Ciências ULisboa

Indexing and Optimization

ADVANCED DATABASES

GROUP 1: DANIELA VIEIRA, JOÃO RAIMUNDO, JOÃO RATO, MARIA VIEIRA
PROFESSOR: CÁTIA PESQUITA

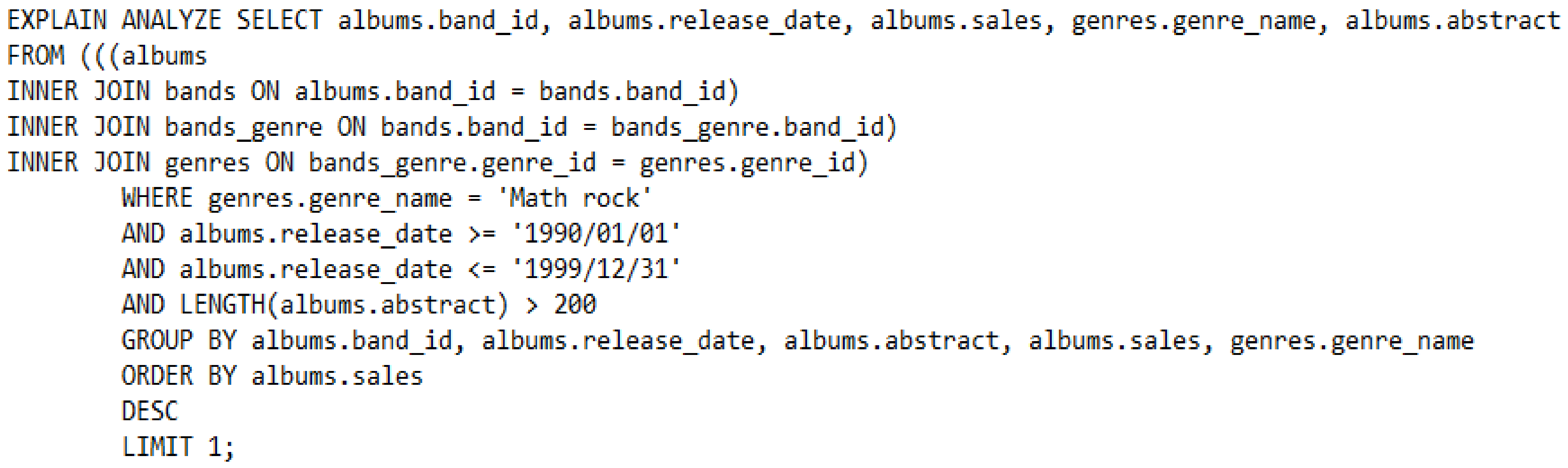
13 dezembro 2021

PostgreSQL - Query Optimization

QUERY 1

Update to 1980/01/01 the release date of all albums for the genre Math rock, which were released in the 90’s with an abstract over 200 characters, and that had most sales.

Query 1 – Execution Plan Without Indexes



```
EXPLAIN ANALYZE SELECT albums.band_id, albums.release_date, albums.sales, genres.genre_name, albums.abstract
FROM (((albums
INNER JOIN bands ON albums.band_id = bands.band_id)
INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date >= '1990/01/01'
        AND albums.release_date <= '1999/12/31'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.band_id, albums.release_date, albums.abstract, albums.sales, genres.genre_name
        ORDER BY albums.sales
        DESC
        LIMIT 1;
```

