5 solutions to the preload top N problem

Mike Buhot
Elixir Australia
2020-06-17

EXPERT360

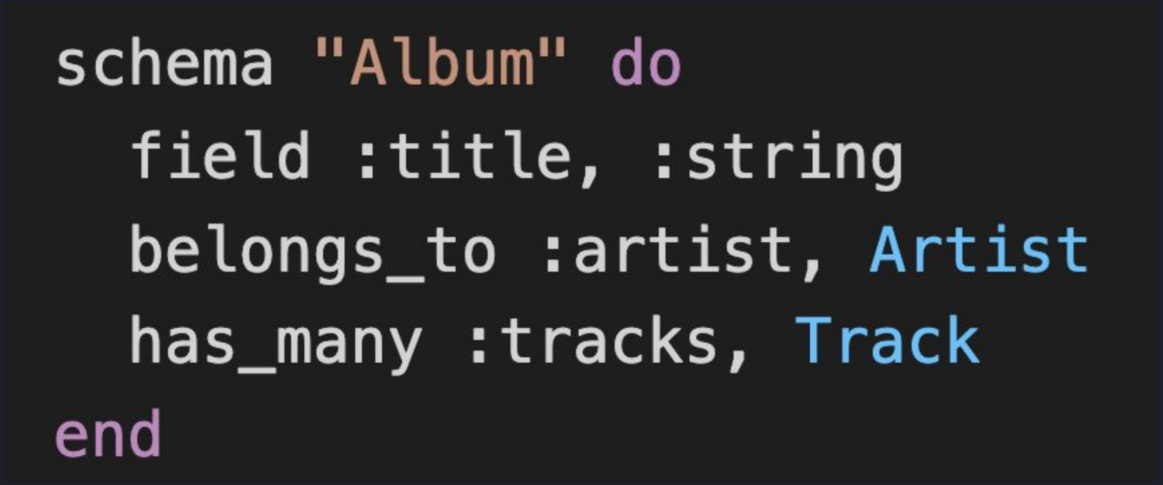
Overview

- Ecto Associations & Preloads
- The Preload Top N Problem
- Customising preloads
- Using Joins
- Using Queries
- Using Functions
- Modern SQL in Ecto

Ecto Associations & Preloads

Associations

- Define relationships between schemas
- has_one, has_many, belongs_to, many_to_many



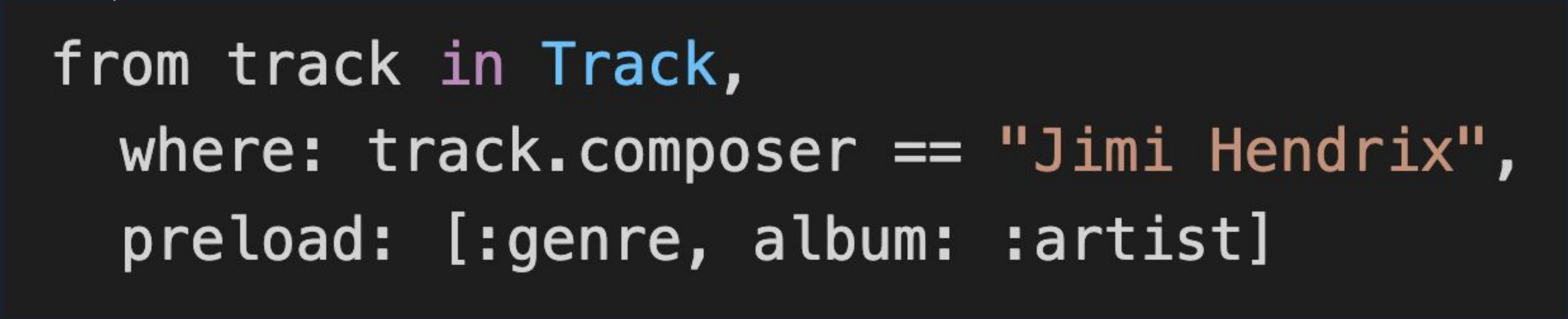
```
schema "Album" do
  field :title, :string
  belongs_to :artist, Artist
  has_many :tracks, Track
end
```

Associations - Joins

```
from album in Album,
  join: track in assoc(album, :tracks),
  join: genre in assoc(track, :genre),
  select: %{album: album.title, track: track.name, genre: genre.name}
```

