New optimizer features in MySQL 6.0

Sergey Petrunia
sergefp@mysql.com

MySQL University, February 2008

Background: subquery processing before 6.0

- FROM subqueries are pre-materialized (early)
- Scalar subqueries use straightforward evaluation
- Predicate subqueries
  - May perform two kinds of rewrites
  - Then use straightforward evaluation
- Originally implemented in MySQL 4.1 by Sinisa (FROM subqueries) and Sanja (all other kinds of subqueries)

Processing subqueries in the FROM clause

```
SELECT ... FROM (SELECT ...) WHERE ...
```

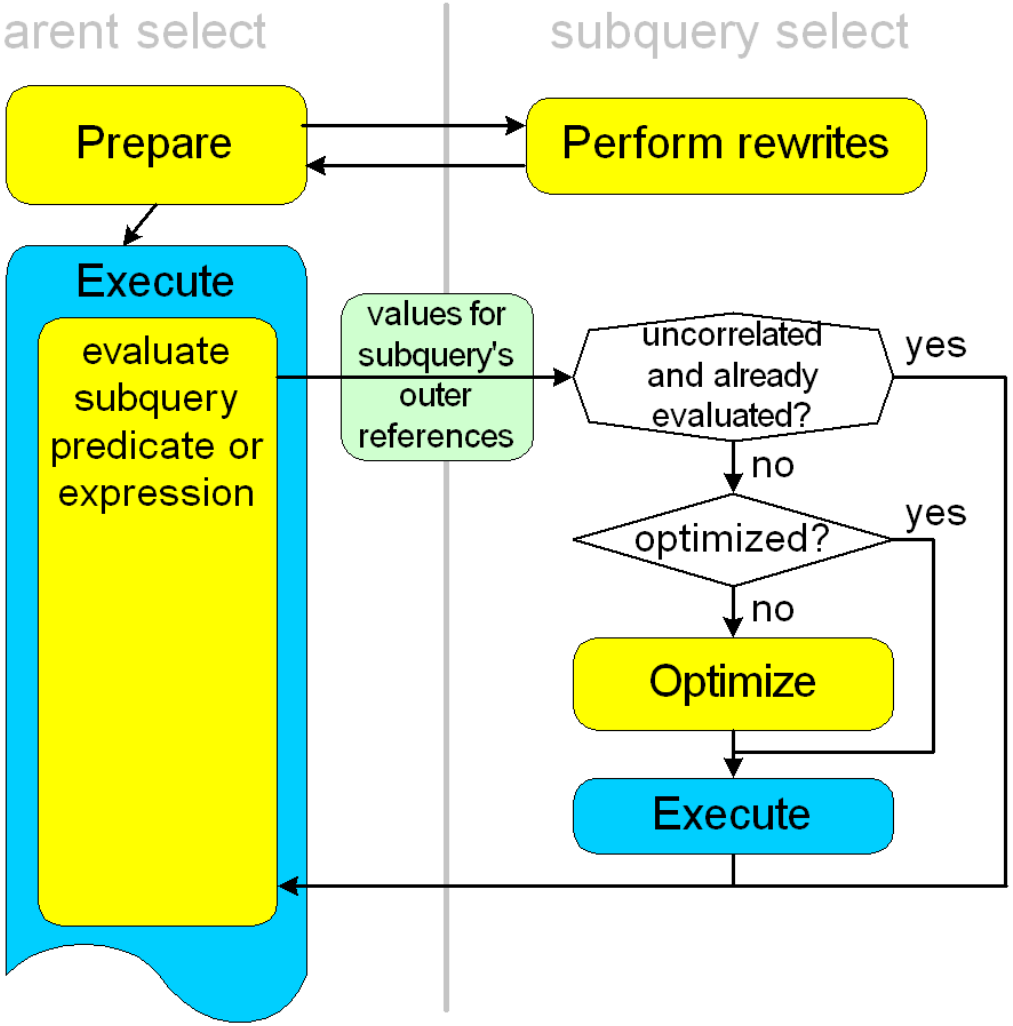
- Execution
  1. Optimize the subquery SELECT (as a separate query)
  2. Execute it and capture its result into a temporary table
  3. Optimize and execute the parent SELECT

