Performance Enhancements In PostgreSQL 8.4

PGDay.EU 2009
Paris, France

Magnus Hagander
Redpill Linpro AB
PostgreSQL 8.4

• Released July 2009
  – 8.4.1 released September 2009
• Major upgrade from 8.3
• New features and enhancements of existing ones
Using PostgreSQL performance

- “ORM-like queries” only get you so far
- Application specific optimizations
- Don't be afraid to let the database work!
Performance enhancements

- Some are application transparent
  - Possibly even DBA transparent
- Some require application changes
Let's get started

- Query execution optimizations
Anti-joins and Semi-joins

- Formalized JOIN methods for inequality joins
- Better performance for EXISTS / NOT EXISTS
Anti-joins and Semi-joins

8.3

```sql
pagila=# EXPLAIN SELECT * FROM actor a WHERE NOT EXISTS
         (SELECT * FROM film_actor fa WHERE fa.actor_id=a.actor_id);

Seq Scan on actor  (cost=0.00..288.99 rows=100 width=25)
  Filter: (NOT (subplan))
  SubPlan
    -> Index Scan using film_actor_pkey on film_actor
       (cost=0.00..38.47 rows=27 width=12)
       Index Cond: (actor_id = $0)
```

```sql
Seq Scan on actor  (cost=0.00..288.99 rows=100 width=25)
  Filter: (NOT (subplan))
  SubPlan
    -> Index Scan using film_actor_pkey on film_actor
       (cost=0.00..38.47 rows=27 width=12)
       Index Cond: (actor_id = $0)
```
Anti-joins and Semi-joins

8.3

```bash
pagila=# EXPLAIN SELECT * FROM actor a WHERE NOT EXISTS (SELECT * FROM film_actor fa WHERE fa.actor_id=a.actor_id);
```

**Nested Loop Anti Join**  (cost=0.00..30.57 rows=1 width=25)
- Seq Scan on actor  (cost=0.00..4.00 rows=200 width=25)
- Index Scan using film_actor_pkey on film_actor  
  (cost=0.00..1.54 rows=27 width=2)
  Index Cond: (film_actor.actor_id = actor.actor_id)
Anti-joins and Semi-joins

8.3

```
pagila=# EXPLAIN SELECT * FROM actor a WHERE EXISTS
   (SELECT * FROM film_actor fa WHERE fa.actor_id=a.actor_id);

Nested Loop Semi Join  (cost=0.00..30.57 rows=200 width=25)
   ->  Seq Scan on actor  (cost=0.00..4.00 rows=200 width=25)
   ->  Index Scan using film_actor_pkey on film_actor
       (cost=0.00..1.54 rows=27 width=2)
       Index Cond: (film_actor.actor_id = actor.actor_id)
```
Hash for DISTINCT/UNION

- Previously, always a sort+unique
- *No longer guaranteed sorted!*
  - Add ORDER BY
  - Both plans will be considered
- Also affects EXCEPT & INTERSECT
Hash improvements

• Faster algorithms
  – WARNING! New hash values!

• Also faster hash indexes
  – Still not WAL-logged

• And optimizations of HASH joins
  – Particularly around large joins
Moving on

- DBA optimizations
Function level statistics

- `pg_stat_user_functions`
- Controlled by "track_functions"
  - `none, pl or all`
- Tracks calls, time, and internal time
postgres=# select * from pg_stat_user_functions ;

<table>
<thead>
<tr>
<th>funcid</th>
<th>101414</th>
</tr>
</thead>
<tbody>
<tr>
<td>schemaname</td>
<td>public</td>
</tr>
<tr>
<td>funcname</td>
<td>foo</td>
</tr>
<tr>
<td>calls</td>
<td>1003</td>
</tr>
<tr>
<td>total_time</td>
<td>6</td>
</tr>
<tr>
<td>self_time</td>
<td>6</td>
</tr>
</tbody>
</table>
Free Space Map (FSM)

- Stores list of free blocks in relations
  - Caused by DELETE and UPDATE
- Used by INSERT & UPDATE
New Free Space Map (FSM)

- No more max_fsm_pages!
- Dynamically tuned
- Uses normal buffer cache
New Free Space Map (FSM)

- No global lock
- Not lost on crash
New Free Space Map (FSM)

