

# MySQL Conference & Expo 2011



Subqueries to the people:  
MariaDB making the impossible possible  
and the slow fast

Igor Babaev

*igor@montyprogram.com*

Sergey Petrunya

*psergey@montyprogram.com*

Timour Katchaounov

*timour@montyprogram.com*

# What is the talk about



- Summarize all subquery improvements in MariaDB 5.3
- Use example queries (DBT3) to show
  - Typical performance improvements
  - Explain optimizer switches
- No complex algorithms/diagrams

# What's possible and what's faster

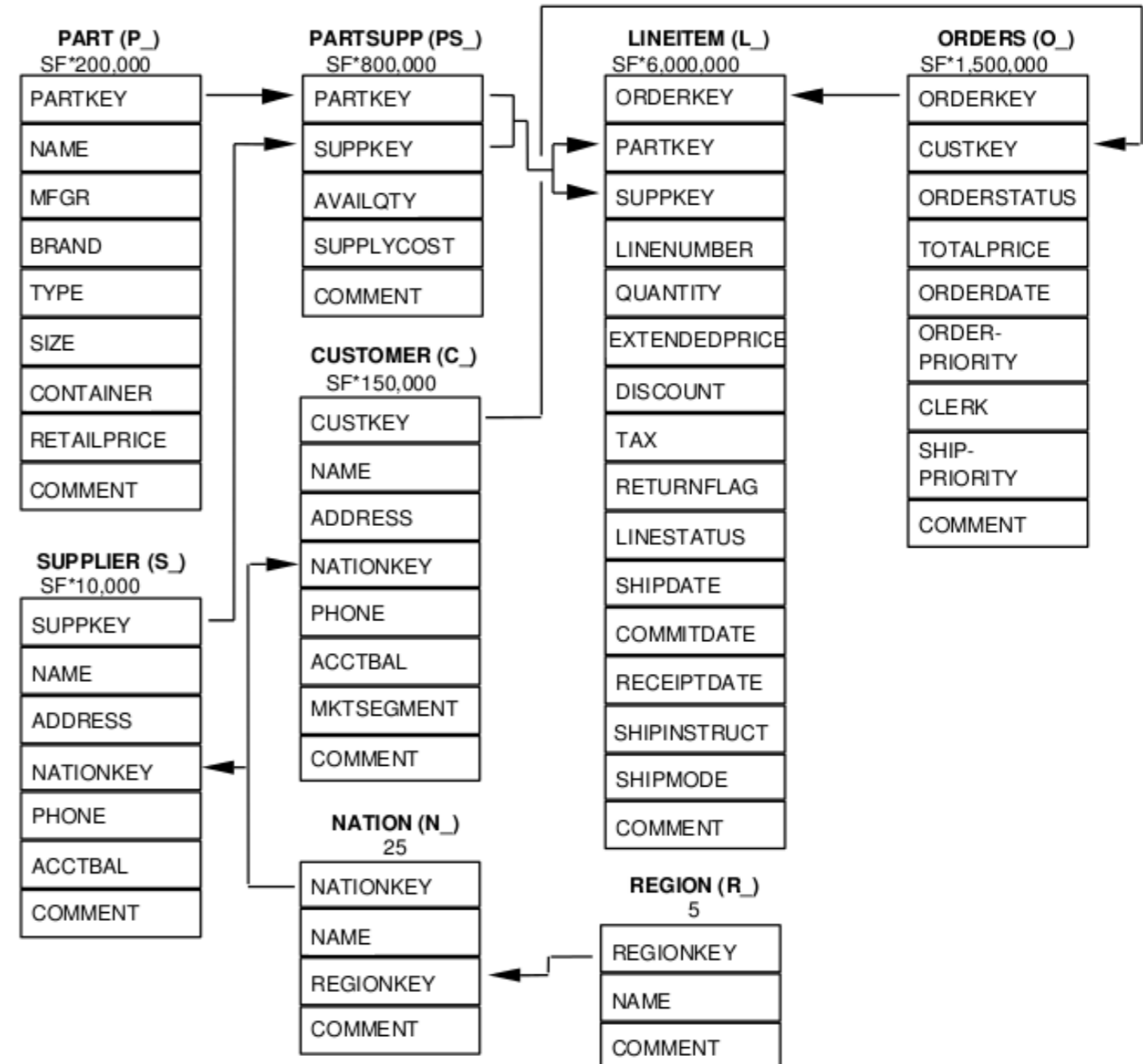


- What's possible (in practice)
  - Efficiently execute any subquery
  - Consistent subquery performance
  - Run EXPLAIN with subqueries
  - Backwards compatibility and fine-grained performance control via switches
- What's faster
  - Save a lot of development time
  - Easier migration from other DBMS
  - Derived tables
  - [NOT IN] Subqueries
  - And any other subquery

# DBT3-based example database & queries



- DBT3 scale 10
- 60 M LINEITEM rows
- 29 GB InnoDB database
- Extended schema – few more
  - indexes
  - columns
- Simplified queries



# Subquery strategies



- Flattenning into JOINS (semijoin):
  - Duplicate elimination, Pull out, Loose scan, First match, Materialization-scan
- Materialization for non-correlated subqueries
  - Subqueries with NOT, GROUP BY, ORDER BY, in the SELECT clause
  - efficient NULL-aware materialized execution
- Derived tables
  - Derived tables are merged as views
  - Late materialization of derived tables and views
  - Dynamic indexing
- Subquery caching
- Cost-based/manual choice of most strategies
- Fast EXPLAIN with subqueries



# Pullout strategy (1)



**Query: orders from customers with negative balance:**

```
SELECT * FROM orders
WHERE o_custkey IN
      (SELECT c_custkey FROM customer
       WHERE c_acctbal < -500);
```

