SQL Query Patterns, Optimized

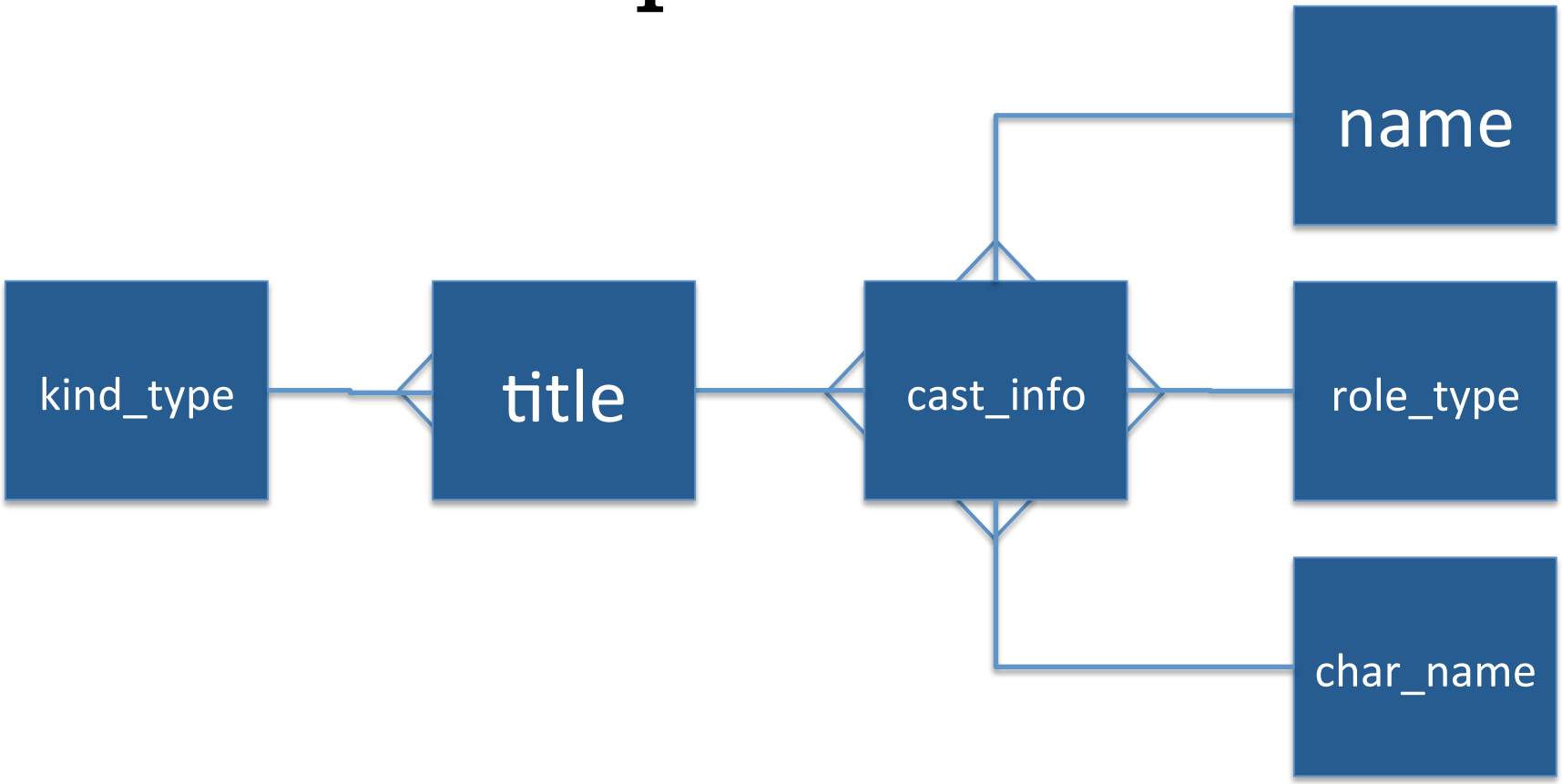
Bill Karwin

Percona University 2013

How Do We Optimize?

- Identify queries.
- Measure optimization plan and performance.
  - EXPLAIN
  - SHOW SESSION STATUS
  - SHOW PROFILES
- Add indexes *and/or* redesign the query.