- Associations make joins easy! 👍

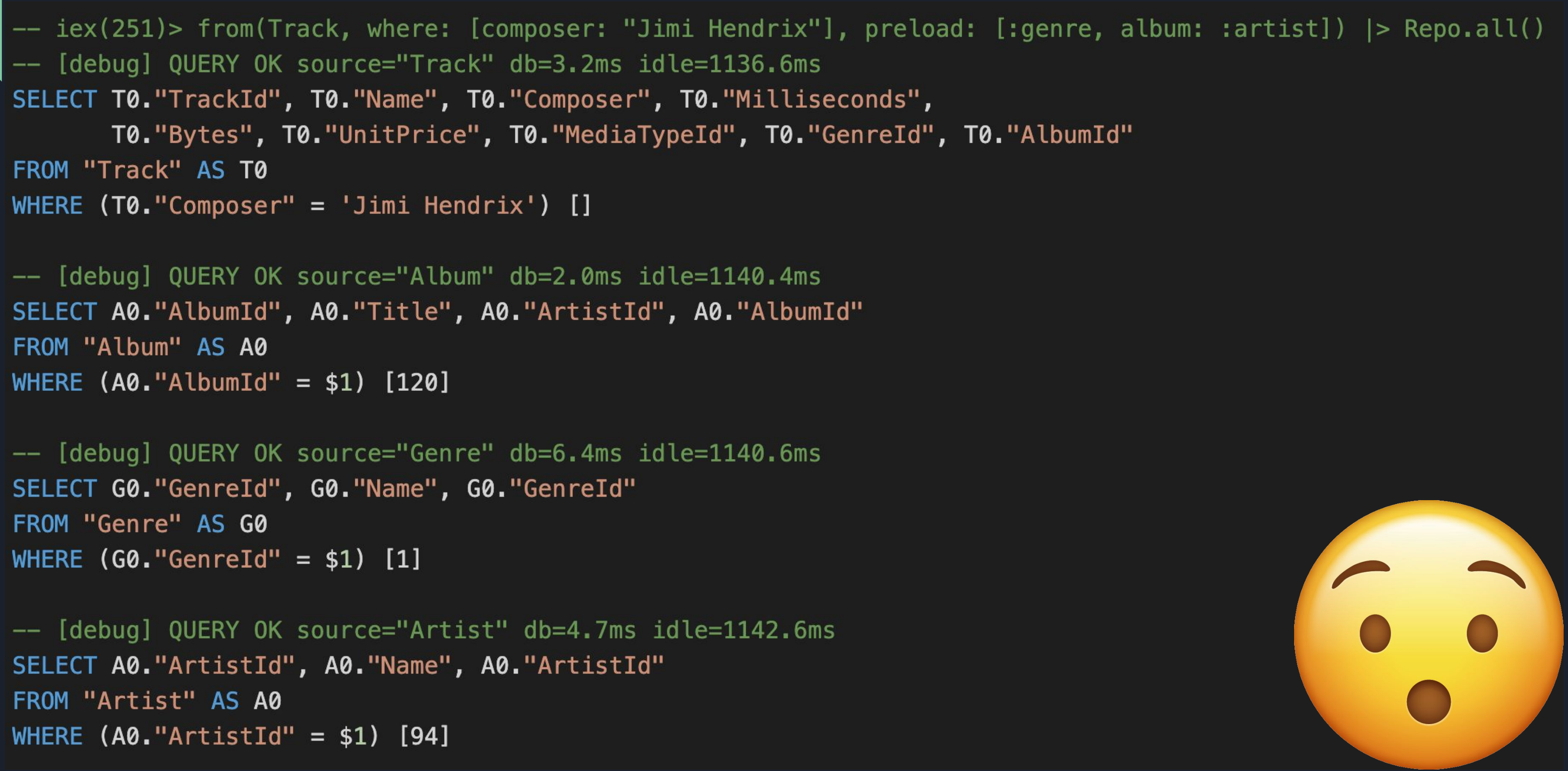
Associations - Preloads



```
from track in Track,
  where: track.composer == "Jimi Hendrix",
  preload: [:genre, album: :artist]
```

- Load related records into struct fields 👍

Associations - Preloads



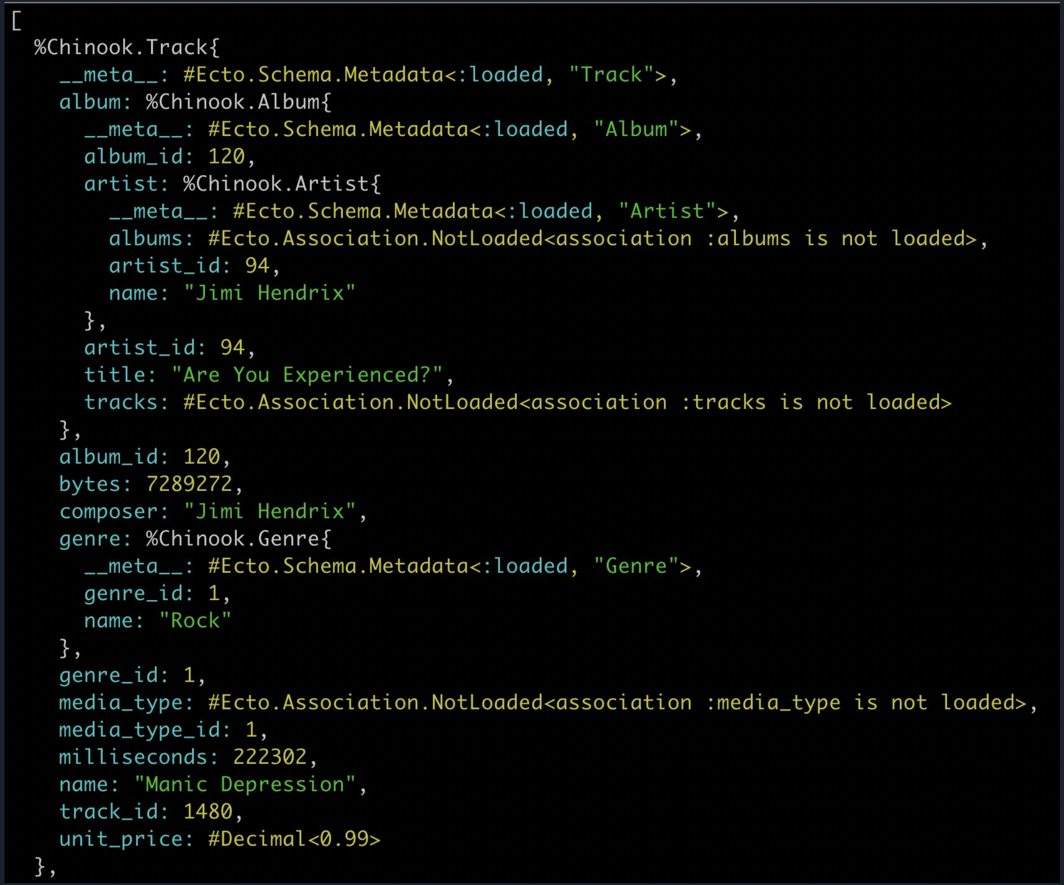
```
-- iex(251)> from(Track, where: [composer: "Jimi Hendrix"], preload: [:genre, album: :artist]) |> Repo.all()
-- [debug] QUERY OK source="Track" db=3.2ms idle=1136.6ms
SELECT T0."TrackId", T0."Name", T0."Composer", T0."Milliseconds",
       T0."Bytes", T0."UnitPrice", T0."MediaTypeId", T0."GenreId", T0."AlbumId"
FROM "Track" AS T0
WHERE (T0."Composer" = 'Jimi Hendrix') []

-- [debug] QUERY OK source="Album" db=2.0ms idle=1140.4ms
SELECT A0."AlbumId", A0."Title", A0."ArtistId", A0."AlbumId"
FROM "Album" AS A0
WHERE (A0."AlbumId" = $1) [120]

-- [debug] QUERY OK source="Genre" db=6.4ms idle=1140.6ms
SELECT G0."GenreId", G0."Name", G0."GenreId"
FROM "Genre" AS G0
WHERE (G0."GenreId" = $1) [1]

-- [debug] QUERY OK source="Artist" db=4.7ms idle=1142.6ms
SELECT A0."ArtistId", A0."Name", A0."ArtistId"
FROM "Artist" AS A0
WHERE (A0."ArtistId" = $1) [94]
```

