Beyond WHERE and GROUP BY

Sergei Golubchik
MariaDB Corporation

Everybody knows

SQL:86

This is SELECT, just SELECT

```
SELECT select_expr [, select_expr ...]
       [ FROM table_references ]
       [ WHERE where_condition ]
       [ GROUP BY {col_name | expr | position} [ASC | DESC], ... ]
       [ HAVING where_condition]
       [ ORDER BY {col_name | expr | position} [ASC | DESC], ... ]
```

UNION, INTERSECT, EXCEPT

SQL:92

UNION — MySQL 4.0

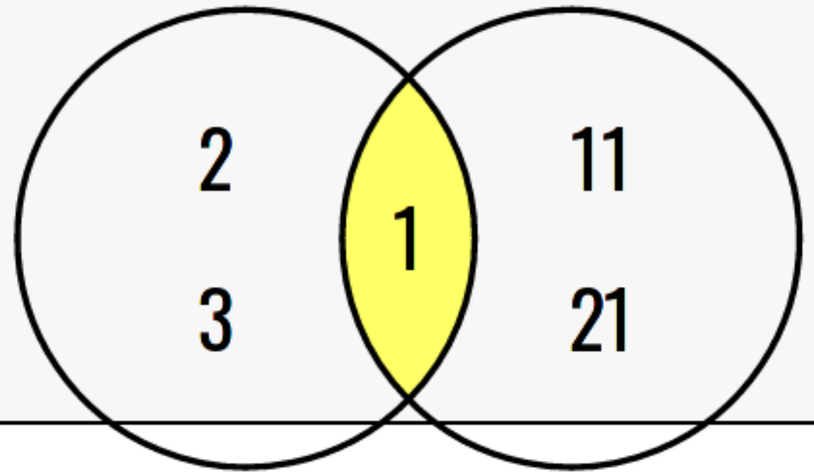
```
CREATE TABLE t1 (a INT); INSERT t1 VALUES (1),(2),(3);
CREATE TABLE t2 (b INT); INSERT t2 VALUES (1),(11),(21);

SELECT * FROM t1 UNION SELECT * FROM t2;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|   11 |
|   21 |
+------+
```

INTERSECT — MariaDB 10.3

```
CREATE TABLE t1 (a INT); INSERT t1 VALUES (1),(2),(3);
CREATE TABLE t2 (b INT); INSERT t2 VALUES (1),(11),(21);


SELECT * FROM t1 INTERSECT SELECT * FROM t2;
+------+
| a    |
+------+
|    1 |
+------+
```

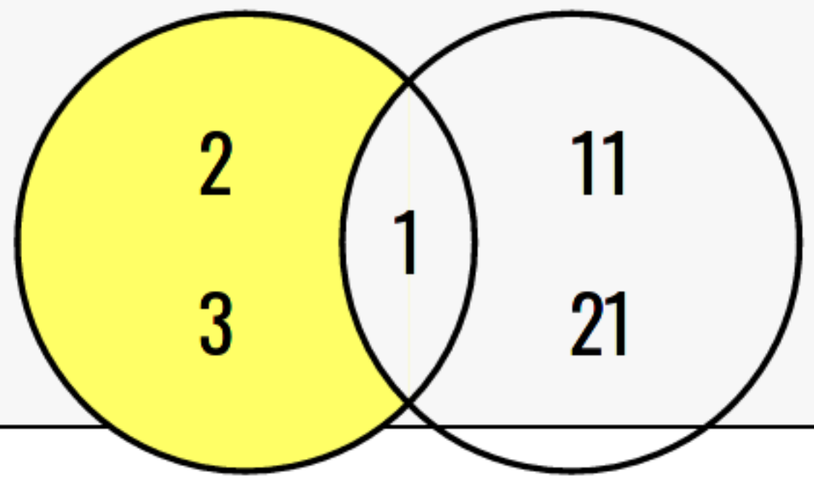
INTERSECT — MariaDB 10.3

```
CREATE TABLE t1 (a INT); INSERT t1 VALUES (1),(2),(3);
CREATE TABLE t2 (b INT); INSERT t2 VALUES (1),(11),(21);


SELECT * FROM t1 INTERSECT SELECT * FROM t2;
+------+
| a    |
+------+
|    1 |
+------+
```