Example Database



Common Query Patterns

1. Exclusion Joins
2. Random Selection
3. Greatest per Group
4. Dynamic Pivot
5. Relational Division

Query Patterns

EXCLUSION JOINS

Assignment:

"I want to find recent movies that had no director."

Not Exists Solution

```
SELECT t.title
FROM title t
WHERE kind_id = 1
AND production_year >= 2005
AND NOT EXISTS (
  SELECT * FROM cast_info c
  WHERE c.movie_id = t.id
    AND c.role_id = 8 /* director */
);
```

Movies

In the range of recent years

Correlated subquery to find a director for each movie

Not Exists Solution


```
SELECT t.title
FROM title t
WHERE kind_id = 1
AND production_year >= 2005
AND NOT EXISTS (
  SELECT * FROM cast_info c
  WHERE c.movie_id = t.id
    AND c.role_id = 8
);
```

I gave up after waiting > 1 hour

Indexes: the Not-Exists Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year);

CREATE INDEX m_r
ON cast_info (movie_id, role_id);
```

EXPLAIN: the Not-Exists Solution

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	t	range	k_py	NULL	189846	Using where
2	DEPENDENT SUBQUERY	c	ref	m_r	t.id, const	105654	Using index

Dependent subquery executes once for each set of values in outer

At least both table references use indexes

The correlated subquery is executed 189k times!

A covering index is best—if the index fits in memory

Not Exists Solution


```
SELECT t.title
FROM title t
WHERE kind_id = 1
AND production_year >= 2005
AND NOT EXISTS (
  SELECT * FROM cast_info c
  WHERE c.movie_id = t.id
    AND c.role_id = 8
);
```

Better, but when the indexes aren't in memory, it's still too slow

Buffer Pool

- It's crucial that queries read an index from memory; I/O during an index scan kills performance.

```
[mysqld]
innodb_buffer_pool_size = 64M # wrong
innodb_buffer_pool_size =  2G # better
```

Not Exists Solution


```
SELECT t.title
FROM title t
WHERE kind_id = 1
AND production_year >= 2005
AND NOT EXISTS (
  SELECT * FROM cast_info c
  WHERE c.movie_id = t.id
    AND c.role_id = 8
);
```

That's a little better

SHOW SESSION STATUS

- Shows the real count of row accesses for your current session.

```
mysql> FLUSH STATUS;
mysql> ... run a query ...
mysql> SHOW SESSION STATUS;
```

Status: the Not-Exists Solution

Variable_name	Value
Handler_commit	7
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	3
Handler_read_key	652715
Handler_read_last	0
Handler_read_next	652710
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	39
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	19

read_key: lookup by index, e.g. each lookup in *cast_info,* plus the first row in *title*

read_next: advancing in index order, e.g. the range query for rows in *title* after the first row

SHOW PROFILE

- Enable query profiler for the current session.

```
mysql> SET PROFILING = 1;
```

- Run a query.

```
mysql> SELECT t.title FROM title t ...
```

- Query the real execution time.

```
mysql> SHOW PROFILES;
```

- Query detail for a specific query.

```
mysql> SHOW PROFILE FOR QUERY 1;
```

Profile: the Not-Exists Solution

```
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| executing          | 0.000002 |
| Sending data       | 0.000011 |
. . .
| executing          | 0.000002 |
| Sending data       | 0.000011 |
| executing          | 0.000002 |
| Sending data       | 0.000011 |
| executing          | 0.000002 |
| Sending data       | 0.000011 |
| executing          | 0.000001 |
| Sending data       | 0.000094 |
| end                | 0.000006 |
| query end          | 0.000004 |
| closing tables     | 0.000032 |
| freeing items      | 0.001833 |
| logging slow query | 0.000002 |
| cleaning up        | 0.000004 |
+--------------------+----------+
```

Thousands of iterations of correlated subqueries cause the profile information to overflow!

Not-In Solution

0.81s

```
SELECT title
FROM title
WHERE kind_id = 1
AND production_year >= 2005
AND id NOT IN (
  SELECT movie_id FROM cast_info
  WHERE role_id = 8
);
```

Not a correlated subquery

Indexes: the Not-In Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year);

CREATE INDEX r_m
ON cast_info (role_id, movie_id);
```

EXPLAIN: the Not-In Solution

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	title	range	k_py	NULL	189846	Using where
1	DEPENDENT SUBQUERY	cast_ info	index_subquery	m_r	func, const	1	Using index; Using where