- Properties

  No optimization (you may get some optimization and better performance if you define/use a VIEW equivalent to subquery)

Straightforward subquery evaluation

- Applied to all subqueries other than FROM:
  - IN
  - EXISTS
  - scalar context subqueries



Subquery rewrites: IN->EXISTS (1)

"Inform the subquery about which part of its resultset we're interested in"

- IN→EXISTS transformation

```
OuterExpr IN (SELECT InnerExpr FROM ...
              WHERE subq_where)
```

→

```
EXISTS (SELECT 1 FROM ...
        WHERE subq_where AND
              InnerExpr = OuterExpr)
```

NOTE: simplifed description, not counting cases with NULLs.

Subquery rewrites: MIN/MAX (2)

"Inform the subquery about which part of its resultset we're interested in"

- MIN/MAX Transformation

```
OuterExpr > ALL(SELECT InnerExpr FROM ...)
```

→

```
OuterExpr > (SELECT MAX(InnerExpr)
               FROM ... )
```

handles all similar cases with

OuterExpr {less|more}[or equal] {ALL|ANY} (SELECT...)

NOTE: simplifed description, not counting cases with NULLs or subqueries returning zero rows

Current state of subqueries: summary

- FROM subqueries
  - are always pre-materialized
- Scalar and predicate subqueries
  - Are evaluated using straightforward outer-to-inner strategy
    - (no re-evaluation for uncorellated subqueries)
  - The optimizations are two rule-based rewrites:
    - IN→EXISTS (pushdown the IN-equality)
    - ALL/ANY→MIN/MAX

New subquery optimizations

- Semi-join (WL#2980)
  - Table pull-out (WL#3740)
  - Duplicate elimination (WL#3741)
  - First match (WL#3750)
  - Inside-out (WL#3751)
- Materialization (WL#1110)
- @@optimizer_switch to turn the above on/off (WL#3952)

- Cost-based choice between semi-join and materialization (WL#3985)
- FROM subquery flattening (WL#3485)

Semi-join optimizaations

- Semi-join operation is defined as:

  t1 SEMI JOIN *t2* ON *sj_cond* =

  $\{ t1.\text{row} \mid \exists\ t2.\text{row}, sj\_cond(t1.\text{row}, t2.\text{row}) = \text{true} \}$

- Semi-join in SQL form:

```
SELECT * FROM t1
 WHERE (a1, a2, ...) IN (SELECT b1, b2, ...
                           FROM t2 ...)
```

  (here *sj_cond* is "(a1, a2, ...) = (b1, b2, ...)")

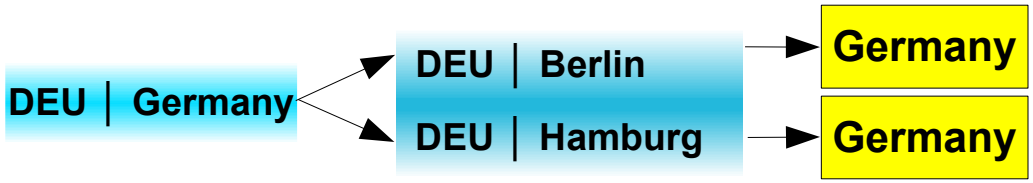
- A semi-join subquery is
  - IN subquery
  - AND-part of the WHERE clause
  - [MySQL-specific] Has no aggregates, GROUP BY, ORDER BY ... LIMIT, is not a UNION etc:
    (same criteria as in mergeable VIEWs but DISTINCT is allowed)

Semi-join vs. inner join semantics

The difference is in duplicate outer row combinations

```
SELECT Country.Name FROM Country
  WHERE Code IN (SELECT CountryCode FROM City
                 WHERE Population > 1M)
```


```
SELECT Country.Name FROM Country, City
  WHERE Country.Code=City.CountryCode AND
        City.Population > 1M
```



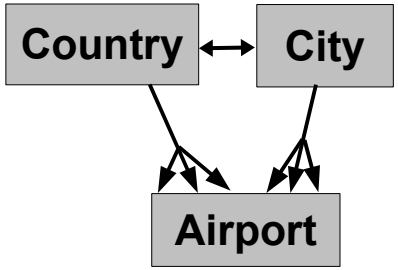
=> *semi-join is like inner join but we need some way to remove the duplicates*