EXCEPT — MariaDB 10.3

```
CREATE TABLE t1 (a INT); INSERT t1 VALUES (1),(2),(3);
CREATE TABLE t2 (b INT); INSERT t2 VALUES (1),(11),(21);

SELECT * FROM t1 EXCEPT SELECT * FROM t2;
+------+
| a    |
+------+
|    2 |
|    3 |
+------+
```



Common Table Expressions

SQL:99

Sergei Golubchik, MariaDB Corporation

Subquery in the FROM clause

```
SELECT name
FROM (SELECT * FROM employees WHERE dept='Engineering') AS engineers
WHERE expertise = 'optimizer'
```

Common Table Expression

```
WITH engineers AS (SELECT * FROM employees WHERE dept='Engineering')
SELECT name FROM engineers WHERE expertise = 'optimizer'
```

Readability

```
SELECT name
FROM (SELECT *
      FROM (SELECT * FROM employees WHERE dept='Engineering') AS engineers
      WHERE expertise = 'optimizer') AS optimizers
WHERE language='russian'
```

```
WITH engineers AS (SELECT * FROM employees WHERE dept='Engineering'),
     optimizers AS (SELECT * FROM engineers WHERE expertise = 'optimizer')
SELECT name FROM optimizers WHERE language='russian'
```

Readability, repeated

```
SELECT coder.name, reviewer.name
FROM (SELECT * FROM employees WHERE dept='Engineering') AS coder,
     (SELECT * FROM employees WHERE dept='Engineering') AS reviewer,
WHERE coder.expertise = reviewer.expertise
```

```
WITH engineers AS (SELECT * FROM employees WHERE dept='Engineering')
SELECT coder.name, reviewer.name FROM engineers coder, engineers reviewer
WHERE coder.expertise = reviewer.expertise
```

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

425	Sergei

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

425	Sergei
767	father
127	mother

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

425	Sergei
767	father
127	mother
892	paternal grandfather
395	paternal grandmother
176	maternal grandfather
881	maternal grandmother

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

425	Sergei
767	father
127	mother
892	paternal grandfather
395	paternal grandmother
176	maternal grandfather
881	maternal grandmother
...	... 8 more rows

Recursive CTE

```
WITH RECURSIVE ancestors AS (
  SELECT * FROM folks WHERE name = 'Sergei'
  UNION ALL
  SELECT folks.* FROM folks, ancestors
    WHERE folks.id = ancestors.father
    OR    folks.id = ancestors.mother
)
SELECT * FROM ancestors;
```

425	Sergei
767	father
127	mother
892	paternal grandfather
395	paternal grandmother
176	maternal grandfather
881	maternal grandmother
...	... 8 more rows
...	...

Weekend days in 2020

```
WITH RECURSIVE days AS (
  SELECT DATE'2020-01-01' d
  UNION ALL
  SELECT d + INTERVAL 1 DAY FROM days WHERE YEAR(d) = 2020
) SELECT * FROM days WHERE WEEKDAY(d) IN (5,6);
+------------+
| 2020-01-04 |
| 2020-01-05 |
| 2020-01-11 |
| 2020-01-12 |
| 2020-01-18 |
| 2020-01-19 |
| 2020-01-25 |
| 2020-01-26 |
```

Weekend days in 2020

```
WITH RECURSIVE days AS (
  SELECT DATE'2020-01-01' d
  UNION ALL
  SELECT d + INTERVAL 1 DAY FROM days WHERE YEAR(d) = 2020
) SELECT * FROM days WHERE WEEKDAY(d) IN (5,6);
+------------+
| 2020-01-04 |
| 2020-01-05 |
| 2020-01-11 |
| 2020-01-12 |
| 2020-01-18 |
| 2020-01-19 |
| 2020-01-25 |
| 2020-01-26 |
```

Weekend days in 2020

```
WITH RECURSIVE days AS (
  SELECT DATE'2020-01-01' d
  UNION ALL
  SELECT d + INTERVAL 1 DAY FROM days WHERE YEAR(d) = 2020
) SELECT * FROM days WHERE WEEKDAY(d) IN (5,6);
+------------+
| 2020-01-04 |
| 2020-01-05 |
| 2020-01-11 |
| 2020-01-12 |
| 2020-01-18 |
| 2020-01-19 |
| 2020-01-25 |
| 2020-01-26 |
```