Query 1 – Execution Plan - Output

```
                                                                              QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=3563.78..3563.80 rows=1 width=504) (actual time=148.644..148.673 rows=1 loops=1)
   ->  Group  (cost=3563.78..3563.93 rows=10 width=504) (actual time=148.638..148.664 rows=1 loops=1)
         Group Key: albums.sales, albums.band_id, albums.release_date, albums.abstract, genres.genre_name
         ->  Sort  (cost=3563.78..3563.81 rows=10 width=504) (actual time=148.631..148.655 rows=1 loops=1)
               Sort Key: albums.sales DESC, albums.band_id, albums.release_date, albums.abstract
               Sort Method: quicksort  Memory: 35kB
               ->  Nested Loop  (cost=415.34..3563.62 rows=10 width=504) (actual time=83.918..148.549 rows=14 loops=1)
                     Join Filter: (albums.band_id = bands.band_id)
                     ->  Hash Join  (cost=415.06..3560.10 rows=11 width=508) (actual time=83.871..148.236 rows=14 loops=1)
                           Hash Cond: (albums.band_id = bands_genre.band_id)
                           ->  Seq Scan on albums  (cost=0.00..3136.76 rows=2179 width=492) (actual time=8.642..81.148 rows=4841 loops=1)
                                 Filter: ((release_date >= '1990-01-01'::date) AND (release_date <= '1999-12-31'::date) AND (length(abstract) > 200))
                                 Rows Removed by Filter: 29247
                           ->  Hash  (cost=414.51..414.51 rows=44 width=16) (actual time=61.220..61.232 rows=62 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                 ->  Hash Join  (cost=10.66..414.51 rows=44 width=16) (actual time=0.270..61.105 rows=62 loops=1)
                                       Hash Cond: (bands_genre.genre_id = genres.genre_id)
                                       ->  Seq Scan on bands_genre  (cost=0.00..341.32 rows=23632 width=8) (actual time=0.060..33.324 rows=23632 loops=1)
                                       ->  Hash  (cost=10.65..10.65 rows=1 width=16) (actual time=0.142..0.146 rows=1 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                             ->  Seq Scan on genres  (cost=0.00..10.65 rows=1 width=16) (actual time=0.021..0.111 rows=1 loops=1)
                                                   Filter: ((genre_name)::text = 'Math rock'::text)
                                                   Rows Removed by Filter: 531
                     ->  Index Only Scan using bands_pkey on bands  (cost=0.29..0.31 rows=1 width=4) (actual time=0.017..0.017 rows=1 loops=14)
                           Index Cond: (band_id = bands_genre.band_id)
                           Heap Fetches: 0
 Planning Time: 86.578 ms
 Execution Time: 148.778 ms
(28 rows)
```

```
 Planning Time: 86.578 ms
 Execution Time: 148.778 ms
(28 rows)
```

