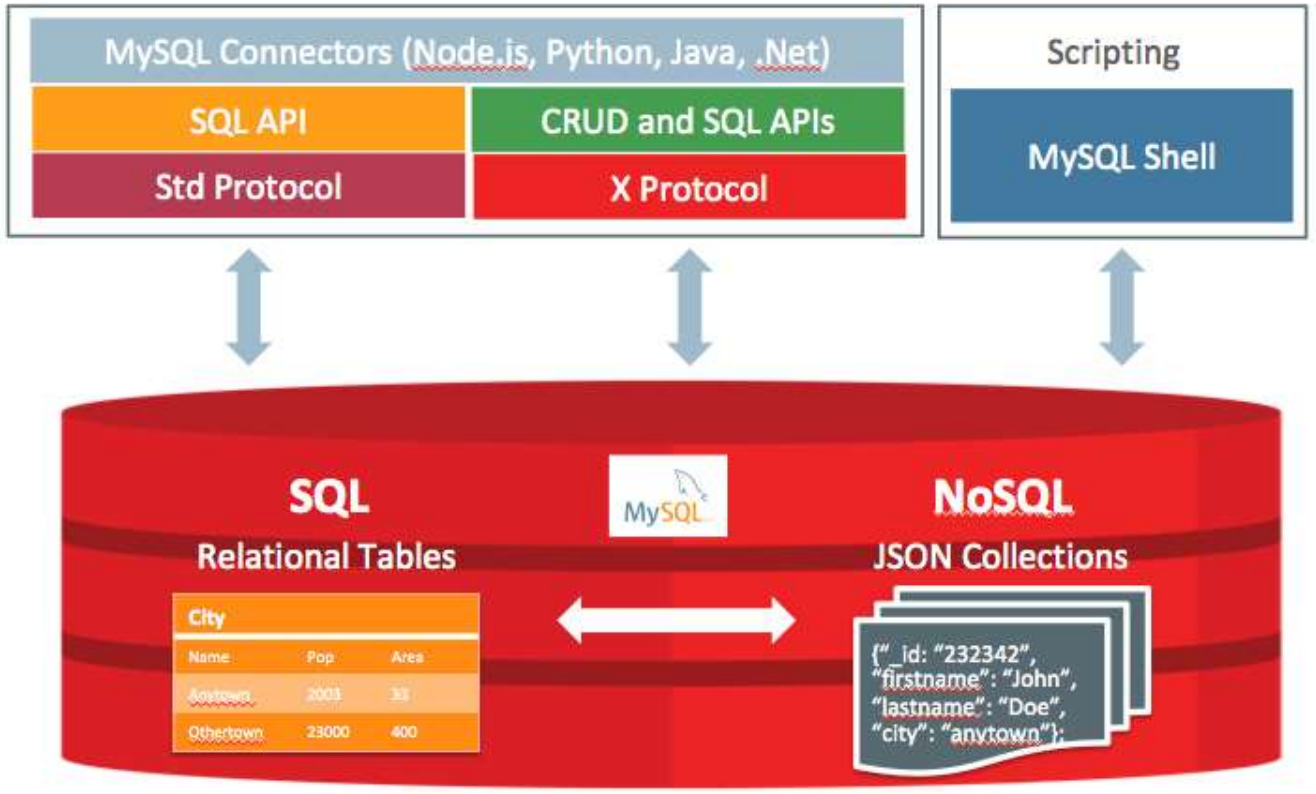
New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- **The Document store**
- GIS improvements
- Datatype improvements
- Descending indexes



The Document store (1/8): General

- GA and default with 8.0, but available as plugin since 5.7.12 (2016-04-11)
- Schema-less (no SQL knowledge required), rapid prototyping
- The *document* is set of key and value pairs (i.e. a JSON object)
- The *collection* is a container for documents
- X Protocol (default port 33060)
- **C**reate/**R**ead/**U**pdate/**D**elete operations through X DevAPI
- MySQL Connectors (Node.js, Python, C++, Java, .Net), no ODBC support)
- MySQL Shell (commands and scripting)