Weekend days in 2020

```
WITH RECURSIVE days AS (
  SELECT DATE'2020-01-01' d
  UNION ALL
  SELECT d + INTERVAL 1 DAY FROM days WHERE YEAR(d) = 2020
) SELECT * FROM days WHERE WEEKDAY(d) IN (5,6);
+------------+
| 2020-01-04 |
| 2020-01-05 |
| 2020-01-11 |
| 2020-01-12 |
| 2020-01-18 |
| 2020-01-19 |
| 2020-01-25 |
| 2020-01-26 |
```

Weekend days in 2020

```
WITH RECURSIVE days AS (
  SELECT DATE'2020-01-01' d
  UNION ALL
  SELECT d + INTERVAL 1 DAY FROM days WHERE YEAR(d) = 2020
) SELECT * FROM days WHERE WEEKDAY(d) IN (5,6);
+------------+
| 2020-01-04 |
| 2020-01-05 |
| 2020-01-11 |
| 2020-01-12 |
| 2020-01-18 |
| 2020-01-19 |
| 2020-01-25 |
| 2020-01-26 |
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
CREATE TABLE users (id BIGINT, name VARCHAR(200));
CREATE TABLE friends (id1 BIGINT, id2 BIGINT);
INSERT users VALUES (1, 'Jennifer'),(2, 'Michael'),(3, 'Amy'),
(4, 'Jason'),(5, 'Michelle'),(6, 'Chris'),(7, 'Angela'),(8, 'James');
INSERT friends VALUES (1,2),(1,3),(1,7),(2,1),(2,4),(3,1),(3,5),(3,6),
(3,7),(3,8),(4,2),(4,5),(4,7),(4,8),(5,3),(5,4),(6,3),(7,1),(7,3),
(7,4),(8,3),(8,4);

WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE name='Chris'
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
```

"Social network"

```
WITH RECURSIVE x AS (
  SELECT id, name AS path FROM users WHERE id=6
  UNION ALL
  SELECT users.id, CONCAT(x.path,',',users.name) FROM users,x,friends
    WHERE x.id=friends.id1 AND users.id=friends.id2 AND
    NOT FIND_IN_SET(users.name, x.path)
) SELECT path FROM x WHERE x.id=4;
+-----------------------------------+
| Chris,Amy,Michelle,Jason          |
| Chris,Amy,Angela,Jason            |
| Chris,Amy,James,Jason             |
| Chris,Amy,Jennifer,Michael,Jason  |
| Chris,Amy,Jennifer,Angela,Jason   |
| Chris,Amy,Angela,Jennifer,Michael,Jason |
+-----------------------------------+
```

Summary: CTE

- Not recursive
  - Alternative syntax for subqueries in the FROM clause
  - More readable and (sometimes) optimizeable
- Recursive
  - Hierarchical data, graphs
  - Data generation
  - Turing complete

Window Functions

SQL:2003

Sergei Golubchik, MariaDB Corporation

Functions in SQL

Functions in SQL

- Normal
  - One result per row, depend on that row only

Functions in SQL

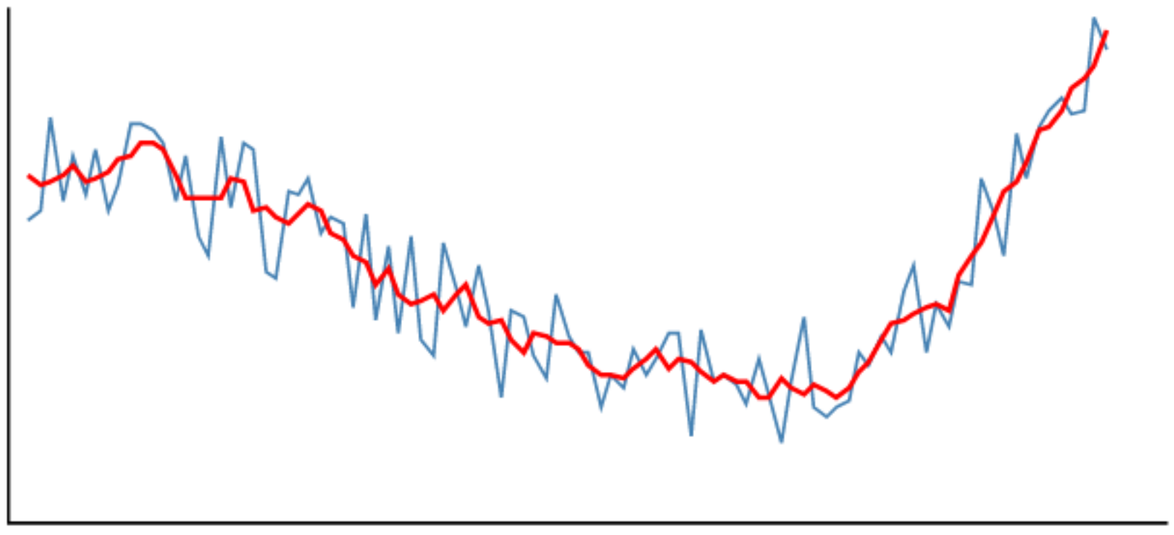
- Normal
  - One result per row, depend on that row only
- Aggregate
  - One result per group, depend on the whole group