Query 1 – Query Plan Output with Index

```
CREATE INDEX date_abstract ON albums USING btree (release_date, length(abstract));


--- Output Query 1 Plan with data_abstract index

                                                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=3112.42..3112.43 rows=1 width=504) (actual time=83.892..83.926 rows=1 loops=1)
   ->  Group  (cost=3112.42..3112.57 rows=10 width=504) (actual time=83.885..83.916 rows=1 loops=1)
         Group Key: albums.sales, albums.band_id, albums.release_date, albums.abstract, genres.genre_name
         ->  Sort  (cost=3112.42..3112.44 rows=10 width=504) (actual time=83.879..83.908 rows=1 loops=1)
               Sort Key: albums.sales DESC, albums.band_id, albums.release_date, albums.abstract
               Sort Method: quicksort  Memory: 35kB
               ->  Nested Loop  (cost=569.88..3112.25 rows=10 width=504) (actual time=63.279..83.706 rows=14 loops=1)
                     Join Filter: (albums.band_id = bands.band_id)
                     ->  Hash Join  (cost=569.59..3108.73 rows=11 width=508) (actual time=63.207..83.384 rows=14 loops=1)
                           Hash Cond: (albums.band_id = bands_genre.band_id)
                           ->  Bitmap Heap Scan on albums  (cost=154.53..2685.40 rows=2179 width=492) (actual time=1.638..17.599 rows=4841 loops=1)
                                 Recheck Cond: ((release_date >= '1990-01-01'::date) AND (release_date <= '1999-12-31'::date) AND (length(abstract) > 200))
                                 Heap Blocks: exact=474
                                 ->  Bitmap Index Scan on date_abstract  (cost=0.00..153.99 rows=2179 width=0) (actual time=1.496..1.499 rows=4841 loops=1)
                                       Index Cond: ((release_date >= '1990-01-01'::date) AND (release_date <= '1999-12-31'::date) AND (length(abstract) > 200))
                           ->  Hash  (cost=414.51..414.51 rows=44 width=16) (actual time=59.980..59.992 rows=62 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                 ->  Hash Join  (cost=10.66..414.51 rows=44 width=16) (actual time=0.340..59.831 rows=62 loops=1)
                                       Hash Cond: (bands_genre.genre_id = genres.genre_id)
                                       ->  Seq Scan on bands_genre  (cost=0.00..341.32 rows=23632 width=8) (actual time=0.076..30.948 rows=23632 loops=1)
                                       ->  Hash  (cost=10.65..10.65 rows=1 width=16) (actual time=0.177..0.180 rows=1 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                             ->  Seq Scan on genres  (cost=0.00..10.65 rows=1 width=16) (actual time=0.029..0.156 rows=1 loops=1)
                                                   Filter: ((genre_name)::text = 'Math rock'::text)
                                                   Rows Removed by Filter: 531
                     ->  Index Only Scan using bands_pkey on bands  (cost=0.29..0.31 rows=1 width=4) (actual time=0.018..0.018 rows=1 loops=14)
                           Index Cond: (band_id = bands_genre.band_id)
                           Heap Fetches: 0
 Planning Time: 30.279 ms
 Execution Time: 84.312 ms
(30 rows)
```

B-tree Composite Index

```
 Planning Time: 30.279 ms
 Execution Time: 84.312 ms
(30 rows)
```

