sql-opt-1

August 21, 2025

1 SQL - Optimization I

1.0.1 1. Introduction for Index Creation

The following code is how we initialize the sql magic for our course

[1]:
```
%load_ext sql
%config SqlMagic.displaycon = 0
%config SqlMagic.displaylimit = 100
%config SqlMagic.feedback = 0
%sql postgresql+psycopg://bank:bank@postgres/bank_index
```

First we show the firts 20 rows of the table account in this practice so we can see the information filled in the table.

[4]:
```
%%sql
SELECT
  *
FROM
  account
LIMIT
  15;
```

[4]:
```
+----------------+-------------+-----------+
| account_number | branch_name |  balance  |
+----------------+-------------+-----------+
|   A-000000     |   Downtown  | 3467.0000 |
|   A-000001     |   Downtown  | 1500.0000 |
|   A-000002     |    Uptown   |  724.0000 |
|   A-000003     |    Metro    | 4358.0000 |
|   A-000004     |    Metro    | 4464.0000 |
|   A-000005     |  University | 3145.0000 |
|   A-000006     |   Downtown  | 1827.0000 |
|   A-000007     |  University |  491.0000 |
|   A-000008     |   Downtown  | 1942.0000 |
|   A-000009     |  University |  436.0000 |
|   A-000010     |  Wine Celar | 4604.0000 |
|   A-000011     |    Uptown   |  153.0000 |
|   A-000012     |    Metro    | 2382.0000 |
```

```
|   A-000013     |    Metro    | 3716.0000 |
|   A-000014     |   Downtown  | 4895.0000 |
+----------------+-------------+-----------+
```

Now we run a query to look for a specific account number with EXPLAIN to see the execution plan.

[5]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  *
FROM
  account
WHERE
  account_number = 'A-012345';
```

[5]:
```
+-----------------------------------------------------------------------------
----------------------------------------+
|                                                        QUERY PLAN
|
+-----------------------------------------------------------------------------
----------------------------------------+
| Index Scan using account_pkey on account  (cost=0.42..8.44 rows=1 width=23)
(actual time=0.030..0.031 rows=1 loops=1) |
|                                    Index Cond: (account_number =
'A-012345'::bpchar)                                    |
|                                                  Buffers: shared hit=4
|
|                                                       Planning:
|
|                                                  Buffers: shared hit=9
|
|                                             Memory: used=10kB  allocated=16kB
|
|                                                 Planning Time: 0.111 ms
|
|                                 Serialization: time=0.002 ms  output=1kB
format=text                                 |
|                                                 Execution Time: 0.055 ms
|
+-----------------------------------------------------------------------------
----------------------------------------+
```

We have an index scan because the search is done on account_number which is the primary key for the table (and it is clustered).

To see how the query would behave without index and compare results, we delete the primary key from the table with the command:

[6]:
```
%%sql
ALTER TABLE account
DROP CONSTRAINT account_pkey;
```

[6]:
```
++
||
++
++
```

Now we repeat the query and we can see the difference in time with the previous one.

[7]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  *
FROM
  account
WHERE
  account_number = 'A-012345';
```

[7]:
```
+-----------------------------------------------------------------------------
----------------------+
|                                                 QUERY PLAN
|
+-----------------------------------------------------------------------------
----------------------+
| Seq Scan on account  (cost=0.00..1931.00 rows=1 width=23) (actual
time=1.163..5.947 rows=1 loops=1) |
|                              Filter: (account_number = 'A-012345'::bpchar)
|
|                                    Rows Removed by Filter: 99999
|
|                                       Buffers: shared hit=681
|
|                                             Planning:
|
|                                    Buffers: shared hit=5 dirtied=1
|
|                                    Memory: used=8kB  allocated=24kB
|
|                                       Planning Time: 0.103 ms
|
|                        Serialization: time=0.006 ms  output=1kB  format=text
|
|                                       Execution Time: 5.974 ms
|
+-----------------------------------------------------------------------------
----------------------+
```

As there is not PK and for then not index, the database does not have fast way to locate account_number='A-012345' and for that reason it performs a sequential scan which means read all rows in the table to find a match.