Associations - Preloads



```
[
  %Chinook.Track{
    __meta__: #Ecto.Schema.Metadata<:loaded, "Track">,
    album: %Chinook.Album{
      __meta__: #Ecto.Schema.Metadata<:loaded, "Album">,
      album_id: 120,
      artist: %Chinook.Artist{
        __meta__: #Ecto.Schema.Metadata<:loaded, "Artist">,
        albums: #Ecto.Association.NotLoaded<association :albums is not loaded>,
        artist_id: 94,
        name: "Jimi Hendrix"
      },
      artist_id: 94,
      title: "Are You Experienced?",
      tracks: #Ecto.Association.NotLoaded<association :tracks is not loaded>
    },
    album_id: 120,
    bytes: 7289272,
    composer: "Jimi Hendrix",
    genre: %Chinook.Genre{
      __meta__: #Ecto.Schema.Metadata<:loaded, "Genre">,
      genre_id: 1,
      name: "Rock"
    },
    genre_id: 1,
    media_type: #Ecto.Association.NotLoaded<association :media_type is not loaded>,
    media_type_id: 1,
    milliseconds: 222302,
    name: "Manic Depression",
    track_id: 1480,
    unit_price: #Decimal<0.99>
  },
```

Preload Top N

Preload Top N

Load the top 10 Artists, their first album, and the first 3 tracks for those albums.

Without making 10+ queries

Customising Preloads

Customising Preloads

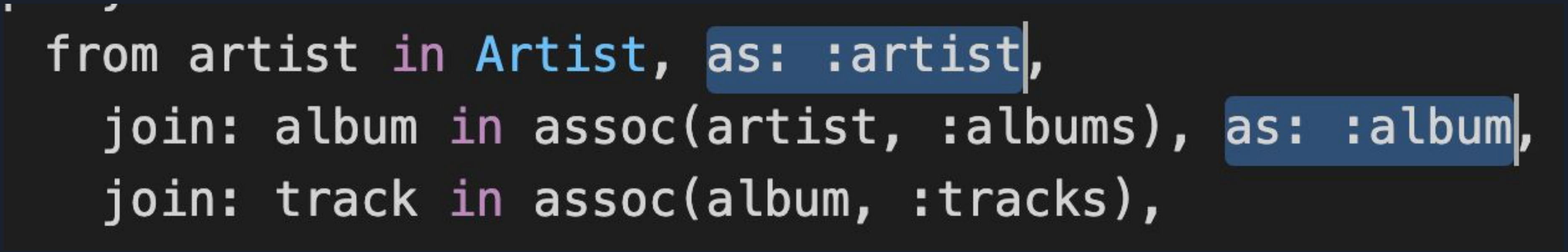
3 tools for customisation

- Joins
- Queries
- Functions

Solution #1 - joins & where-in subqueries

```elixir
from artist in Artist, as: :artist,
  join: album in assoc(artist, :albums), as: :album,
  join: track in assoc(album, :tracks),
  where: artist.artist_id in subquery(
    from a in Artist,
      order_by: a.artist_id,
      limit: 10,
      select: a.artist_id
  ),
  where: album.album_id in subquery(
    from a in Album,
      where: a.artist_id == parent_as(:artist).artist_id,
      order_by: :title,
      limit: 1,
      select: a.album_id
  ),
  where: track.track_id in subquery(
    from t in Track,
    where: t.album_id == parent_as(:album).album_id,
    order_by: :name,
    limit: 3,
    select: t.track_id
  ),
  order_by: [artist.artist_id, album.album_id, track.track_id],
  select: artist,
  preload: [albums: {album, tracks: track}]
```

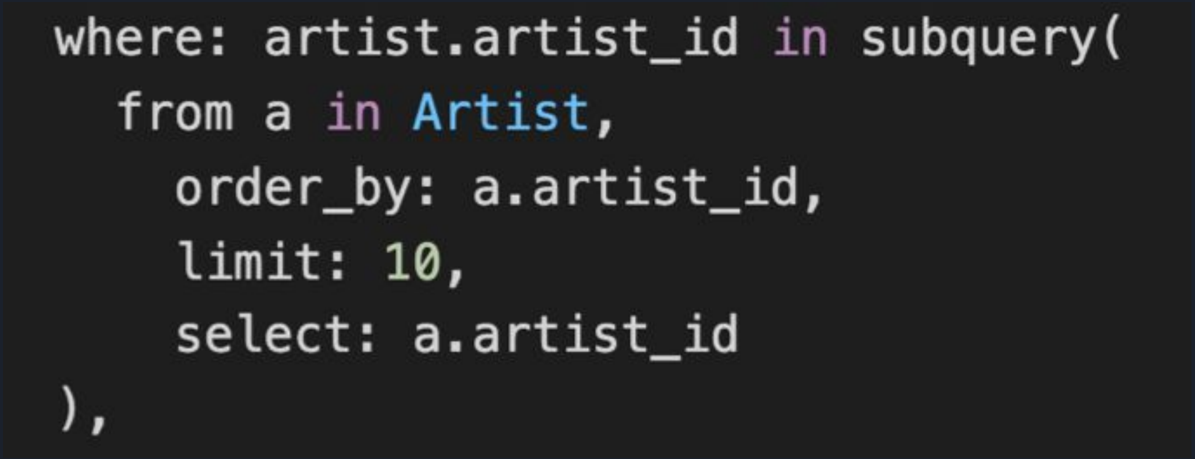
Named Bindings



```
from artist in Artist, as: :artist,
  join: album in assoc(artist, :albums), as: :album,
  join: track in assoc(album, :tracks),
```