Semi-join strategy #1: WL#3740 table pullout

```
SELECT Country.Name FROM Country
  WHERE Capital IN (SELECT ID FROM City
                    WHERE Population > 1M)
```

- In general case can only pull some tables:

```
SELECT Country.Name FROM Country
  WHERE Capital IN (SELECT City.ID
                    FROM City, Airport
                    WHERE Airport.City=City.Name)
```



If a subquery table is functionally dependent on the parent query tables, it can be "pulled out" of the subquery

WL#3740 table pullout : example

```
SELECT Name FROM Country
  WHERE Capital IN (SELECT City.ID FROM City
                    WHERE City.Population > 7000000);
```

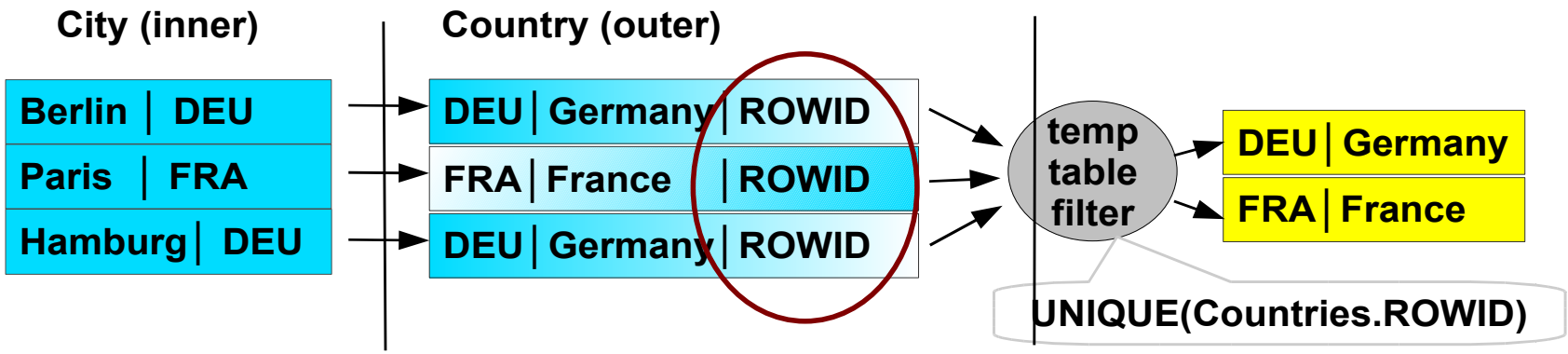
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	City	range	PRIMARY, Population	Population	4	NULL	15	Using index condition; Using MRR
1	PRIMARY	Country	ref	Capital	Capital	5	City.ID	1	

select `world`.`Country`.`Name` AS `Name` **from (`world`.`City`) join `world`.`Country`** where ((`world`.`Country`.`Capital` = `world`.`City`.`ID`) and (`world`.`City`.`Population` > 7000000))

Semi-join strategy #2: WL#3741 duplicate elimination

Use temporary table with unique key (or constraint) to eliminate duplicate row combinations of outer tables

```
SELECT Country.Name FROM Country
  WHERE Code IN (SELECT CountryCode FROM City
                 WHERE Population > 1M)
```