But somehow MySQL doesn't report a different select type

Status: the Not-In Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	93245
Handler_read_last	0
Handler_read_next	93244
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

A fraction as many rows accessed as the NOT EXISTS solution

Profile: the Not-In Solution

Status	Duration
starting	0.000095
checking permissions	0.000004
checking permissions	0.000007
Opening tables	0.000030
System lock	0.000012
init	0.000038
optimizing	0.000015
statistics	0.000105
preparing	0.000018
executing	0.000002
Sending data	0.000061
optimizing	0.000006
statistics	0.000054
preparing	0.809558
end	0.000014
query end	0.000075
closing tables	0.000037
freeing items	0.000356
logging slow query	0.000002
cleaning up	0.000005

Most of the time spent in “preparing”—???

Outer-Join Solution

0.71s

```
SELECT t.title
FROM title t
LEFT OUTER JOIN cast_info c
  ON t.id = c.movie_id
  AND c.role_id = 8
WHERE t.kind_id = 1
AND t.production_year >= 2005
AND c.movie_id IS NULL;
```

Try to find a director for each movie using a join

If no director is found, that's the one we want

Indexes: the Outer-Join Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year);

CREATE INDEX m_r
ON cast_info (movie_id, role_id);
```

EXPLAIN: the Outer-Join Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	t	range	k_py	NULL	189846	Using where
1	SIMPLE	c	ref	m_r	t.id, const	105654	Using where; Using index; Not exists

Special “not exists” optimization

Status: the Outer-Join Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	93245
Handler_read_last	0
Handler_read_next	93244
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

Curiously, this is exactly the same as the NOT IN solution

Profile: the Outer-Join Solution

Status	Duration
starting	0.000096
checking permissions	0.000004
checking permissions	0.000007
Opening tables	0.000029
System lock	0.000012
init	0.000028
optimizing	0.000016
statistics	0.000375
preparing	0.000023
executing	0.000002
Sending data	0.709222
end	0.000014
query end	0.000004
closing tables	0.000031
freeing items	0.000421
logging slow query	0.000085
cleaning up	0.000039

Similar here too, but even a little better

Summary: Exclusion Joins

Solution	Time	Notes
Not-Exists	1.20s	correlated subquery
Not-In	0.81s	
Outer-Join	0.71s	“not exists” optimization

Query Patterns

RANDOM SELECTION

Assignment:

"I want a query that picks a random movie."

Naïve Order-By Solution

8.80s

```
SELECT *
FROM title
WHERE kind_id = 1 /* movie */
ORDER BY RAND()
LIMIT 1;
```

Indexes: the Outer-Join Solution

```
CREATE INDEX k
ON title (kind_id);
```

EXPLAIN: the Order-By Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	title	ref	k	const	787992	Using temporary; Using filesort

Status: the Order-By Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	1
Handler_read_last	0
Handler_read_next	473582
Handler_read_prev	0
Handler_read_rnd	1
Handler_read_rnd_next	473583
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	473582

Profile: the Order-By Solution

Status	Duration
starting	0.000074
checking permissions	0.000032
Opening tables	0.000035
System lock	0.000012
init	0.000025
optimizing	0.000004
statistics	0.000014
preparing	0.000010
Creating tmp table	0.000245
executing	0.000003
Copying to tmp table	4.875666
Sorting result	3.871513
Sending data	0.000059
end	0.000005
removing tmp table	0.058239
end	0.000018
query end	0.000064
closing tables	0.000034
freeing items	0.000210
logging slow query	0.000003
cleaning up	0.000005

Offset Solution

0.00-1.12s

```
SELECT ROUND(RAND() * COUNT(*))
FROM title
WHERE kind_id = 1;

SELECT *
FROM title
WHERE kind_id = 1
LIMIT 1 OFFSET $random;
```

Indexes: the Offset Solution

```
CREATE INDEX k
ON title (kind_id);
```

EXPLAIN: the Offset Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	title	ref	k	const	787992	

Status: the Offset Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	1
Handler_read_last	0
Handler_read_next	470000
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

Profile: the Offset Solution

Status	Duration
starting	0.000069
checking permissions	0.000009
Opening tables	0.000026
System lock	0.000037
init	0.000036
optimizing	0.000009
statistics	0.000081
preparing	0.000015
executing	0.000002
Sending data	1.118662
end	0.000013
query end	0.000005
closing tables	0.000029
freeing items	0.000202
logging slow query	0.000002
cleaning up	0.000004

Primary Key Solution

0.0005s