- Introduced in Ecto 3.0
- Better query composition for joins & subqueries

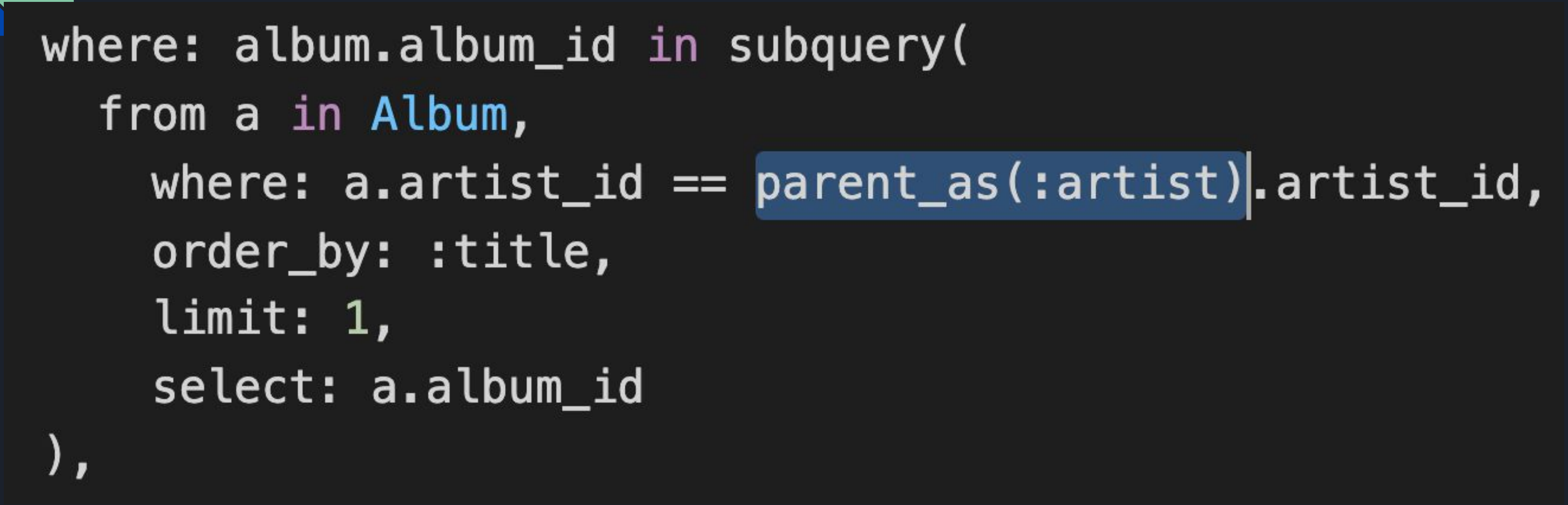
Where: in Subquery



```
where: artist.artist_id in subquery(
  from a in Artist,
    order_by: a.artist_id,
    limit: 10,
    select: a.artist_id
),
```

- Introduced in Ecto 3.4.3 🔥
- Find the IDs of the first 10 Artist
- Filter the outer query to only rows matching these IDs

Correlated Subquery



```
where: album.album_id in subquery(
  from a in Album,
    where: a.artist_id == parent_as(:artist).artist_id,
    order_by: :title,
    limit: 1,
    select: a.album_id
),
```

- Introduced in Ecto 3.4.3 🔥
- Allows subquery to refer to parent query named binding
- Filters albums to first 1 by title for artist

```sql
-- [debug] QUERY OK source="Artist" db=8.1ms idle=1791.9ms
SELECT A0."ArtistId", A0."Name",
       A1."AlbumId", A1."Title", A1."ArtistId",
       T2."TrackId", T2."Name", T2."Composer", T2."Milliseconds",
       T2."Bytes", T2."UnitPrice", T2."MediaTypeId", T2."GenreId", T2."AlbumId"
FROM "Artist" AS A0
INNER JOIN "Album" AS A1 ON A1."ArtistId" = A0."ArtistId"
INNER JOIN "Track" AS T2 ON T2."AlbumId" = A1."AlbumId"
WHERE (A0."ArtistId" IN (
    SELECT sA0."ArtistId" AS "artist_id"
    FROM "Artist" AS sA0
    ORDER BY sA0."ArtistId"
    LIMIT 10)
) AND (A1."AlbumId" IN (
    SELECT sA0."AlbumId" AS "album_id"
    FROM "Album" AS sA0
    WHERE (sA0."ArtistId" = A0."ArtistId")
    ORDER BY sA0."Title"
    LIMIT 1)
) AND (T2."TrackId" IN (
    SELECT sT0."TrackId" AS "track_id"
    FROM "Track" AS sT0
    WHERE (sT0."AlbumId" = A1."AlbumId")
    ORDER BY sT0."Name"
    LIMIT 3)
)
ORDER BY A0."ArtistId", A1."AlbumId", T2."TrackId"
```

Solution #2 - lateral joins

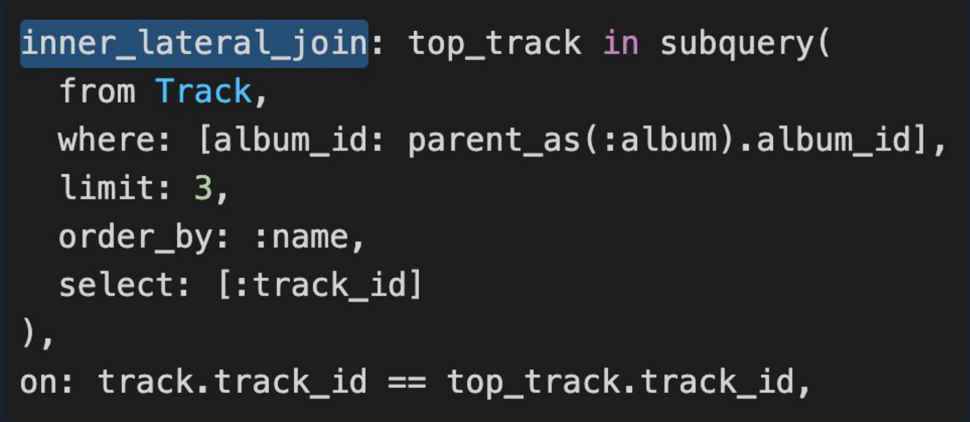
#2 - lateral joins?