WL#3741: Duplicate elimination: example

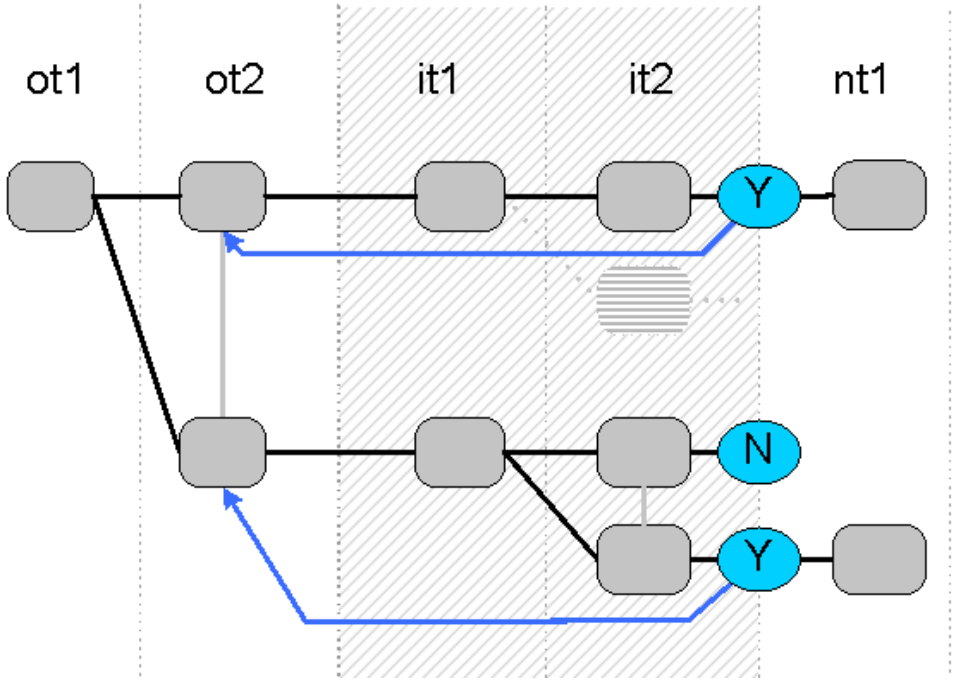
```
SELECT Name FROM City
WHERE
  City.ID IN (SELECT Capital FROM Country
              WHERE SurfaceArea > 1000000 AND
                    Country.Population >
                    10*City.Population);
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Country	ALL	Population	NULL	NULL	NULL	239	Using where; Start temporary
1	PRIMARY	City	eq_ref	PRIMARY	PRIMARY	4	Country.Capital	1	Using where; End temporary

select `world`.`City`.`Name` AS `Name` **from `world`.`City` semi join (`world`.`Country`)** where ((`world`.`City`.`ID` = `world`.`Country`.`Capital`) and (`world`.`Country`.`SurfaceArea` > 1000000) and (`world`.`Country`.`Population` > (10 * `world`.`City`.`Population`)))

Semi-join strategy #3: WL#3750 FirstMatch

If all correlated outer tables are followed by the inner, short-cut enumeration of subquery tables as soon as we get a matching row combination




```
if (table condition satisfied) {
  do join with next tables;
  jump out to the last otN;
} else {
  discard row combination;
  continue current table scan;
}
```

itN – inner; otN – outer correlated; ntN – outer uncorrelated

WL#3750: FirstMatch: example