```
SELECT ROUND(RAND() * COUNT(*))
FROM title
WHERE kind_id = 1;

SELECT *
FROM title
WHERE id > $random
LIMIT 1;
```

EXPLAIN: the Primary Key Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	title	range	PRIMARY	NULL	787992	Using where

Status: the Primary Key Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	1
Handler_read_last	0
Handler_read_next	0
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

Profile: the Primary Key Solution

Status	Duration
starting	0.000070
checking permissions	0.000009
Opening tables	0.000027
System lock	0.000017
init	0.000029
optimizing	0.000008
statistics	0.000088
preparing	0.000030
executing	0.000003
Sending data	0.000235
end	0.000011
query end	0.000003
closing tables	0.000022
freeing items	0.000018
logging slow query	0.000001
cleaning up	0.000001

Summary: Random Selection

Solution	Time	Notes
Order-By Solution	8.80s	
Offset Solution	1.12s	Requires the count
Primary Key Solution	0.0005s	Requires the count

Query Patterns

GREATEST PER GROUP

Assignment:

"I want the last episode of every TV series."

Getting the Last Episode

```
SELECT tv.title, ep.title,
  MAX(ep.episode_nr) AS last_ep
FROM title ep
JOIN title tv ON tv.id = ep.episode_of_id
WHERE ep.kind_id = 7 /* TV show */
GROUP BY ep.episode_of_id ORDER BY NULL;
```

This is *not* the title of the last episode!

Why Isn't It?

- The query doesn't necessarily return the title from the row where `MAX(ep.episode_nr)` occurs.
- Should the following return the title of the *first* episode or the *last* episode?

```
SELECT tv.title, ep.title,
  MIN(ep.episode_nr) AS first_ep
  MAX(ep.episode_nr) AS last_ep
FROM . . .
```

Exclusion Join Solution


```
SELECT tv.title, ep1.title, ep1.episode_nr
FROM title ep1
LEFT OUTER JOIN title ep2
  ON ep1.kind_id = ep2.kind_id
  AND ep1.episode_of_id = ep2.episode_of_id
  AND ep1.episode_nr < ep2.episode_nr
JOIN title tv ON tv.id = ep1.episode_of_id
WHERE ep1.kind_id = 7
  AND ep1.episode_of_id IS NOT NULL
  AND ep1.episode_nr >= 1
  AND ep2.episode_of_id IS NULL;
```

Try to find a row *ep2* for the same show with a greater *episode_nr*

If no such row is found, then *ep1* must be the last episode for the show

Indexes: the Exclusion-Join Solution

```
CREATE INDEX k_ep_nr
ON title (kind_id, episode_of_id, episode_nr);
```

EXPLAIN: the Exclusion-Join Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	ep1	ref	k_py	const	787992	
1	SIMPLE	ep2	ref	k_ep_nr	const, ep1.episode_of_id	7879	Using where; Using index
1	SIMPLE	tv	eq_ref	PRIMARY	ep1.episode_of_id	1	

Status: the Exclusion-Join Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	673526
Handler_read_last	0
Handler_read_next	254373071
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

Unfortunately, this seems to be $O(n^2)$

Profile: the Exclusion-Join Solution

Status	Duration
starting	0.000104
checking permissions	0.000004
checking permissions	0.000001
checking permissions	0.000007
Opening tables	0.000029
System lock	0.000012
init	0.000034
optimizing	0.000020
statistics	0.000150
preparing	0.000090
executing	0.000011
Sending data	89.586871
end	0.000006
query end	0.000018
closing tables	0.000197
freeing items	0.001682
logging slow query	0.000073
logging slow query	0.000305
cleaning up	0.000091

A lot of time is spent moving rows around

Derived-Table Solution

0.60s

```
SELECT tv.title, ep.title, ep.episode_nr
FROM (
  SELECT kind_id, episode_of_id,
    MAX(episode_nr) AS episode_nr
  FROM title
  WHERE kind_id = 7
  GROUP BY kind_id, episode_of_id
) maxep
JOIN title ep USING (kind_id, episode_of_id, episode_nr)
JOIN title tv ON tv.id = ep.episode_of_id;
```

Generate a list of the greatest episode number per show

Indexes: the Derived-Table Solution