Subqueries appearing in FROM can be preceded by the key word LATERAL. This allows them to reference columns provided by preceding FROM items. (Without LATERAL, each subquery is evaluated independently and so cannot cross-reference any other FROM item.)

https://www.postgresql.org/docs/current/queries-table-expressions.html#QUERIES-FROM

```elixir
from(artist in Artist, as: :artist,
  join: album in assoc(artist, :albums), as: :album,
  join: track in assoc(album, :tracks),
  join: top_artist in subquery(
    from Artist,
      order_by: :artist_id,
      limit: 10,
      select: [:artist_id]
  ),
  on: artist.artist_id == top_artist.artist_id,
  inner_lateral_join: top_album in subquery(
    from Album,
    where: [artist_id: parent_as(:artist).artist_id],
    limit: 1,
    order_by: :title,
    select: [:album_id]
  ),
  on: album.album_id == top_album.album_id,
  inner_lateral_join: top_track in subquery(
    from Track,
    where: [album_id: parent_as(:album).album_id],
    limit: 3,
    order_by: :name,
    select: [:track_id]
  ),
  on: track.track_id == top_track.track_id,
  order_by: [artist.artist_id, album.album_id, track.track_id],
  select: artist,
  preload: [albums: {album, tracks: track}]
)
```

#2 - inner_lateral_join



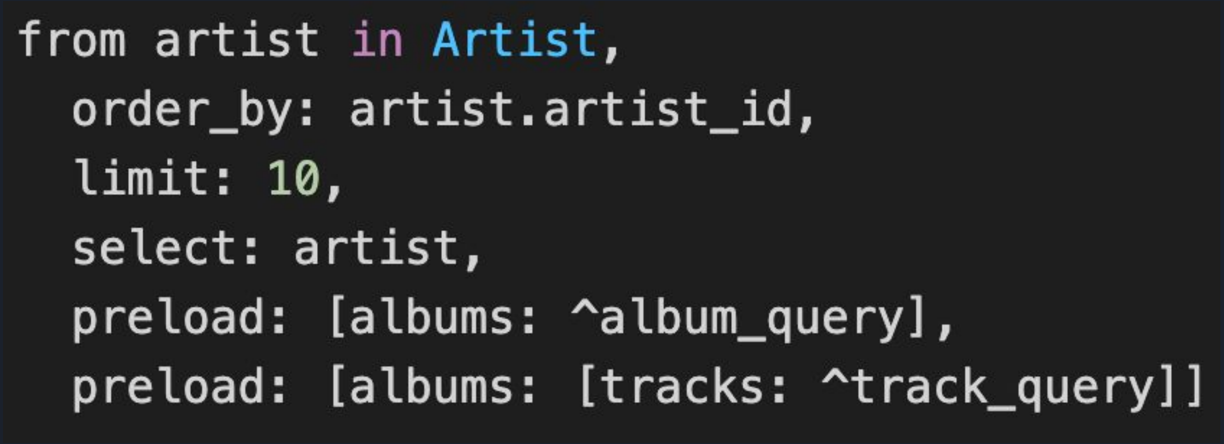
```
inner_lateral_join: top_track in subquery(
  from Track,
  where: [album_id: parent_as(:album).album_id],
  limit: 3,
  order_by: :name,
  select: [:track_id]
),
on: track.track_id == top_track.track_id,
```

- Available since Ecto 2.0
- Originally required using fragment with raw SQL 🤮
- Now works with subqueries! 💖

```
-- [debug] QUERY OK source="Artist" db=3.3ms idle=880.8ms
SELECT A0."ArtistId", A0."Name",
       A1."AlbumId", A1."Title", A1."ArtistId",
       T2."TrackId", T2."Name", T2."Composer", T2."Milliseconds",
       T2."Bytes", T2."UnitPrice", T2."MediaTypeId", T2."GenreId", T2."AlbumId"
FROM "Artist" AS A0
INNER JOIN "Album" AS A1 ON A1."ArtistId" = A0."ArtistId"
INNER JOIN "Track" AS T2 ON T2."AlbumId" = A1."AlbumId"
INNER JOIN (
    SELECT sA0."ArtistId" AS "artist_id"
    FROM "Artist" AS sA0
    ORDER BY sA0."ArtistId"
    LIMIT 10) AS s3 ON A0."ArtistId" = s3."artist_id"
INNER JOIN LATERAL (
    SELECT sA0."AlbumId" AS "album_id"
    FROM "Album" AS sA0
    WHERE (sA0."ArtistId" = A0."ArtistId")
    ORDER BY sA0."Title"
    LIMIT 1) AS s4 ON A1."AlbumId" = s4."album_id"
INNER JOIN LATERAL (
    SELECT sT0."TrackId" AS "track_id"
    FROM "Track" AS sT0
    WHERE (sT0."AlbumId" = A1."AlbumId")
    ORDER BY sT0."Name"
    LIMIT 3) AS s5 ON T2."TrackId" = s5."track_id"
ORDER BY A0."ArtistId", A1."AlbumId", T2."TrackId"
```

Preload Queries

Preload Queries



```
from artist in Artist,
  order_by: artist.artist_id,
  limit: 10,
  select: artist,
  preload: [albums: ^album_query],
  preload: [albums: [tracks: ^track_query]]
```

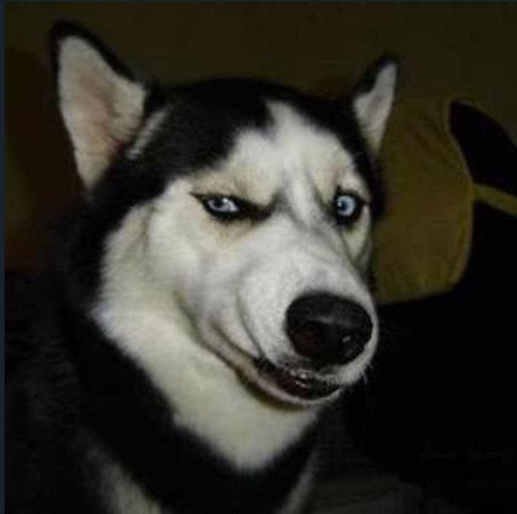
- Ecto issues separate queries to preload data
- Customised query provided to limit rows
- Keeps main query tidy 👍

