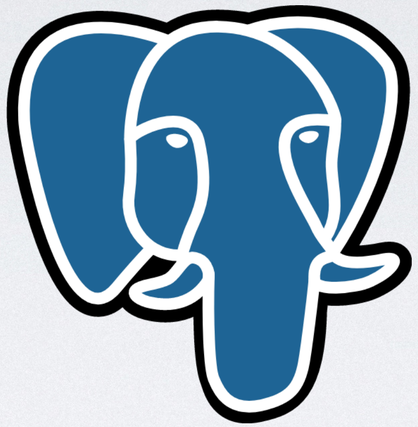
Mark Haylock

https://github.com/mhaylock



Faster Pagination with Postgres

(and other SQL servers too!)

Uses of Pagination

Previous / Next

+ Jump to page

Infinite scrolling

Batched iteration through records

Common pagination libraries in Ruby

will_paginate
https://github.com/ddnexus/pagy

kaminari
https://github.com/kaminari/kaminari

pagy
https://github.com/ddnexus/pagy

Common features:

- Paging using OFFSET
- Counting all matching records to calculate a total page count
- Rendering page navigation frontends

Two things that scale terribly in Postgres

Paging using OFFSET

&

COUNT

Apply caution when using these!

OFFSET Pagination

```sql
SELECT *
FROM movies
ORDER BY start_year DESC NULLS LAST, id ASC
LIMIT 500
OFFSET 50000; -- Page 101
```

EXPLAIN ANALYZE

```
EXPLAIN (ANALYZE, TIMING OFF) SELECT *
FROM movies
…
```

```
Limit  (cost=57009.16..57067.49 rows=500 width=122) (actual rows=500 loops=1)
  ->  Gather Merge  (cost=51175.41..106908.17 rows=477676 width=122) (actual rows=50500 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Sort  (cost=50175.39..50772.49 rows=238838 width=122) (actual rows=17032 loops=3)
              Sort Key: start_year DESC NULLS LAST, id
              Sort Method: external merge  Disk: 27768kB
              Worker 0:  Sort Method: external merge  Disk: 24472kB
              Worker 1:  Sort Method: external merge  Disk: 24680kB
              ->  Parallel Seq Scan on movies  (cost=0.00..13328.38 rows=238838 width=122) (actual
rows=191071 loops=3)
Planning Time: 0.092 ms
Execution Time: 212.477 ms
```

Adding an Index

```
CREATE INDEX index_movies_on_start_year_and_id
  ON movies(start_year DESC NULLS LAST, id);
```

```
Limit  (cost=4685.48..4732.33 rows=500 width=122) (actual rows=500 loops=1)
  ->  Index Scan using index_movies_on_start_year_and_id on movies  (cost=0.42..53711.05 rows=573212
width=122) (actual rows=50500 loops=1)
Planning Time: 0.196 ms
Execution Time: 16.226 ms
```

Seek Method / Keyset Pagination

What if we could help Postgres jump straight to the record after the last record on the previous page?

```
#<Movie:0x00007fb08b205128
 id: 480934,
 imdb_title_id: "tt5657280",
 primary_title: "Viking Destiny",
 original_title: "Viking Destiny",
 start_year: 2018,
 end_year: nil,
 runtime_minutes: 91,
 genres: ["Action", "Adventure", "Fantasy"],
 created_at: Wed, 14 Jul 2021 10:15:46.191499000 UTC +00:00,
 updated_at: Wed, 14 Jul 2021 10:15:46.191499000 UTC +00:00>
```

```sql
SELECT *
FROM movies
WHERE start_year < 2018
ORDER BY start_year DESC NULLS LAST, id ASC
LIMIT 500;
```