```
CREATE INDEX k_ep_nr
ON title (kind_id, episode_of_id, episode_nr);
```

EXPLAIN: the Derived-Table Solution

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	30323	
1	PRIMARY	ep	ref	k_ep_nr	maxep.kind_id, maxep.episode_of_id, maxep.episode_nr	7646	Using where
1	PRIMARY	tv	eq_ref	PRIMARY	maxep.episode_of_id	1	
2	DERIVED	title	range	k_ep_nr	NULL	100	Using where; Using index; Using index for group-by

Status: the Derived-Table Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	110312
Handler_read_last	1
Handler_read_next	28989
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	30324
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	30323

Evidence of a temporary table, even though EXPLAIN didn't report it

Profile: the Derived-Table Solution

Status	Duration
starting	0.000119
checking permissions	0.000005
checking permissions	0.000002
checking permissions	0.000008
Opening tables	0.000035
System lock	0.000086
optimizing	0.000011
statistics	0.000149
preparing	0.000017
executing	0.000006
Sorting result	0.000002
Sending data	0.319519
init	0.000059
optimizing	0.000014
statistics	0.000036
preparing	0.000019
executing	0.000002
Sending data	0.280462
end	0.000010
query end	0.000005
closing tables	0.000002
removing tmp table	0.000010
closing tables	0.000029
freeing items	0.001882
logging slow query	0.000082
cleaning up	0.000139

Evidence of a temporary table, even though EXPLAIN didn't report it

Summary: Greatest per Group

Solution	Time	Notes
Exclusion-join solution	89.59s	Bad when each group has many entries.
Derived-table solution	0.60s	

Query Patterns

DYNAMIC PIVOT

Assignment:

"I want the count of movies, TV, and video games per year—in columns."

Not Like This

```
SELECT k.kind, t.production_year, COUNT(*) AS Count
FROM kind_type k
JOIN title t ON k.id = t.kind_id
WHERE production_year BETWEEN 2005 AND 2009
GROUP BY k.id, t.production_year;
```

kind	production_year	Count
movie	2005	13807
movie	2006	13916
movie	2007	14494
movie	2008	18354
movie	2009	23714
tv series	2005	3248
tv series	2006	3588
tv series	2007	3361
tv series	2008	3026
tv series	2009	2572

Like This

kind	Count2005	Count2006	Count2007	Count2008	Count2009
episode	36138	24745	22335	16448	12917
movie	13807	13916	14494	18354	23714
tv movie	3541	3561	3586	3025	2778
tv series	3248	3588	3361	3026	2572
video game	383	367	310	300	215
video movie	7693	7671	6955	5808	4090

Do It in One Pass

1.78s

SUM of 1's = COUNT where condition is true

```
SELECT k.kind,
 SUM(production_year=2005) AS Count2005,
 SUM(production_year=2006) AS Count2006,
 SUM(production_year=2007) AS Count2007,
 SUM(production_year=2008) AS Count2008,
 SUM(production_year=2009) AS Count2009
FROM title t
JOIN kind_type k ON k.id = t.kind_id
GROUP BY t.kind_id ORDER BY NULL;
```

Indexes: the One-Pass Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year);
```

EXPLAIN: the One-Pass Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	k	index	PRIMARY	NULL	7	Using index; Using temporary
1	SIMPLE	t	ref	k_py	k.id	7687	Using index

Status: the One-Pass Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	1
Handler_read_key	1543727
Handler_read_last	0
Handler_read_next	1543726
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	7
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	1543713
Handler_write	6

title table has 1.5M rows; that's how many times it increments counts in the temp table

Profile: the One-Pass Solution

Status	Duration
starting	0.000135
checking permissions	0.000003
checking permissions	0.000007
Opening tables	0.000031
System lock	0.000014
init	0.000032
optimizing	0.000010
statistics	0.000035
preparing	0.000014
Creating tmp table	0.022591
executing	0.000006
Copying to tmp table	1.754729
Sending data	0.000046
end	0.000003
removing tmp table	0.000031
end	0.000005
query end	0.000004
closing tables	0.000033
freeing items	0.000211
logging slow query	0.000009
cleaning up	0.000004

majority of time spent building temp table

One-Pass with Straight-Join Optimizer Override


```
SELECT STRAIGHT_JOIN k.kind,
 SUM(production_year=2005) AS Count2005,
 SUM(production_year=2006) AS Count2006,
 SUM(production_year=2007) AS Count2007,
 SUM(production_year=2008) AS Count2008,
 SUM(production_year=2009) AS Count2009
FROM title t
JOIN kind_type k ON k.id = t.kind_id
GROUP BY t.kind_id ORDER BY NULL;
```