Solution #3 - Preload Queries with Window Functions

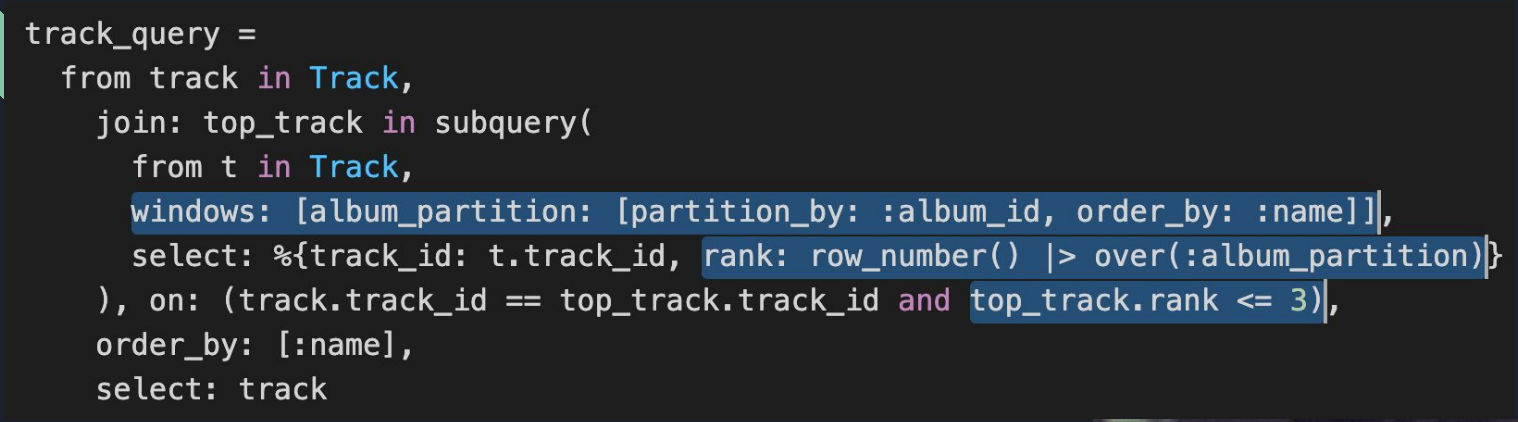
#3 - window functions?

A window function performs a calculation across a set of table rows that are somehow related to the current row. This is comparable to the type of calculation that can be done with an aggregate function. However, window functions do not cause rows to become grouped into a single output row like non-window aggregate calls would. Instead, the rows retain their separate identities. Behind the scenes, the window function is able to access more than just the current row of the query result.

https://www.postgresql.org/docs/current/tutorial-window.html

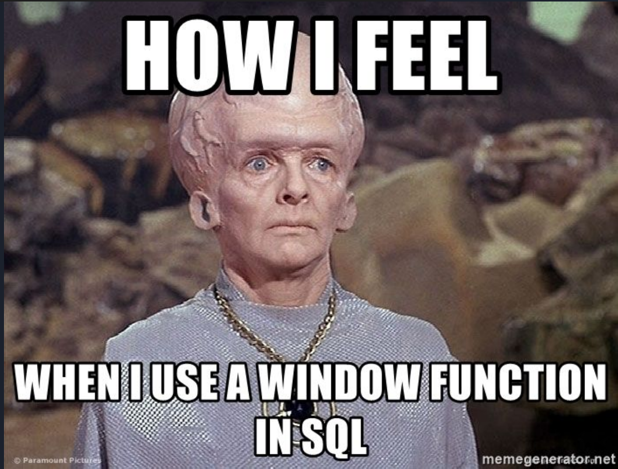


#3 - Window Functions



```
track_query =
  from track in Track,
    join: top_track in subquery(
      from t in Track,
      windows: [album_partition: [partition_by: :album_id, order_by: :name]],
      select: %{track_id: t.track_id, rank: row_number() |> over(:album_partition)}
    ), on: (track.track_id == top_track.track_id and top_track.rank <= 3),
    order_by: [:name],
    select: track
```

- Added in Ecto 3.0
- Subquery ranks tracks within each album partition
- Outer query uses rank in join condition to limit rows



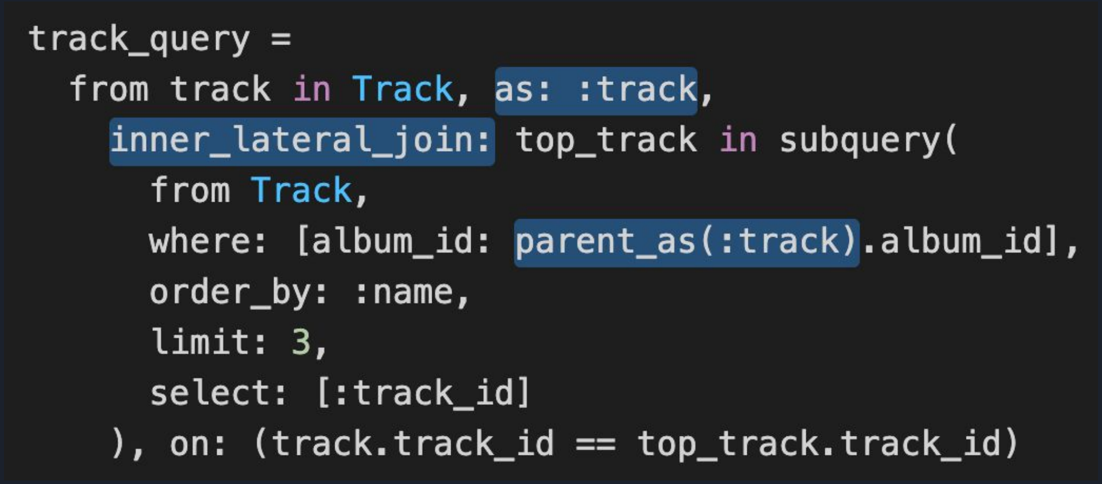
```
-- [debug] QUERY OK source="Artist" db=3.9ms idle=289.3ms
SELECT A0."ArtistId", A0."Name"
FROM "Artist" AS A0
ORDER BY A0."ArtistId"
LIMIT 10;

-- [debug] QUERY OK source="Album" db=6.1ms idle=293.6ms
SELECT A0."AlbumId", A0."Title", A0."ArtistId", A0."ArtistId"
FROM "Album" AS A0 INNER JOIN (
    SELECT sA0."AlbumId" AS "album_id", row_number() OVER "artist_partition" AS "rank"
    FROM "Album" AS sA0
    WINDOW "artist_partition" AS (PARTITION BY sA0."ArtistId" ORDER BY sA0."Title")
    ) AS s1 ON (A0."AlbumId" = s1."album_id") AND (s1."rank" = 1)
WHERE (A0."ArtistId" = ANY($1))
ORDER BY A0."ArtistId", A0."Title" [[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]];

-- [debug] QUERY OK source="Track" db=8.4ms idle=300.0ms
SELECT T0."TrackId", T0."Name", T0."Composer", T0."Milliseconds",
       T0."Bytes", T0."UnitPrice", T0."MediaTypeId", T0."GenreId", T0."AlbumId", T0."AlbumId"
FROM "Track" AS T0 INNER JOIN (
    SELECT sT0."TrackId" AS "track_id", row_number() OVER "album_partition" AS "rank"
    FROM "Track" AS sT0
    WINDOW "album_partition" AS (PARTITION BY sT0."AlbumId" ORDER BY sT0."Name")
    ) AS s1 ON (T0."TrackId" = s1."track_id") AND (s1."rank" <= 3)
WHERE (T0."AlbumId" = ANY($1))
ORDER BY T0."AlbumId", T0."Name" [[1, 2, 5, 6, 7, 34, 9, 10, 12, 13]]
```