Seek Method / Keyset Pagination

What if we could help Postgres jump straight to the record after the last record on the previous page?

```
#<Movie:0x00007fb08b205128
 id: 480934,
 imdb_title_id: "tt5657280",
 primary_title: "Viking Destiny",
 original_title: "Viking Destiny",
 start_year: 2018,
 end_year: nil,
 runtime_minutes: 91,
 genres: ["Action", "Adventure", "Fantasy"],
 created_at: Wed, 14 Jul 2021 10:15:46.191499000 UTC +00:00,
 updated_at: Wed, 14 Jul 2021 10:15:46.191499000 UTC +00:00>
```

```
SELECT *
FROM movies
WHERE start_year <= 2018
  AND NOT (start_year = 2018 AND id <= 480934)
ORDER BY start_year DESC NULLS LAST, id ASC
LIMIT 500;
```

EXPLAIN for Seek Method

```
Limit  (cost=0.42..60.14 rows=500 width=122) (actual rows=500 loops=1)
  ->  Index Scan using index_movies_on_start_year_and_id on movies  (cost=0.42..52868.83 rows=442690
width=122) (actual rows=500 loops=1)
        Index Cond: (start_year <= 2018)
        Filter: ((start_year <> 2018) OR (id < 480934))
Planning Time: 0.122 ms
Execution Time: 0.295 ms
```

vs the previous EXPLAIN:

```
Limit  (cost=4685.48..4732.33 rows=500 width=122) (actual rows=500 loops=1)
  ->  Index Scan using index_movies_on_start_year_and_id on movies  (cost=0.42..53711.05 rows=573212
width=122) (actual rows=50500 loops=1)
Planning Time: 0.196 ms
Execution Time: 16.226 ms
```

If column sort order is consistent

Use "Row Values" comparison syntax:

```sql
SELECT *
FROM movies
WHERE (primary_title, id) > ('Viking Destiny', 480934)
ORDER BY primary_title ASC, id ASC
LIMIT 500;
```

```
Limit  (cost=0.42..784.47 rows=500 width=122) (actual rows=500 loops=1)
  ->  Index Scan using index_movies_on_primary_title_and_id on movies  (cost=0.42..44927.73 rows=28651
width=122) (actual rows=500 loops=1)
        Index Cond: (ROW((primary_title)::text, id) > ROW('Viking Destiny'::text, 480934))
Planning Time: 0.111 ms
Execution Time: 0.567 ms
```

Going backwards

Means you have to reverse everything,
including the results!

```sql
SELECT *
FROM movies
WHERE (primary_title, id) > ('Viking Destiny', 480934)
ORDER BY primary_title ASC, id ASC
LIMIT 500;
```

```sql
SELECT *
FROM movies
WHERE (primary_title, id) < ('Viking Destiny', 480934)
ORDER BY primary_title DESC, id DESC
LIMIT 500;
```

```sql
SELECT *
FROM movies
WHERE start_year <= 2018
  AND NOT (start_year = 2018 AND id <= 480934)
ORDER BY start_year DESC NULLS LAST, id ASC
LIMIT 500;
```

```sql
SELECT *
FROM movies
WHERE start_year >= 2018
  AND NOT (start_year = 2018 AND id >= 480934)
ORDER BY start_year ASC NULLS FIRST, id DESC
LIMIT 500;
```

DEMO TIME

Pros & Cons of Seek/Keyset Pagination

Pros:

- Fast
- No page drift as rows are inserted

Cons:

- Every sort order needs an index
- WHERE clause can be complicated
- Going backwards is confusing
- Can't identify current page
- Can't jump to a particular page

Ruby Library

https://github.com/glebm/order_query

References

Pros:

- ***Five ways to paginate in Postgres, from the basic to the exotic***
  https://www.citusdata.com/blog/2016/03/30/five-ways-to-paginate/
- ***Pagination done the Right way***
  https://www.slideshare.net/MarkusWinand/p2d2-pagination-done-the-postgresql-way
- ***Paging Through Results***
  https://use-the-index-luke.com/sql/partial-results/fetch-next-page