The Document store (2/8): Example Connecting

```
MySQL Shell 8.0.11
Copyright (c) 2016, 2018, Oracle and/or its affiliates. All rights reserved.
Type '\help' or '\?' for help; '\quit' to exit.

MySQL JS> session
MySQL JS> \connect root@localhost
Creating a session to 'root@localhost'
Enter password: ************
Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 1024235 (X protocol)
Server version: 5.7.22-log Source distribution
No default schema selected; type \use <schema> to set one.

MySQL [localhost+ ssl] JS> session
<Session:root@localhost>
```

The Document store (3/8): Example Working with objects (1/3)

```
MySQL [localhost+ ssl] JS> session.createSchema('docstore')
<Schema:docstore>
MySQL [localhost+ ssl/docstore] JS> db
<Schema:docstore>
MySQL [localhost+ ssl/docstore] JS> dcol = db.createCollection('dept')
<Collection:dept>
MySQL [localhost+ ssl/docstore] JS> dcol.add([{_id:10,dname:"ACCOUNTING"},
                                              {_id:20,dname:"RESEARCH"},
                                              {_id:30,dname:"SALES"},
                                              {_id:40,dname:"OPERATIONS"}])

Query OK, 4 items affected (0.2549 sec)
MySQL [localhost+ ssl/docstore] JS> dcol.add({_id:50,dname:"REPAIRS"})
Query OK, 1 item affected (0.1997 sec)
```

The Document store (4/8): Example Working with objects (2/3)

```
MySQL [localhost+ ssl/docstore] JS> dcol.find()
[   {"_id": 10, "dname": "ACCOUNTING"},
    {"_id": 20, "dname": "RESEARCH"},
    {"_id": 30, "dname": "SALES"},
    {"_id": 40, "dname": "OPERATIONS"},
    {"_id": 50, "dname": "REPAIRS"}]
5 documents in set (0.0041 sec)

MySQL [localhost+ ssl/docstore] JS> dcol.find('_id=50')
[   {"_id": 50,  "dname": "REPAIRS" }]
1 document in set (0.0048 sec)
MySQL [localhost+ ssl/docstore] JS> dcol.find('dname="ACCOUNTING"')
[   {"_id": 10,  "dname": "ACCOUNTING" }]
1 document in set (0.0044 sec)
```

The Document store (5/8): Example Working with objects (3/3)

```
MySQL [localhost+ ssl/docstore] JS> dcol.modify('_id = 50')
                                        .set('dname', 'ALL REPAIRS')
Query OK, 1 item affected (0.0143 sec)

MySQL [localhost+ ssl/docstore] JS> dcol.find('_id=50')
[   {"_id": 50, "dname": "ALL REPAIRS"}]
1 document in set (0.0042 sec)

MySQL [localhost+ ssl/docstore] JS> dcol.remove('_id=50')
Query OK, 1 item affected (0.0160 sec)

MySQL [localhost+ ssl/docstore] JS> dcol.find().fields('_id')
[   {"_id": 10},{"_id": 20},{"_id": 30},{"_id": 40}]
4 documents in set (0.0039 sec)
```

The Document store (6/8): Example Working with regular tables

```
MySQL [localhost+ ssl/docstore] JS> \use dept_emp
Default schema `dept_emp` accessible through db.
MySQL [localhost+ ssl/dept_emp] JS> dept = db.getTable('dept')
<Table:dept>
MySQL [localhost+ ssl/dept_emp] JS> dept.select().orderBy('dname')
+--------+------------+----------+
| deptno | dname      | loc      |
+--------+------------+----------+
|     10 | ACCOUNTING | NEW YORK |
|     40 | OPERATIONS | BOSTON   |
|     20 | RESEARCH   | DALLAS   |
|     30 | SALES      | CHICAGO  |
+--------+------------+----------+
4 rows in set (0.5590 sec)
MySQL [localhost+ ssl/dept_emp] JS> dept.insert('deptno', 'dname', 'loc')
                                        .values('50','REPAIRS','SOFIA')
Query OK, 1 item affected (0.2108 sec)
```