```
SELECT Name FROM Country
WHERE
  Country.Population > 100000000 AND
  Country.Code IN (SELECT City.Country FROM City
                   WHERE City.ID != Country.Capital AND
                         Population > 5000000);
```

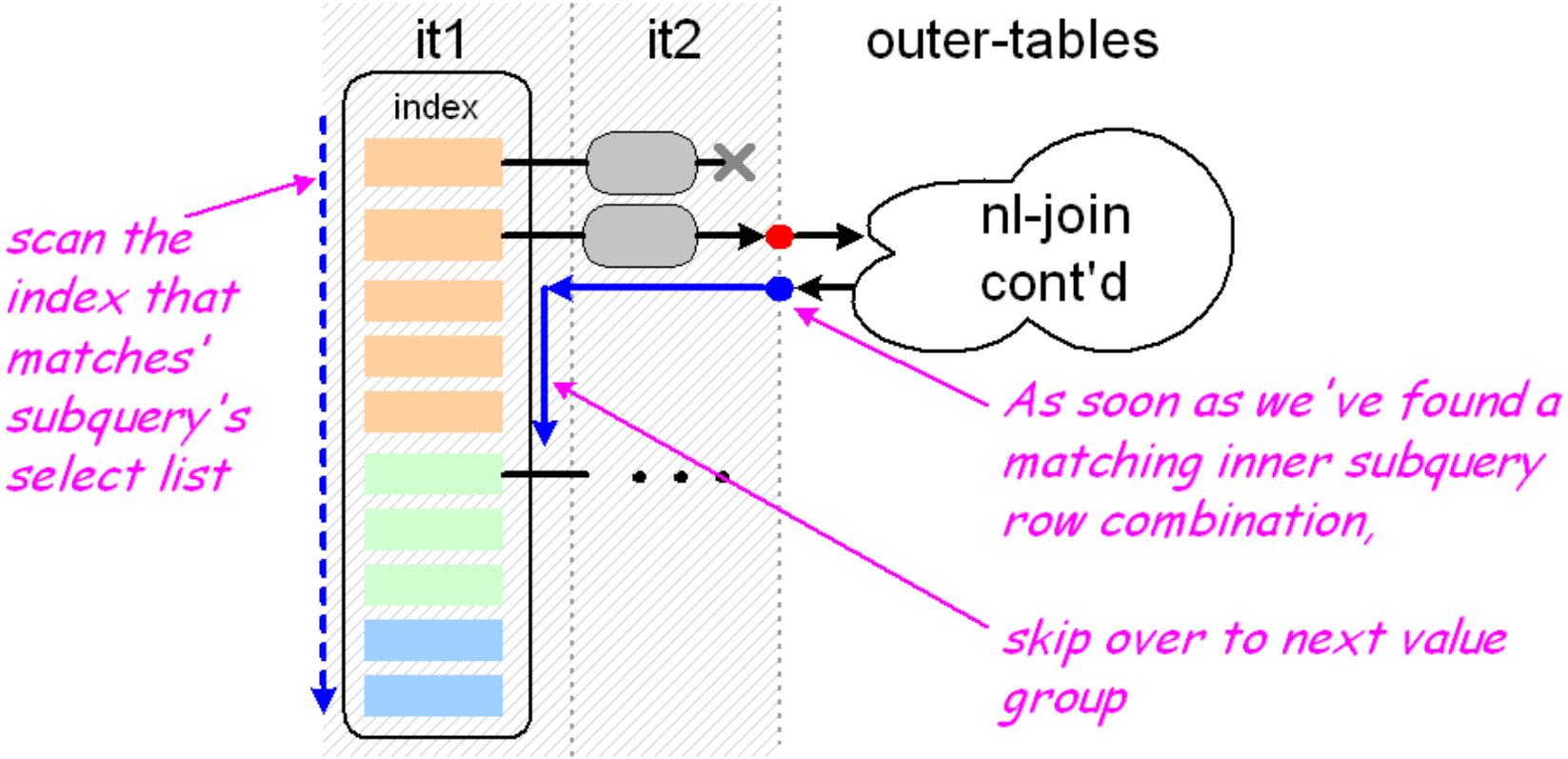
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	Country	range	PRIMARY, Population	Population	4	NULL	23	Using index condition; Using MRR
1	PRIMARY	City	ref	Population, Country	Country	3	Country.Code	18	Using where; **FirstMatch(Country)**

select `world`.`Country`.`Name` AS `Name` **from `world`.`Country` semi join (`world`.`City`)** where ((`world`.`City`.`Country` = `world`.`Country`.`Code`) and (`world`.`Country`.`Population` > 100000000) and (`world`.`City`.`ID` <> `world`.`Country`.`Capital`) and (`world`.`City`.`Population` > 5000000))

Semi join strategy #4: WL#3751 InsideOut

Scan inner table(s) in a way that doesn't produce duplicates

```
SELECT … FROM ot1, …
  WHERE outer_expr IN
    (SELECT it1.key FROM it1,it2 WHERE cond(it1, it2))
```



WL#3751: InsideOut: example

```
SELECT COUNT(Name) FROM Country
WHERE
  Country.Code IN (SELECT City.Country FROM City);
```