- No global lock
- Not lost on crash

- VACUUM is still needed, of course...
Visibility Map

• Tracks pages that are “visible to all transactions” in bitmap

• Set by VACUUM

• Cleared by INSERT/UPDATE/DELETE
Partial VACUUM

• “Visible to all” pages skipped by VACUUM
• Only heap tables, not indexes
• Still requires freezing
VACUUM snapshot tracking

- Snapshot tracking for idle sessions
- Makes VACUUM clean up better with long running transactions
- `<IDLE>` In Transaction
Stats temp file improvements

- Previously, unconditionally written twice/sec in data dir
- Now, written only on demand
- And in configurable location (tmpfs!)
Parallel pg_restore

- Restore from dump was single threaded
- Can now load in \(<n>\) sessions
- At least one table per session
- No single-transaction!
int8 pass by value

- 64-bit integers finally take advantage of 64-bit CPUs
Moving on

• Application features
Subselects in LIMIT/OFFSET

- Previously, only constants allowed
- Required two queries / roundtrips
  - Or cursor in function

\textbf{SELECT} * \textbf{FROM} ... \textbf{LIMIT} ( \\
  \textbf{SELECT} something \textbf{FROM} other \\
)
WINDOW aggregates

- Perform aggregates over parts of data
- Avoid requiring multiple queries
- Avoid multiple scans
SELECT  name, department, salary,
       rank() OVER ( PARTITION BY department
                      ORDER BY salary DESC
       )
FROM    employees
<table>
<thead>
<tr>
<th>name</th>
<th>department</th>
<th>salary</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Berra</td>
<td>Ekonomi</td>
<td>29400</td>
<td>1</td>
</tr>
<tr>
<td>Åke</td>
<td>Ekonomi</td>
<td>29400</td>
<td>1</td>
</tr>
<tr>
<td>Sune</td>
<td>Ekonomi</td>
<td>24000</td>
<td>3</td>
</tr>
<tr>
<td>Arne</td>
<td>IT</td>
<td>24000</td>
<td>1</td>
</tr>
<tr>
<td>Pelle</td>
<td>IT</td>
<td>22000</td>
<td>2</td>
</tr>
<tr>
<td>Kalle</td>
<td>IT</td>
<td>18000</td>
<td>3</td>
</tr>
</tbody>
</table>

(6 rows)
SELECT name, department, salary,
   rank() OVER (PARTITION BY department
                  ORDER BY salary DESC)
  ,
   rank() OVER (ORDER BY salary DESC)
FROM employees
<table>
<thead>
<tr>
<th>name</th>
<th>department</th>
<th>salary</th>
<th>rank</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Åke</td>
<td>Ekonomi</td>
<td>29400</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Berra</td>
<td>Ekonomi</td>
<td>29400</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Sune</td>
<td>Ekonomi</td>
<td>24000</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Arne</td>
<td>IT</td>
<td>24000</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Pelle</td>
<td>IT</td>
<td>22000</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Kalle</td>
<td>IT</td>
<td>18000</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

(6 rows)
Common Table Expressions

- WITH RECURSIVE
- Traverse trees and graphs in SQL
- .. avoid multiple queries
  - (also makes your life easier)
WITH RECURSIVE t(id, department, name, manager) AS (  
    SELECT id, department, name, manager  
    FROM emp WHERE name='Kalle'  
    UNION ALL  
    SELECT emp.id, emp.department, emp.name, emp.manager  
    FROM emp JOIN t ON t.manager=emp.id
    
)  
SELECT * FROM t;
<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>name</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IT</td>
<td>Kalle</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>IT</td>
<td>Arne</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Ekonomi</td>
<td>Berra</td>
<td></td>
</tr>
</tbody>
</table>

(3 rows)
<table>
<thead>
<tr>
<th>id</th>
<th>department</th>
<th>name</th>
<th>manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IT</td>
<td>Kalle</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>IT</td>
<td>Arne</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Ekonomi</td>
<td>Berra</td>
<td></td>
</tr>
</tbody>
</table>

(3 rows)

Very important!
Lots of more improvements!

• But that's it for now..
• Go download and test!
Performance Enhancements in PostgreSQL 8.4

http://2009.pgday.eu/feedback
Questions?

magnus@hagander.net
Twitter: @magnushagander
http://blog.hagander.net