The Document store (7/8): Behind the scenes

- In the database *collections* are tables and *documents* are JSON data

```
CREATE TABLE `docstore`.`dept` (
   `doc` JSON DEFAULT NULL,
   `_id` VARCHARBINARY(32) GENERATED ALWAYS AS
         (json_unquote(json_extract(`doc`,'$._id'))) STORED NOT NULL,
   PRIMARY KEY (`_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Before MySQL 8.0.11 document identifiers were generated as UUIDs by client (e.g. MySQL Shell)
- Now the server generates Documents identifiers with specific format:
  - *unique prefix* (4 bytes) – from `mysqlx_document_id_unique_prefix` system variable (default 0);
  - *start timestamp* (8 bytes) – startup time of server instance;
  - *serial* (16 bytes) – per instance auto-incremented integer value (hex encoded)

The Document store (8/8): Relational tables as JSON collections

- Need to set output format with `--json` command line option (valid options are `raw` and `pretty`)
- Or with `\option outputFormat` on the command prompt (valid options are `json` (pretty) and `json/raw`)
- Or through `shell.options` scripting interface (same options)

```
MySQL [localhost+ ssl/dept_emp] SQL>
select * from dept;
{"executionTime":"0.0049
sec","warningCount":0,"warnings":[],"rows":[{"de
ptno":10,"dname":"ACCOUNTING","loc":"NEW
YORK"},{"deptno":20,"dname":"RESEARCH","loc":"
DALLAS"},{"deptno":30,"dname":"SALES","loc":"CHI
CAGO"},{"deptno":40,"dname":"OPE
RATIONS","loc":"BOSTON"}],"hasData
":true,"affectedRowCount":0,"autoIncrementValue"
:0}
```

```
MySQL [localhost+ ssl/dept_emp] SQL>
select * from dept;
{
    "executionTime": "0.0070 sec",
    "warningCount": 0,
    "warnings": [],
    "rows": [
        {
            "deptno": 10,
            "dname": "ACCOUNTING",
            "loc": "NEW YORK"
        },
        {
            "deptno": 20,
        ...
    ],
    "hasData": true,
    "affectedRowCount": 0,
    "autoIncrementValue": 0
}
```

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The document store
- **GIS improvements**
- Datatype improvements
- Descending indexes

GIS Improvements

- Support for 5108 Spatial Reference Systems or SRS (4628 projections or flat maps, 479 geographic or ellipsoidal, and SRID 0 – Cartesian all-purpose abstract plane)
- Can correctly calculate the distances between two points on the earths surface in any of the supported SRSes
- Meta data support for SRSes through `ST_SPATIAL_REFERENCE_SYSTEMS` view in `INFORMATION_SCHEMA`
- SRID aware spatial data types (e.g. `CREATE TABLE t1 (g GEOMETRY SRID 4326)`)
- SRID aware spatial indexes and functions
- Performance optimizations