Functions in SQL

- Normal
  - One result per row, depend on that row only
- Aggregate
  - One result per group, depend on the whole group
- Window
  - One result per row, depend on the whole group

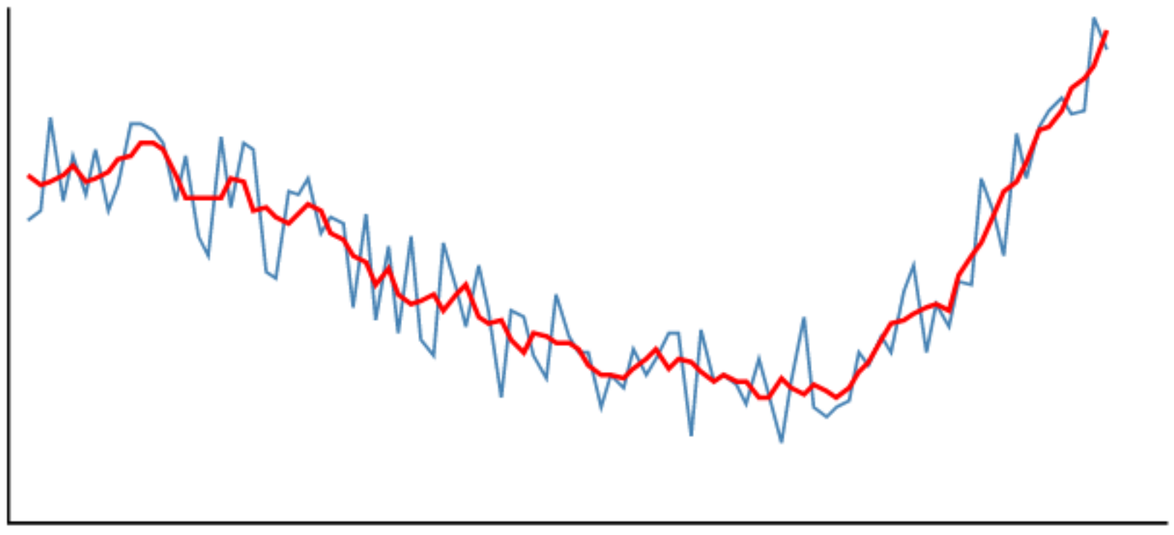
Example: Moving Average

```
SELECT time, value,
  AVG(value) OVER (ORDER BY time ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
FROM visitors ORDER BY time;
```



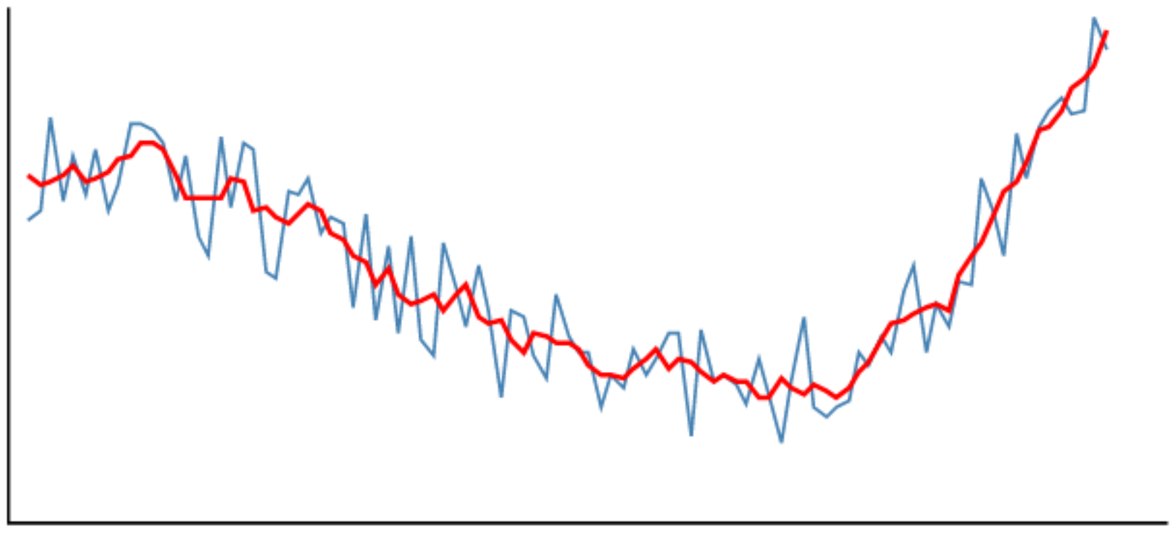
Example: Moving Average

```
SELECT time, value,
  AVG(value) OVER (ORDER BY time ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
FROM visitors ORDER BY time;
```