Indexes: the Straight-Join Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year);
```

EXPLAIN: the Straight-Join Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	t	index	k_py	NULL	1537429	Using index
1	SIMPLE	k	eq_ref	PRIMARY	t.kind_id	1	

Status: the Straight-Join Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	1
Handler_read_key	7
Handler_read_last	0
Handler_read_next	1543719
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

really one-pass

Profile: the Straight-Join Solution

Status	Duration
starting	0.000161
checking permissions	0.000005
checking permissions	0.000008
Opening tables	0.000031
System lock	0.000014
init	0.000032
optimizing	0.000010
statistics	0.000032
preparing	0.000016
executing	0.000008
Sorting result	0.000002
Sending data	0.848812
end	0.000013
query end	0.000078
closing tables	0.000272
freeing items	0.000215
logging slow query	0.000009
cleaning up	0.000005

no temporary table!

majority of time spent just moving rows

Scalar Subquery Solution

0.08s

```
SELECT k.kind,
(SELECT COUNT(*) FROM title WHERE kind_id = k.id AND production_year = 2005) AS Count2005,
(SELECT COUNT(*) FROM title WHERE kind_id = k.id AND production_year = 2006) AS Count2006,
(SELECT COUNT(*) FROM title WHERE kind_id = k.id AND production_year = 2007) AS Count2007,
(SELECT COUNT(*) FROM title WHERE kind_id = k.id AND production_year = 2008) AS Count2008,
(SELECT COUNT(*) FROM title WHERE kind_id = k.id AND production_year = 2009) AS Count2009
FROM kind_type k;
```

Indexes: the Scalar Subquery Solution

```
CREATE INDEX k_py
ON title (kind_id, production_year)

CREATE UNIQUE INDEX kind
ON kind_type (kind);
```

EXPLAIN: the Scalar Subquery Solution

id	select_type	table	type	key	ref	rows	Extra
1	PRIMARY	k	index	kind	NULL	7	Using index
6	DEPENDENT SUBQUERY	title	ref	k_py	k.id, const	87554	Using where; Using index
5	DEPENDENT SUBQUERY	title	ref	k_py	k.id, const	87554	Using where; Using index
4	DEPENDENT SUBQUERY	title	ref	k_py	k.id, const	87554	Using where; Using index
3	DEPENDENT SUBQUERY	title	ref	k_py	k.id, const	87554	Using where; Using index
2	DEPENDENT SUBQUERY	title	ref	k_py	k.id, const	87554	Using where; Using index

Status: the Scalar Subquery Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	1
Handler_read_key	36
Handler_read_last	0
Handler_read_next	262953
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

really good use of indexes

Profile: the Scalar Subquery Solution

```
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000009 |
| Opening tables       | 0.000042 |
| System lock          | 0.000017 |
| init                 | 0.000065 |
| optimizing           | 0.000006 |
| statistics           | 0.000016 |
| preparing            | 0.000010 |
. . .
| executing            | 0.000059 |
| Sending data         | 0.001621 |
| executing            | 0.000002 |
| Sending data         | 0.000854 |
| end                  | 0.000002 |
| query end            | 0.000000 |
| closing tables       | 0.000200 |
| freeing items        | 0.000293 |
| logging slow query   | 0.000015 |
| cleaning up          | 0.000019 |
+----------------------+----------+
```

Summary: Dynamic Pivot

Solution	Time	Notes
One-pass solution	1.78s	
Straight-join solution	0.85s	
Scalar Subquery solution	0.08s	

Query Patterns

RELATIONAL DIVISION

Assignment:

"I want to see movies with all three of keywords *espionage*, *nuclear-bomb*, and *ejector-seat*."

Not Movies with *One* Keyword

```
SELECT t.title, k.keyword FROM keyword k
JOIN movie_keyword mk ON k.id = mk.keyword_id
JOIN title t ON mk.movie_id = t.id
WHERE k.keyword IN ('espionage', 'nuclear-bomb', 'ejector-seat');
```