```
+----+-------------+---------+--------+---------------+---------+---------+--------------+------+-----------+
| id | select_type | table   | type   | possible_keys | key     | key_len | ref          | rows | Extra     |
+----+-------------+---------+--------+---------------+---------+---------+--------------+------+-----------+
|  1 | PRIMARY     | City    | index  | Country       | PRIMARY | 4       | NULL         | 4079 | LooseScan |
|  1 | PRIMARY     | Country | eq_ref | PRIMARY       | PRIMARY | 3       | City.Country |    1 |           |
+----+-------------+---------+--------+---------------+---------+---------+--------------+------+-----------+
```

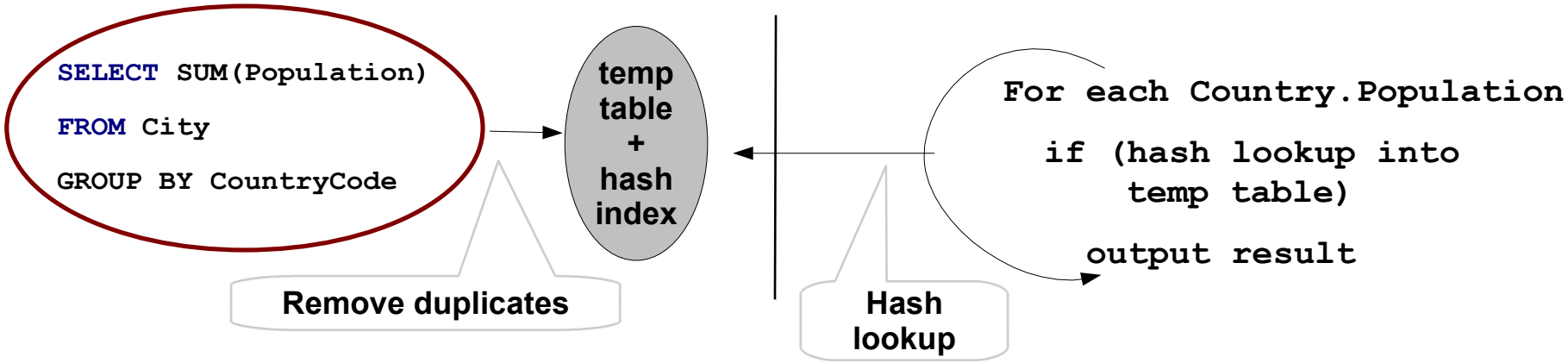
```
select count(`world`.`Country`.`Name`) AS `COUNT(Name)` from
`world`.`Country` semi join (`world`.`City`) where (`world`.`Country`.`Code` =
`world`.`City`.`Country`)
```

WL#1110 Materialization

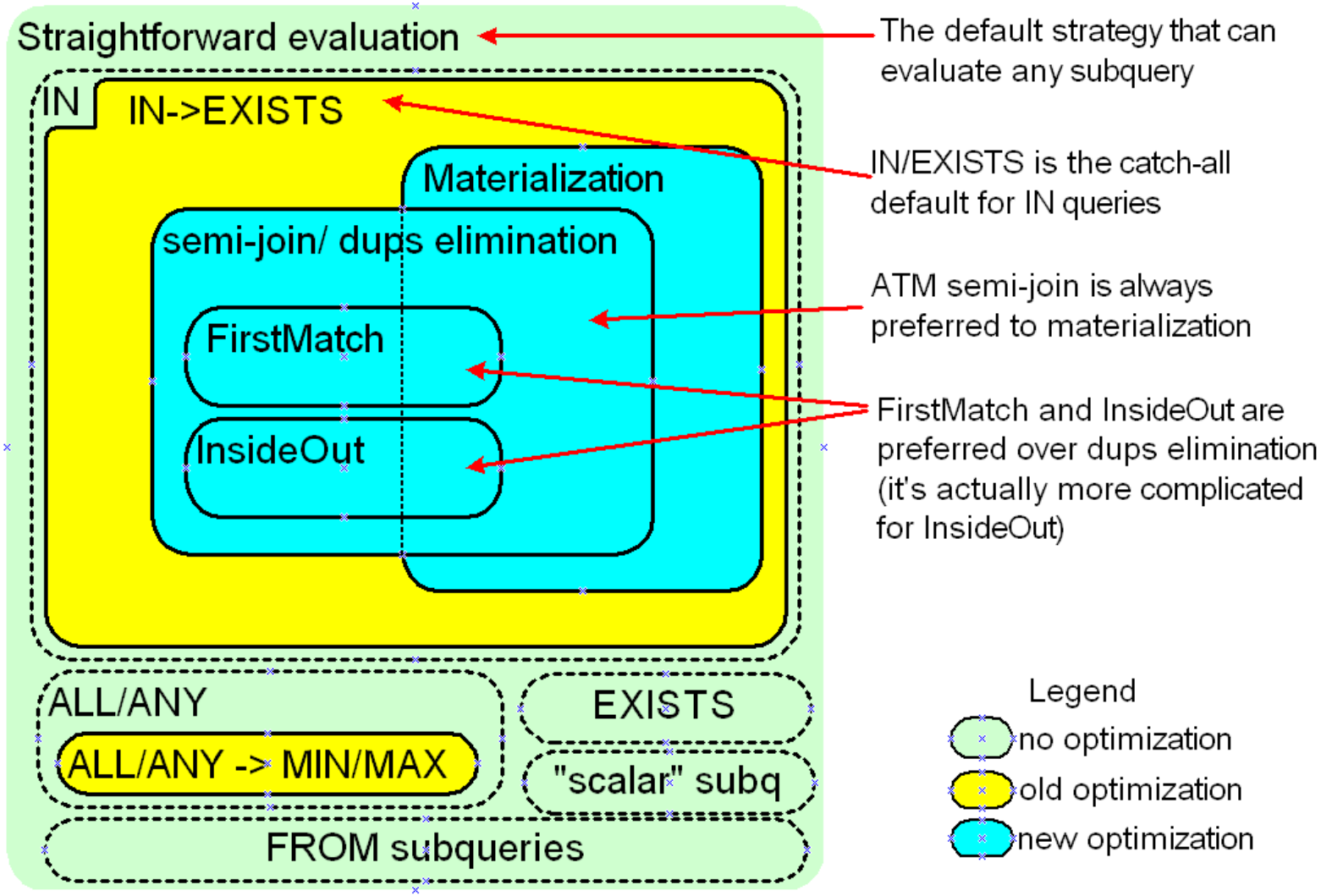
Use temporary table with unique constraint to

- Materialize the subquery result
- Create unique hash index on it
- Lookup outer tuples into temp table index