Example: Moving Average

```
SELECT time, value,
  AVG(value) OVER (ORDER BY time ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING)
FROM visitors ORDER BY time;
```



Example: Running Total

```
CREATE TABLE transactions (
  trn_id BIGINT AUTO_INCREMENT PRIMARY KEY,
  cust_id BIGINT,
  amount DECIMAL(10,2)
);

SELECT trn_id, cust_id, amount FROM transactions;
+--------+---------+---------+
|      1 |       1 |  -50.00 |
|      2 |       3 |   50.00 |
|      3 |       2 |  100.00 |
|      4 |       1 |  950.00 |
|      5 |       3 |  300.00 |
|      6 |       3 | -350.00 |
```

Example: Running Total

```
CREATE TABLE transactions (
  trn_id BIGINT AUTO_INCREMENT PRIMARY KEY,
  cust_id BIGINT,
  amount DECIMAL(10,2)
);

SELECT trn_id, cust_id, amount FROM transactions;
+--------+---------+---------+------------+
|      1 |       1 |  -50.00 | -50        |
|      2 |       3 |   50.00 | 50         |
|      3 |       2 |  100.00 | 100        |
|      4 |       1 |  950.00 | -50+950    |
|      5 |       3 |  300.00 | 50+300     |
|      6 |       3 | -350.00 | 50+300-350 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
  (SELECT SUM(amount) FROM transactions t2
   WHERE t2.cust_id=t1.cust_id AND t2.trn_id <= t1.trn_id)
FROM transactions t1;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
   (SELECT SUM(amount) FROM transactions t2
    WHERE t2.cust_id=t1.cust_id AND t2.trn_id <= t1.trn_id)
FROM transactions t1;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
   (SELECT SUM(amount) FROM transactions t2
    WHERE t2.cust_id=t1.cust_id AND t2.trn_id <= t1.trn_id)
FROM transactions t1;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
   (SELECT SUM(amount) FROM transactions t2
    WHERE t2.cust_id=t1.cust_id AND t2.trn_id <= t1.trn_id)
FROM transactions t1;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
  SUM(amount) OVER (PARTITION BY cust_id ORDER BY trn_id
                    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM transactions;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
  SUM(amount) OVER (PARTITION BY cust_id ORDER BY trn_id
                    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM transactions;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
  SUM(amount) OVER (PARTITION BY cust_id ORDER BY trn_id
                    ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM transactions;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