Query 1 – Clustered Index

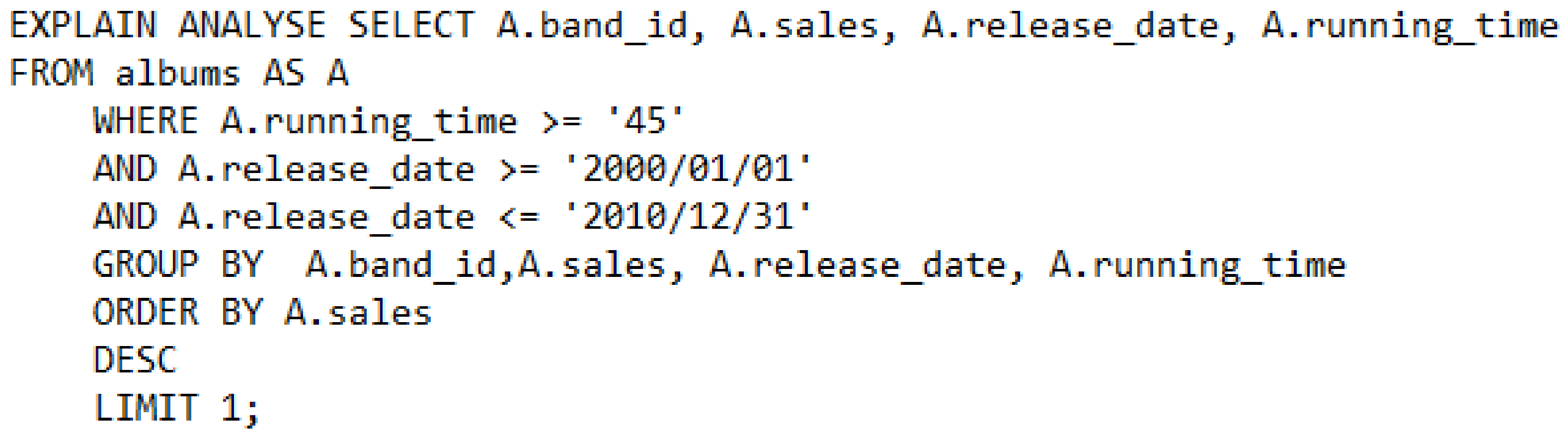
```
CREATE INDEX date ON albums USING btree (release_date);

CLUSTER albums USING date;
ANALYZE albums;


--- EXPLAIN ANALYSE QUERY 1 WITH the albums table clustered by release date


                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=1126.69..1126.71 rows=1 width=506) (actual time=80.687..80.716 rows=1 loops=1)
   ->  Group  (cost=1126.69..1127.02 rows=22 width=506) (actual time=80.682..80.708 rows=1 loops=1)
         Group Key: albums.sales, albums.band_id, albums.release_date, albums.abstract, genres.genre_name
         ->  Sort  (cost=1126.69..1126.75 rows=22 width=506) (actual time=80.677..80.701 rows=1 loops=1)
               Sort Key: albums.sales DESC, albums.band_id, albums.release_date, albums.abstract
               Sort Method: quicksort  Memory: 35kB
               ->  Nested Loop  (cost=415.63..1126.20 rows=22 width=506) (actual time=58.888..80.652 rows=14 loops=1)
                     Join Filter: (albums.band_id = bands.band_id)
                     ->  Hash Join  (cost=415.35..1118.20 rows=25 width=510) (actual time=58.871..80.525 rows=14 loops=1)
                           Hash Cond: (albums.band_id = bands_genre.band_id)
                           ->  Index Scan using date on albums  (cost=0.29..684.44 rows=4924 width=494) (actual time=0.047..17.814 rows=4841 loops=1)
                                 Index Cond: ((release_date >= '1990-01-01'::date) AND (release_date <= '1999-12-31'::date))
                                 Filter: (length(abstract) > 200)
                                 Rows Removed by Filter: 1717
                           ->  Hash  (cost=414.51..414.51 rows=44 width=16) (actual time=56.719..56.731 rows=62 loops=1)
                                 Buckets: 1024  Batches: 1  Memory Usage: 11kB
                                 ->  Hash Join  (cost=10.66..414.51 rows=44 width=16) (actual time=0.218..56.609 rows=62 loops=1)
                                       Hash Cond: (bands_genre.genre_id = genres.genre_id)
                                       ->  Seq Scan on bands_genre  (cost=0.00..341.32 rows=23632 width=8) (actual time=0.074..28.020 rows=23632 loops=1)
                                       ->  Hash  (cost=10.65..10.65 rows=1 width=16) (actual time=0.086..0.090 rows=1 loops=1)
                                             Buckets: 1024  Batches: 1  Memory Usage: 9kB
                                             ->  Seq Scan on genres  (cost=0.00..10.65 rows=1 width=16) (actual time=0.029..0.075 rows=1 loops=1)
                                                   Filter: ((genre_name)::text = 'Math rock'::text)
                                                   Rows Removed by Filter: 531
                     ->  Index Only Scan using bands_pkey on bands  (cost=0.29..0.31 rows=1 width=4) (actual time=0.003..0.004 rows=1 loops=14)
                           Index Cond: (band_id = bands_genre.band_id)
                           Heap Fetches: 0
 Planning Time: 1.158 ms
 Execution Time: 80.790 ms
(29 rows)
```

```
Planning Time: 1.158 ms
Execution Time: 80.790 ms
```