```
GEOGCS[
  "WGS 84",
  DATUM[
    "World Geodetic System 1984",
    SPHEROID[
      "WGS 84",
      6378137,
      298.257223563,
      AUTHORITY["EPSG","7030"]
    ],
    AUTHORITY["EPSG","6326"]
  ],
  PRIMEM[
    "Greenwich",
    0,
    AUTHORITY["EPSG","8901"]
  ],
  UNIT[
    "degree",
    0.0174532925199433278,
    AUTHORITY["EPSG","9122"]
  ],
  AXIS["Lat", NORTH],
  AXIS["Lon", EAST],
  AUTHORITY["EPSG","4326"]
]
```

GIS Improvements (2/2): Example

MySQL 5.7 (SRID 0)

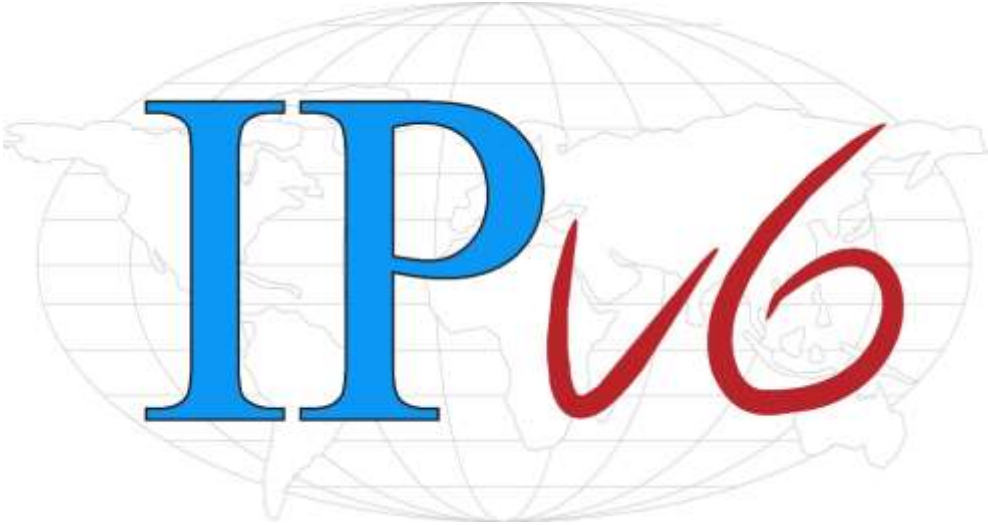
```
SELECT
ST_Distance_Sphere(POINT( 23.32415,
            /* Sofia  */  42.69751),
                    POINT(151.20000,
            /* Sydney */ -33.86667)
) / 1000 dist_km;

+---------------------+
| dist_km             |
+---------------------+
| 15435.226080478988 |
+---------------------+
1 row in set (0,00 sec)
```

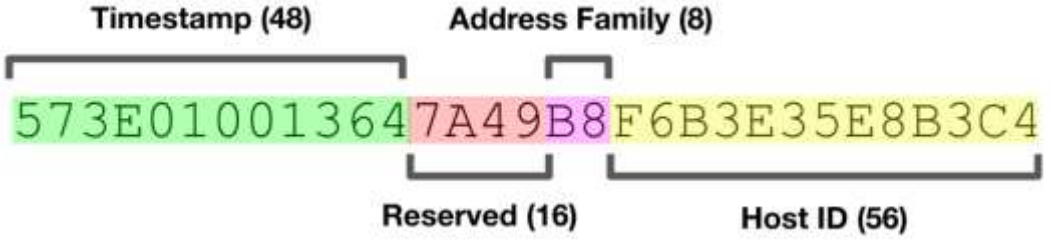
MySQL 8.0 (SRID 4326)

```
SELECT ST_Distance(
ST_PointFromText('POINT( 42.69751
                    23.32415)', 4326),
ST_PointFromText('POINT(-33.86667
                   151.20000)', 4326)
) / 1000 dist_km;