**MariaDB 5.2 (any MySQL): 45 sec (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	orders	index	i_o_custkey	NULL	1493631	Using where; Using index
2	SUBQUERY	customer	range	c_acctbal	NULL	10536	Using where; Using index

**MariaDB 5.3: 0.43 sec (faster ~ 100x)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	customer	range	c_acctbal	NULL	10536	Using where; Using index
1	PRIMARY	orders	ref	i_o_custkey	dbt3sf1.customer.c_custkey	7	Using index

# Pullout strategy (2)



- General idea  
Based on unique indexes, rewrite subqueries into equivalent joins
- Optimizer switches
  - `optimizer_switch='semijoin=on';`

# Materialization-scan (1)



**Query: find customers with top balance in their nations:**

```
SELECT c_name, c_address
FROM customer
WHERE c_acctbal IN (SELECT max(c_acctbal)
                    FROM customer GROUP BY c_nationkey);
```

**MariaDB 5.2 (any MySQL): > 1.5 hours (impossible)**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	customer	ALL	NULL	NULL	NULL	NULL	149637	Using where
2	DEPENDENT SUBQUERY	customer	index	NULL	i_c_nationkey	5	NULL	3117	

**MariaDB 5.3: 3.2 sec (faster ~ INF)**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<subquery2>	ALL	distinct_key	NULL	NULL	NULL	149637	Using where
1	PRIMARY	customer	ref	c_acctbal	c_acctbal	9	<subquery2>.max(c_acctbal)	1	
2	SUBQUERY	customer	index	NULL	i_c_nationkey	5	NULL	149637	

# Materialization-scan (2)



- General idea:
  - The subquery materializes into a small dataset
  - The materialized subquery drives index-based access to the outer table
- Relevant optimizer switches:
  - semijoin= on,
  - materialization=on

# Duplicate elimination (1)



**Query: find customers who have orders exceeding their balance:**

```
SELECT * FROM customer
WHERE c_custkey IN
      (SELECT o_custkey FROM orders
       WHERE o_totalprice > c_acctbal and
            o_orderdate between '1997-01-01' and '1997-01-02');
```