Solution #4 - Preload Queries with lateral joins

#4 - Preload Query w’ lateral join



```
track_query =
  from track in Track, as: :track,
    inner_lateral_join: top_track in subquery(
      from Track,
      where: [album_id: parent_as(:track).album_id],
      order_by: :name,
      limit: 3,
      select: [:track_id]
    ), on: (track.track_id == top_track.track_id)
```

- Ecto requires the schema in from match the type being preloaded
- Can’t use Album in outer query :(

```
-- [debug] QUERY OK source="Artist" db=2.2ms idle=1212.5ms
SELECT A0."ArtistId", A0."Name"
FROM "Artist" AS A0
ORDER BY A0."ArtistId"
LIMIT 10 [];

-- [debug] QUERY OK source="Album" db=2.2ms idle=1214.9ms
SELECT A0."AlbumId", A0."Title", A0."ArtistId", A0."ArtistId"
FROM "Album" AS A0 INNER JOIN LATERAL (
    SELECT sA0."AlbumId" AS "album_id"
    FROM "Album" AS sA0
    WHERE (sA0."ArtistId" = A0."ArtistId")
    ORDER BY sA0."Title"
    LIMIT 1
) AS s1 ON A0."AlbumId" = s1."album_id"
WHERE (A0."ArtistId" = ANY($1))
ORDER BY A0."ArtistId" [[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]];

-- [debug] QUERY OK source="Track" db=3.6ms idle=1217.5ms
SELECT T0."TrackId", T0."Name", T0."Composer", T0."Milliseconds",
       T0."Bytes", T0."UnitPrice", T0."MediaTypeId", T0."GenreId", T0."AlbumId", T0."AlbumId"
FROM "Track" AS T0 INNER JOIN LATERAL (
    SELECT sT0."TrackId" AS "track_id"
    FROM "Track" AS sT0
    WHERE (sT0."AlbumId" = T0."AlbumId")
    ORDER BY sT0."Name" LIMIT 3) AS s1 ON T0."TrackId" = s1."track_id"
WHERE (T0."AlbumId" = ANY(ARRAY[1, 2, 5, 6, 7, 34, 9, 10, 12, 13]))
ORDER BY T0."AlbumId";
```

Generic Helpers!

```
from artist in Artist,
order_by: artist.artist_id,
limit: 10,
select: artist,
preload: [albums: ^top_n(Artist, :albums, order_by: :title, limit: 1)],
preload: [albums: [tracks: ^top_n(Album, :tracks, order_by: :name, limit: 3)]]
```

Preload Functions

Preload Functions

```
albums_func =
  fn artist_ids ->
    Repo.all(from a in Album, where: a.artist_id in ^artist_ids)
  end

from a in Artist,
  preload: [albums: ^albums_func],
  select: a
```

- Function is given IDs of parent records
- Can query DB or external sources

Common Table Expressions

Common Table Expressions

```
cte_query =
  from a in Artist,
  where: a.artist_id < 10,
  select: a

query =
  Album
  |> with_cte("artist", as: ^cte_query)
  |> join(:inner, [album], a in "artist", on: album.artist_id == a.artist_id)
  |> select([album, artist], %{title: album.title, name: artist.name})
```

WITH provides a way to write auxiliary statements for use in a larger query. These statements, which are often referred to as Common Table Expressions or CTEs, can be thought of as defining temporary tables that exist just for one query.

https://www.postgresql.org/docs/current/queries-with.html

Added in Ecto 3.2

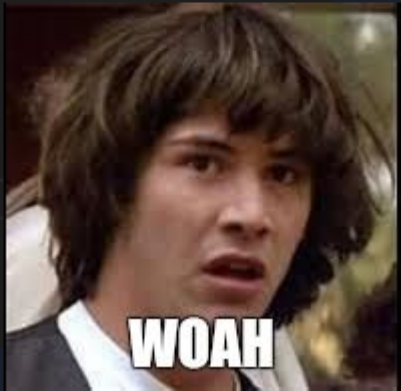
Solution #5 - Preload Functions with CTEs

#5 - Preload function with CTE

```
def albums_for_artist(order_by: order_by, limit: limit) do
  fn artist_ids ->
    cte_query =
      "artist"
      |> with_cte("artist", as: fragment("select unnest(? :: int[]) as artist_id", ^artist_ids))

    query =
      from artist in cte_query, as: :artist,
        inner_lateral_join: album in subquery(
          from a in Album,
            where: a.artist_id == parent_as(:artist).artist_id,
            order_by: ^order_by,
            limit: ^limit,
            select: a
          ),
        select: album

    Repo.all(query)
  end
end
```