```
+--------------------------+--------------+
| title                    | keyword      |
+--------------------------+--------------+
| 2 Fast 2 Furious         | ejector-seat |
| Across the Pacific       | espionage    |
| Action in Arabia         | espionage    |
. . .
| You Only Live Twice      | espionage    |
| Zombie Genocide          | nuclear-bomb |
| Zombies of the Strat     | espionage    |
+--------------------------+--------------+
705 rows in set (12.97 sec)
```

This Won't Work

```
SELECT t.title, k.keyword
FROM keyword k
JOIN movie_keyword mk ON k.id = mk.keyword_id
JOIN title t ON mk.movie_id = t.id
WHERE k.keyword = 'espionage'
  AND k.keyword = 'nuclear-bomb'
  AND k.keyword = 'ejector-seat';

0 rows in set (12.97 sec)
```

It's impossible for one column to have three values on a given row

Only Movies with All Three

```
+------------+-------------------------------------+
| title      | keywords                            |
+------------+-------------------------------------+
| Goldfinger | ejector-seat,espionage,nuclear-bomb |
+------------+-------------------------------------+
```

Group-by Solution

0.10s

```
SELECT t.title, GROUP_CONCAT(k.keyword) AS keywords
FROM title t
JOIN movie_keyword mk ON t.id = mk.movie_id
JOIN keyword k ON k.id = mk.keyword_id
WHERE k.keyword IN
  ('espionage', 'nuclear-bomb', 'ejector-seat')
GROUP BY mk.movie_id
HAVING COUNT(DISTINCT mk.keyword_id) = 3
ORDER BY NULL;
```

Indexes

```
CREATE INDEX k_i
ON keyword (keyword, id);

CREATE INDEX k_m
ON movie_keyword (keyword_id, movie_id);
```

EXPLAIN: the Group-by Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	k	range	k_i	NULL	3	Using where; Using index; Using temporary
1	SIMPLE	mk	ref	k_m	k.id	13884	Using index
1	SIMPLE	t	eq_ref	PRIMARY	mk.movie_id	1	

Status: the Group-by Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	710
Handler_read_last	0
Handler_read_next	708
Handler_read_prev	0
Handler_read_rnd	705
Handler_read_rnd_next	706
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	705

building and reading a temporary table

Profile: the Group-by Solution

Status	Duration
starting	0.000163
checking permissions	0.000004
checking permissions	0.000001
checking permissions	0.000009
Opening tables	0.000045
System lock	0.000017
init	0.000054
optimizing	0.000022
statistics	0.000113
preparing	0.000019
Creating tmp table	0.077638
executing	0.000008
Copying to tmp table	0.026402
Sorting result	0.000648
Sending data	0.002490
end	0.000020
removing tmp table	0.000009
end	0.000003
removing tmp table	0.000194
end	0.000005
query end	0.000006
closing tables	0.000038
freeing items	0.000017
removing tmp table	0.000007
freeing items	0.000005
removing tmp table	0.000003
freeing items	0.000185
logging slow query	0.000010
cleaning up	0.000019

building & tearing down temp table

Self-Join Solution

0.005s