+---------------------+
| dist_km             |
+---------------------+
| 15431.933058990675 |
+---------------------+
1 row in set (0.00 sec)
```

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The document store
- GIS improvements
- **Datatype improvements**
- Descending indexes



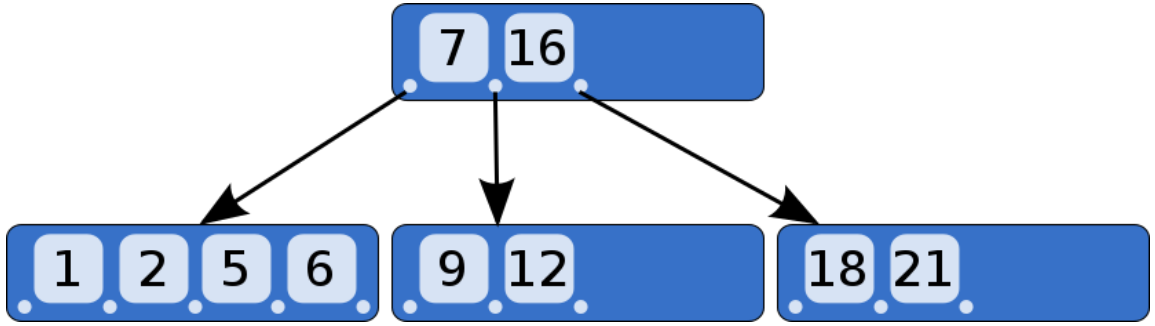


Datatype improvements

- Bit-wise operations on binary data types (now also on `[VAR]BINARY`/`[TINY|MEDIUM|LONG]BLOB` data types)
- IPv6 manipulation improved with bit-wise on binary data types (now it's possible to test an IPv6 address against a network mask with `INET6_ATON(address) & INET6_ATON(network)`
- Universally unique identifier (UUID) manipulations – three new functions `UUID TO BIN`, `BIN TO UUID` and `IS UUID`

New features for developers

- New default character set
- Common Table Expressions
- Window functions
- Locking Read Concurrency
- Regular expression improvements
- GROUPING function
- JSON enhancements
- The document store
- GIS improvements
- Datatype improvements
- **Descending indexes**



Descending indexes (1/3): General

- Before `DESC` keyword in index definition was **silently** ignored
- In case of `ORDER BY ... DESC` previously indexes were scanned in reverse order
- Only for B-tree indexes
- Only available with InnoDB storage engine
- Not supported for full-text and spatial indexes (error raised)
- Forward scan of descending indexes is up to 15% better

Descending indexes (2/3): Example

MySQL 5.7

```
ALTER TABLE emp
  ADD INDEX idx_hiredate (hiredate DESC,
                          ename    ASC);
Query OK, 0 rows affected (0,17 sec)
Records: 0  Duplicates: 0  Warnings: 0

SELECT `column_name`, `collation`,
       index_type
  FROM information_schema.STATISTICS
 WHERE index_name = 'idx_hiredate';
+-------------+-----------+------------+
| column_name | collation | index_type |
+-------------+-----------+------------+
| hiredate    | A         | BTREE      |
| ename       | A         | BTREE      |
+-------------+-----------+------------+
2 rows in set (0,02 sec)
```

MySQL 8.0

```
ALTER TABLE emp
  ADD INDEX idx_hiredate (hiredate DESC,
                          ename    ASC);
Query OK, 0 rows affected (0.63 sec)
Records: 0  Duplicates: 0  Warnings: 0

SELECT `column_name`, `collation`,
       index_type
  FROM information_schema.STATISTICS
 WHERE index_name = 'idx_hiredate';
+-------------+-----------+------------+
| COLUMN_NAME | COLLATION | INDEX_TYPE |
+-------------+-----------+------------+
| hiredate    | D         | BTREE      |
| ename       | A         | BTREE      |
+-------------+-----------+------------+
2 rows in set (0.04 sec)
```

Descending indexes (3/3): Explain plan

MySQL 5.7 **SELECT** ename, hiredate **FROM** emp **ORDER BY** hiredate **DESC**/**ASC**;

```
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
| id | select_type | table | type  | possible_keys | key          | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | emp   | index | NULL          | idx_hiredate | 37      | NULL |   14 |   100.00 | Using index |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0,00 sec)
```

MySQL 8.0 **SELECT** ename, hiredate **FROM** emp **ORDER BY** hiredate **DESC**;

```
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
| id | select_type | table | type  | possible_keys | key          | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | emp   | index | NULL          | idx_hiredate | 37      | NULL |   14 |   100.00 | Using index |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