```
SELECT trn_id,cust_id,amount,
  SUM(amount) OVER (PARTITION BY cust_id ORDER BY trn_id)
FROM transactions;
+--------+---------+---------+---------+
|      1 |       1 |  -50.00 |  -50.00 |
|      2 |       3 |   50.00 |   50.00 |
|      3 |       2 |  100.00 |  100.00 |
|      4 |       1 |  950.00 |  900.00 |
|      5 |       3 |  300.00 |  350.00 |
|      6 |       3 | -350.00 |    0.00 |
|      7 |       4 | -100.00 | -100.00 |
|      8 |       2 |  500.00 |  600.00 |
|      9 |       1 | -700.00 |  200.00 |
```

Example: Running Total

	subquery	subquery + index	window function
100 rows	0.02 sec	0.01 sec	0.01 sec
1,000 rows	1.09 sec	0.10 sec	0.01 sec
10,000 rows	1 min 45.50 sec	2.71 sec	0.12 sec
100,000 rows	2 hours 55 min 4.40 sec	1 min 24.99 sec	1.19 sec

Summary: window functions

- A way to avoid slow subqueries
  - and self-joins
- Better readability
  - and "optimizeability"
- Often much faster
  - not always

System-versioned tables

SQL:2011

Sergei Golubchik, MariaDB Corporation

Problems

Problems

- DELETE FROM t1 WHERE pk Enter Oops...
  - Undo erroneous statements

Problems

- DELETE FROM t1 WHERE pk Enter Oops...
  - Undo erroneous statements
- “How did our user base change since last yeat?”
  - Analytics on historical data

Problems

- DELETE FROM t1 WHERE pk Enter Oops...
  - Undo erroneous statements
- “How did our user base change since last yeat?”
  - Analytics on historical data
- “We were hacked! Half a year ago!!!”
  - Forensic data analysis

System-versioned tables

```
CREATE TABLE t1 (
  pk BIGINT AUTO_INCREMENT PRIMARY KEY,
  data TEXT
);
```

System-versioned tables

```
CREATE TABLE t1 (
  pk BIGINT AUTO_INCREMENT PRIMARY KEY,
  data TEXT,
  v_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
  v_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END
);
```

System-versioned tables

```
CREATE TABLE t1 (
  pk BIGINT AUTO_INCREMENT PRIMARY KEY,
  data TEXT,
  v_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
  v_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (v_start, v_end)
);
```

System-versioned tables

```
CREATE TABLE t1 (
  pk BIGINT AUTO_INCREMENT PRIMARY KEY,
  data TEXT,
  v_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
  v_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (v_start, v_end)
) WITH SYSTEM VERSIONING;
```

System-versioned tables

```
CREATE TABLE t1 (
  pk BIGINT AUTO_INCREMENT PRIMARY KEY,
  data TEXT,
  v_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
  v_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (v_start, v_end)
) WITH SYSTEM VERSIONING;
```

AS OF

```
SELECT * FROM t1 FOR SYSTEM_TIME AS OF '2016-10-11 12:13:14';
```

Solutions

Solutions

```
INSERT INTO t1
  SELECT * FROM t1 FOR SYSTEM_TIME AS OF NOW() - INTERVAL 5 MINUTE
```

Solutions

```
INSERT INTO t1
  SELECT * FROM t1 FOR SYSTEM_TIME AS OF NOW() - INTERVAL 5 MINUTE
```

```
SELECT * FROM t2 AS new,
  t2 FOR SYSTEM_TIME AS OF '2016-12-31 23:59:59' AS old
WHERE old.id=new.id AND old.amount > new.amount
```

Solutions

```
INSERT INTO t1
  SELECT * FROM t1 FOR SYSTEM_TIME AS OF NOW() - INTERVAL 5 MINUTE
```

```
SELECT * FROM t2 AS new,
  t2 FOR SYSTEM_TIME AS OF '2016-12-31 23:59:59' AS old
WHERE old.id=new.id AND old.amount > new.amount
```

```
SELECT * FROM t3 FOR SYSTEM_TIME AS OF '2016-10-11 12:13:14.567890';
```

Partitioning

```
CREATE TABLE t1 (
  x INT
) WITH SYSTEM VERSIONING
PARTITION BY SYSTEM_TIME (
    PARTITION ph HISTORY,
    PARTITION pc CURRENT
);
```

JSON

SQL:2016

This is the Standard

```
SELECT JSON_QUERY('{"key1":{"a":1, "b":[1,2]}}', '$.key1.b');
+-----------------------------------------------------------+
| [1,2]                                                     |
+-----------------------------------------------------------+
```

- JSON_OBJECT, JSON_ARRAY
- JSON_EXISTS, JSON_VALUE, JSON_QUERY
- JSON_TABLE

This is the Standard