QUERY 2

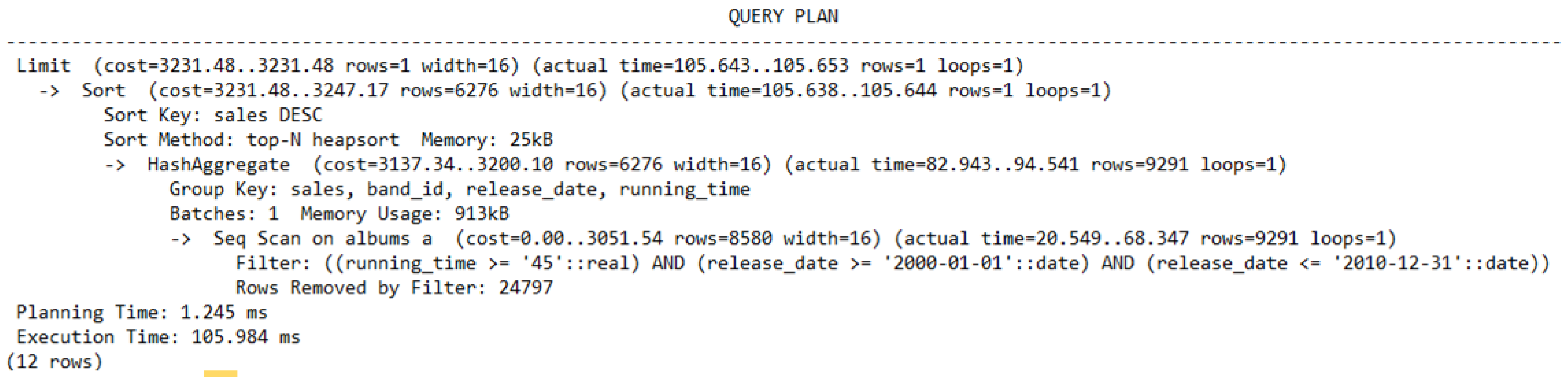
Update to 0 the sales from the album with the most sales in the first decade of the year 2000, and which the running time is longer than 45 minutes.

Query 2 – Execution Plan Without Indexes



```
EXPLAIN ANALYSE SELECT A.band_id, A.sales, A.release_date, A.running_time
FROM albums AS A
    WHERE A.running_time >= '45'
    AND A.release_date >= '2000/01/01'
    AND A.release_date <= '2010/12/31'
    GROUP BY  A.band_id,A.sales, A.release_date, A.running_time
    ORDER BY A.sales
    DESC
    LIMIT 1;
```

Query 2 – Execution Plan - Output



```
                                                              QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=3231.48..3231.48 rows=1 width=16) (actual time=105.643..105.653 rows=1 loops=1)
   ->  Sort  (cost=3231.48..3247.17 rows=6276 width=16) (actual time=105.638..105.644 rows=1 loops=1)
         Sort Key: sales DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=3137.34..3200.10 rows=6276 width=16) (actual time=82.943..94.541 rows=9291 loops=1)
               Group Key: sales, band_id, release_date, running_time
               Batches: 1  Memory Usage: 913kB
               ->  Seq Scan on albums a  (cost=0.00..3051.54 rows=8580 width=16) (actual time=20.549..68.347 rows=9291 loops=1)
                     Filter: ((running_time >= '45'::real) AND (release_date >= '2000-01-01'::date) AND (release_date <= '2010-12-31'::date))
                     Rows Removed by Filter: 24797
 Planning Time: 1.245 ms
 Execution Time: 105.984 ms
(12 rows)
```

```
 Planning Time: 1.245 ms
 Execution Time: 105.984 ms
(12 rows)
```