```
SELECT Country.Name FROM Country
WHERE Population IN (SELECT SUM(Population) FROM
City
                     GROUP BY CountryCode)
```



Summing up: query/strategy coverage



Controlling new subquery optimizations

- Currently, a server variable:

```
@@optimizer_switch =
    'no_semijoin,no_materialization'
```

(like set-type column: any order, no space after comma)

  - this will likely to change into being a part of a bigger optimization on/off scheme (WL#4046, public)

- Already seeing a need for hints but no WL for this yet
  - thinking of syntax like

```
outer_expr IN (SELECT no_materialize ...)
```

  - No idea how to specify sort order after flattening (input welcome)

Future subquery work

- Doing now
  - Fixing bugs in what is already implemented
  - WL#3485 FROM subquery flattening (Evgen)
  - WL#3985 Make smart choice between semi-join and materialization
  - WL#3830 Partial matching of tuples with NULL components
    - let materialization handle
      - NULL IN (SELECT ...)
      - smth IN (SELECT something_that_maybe_null)
- Intend to do
  - Subquery Hints

A simple benchmark: query 1

```
SELECT COUNT(l_orderkey) FROM lineitem
  WHERE l_linenumber=1 AND
        l_orderkey IN
        (SELECT o_orderkey FROM orders
           WHERE o_totalprice > 1000 AND
                 o_custkey IN
                 (SELECT c_custkey FROM customer
                    WHERE c_address LIKE 'Le%'));
```

	Wallclock time	# reads
MySQL 5.1	12 min	9,001,055
MySQL 5.2	1.8 sec	153,008
MySQL 5.2 no_semijoin	25 sec	7,651,215
PostgreSQL 8.2.5	0.1 sec	2,413

A simple benchmark: query 1: EXPLAINs

MySQL 5.1:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	lineitem	index	NULL	i_l_shipdate	4	NULL	5994679	Using where; Using index
2	DEPENDENT SUBQUERY	orders	unique_subquery	PRIMARY	PRIMARY	4	func	1	Using where
3	DEPENDENT SUBQUERY	customer	unique_subquery	PRIMARY	PRIMARY	4	func	1	Using where

MySQL 5.2:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	customer	ALL	PRIMARY	NULL	NULL	NULL	151635	Using where; Start temporary
1	PRIMARY	orders	ref	PRIMARY, i_o_custkey	i_o_custkey	5	dbt3.customer.c_custkey	8	Using where
1	PRIMARY	lineitem	eq_ref	PRIMARY, i_l_orderkey, i_l_orderkey_quantity	PRIMARY	8	dbt3.orders.o_orderkey, const	1	Using index; End temporary

select count(`dbt3`.`lineitem`.`l_orderkey`) AS `COUNT(l_orderkey)` **from `dbt3`.`lineitem` semi join (`dbt3`.`customer` join `dbt3`.`orders`)** where ((`dbt3`.`lineitem`.`l_orderkey` = `dbt3`.`orders`.`o_orderkey`) and (`dbt3`.`orders`.`o_custkey` = `dbt3`.`customer`.`c_custkey`) and (`dbt3`.`lineitem`.`l_linenumber` = 1) and (`dbt3`.`orders`.`o_totalprice` > 1000) and (`dbt3`.`customer`.`c_address` like _latin1'Le%'))

A simple benchmark: query 2