```
SELECT JSON_QUERY('{"key1":{"a":1, "b":[1,2]}}', '$.key1.b');
+-----------------------------------------------------------+
| [1,2]                                                     |
+-----------------------------------------------------------+
```

- JSON_OBJECT, JSON_ARRAY
- JSON_EXISTS, JSON_VALUE, JSON_QUERY
- JSON_TABLE

And this is not

- JSON_CONTAINS, JSON_CONTAINS_PATH, JSON_EXTRACT, JSON_KEYS, JSON_SEARCH
- JSON_APPEND, JSON_ARRAY_APPEND, JSON_ARRAY_INSERT, JSON_INSERT, JSON_MERGE, JSON_REMOVE, JSON_REPLACE, JSON_SET, JSON_QUOTE, JSON_UNQUOTE
- JSON_DEPTH, JSON_LENGTH, JSON_TYPE, JSON_VALID
- JSON_COMPACT, JSON_DETAILED, JSON_LOOSE

And this is not

- JSON_CONTAINS, JSON_CONTAINS_PATH, JSON_EXTRACT, JSON_KEYS, JSON_SEARCH
- JSON_APPEND, JSON_ARRAY_APPEND, JSON_ARRAY_INSERT, JSON_INSERT, JSON_MERGE, JSON_REMOVE, JSON_REPLACE, JSON_SET, JSON_QUOTE, JSON_UNQUOTE
- JSON_DEPTH, JSON_LENGTH, JSON_TYPE, JSON_VALID
- JSON_COMPACT, JSON_DETAILED, JSON_LOOSE ← not in MySQL

Aggregate stored functions

SQL:????

Example: NOT an aggregate function

```
CREATE FUNCTION sum2() RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;
  DECLARE x DOUBLE;
  DECLARE c CURSOR FOR SELECT col FROM t1;
  OPEN c;
  LOOP
    FETCH c INTO x;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: an aggregate function

```
CREATE AGGREGATE FUNCTION sum2(x DOUBLE) RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;



  LOOP
    FETCH GROUP NEXT ROW;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: an aggregate function

```
CREATE AGGREGATE FUNCTION sum2(x DOUBLE) RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;



  LOOP
    FETCH GROUP NEXT ROW;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: an aggregate function

```
CREATE AGGREGATE FUNCTION sum2(x DOUBLE) RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;



  LOOP
    FETCH GROUP NEXT ROW;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: an aggregate function

```
CREATE AGGREGATE FUNCTION sum2(x DOUBLE) RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;

  -- no explicit cursor declaration

  LOOP
    FETCH GROUP NEXT ROW;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: an aggregate function

```
CREATE AGGREGATE FUNCTION sum2(x DOUBLE) RETURNS DOUBLE
BEGIN
  DECLARE acc DOUBLE;
  DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN acc;

  -- no explicit cursor declaration

  LOOP
    FETCH GROUP NEXT ROW;
    IF x IS NOT NULL THEN
      SET acc = acc + x*x;
    END IF;
  END LOOP;
END
```

Example: median

```
CREATE AGGREGATE FUNCTION medi_int(x INT) RETURNS DOUBLE
BEGIN
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
      DECLARE res DOUBLE;
      DECLARE cnt INT DEFAULT (SELECT count(*) FROM tt);
      DECLARE lim INT DEFAULT (cnt-1) DIV 2;
      IF cnt % 2 = 0 THEN
        SET res = (SELECT AVG(a) FROM (SELECT a FROM tt ORDER BY a LIMIT lim, 2) ttt);
      ELSE
        SET res = (SELECT a FROM tt ORDER BY a LIMIT lim, 1);
      END IF;
      DROP TEMPORARY TABLE tt;
      RETURN res;
    END;
  CREATE TEMPORARY TABLE tt (a INT);
  LOOP
    FETCH GROUP NEXT ROW;
    INSERT INTO tt VALUES (x);
  END LOOP;
END
```

Example: median

```
CREATE AGGREGATE FUNCTION medi_int(x INT) RETURNS DOUBLE
BEGIN
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
      DECLARE res DOUBLE;
      DECLARE cnt INT DEFAULT (SELECT count(*) FROM tt);
      DECLARE lim INT DEFAULT (cnt-1) DIV 2;
      IF cnt % 2 = 0 THEN
        SET res = (SELECT AVG(a) FROM (SELECT a FROM tt ORDER BY a LIMIT lim, 2) ttt);
      ELSE
        SET res = (SELECT a FROM tt ORDER BY a LIMIT lim, 1);
      END IF;
      DROP TEMPORARY TABLE tt;
      RETURN res;
    END;
  CREATE TEMPORARY TABLE tt (a INT);
  LOOP
    FETCH GROUP NEXT ROW;
    INSERT INTO tt VALUES (x);
  END LOOP;
END
```

Questions?