Query 2 – Query Plan Output with Index

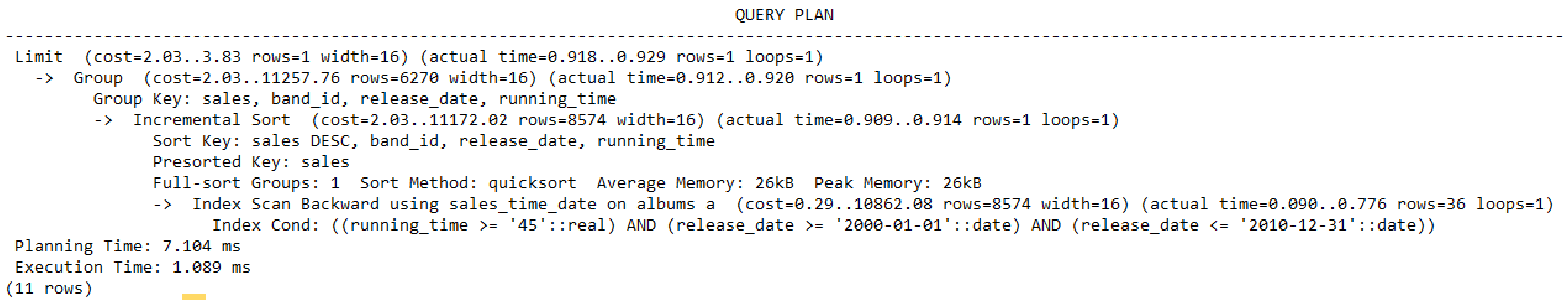
B-tree Composite Index

```
CREATE INDEX sales_time_date ON albums USING btree (sales,running_time,release_date);
```

```
                                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=2.03..3.82 rows=1 width=16) (actual time=0.848..0.859 rows=1 loops=1)
   ->  Group  (cost=2.03..11258.45 rows=6276 width=16) (actual time=0.842..0.850 rows=1 loops=1)
         Group Key: sales, band_id, release_date, running_time
         ->  Incremental Sort  (cost=2.03..11172.65 rows=8580 width=16) (actual time=0.839..0.844 rows=1 loops=1)
               Sort Key: sales DESC, band_id, release_date, running_time
               Presorted Key: sales
               Full-sort Groups: 1  Sort Method: quicksort  Average Memory: 26kB  Peak Memory: 26kB
               ->  Index Scan Backward using sales_time_date on albums a  (cost=0.29..10862.49 rows=8580 width=16) (actual time=0.087..0.717 rows=36 loops=1)
                     Index Cond: ((running_time >= '45'::real) AND (release_date >= '2000-01-01'::date) AND (release_date <= '2010-12-31'::date))
 Planning Time: 13.946 ms
 Execution Time: 1.019 ms
(11 rows)
```

```
 Planning Time: 13.946 ms
 Execution Time: 1.019 ms
(11 rows)
```

Query 2 – Clustered Index



```
                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=2.03..3.83 rows=1 width=16) (actual time=0.918..0.929 rows=1 loops=1)
   ->  Group  (cost=2.03..11257.76 rows=6270 width=16) (actual time=0.912..0.920 rows=1 loops=1)
         Group Key: sales, band_id, release_date, running_time
         ->  Incremental Sort  (cost=2.03..11172.02 rows=8574 width=16) (actual time=0.909..0.914 rows=1 loops=1)
               Sort Key: sales DESC, band_id, release_date, running_time
               Presorted Key: sales
               Full-sort Groups: 1  Sort Method: quicksort  Average Memory: 26kB  Peak Memory: 26kB
               ->  Index Scan Backward using sales_time_date on albums a  (cost=0.29..10862.08 rows=8574 width=16) (actual time=0.090..0.776 rows=36 loops=1)
                     Index Cond: ((running_time >= '45'::real) AND (release_date >= '2000-01-01'::date) AND (release_date <= '2010-12-31'::date))
 Planning Time: 7.104 ms
 Execution Time: 1.089 ms
(11 rows)
```

```
Planning Time: 7.104 ms
Execution Time: 1.089 ms
```

NoSQL - Query Optimization

- Permission problems between the MongoDB Shell (mongosh) and MongoDB Compass
- We used the MongoDB Compass GUI tabs to create the indexes and the explain query plans