**MariaDB 5.2 (any MySQL): 1:28 min (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	customer	index	c_acctbal	NULL	150081	Using where; Using index
2	DEPENDENT SUBQUERY	orders	index_subquery	i_o_custkey	func	7	Using where

**MariaDB 5.3: 36 sec (faster ~ 2.4x)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	orders	range	i_o_orderdate	NULL	5184	Using index condition; Using where;
							Rowid-ordered scan; Start temporary
1	PRIMARY	customer	eq_ref	PRIMARY	o_custkey	1	Using where; End temporary

# Duplicate elimination (2)



- General idea:
  - Convert to JOIN,
  - Remove duplicates (since it is semi-join)
- Relevant optimizer switches:
  - `optimizer_switch='semijoin=on'`;
  - No other strategy specific switches,
  - it is the “catch-all” strategy



- Same as original IN-TO-EXISTS, but:
  - Cost-based choice
  - First match compared to other strategies
- General idea:
  - Cheap when subquery re-execution cost is very low
  - Typically REF or EQ\_REF access types

# Subquery materialization



- Non-correlated subqueries, with either/or:
  - GROUP / ORDER BY,
  - NOT IN
  - Aggregation
  - Disjunctive (OR-ed) WHERE clause,
  - Subquery in the SELECT clause

# Materialization lookup (1)



**Query:**

```
SELECT * FROM part
WHERE p_partkey IN
      (SELECT l_partkey FROM lineitem
       WHERE l_shipdate between '1997-01-01' and '1997-02-01')
ORDER BY p_retailprice DESC LIMIT 10;
```

**MariaDB 5.2 (any MySQL): > 1 h (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	part	ALL	NULL	NULL	199755	Using where; Using filesort
2	DEPENDENT SUBQUERY	lineitem	index_subquery	i_l_suppkey_partkey	func	14	Using where

**MariaDB 5.3: 43 sec (faster ~ INF)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	part	ALL	NULL	NULL	199755	Using temporary; Using filesort
1	PRIMARY	<subquery2>	eq_ref	distinct_key	func	1	
2	SUBQUERY	lineitem	range	l_shipdate_partkey	NULL	160060	Using where; Using index

# Materialization lookup (2)



- General guidelines
  - Fast when 'pre-processing' the subquery and
  - making lookups is cheaper than re-executing it every time.
  - Typically when the subquery has relatively small result set, and
  - The outer query performs many lookups
- Optimizer switches
  - `semijoin=on`,
  - `materialization=on`

# Materialization with NULLs (1)



Find all customers that didn't buy from a preferred country, and from a preferred brand.

```
SELECT count (*)
FROM customer
WHERE (c_custkey, c_pref_nationkey_05, c_pref_brand_05)
NOT IN
  (SELECT o_custkey, s_nationkey, p_brand
FROM orders, supplier, part, lineitem
WHERE l_orderkey = o_orderkey and
        l_suppkey = s_suppkey and
        l_partkey = p_partkey and
        p_retailprice < 1200 and
        l_shipdate >= '1996-04-01' and
        l_shipdate < '1996-04-05' and
        o_orderdate >= '1996-04-01' and
        o_orderdate < '1996-04-05');
```

# Materialization with NULLs (2)



**MariaDB 5.2 (any MySQL) : 40 sec (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	customer	ALL	NULL	NULL	150841	Using where
2	DEPENDENT SUBQUERY	orders	ref_or_null	i_o_custkey	func	14	Using index condition; Using where
2	DEPENDENT SUBQUERY	lineitem	ref	PRIMARY	o_orderkey	1	Using where
2	DEPENDENT SUBQUERY	supplier	eq_ref	PRIMARY	l_suppkey	1	Using where
2	DEPENDENT SUBQUERY	part	eq_ref	PRIMARY	l_partkey	1	Using where