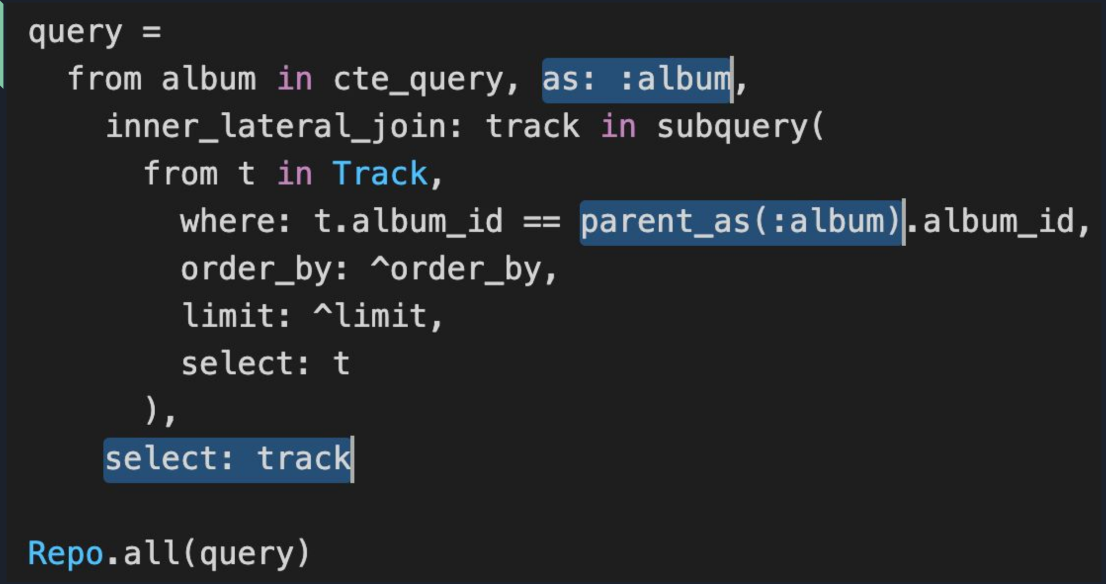


#5 - Preload function with CTE

```
fn artist_ids ->
  cte_query =
    "artist"
    |> with_cte("artist", as: fragment("select unnest(? :: int[]) as artist_id", ^artist_ids))
```

- Magical incantation required to drive a query from a CTE
- unnest converts array of IDs to rows

#5 - Preload function with CTE



```
query =
  from album in cte_query, as: :album,
    inner_lateral_join: track in subquery(
      from t in Track,
        where: t.album_id == parent_as(:album).album_id,
        order_by: ^order_by,
        limit: ^limit,
        select: t
      ),
    select: track

Repo.all(query)
```

- Apply named binding to CTE rows
- Lateral join
- Select whole row in subquery

```
-- [debug] QUERY OK source="Artist" db=2.7ms idle=991.8ms
SELECT A0."ArtistId", A0."Name"
FROM "Artist" AS A0
ORDER BY A0."ArtistId"
LIMIT 10;

-- [debug] QUERY OK source="artist" db=2.8ms idle=988.9ms
WITH "artist" AS (select unnest($1 :: int[]) as artist_id)
SELECT s1."album_id", s1."title", s1."artist_id"
FROM "artist" AS a0
INNER JOIN LATERAL (
    SELECT sA0."AlbumId" AS "album_id", sA0."Title" AS "title", sA0."ArtistId" AS "artist_id"
    FROM "Album" AS sA0
    WHERE (sA0."ArtistId" = a0."artist_id")
    ORDER BY sA0."Title"
    LIMIT $2) AS s1 ON TRUE [[10, 9, 8, 7, 6, 5, 4, 3, 2, 1], 1];

-- [debug] QUERY OK source="album" db=2.3ms idle=987.5ms
WITH "album" AS (select unnest($1 :: int[]) as album_id)
SELECT s1."track_id", s1."name", s1."composer", s1."milliseconds",
       s1."bytes", s1."unit_price", s1."media_type_id", s1."genre_id", s1."album_id"
FROM "album" AS a0
INNER JOIN LATERAL (
    SELECT sT0."TrackId" AS "track_id", sT0."Name" AS "name",
           sT0."Composer" AS "composer", sT0."Milliseconds" AS "milliseconds",
           sT0."Bytes" AS "bytes", sT0."UnitPrice" AS "unit_price",
           sT0."MediaTypeId" AS "media_type_id", sT0."GenreId" AS "genre_id", sT0."AlbumId" AS "album_id"
    FROM "Track" AS sT0
    WHERE (sT0."AlbumId" = a0."album_id")
    ORDER BY sT0."Name"
    LIMIT $2) AS s1 ON TRUE [[1, 2, 5, 6, 7, 34, 9, 10, 12, 13], 3];
```

Recap

Preload Top-N

- Use lateral joins and a single query for performance
- Use preload queries for easy composition
- Use preload functions when necessary
- Use windows when rank field needs to be included in response

Modern SQL in Ecto

- Ecto has native support for some ‘Modern SQL’ constructs
  - Correlated Subquery
  - Lateral Join
  - CTEs
  - Windows
  - JSON

- Approach problems SQL-first, then translate to Ecto syntax
- Use fragments and raw SQL as an escape hatch

Thanks!

Bonus Slide 1

```elixir
def longest_tracks_per_album(limit: limit) do
  fn album_ids ->
    Repo.query!(
      """
      SELECT track.*
      FROM unnest($1::int[]) as album(album_id)
      LEFT JOIN LATERAL (
        SELECT *
        FROM "Track"
        WHERE "AlbumId" = album.album_id
        ORDER BY "Name" DESC
        LIMIT $2) track ON true
      """,
      [album_ids, limit]
    )
    |> case do
      %{rows: rows, columns: cols} -> Enum.map(rows, &Repo.load(Track, {cols, &1}))
    end
  end
end
```

Bonus Slide 2: GraphQL Relay Resolvers

```elixir
@spec tracks_for_playlist_ids(PagingOptions.t(), [playlist_id]) :: %{playlist_id => Track.t()}
      when playlist_id: integer
def tracks_for_playlist_ids(args, playlist_ids) do
  from(playlist_track in PlaylistTrack,
    as: :playlist_track,
    join: track in assoc(playlist_track, :track),
    as: :track,
    select: track
  )
  |> paginate(:track, args)
  |> batch_by(:playlist_track, :playlist_id, playlist_ids)
  |> select([playlist, track], %{playlist_id: playlist.id, track: track})
  |> Repo.all()
  |> Enum.group_by(& &1.playlist_id, & &1.track)
end
```