MySQL 8.0 **SELECT** ename, hiredate **FROM** emp **ORDER BY** hiredate **ASC**;

```
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
| id | select_type | table | type  | possible_keys | key          | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | emp   | index | NULL          | idx_hiredate | 37      | NULL |   14 |   100.00 | Backward index scan;  |
|                                                                                                      Using index           |
+----+-------------+-------+-------+---------------+--------------+---------+------+------+----------+-----------------------+
1 row in set, 1 warning (0.00 sec)
```

Other new features and improvements (1/2)

- Administrations/Operations
  - **Transactional data dictionary** & **Atomic DDLs**
  - Persistent global variables and `RESTART` command (cloud friendly)
  - Query cache removed (see ProxySQL as alternative)
  - Resource groups (associate threads with resources – e.g. virtual CPUs)
  - REDO & UNDO logs are now encrypted if file-per-table tablespace is encrypted
  - InnoDB Instant `ADD COLUMN`
  - New backup lock (`LOCK INSTANCE FOR BACKUP` / `UNLOCK INSTANCE`)
  - **Roles** (check Giuseppe Maxia's presentation)
  - **New default authentication plugin** (caching_sha2_password) that improves security, but could break old applications (!!!)
  - Components providing services for extending server capabilities (instead of plugins)

Other new features and improvements (2/2)

- Performance & tuning:
  - Descending indexes
  - Invisible indexes (`INDEX ... INVISIBLE`)
  - Histograms (ANALYSE TABLE ... [UPDATE|DROP] HISTOGRAM ON ...)
- Improvements:
  - General performance up to 2 times faster compared to MySQL 5.7 [2]
  - Unicode performance up to 20 times faster [10l]
  - JSON performance up to 18 times faster sorting [10l]
  - Up to 30 times faster PERFORMANCE_SCHEMA and 100 times faster INFORMATION SCHEMA
  - Better optimizer cost model (considering how much data is cached in memory and how much still on disk)

What is still missing in MySQL

- What is missing in SQL
- What is missing in DDL
- What is missing for views
- What is missing for routines and triggers

What is missing in SQL (1/3)

- **Modify table data** based on **subquery on the same table** (still results in Error 1093)
- Full set of **SQL set operations** (e.g. `EXCEPT` or `MINUS` and `INTERSECT`). Workarounds exists:
  - `NOT IN` or `LEFT JOIN` instead of `MINUS`;
  - `INNER JOIN` instead of `INTERSECT`;
- **Temporal queries** (`AS OF`).
- ~~**Query factoring** with **`WITH`** clause~~ (already in 8.0.1)
- ~~**NOWAIT** and **SKIP LOCKED** statements~~ (already in 8.0.1)
- ~~Window functions with **`ORDER (PARTITION BY )`**~~ (already in 8.0.2)

What is missing in SQL (2/3): Update

- From MySQL 8.0's reference manual, chapter 13.2.11 UPDATE Syntax:
  - *"You cannot update a table and select from the same table in a subquery."*

```
UPDATE dept
   SET dname = CONCAT(dname, ' MAX')
 WHERE deptno = (SELECT MAX(deptno) FROM dept);

Error Code: 1093. You can't specify target table
'dept' for update in FROM clause
```

What is missing in SQL (3/3): Set operations

- EXCEPT or MINUS

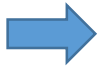
```
SELECT id, name FROM a
MINUS
SELECT id, name FROM b;
```

```
SELECT DISTINCT id, name
  FROM a
 WHERE (id, name) NOT IN
       (SELECT id, name FROM b);
```

```
SELECT DISTINCT a.id, a.name
  FROM a LEFT JOIN b USING (id, name)
WHERE b.id IS NULL
```

- INTERSECT

```
SELECT id, name FROM a
INTERSECT
SELECT id, name FROM b;
```



```
SELECT a.id, a.name
  FROM a INNER JOIN b USING (id, name);
```

What is missing in DDL (1/2)