Query 1 – Execution Plan Without Index – Output

```
Query Performance Summary
          Documents Returned: 1
          Index Keys Examined:0
          Documents Examined: 34088
          Actual Query Execution Time (ms): 27
          Sorted in Memory: yes
          No index available for this query.
```

```
{
 "stage": "SORT",
 "nReturned": 1,
 "executionTimeMillisEstimate": 3,
 "works": 34092,
 "advanced": 1,
 "needTime": 34090,
 "needYield": 0,
 "saveState": 34,
 "restoreState": 34,
 "isEOF": 1,
 "sortPattern": {
  "sales": -1
 },
 "memLimit": 33554432,
 "limitAmount": 1,
 "type": "simple",
 "totalDataSizeSorted": 24526,
 "usedDisk": false
}
```

```
{
 "stage": "COLLSCAN",
 "filter": {
  "$and": [
   {
    "genres": {
     "$eq": "Math rock"
    }
   },
   {
    "release_date": {
     "$lte": "1999-12-31T00:00:00.000Z"
    }
   },
   {
    "abstract": {
     "$regex": "^[\\s\\S]{200,}$"
    }
   }
  ]
 },
 "nReturned": 22,
 "executionTimeMillisEstimate": 3,
 "works": 34090,
 "advanced": 22,
 "needTime": 34067,
 "needYield": 0,
 "saveState": 34,
 "restoreState": 34,
 "isEOF": 1,
 "direction": "forward",
 "docsExamined": 34088
}
```

Query 1 – Execution Plan With Index - Output

```
db.albums.createIndex({"sales":-1}, {"release_date":1}, name: "sales_date")
```

```
Query Performance Summary:
          Documents Returned: 1
          Index Keys Examined: 1926
          Documents Examined: 1926
          Actual Query Execution Time (ms): 5
          Sorted in Memory: no
          Query used the following index: sales_date
```

```
{
 "stage": "LIMIT",
 "nReturned": 1,
 "executionTimeMillisEstimate": 4,
 "works": 1927,
 "advanced": 1,
 "needTime": 1925,
 "needYield": 0,
 "saveState": 1,
 "restoreState": 1,
 "isEOF": 1,
 "limitAmount": 1
}
```

```
{
 "stage": "FETCH",
 "filter": {
  "$and": [
   {
    "genres": {
     "$eq": "Math rock"
    }
   },
   {
    "release_date": {
     "$lte": "1999-12-31T00:00:00.000Z"
    }
   },
   {
    "abstract": {
     "$regex": "^[\\s\\S]{200,}$"
    }
   }
  ]
 },
 "nReturned": 1,
 "executionTimeMillisEstimate": 4,
 "works": 1926,
 "advanced": 1,
 "needTime": 1925,
 "needYield": 0,
 "saveState": 1,
 "restoreState": 1,
 "isEOF": 0,
 "docsExamined": 1926,
 "alreadyHasObj": 0
}
```

```
{
 "stage": "IXSCAN",
 "nReturned": 1926,
 "executionTimeMillisEstimate": 1,
 "works": 1926,
 "advanced": 1926,
 "needTime": 0,
 "needYield": 0,
 "saveState": 1,
 "restoreState": 1,
 "isEOF": 0,
 "keyPattern": {
  "sales": -1,
  "release_date": 1
 },
 "indexName": "sales_date",
 "isMultiKey": false,
 "multiKeyPaths": {
  "sales": [],
  "release_date": []
 },
 "isUnique": false,
 "isSparse": false,
 "isPartial": false,
 "indexVersion": 2,
 "direction": "forward",
 "indexBounds": {
  "sales": [
   "[MaxKey, MinKey]"
  ],
  "release_date": [
   "[MinKey, MaxKey]"
  ]
 },
 "keysExamined": 1926,
 "seeks": 1,
 "dupsTested": 0,
 "dupsDropped": 0
}
```