**MariaDB 5.3: 2 sec (faster ~ 20x)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	customer	ALL	NULL	NULL	150841	Using where
2	SUBQUERY	orders	range	i_o_orderdate	NULL	4680	Using index condition; Using MRR
2	SUBQUERY	lineitem	ref	PRIMARY	o_orderkey	1	Using where
2	SUBQUERY	supplier	eq_ref	PRIMARY	l_suppkey	1	
2	SUBQUERY	part	eq_ref	PRIMARY	l_partkey	1	Using where

# Materialization with NULLs (3)



- General idea
  - NULL acts as a “match-any” operator
  - If possible perform a lookup in the materialized subquery
  - Otherwise perform partial matching
    - Specialized per-column index merge (big data), or
    - Table scan + NULL test per column
- Optimizer switches
  - materialization=on,
  - partial\_match\_rowid\_merge=on
  - partial\_match\_table\_scan=on

# Derived tables (1)



```
SELECT max(t.l_extendedprice)
FROM orders,
    (SELECT * FROM lineitem
     WHERE l_shipdate between '1992-07-01' and '1992-08-31') AS t
WHERE t.l_orderkey=o_orderkey AND
    o_orderdate between '1992-05-01' AND '1992-08-31';
```

**MariaDB 5.2 (any MySQL): 6:37 min (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	1543786	
1	PRIMARY	orders	eq_ref	PRIMARY	t.l_orderkey	1	Using where
2	DERIVED	lineitem	range	i_l_shipdate	NULL	2420940	Using where

**EXPLAIN**  
**~6-8 min**

**MariaDB 5.3: 4:20 min (faster ~ 1.5x)**

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	orders	range	i_o_orderdate	NULL	353592	Using where; Using index
1	SIMPLE	lineitem	ref	PRIMARY	o_orderkey	2	Using where

**EXPLAIN**  
**0 sec**

# Derived tables – dynamic index (2)



```
SELECT MAX(o_totalprice)
FROM
(SELECT * FROM orders
 WHERE o_orderdate between '1992-04-01' and '1992-06-30') AS r,
(SELECT l_orderkey, MAX(l_shipdate) AS m FROM lineitem
 WHERE l_shipdate BETWEEN '1992-01-01' AND '1992-06-30'
 GROUP BY l_orderkey) AS t
WHERE t.l_orderkey=r.o_orderkey AND
      r.o_orderdate > m - interval 3 month AND
      r.o_orderstatus='F' AND r.o_totalprice > 400000;
```

# Derived tables – dynamic index (3)



**MariaDB 5.2 (any MySQL): 4:14 min (slow)**

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	1543786	
1	PRIMARY	orders	eq_ref	PRIMARY	t.l_orderkey	1	Using where
2	DERIVED	lineitem	range	i_l_shipdate	NULL	2420940	Using where

**EXPLAIN**  
**~1 min**

**MariaDB 5.3: 44.9 sec (faster ~ 5.6x)**

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	orders	range	i_o_orderdate	NULL	353592	Using where; Using index
1	SIMPLE	lineitem	ref	PRIMARY	o_orderkey	2	Using where

**EXPLAIN**  
**0 sec**

# Explain – consistently fast



- **Subquery in FROM**

```
select a, b, c
from (select ...) as t1, t2
where some_cond;
```

- **Subquery in SELECT**

```
select a, (select b, c from t1 ...)
from t2 where ...;
```

- **Single-row subquery**

```
select a, b, c
from t1
where (select sum(d) from ...) /
      (select count(*) from ...) > 5;
```

# Summary



- Subqueries in MariaDB run 2x ,10x, 100x, Infinitely faster
- Subqueries are usually as fast or faster than manually optimized queries
- Instant EXPLAIN in all cases
  
- Much less machine time
- Much less development time
  
- Questions? Suggestions?