- **`CHECK` constraints** (workaround is with row-level triggers, but it may not be that obvious).
- **Partitioning of tables with foreign keys** (no workaround)
- **Temporal tables** (WITH SYSTEM VERSIONING)
- **`CREATE OR REPLACE`** for tables, triggers and stored routines
- **Enable/disable** for triggers (drop & re-create)
- **Sequences** (for more control on identifier generation)
- A working **schema rename** (i.e. `RENAME {DATABASE | SCHEMA}`) statement was added in 5.1.17, but was removed in 5.1.23 due to “serious problems”). The workaround is:

```
CREATE DATABASE new_db_name;
RENAME TABLE db_name.table1 TO new_db_name,
             db_name.table2 TO new_db_name;
DROP DATABASE db_name;
```

What is missing in DDL (2/2): Check constraints with row-level triggers

```
DELIMITER //
CREATE TRIGGER check_hire_date
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
  IF NEW.hiredate < CURDATE() THEN
    SET @errmsg = 'Hire date in the past!';
    SIGNAL SQLSTATE '99999'
      SET MESSAGE_TEXT = @errmsg;
  END IF;
END //
```

```
Error Code: 1644. Hire date in the past!
```

```
DELIMITER //
CREATE TRIGGER check_sal
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
  IF COALESCE(NEW.sal, 0) <= 0 THEN
    SET @errmsg = 'Salary should be higher!';
    SIGNAL SQLSTATE '12345'
      SET MESSAGE_TEXT = @errmsg;
  END IF;
END //
```

```
Error Code: 1644. Salary should be higher!
```

```
GET CURRENT DIAGNOSTICS CONDITION 1
  @errno = MYSQL_ERRNO, @errst = RETURNED_SQLSTATE, @msg = MESSAGE_TEXT;
SELECT @errno, @errst, @msg;
```

```
+--------+--------+-------------------------+
| @errno | @errst | @msg                    |
+--------+--------+-------------------------+
| 1644   | 99999  | Hire date in the past! |
+--------+--------+-------------------------+
1 row in set (0.00 sec)
```

```
+--------+--------+-------------------------+
| @errno | @errst | @msg                    |
+--------+--------+-------------------------+
| 1644   | 12345  | Salary should be higher! |
+--------+--------+-------------------------+
1 row in set (0.00 sec)
```

What is missing for views

- Saving of the **exact view statement** in the server. Rewrites by the server are simply annoying and impacting maintainability (unless using MySQL Workbench, but it still fails)
- **Fully updatable views** with `INSTEAD OF` triggers (no workaround)
- **Materialized views** - snapshots of data with periodical or immediate refresh (workaround is with triggers/stored procedures through event scheduler)

What is missing for routines and triggers

- **Default arguments** for routines (no workaround)
- **`FOR`** statement for looping (no workaround)
- **Anonymous code blocks** (e.g. `BEGIN ... END` outside routine's body)
- **Records**, column and row **anchored types**
- **Stored modules** or **packages** (routine groups in Workbench?)
- **Statement** and **schema level triggers** (no workaround)
- **Debugging facilities**
- **External routines** in JavaScript, Perl, PHP, etc.

Comparison to MariaDB 10.3.7 (2018-05-25)

Features		MariaDB	MySQL	Standard
SQL	Modify table data based on subquery on the same table	10.3.2 (2017-11-09)	n/a	
	EXCEPT or MINUS and INTERSECT set operators	10.3.0 (2017-04-16)	n/a	SQL:2003
	Temporal queries (AS OF)	10.3.4 (2018-01-18)	n/a	SQL:2011
DDL	CHECK constraints	10.2.1 (2016-07-04)	n/a	SQL:1999
	Partitioning of tables with foreign keys	n/a	n/a	
	Temporal tables (WITH SYSTEM VERSIONING)	10.3.4 (2018-01-18)	n/a	SQL:2011
	CREATE OR REPLACE for tables, triggers and stored routines	10.0.8 (2014-02-10), 10.1.4 (2015-04-13) and 10.1.3 (2015-03-02)	n/a	
	Enable/disable for triggers	n/a	n/a	
	Sequences	10.3.0 (2017-04-16)	n/a	SQL:2003
	A working schema rename (i.e. RENAME {DATABASE | SCHEMA}) statement	n/a	n/a	