```
SELECT t.title, CONCAT_WS(',', k1.keyword, k2.keyword,
k3.keyword) AS keywords
FROM title t
JOIN movie_keyword mk1 ON t.id = mk1.movie_id
JOIN keyword k1 ON k1.id = mk1.keyword_id
JOIN movie_keyword mk2 ON mk1.movie_id= mk2.movie_id
JOIN keyword k2 ON k2.id = mk2.keyword_id
JOIN movie_keyword mk3 ON mk1.movie_id = mk3.movie_id
JOIN keyword k3 ON k3.id = mk3.keyword_id
WHERE (k1.keyword, k2.keyword, k3.keyword)
   =  ('espionage', 'nuclear-bomb', 'ejector-seat');
```

EXPLAIN: the Self-Join Solution

id	select_type	table	type	key	ref	rows	Extra
1	SIMPLE	k1	ref	keyword	const	1	Using where; Using index
1	SIMPLE	k2	ref	keyword	const	1	Using where; Using index
1	SIMPLE	k3	ref	keyword	const	1	Using where; Using index
1	SIMPLE	mk1	ref	keyword_id	k1.id	13884	Using index
1	SIMPLE	t	eq_ref	PRIMARY	mk1.movie_id	1	
1	SIMPLE	mk2	ref	keyword_id	k2.id, t.id	13884	Using where; Using index
1	SIMPLE	mk3	ref	keyword_id	k3.id, t.id	13884	Using where; Using index

Status: the Self-Join Solution

Variable_name	Value
Handler_commit	1
Handler_delete	0
Handler_discover	0
Handler_prepare	0
Handler_read_first	0
Handler_read_key	1218
Handler_read_last	0
Handler_read_next	613
Handler_read_prev	0
Handler_read_rnd	0
Handler_read_rnd_next	0
Handler_rollback	0
Handler_savepoint	0
Handler_savepoint_rollback	0
Handler_update	0
Handler_write	0

minimal rows,
good index usage

Profile: the Self-Join Solution

Status	Duration
starting	0.000137
checking permissions	0.000004
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000007
Opening tables	0.000039
System lock	0.000016
init	0.000040
optimizing	0.000024
statistics	0.000172
preparing	0.000026
executing	0.000003
Sending data	0.004121
end	0.000007
query end	0.000003
closing tables	0.000037
freeing items	0.000161
logging slow query	0.000002
cleaning up	0.000005

Summary: Relational Division

Solution	Time	Notes
Group-by solution	0.100s	
Self-join solution	0.005s	

Query Patterns

CONCLUSIONS

Conclusions

- Use all tools to measure query performance
  - EXPLAIN
  - Session Status
  - Query Profiler
- Test with real-world data, because the best solution depends on the volume of data you're querying.
- Allocate enough memory to buffers so the indexes you need stay resident in RAM.

PERCONA TRAINING

Senior Industry Experts

In-Person and Online Classes

Custom Onsite Training

http://percona.com/training

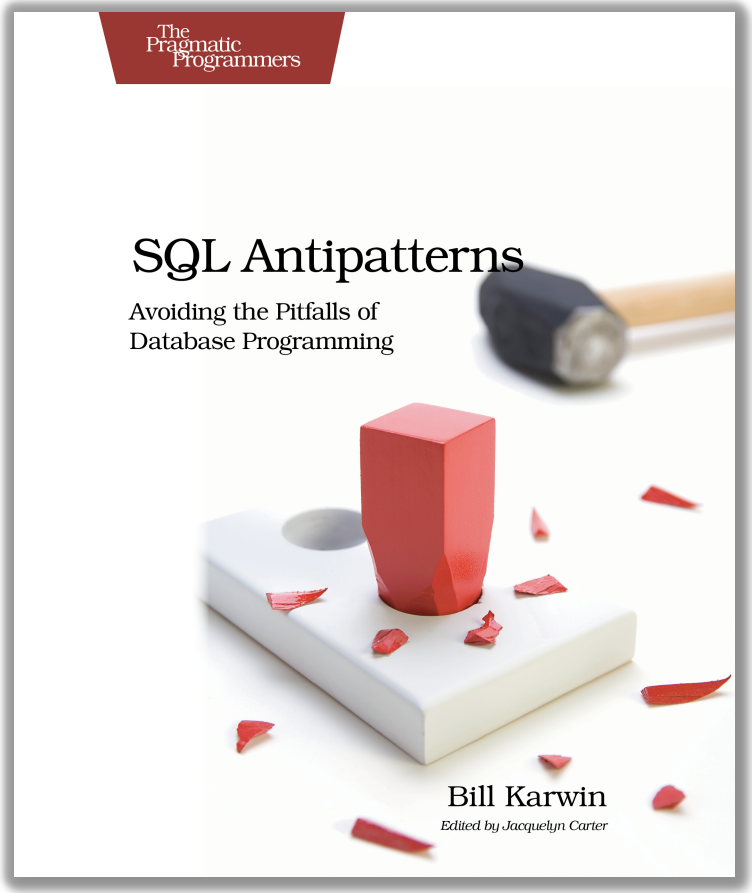
SQL Antipatterns

Avoiding the Pitfalls of Database Programming

by Bill Karwin

Available in print, epub, mobi, pdf.
Delivery options for Kindle or Dropbox.

http://pragprog.com/book/bksqla



Percona Live

MySQL Conference & Expo

Where the MySQL world comes together

Santa Clara Convention Center
April 22 – 25, 2013
http://www.percona.com/live/mysql-conference-2013/

"How to Design Indexes, Really"
Bill Karwin

"Extensible Data Modeling in MySQL"
Bill Karwin