Query 2 – Execution Plan Without Index – Output

```
Query Performance Summary:
         Documents Returned: 1
         Index Keys Examined: 0
         Documents Examined: 34088
         Actual Query Execution Time (ms): 28
         Sorted in Memory: yes
         No index available for this query.
```

```
{
 "stage": "SORT",
 "nReturned": 1,
 "executionTimeMillisEstimate": 9,
 "works": 34092,
 "advanced": 1,
 "needTime": 34090,
 "needYield": 0,
 "saveState": 34,
 "restoreState": 34,
 "isEOF": 1,
 "sortPattern": {
  "sales": -1
 },
 "memLimit": 33554432,
 "limitAmount": 1,
 "type": "simple",
 "totalDataSizeSorted": 18924046,
 "usedDisk": false
}
```

```
{
 "stage": "COLLSCAN",
 "filter": {
  "$and": [
   {
    "release_date": {
     "$lte": "2010-12-31T00:00:00.000Z"
    }
   },
   {
    "running_time": {
     "$gte": 45
    }
   }
  ]
 },
 "nReturned": 14628,
 "executionTimeMillisEstimate": 8,
 "works": 34090,
 "advanced": 14628,
 "needTime": 19461,
 "needYield": 0,
 "saveState": 34,
 "restoreState": 34,
 "isEOF": 1,
 "direction": "forward",
 "docsExamined": 34088
}
```

Query 2 – Execution Plan With Index - Output

```
db.albums.createIndex({"sales":-1},{"release_date":1},{"running_time":1}, name: "sales_date_time”)
```

```
Query Performance Summary
Documents Returned: 1
Index Keys Examined: 1
Documents Examined: 1
Actual Query Execution Time (ms): 1
Sorted in Memory: no
Query used the following index: sales_date_time
```

```
{
 "stage": "LIMIT",
 "nReturned": 1,
 "executionTimeMillisEstimate": 1,
 "works": 2,
 "advanced": 1,
 "needTime": 0,
 "needYield": 0,
 "saveState": 0,
 "restoreState": 0,
 "isEOF": 1,
 "limitAmount": 1
}
```

```
{
 "stage": "FETCH",
 "filter": {
  "$and": [
   {
    "release_date": {
     "$lte": "2010-12-31T00:00:00.000Z"
    }
   },
   {
    "running_time": {
     "$gte": 45
    }
   }
  ]
 },
 "nReturned": 1,
 "executionTimeMillisEstimate": 1,
 "works": 1,
 "advanced": 1,
 "needTime": 0,
 "needYield": 0,
 "saveState": 0,
 "restoreState": 0,
 "isEOF": 0,
 "docsExamined": 1,
 "alreadyHasObj": 0
}
```

```
{
 "stage": "IXSCAN",
 "nReturned": 1,
 "executionTimeMillisEstimate": 1,
 "works": 1,
 "advanced": 1,
 "needTime": 0,
 "needYield": 0,
 "saveState": 0,
 "restoreState": 0,
 "isEOF": 0,
 "keyPattern": {
  "sales": -1,
  "release_date": 1,
  "running_time": 1
 },
 "indexName": "sales_date_time",
 "isMultiKey": false,
 "multiKeyPaths": {
  "sales": [],
  "release_date": [],
  "running_time": []
 },
 "isUnique": false,
 "isSparse": false,
 "isPartial": false,
 "indexVersion": 2,
 "direction": "forward",
 "indexBounds": {
  "sales": [
   "[MaxKey, MinKey]"
  ],
  "release_date": [
   "[MinKey, MaxKey]"
  ],
  "running_time": [
   "[MinKey, MaxKey]"
  ]
 },
 "keysExamined": 1,
 "seeks": 1,
 "dupsTested": 0,
 "dupsDropped": 0
}
```

Discussion

- NoSQL relatively faster than PostgreSQL (query execution times)
  - Query 1 needs 3 joins in PostgreSQL – Takes time!
  - NoSQL was structured in a way that it does not need multiple aggregations, only one -> Albums

- No alterations for RDB, **but** we know a few:
  - Denormalization (genre names) – reducing the number of joins
  - Horizontal decomposition (transforming *release_date* into *decades* and creating new relations [90s, 80s, etc])
  - Not performed since the queries execution times were fast enough (ms)

- No alterations needed for NoSQL