Comparison to MariaDB 10.3.7 (2018-05-25)

Features		MariaDB	MySQL	Standard
Views	Saving of the exact view statement in the server	n/a	n/a	n/a
	Fully updatable views with INSTEAD OF triggers	n/a	n/a	SQL:2008
	Materialized views	n/a	n/a	
Stored routines	Default arguments	n/a	n/a	
	FOR loop	10.3.3 (2017-12-23)	n/a	
	Anonymous code blocks (or compound statements outside routines)	10.1.1 (2014-10-17)	n/a	
	Records, column and row anchored types	10.3.0 (2017-04-16)	n/a	
	Stored modules or packages	10.3.5 (2018-02-26)	n/a	
	Statement and schema level triggers	n/a	n/a	
	Debugging facilities	n/a	n/a	n/a
	External routines in JavaScript, Perl, PHP, etc.	n/a	n/a	

Conclusions

- With MySQL 8.0 Oracle is delivering some functionalities frequently requested by developers and DBAs, but long overdue
- MySQL now provides more **modern SQL features** (e.g. CTE, Window functions, JSON)
- MySQL is becoming even more **standards compliant**, not sacrificing speed and reliability
- MySQL is becoming an even better **processing engine**, so we cold do more on the DB server
- There's now even more *magic* into MySQL

"Any sufficiently advanced technology is indistinguishable from magic."

Arthur C. Clarke's third law

What to expect

- More performance and scalability improvements
- More replication and HA improvements (replication is key technology for MySQL)
- More and more Oracle “look & feel”
- Large user groups would continue to dictate the direction of the product

References (1/2)

1. What Is New in MySQL 8.0
2. Announcing General Availability of MySQL 8.0
3. What's new in the first MySQL 8.0 development milestone by Georgi Kodinov (BGOUG Autumn 2016)
4. One Giant Leap For SQL: MySQL 8.0 Released by Markus Winand
5. SQL window functions for MySQL by Dag H. Wanvik
6. MySQL Document store: SQL and NoSQL united by Giuseppe Maxia
7. A quick tour of MySQL 8.0 roles by Giuseppe Maxia
8. What’s new in SQL:2016 by Markus Winand
9. Calculating distance using MySQL by Logan Henson

References (2/2)

10. MySQL server blog
    a. 2016-09-12 The MySQL 8.0.0 Milestone Release is available
    b. 2016-09-20 MySQL 8.0 Labs: [Recursive] Common Table Expressions (CTEs)
    c. 2016-10-12 MySQL 8.0 Labs – Descending Indexes in MySQL
    d. 2016-10-18 MySQL 8.0 Labs: CTEs, Part Two – how to generate series
    e. 2016-10-25 MySQL 8.0 Labs: CTEs, Part Three – hierarchies
    f. 2017-05-23 MySQL 8.0.1; CTEs, Part Four - depth-first or breadth-first traversal, transitive closure, cycle avoidance
    g. 2017-04-12 MySQL 8.0.1: Using SKIP LOCKED and NOWAIT to handle hot rows
    h. 2017-05-02 GROUPING function
    i. 2017-07-18 MySQL 8.0.2: Introducing Window Functions
    j. 2018-03-15 MySQL 8.0: It Goes to 11!
    k. 2018-04-03 Partial update of JSON values
    l. 2018-04-19 What's New in MySQL 8.0? (Generally Available)
    m. 2018-04-26 New JSON functions in MySQL 5.7.22
    n. 2018-05-18 Changes in MySQL 8.0.11 (General Availability)
    o. 2018-05-25 Geographic Spatial Reference Systems in MySQL 8.0

Questions & Answers