```
SELECT COUNT(s_name) FROM supplier
WHERE s_suppkey IN
    (SELECT l_suppkey FROM lineitem
      WHERE l_quantity > 10 AND
          l_orderkey IN
         (SELECT o_orderkey FROM orders
           WHERE o_totalprice > 1000 AND
               o_custkey IN
               (SELECT c_custkey FROM customer
                 WHERE c_nationkey = s_nationkey AND
                     c_nationkey IN
                     (SELECT n_nationkey FROM nation
                       WHERE n_name IN ('FRANCE','ITALY')))));
```

	Wallclock time	# reads
MySQL 5.1	1 hr 4 min	15,206,858
MySQL 5.2	47.2 sec	511,535
PostgresSQL 8.2.5	14 min 17 sec	~5,200,000

A simple benchmark: query 2: EXPLAINs

MySQL 5.1:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	supplier	ALL	NULL	NULL	NULL	NULL	10149	Using where
2	DEPENDENT SUBQUERY	lineitem	index_subquery	i_l_suppkey	i_l_suppkey	5	func	276	Using where
3	DEPENDENT SUBQUERY	orders	unique_subquery	PRIMARY	PRIMARY	4	func	1	Using where
4	DEPENDENT SUBQUERY	customer	unique_subquery	PRIMARY, i_c_nationkey	PRIMARY	4	func	1	Using where
5	DEPENDENT SUBQUERY	nation	unique_subquery	PRIMARY	PRIMARY	4	func	1	Using where

MySQL 5.2:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	nation	ALL	PRIMARY	NULL	NULL	NULL	25	Using where
1	PRIMARY	customer	ref	PRIMARY, i_c_nationkey	i_c_nationkey	5	dbt3.nation.n_nationkey	3159	Using index; Start temporary
1	PRIMARY	orders	ref	PRIMARY, i_o_custkey	i_o_custkey	5	dbt3.customer.c_custkey	8	Using where
1	PRIMARY	lineitem	ref	PRIMARY, i_l_suppkey, i_l_orderkey, i_l_orderkey_quantity	PRIMARY	4	dbt3.orders.o_orderkey	2	Using where
1	PRIMARY	supplier	eq_ref	PRIMARY, i_s_nationkey	PRIMARY	4	dbt3.lineitem.l_suppkey	1	Using where; End temporary

select count(`dbt3`.`supplier`.`s_name`) AS `COUNT(s_name)` **from `dbt3`.`supplier` semi join (`dbt3`.`nation` join `dbt3`.`customer` join `dbt3`.`orders` join `dbt3`.`lineitem`)** where ((`dbt3`.`supplier`.`s_suppkey` = `dbt3`.`lineitem`.`l_suppkey`) and (`dbt3`.`lineitem`.`l_orderkey` = `dbt3`.`orders`.`o_orderkey`) and (`dbt3`.`orders`.`o_custkey` = `dbt3`.`customer`.`c_custkey`) and (`dbt3`.`customer`.`c_nationkey` = `dbt3`.`nation`.`n_nationkey`) and (`dbt3`.`supplier`.`s_nationkey` = `dbt3`.`nation`.`n_nationkey`) and (`dbt3`.`lineitem`.`l_quantity` > 10) and (`dbt3`.`orders`.`o_totalprice` > 1000) and (`dbt3`.`nation`.`n_name` in (_latin1'FRANCE',_latin1'ITALY')))

The end

Comments or questions?