We add the primary key back, to see how much time it takes to create the PK for this table and we see it is not expensive which means it should be done.

[8]:
```
%%sql
ALTER TABLE account ADD PRIMARY KEY (account_number);
```

[8]:
```
++
||
++
++
```

1.0.2 2. Execution Plans and Index Optimization

Here we consider two queries: one to obtain the accounts with a balance equal to €1000, and another to obtain the maximum balance.

First we run the queries and note the time it takes the system to execute each command.

[3]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  account_number
FROM
  account
WHERE
  balance = 1000;
```

[3]:
```
+-------------------------------------------------------------------------------
-------------------------+
|                                                 QUERY PLAN
|
+-------------------------------------------------------------------------------
-------------------------+
| Seq Scan on account  (cost=0.00..1931.00 rows=20 width=10) (actual
time=6.851..63.904 rows=22 loops=1) |
|                                    Filter: (balance = '1000'::numeric)
|
|                                       Rows Removed by Filter: 99978
|
|                                          Buffers: shared read=681
|
|                                                Planning:
|
|                                  Buffers: shared hit=28 read=7 dirtied=1
|
```

```
|                                        Memory: used=9kB  allocated=24kB
|
|                                           Planning Time: 5.611 ms
|
|                             Serialization: time=0.045 ms  output=1kB  format=text
|
|                                         Execution Time: 64.010 ms
|
+-------------------------------------------------------------------------------
------------------------+
```

[4]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  MAX(balance)
FROM
  account;
```

[4]:
```
+-------------------------------------------------------------------------------
-----------------------------------+
|                                                        QUERY PLAN
|
+-------------------------------------------------------------------------------
-----------------------------------+
|           Aggregate  (cost=1931.00..1931.01 rows=1 width=32) (actual
time=22.351..22.353 rows=1 loops=1)           |
|                                                 Buffers: shared hit=681
|
|   ->  Seq Scan on account  (cost=0.00..1681.00 rows=100000 width=4) (actual
time=0.012..7.733 rows=100000 loops=1) |
|                                                    Buffers: shared hit=681
|
|                                                       Planning:
|
|                                              Buffers: shared hit=17 read=2
|
|                                            Memory: used=25kB  allocated=32kB
|
|                                                Planning Time: 3.157 ms
|
|                                  Serialization: time=0.009 ms  output=1kB
format=text                                  |
|                                               Execution Time: 22.487 ms
|
+-------------------------------------------------------------------------------
-----------------------------------+
```

Now we create an b-tree index for the balance column with the command:

[5]:
```
%%sql
CREATE INDEX balance_idx ON account (balance);
```

[5]:
```
++
||
++
++
```

We run the queries again to analize what impact have in the execution.

[6]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  account_number
FROM
  account
WHERE
  balance = 1000;
```

[6]:
```
+-------------------------------------------------------------------------------
--------------------------------------+
|                                                    QUERY PLAN
|
+-------------------------------------------------------------------------------
--------------------------------------+
|     Bitmap Heap Scan on account  (cost=4.57..74.54 rows=20 width=10) (actual
time=0.102..0.159 rows=22 loops=1)      |
|                                         Recheck Cond: (balance =
'1000'::numeric)                                  |
|                                               Heap Blocks: exact=21
|
|                                            Buffers: shared hit=21 read=3
|
|   ->  Bitmap Index Scan on balance_idx  (cost=0.00..4.57 rows=20 width=0)
(actual time=0.089..0.089 rows=22 loops=1) |
|                                           Index Cond: (balance =
'1000'::numeric)                                  |
|                                                    Buffers: shared read=3
|
|                                                      Planning:
|
|                                            Buffers: shared hit=19 read=1
|
|                                          Memory: used=11kB  allocated=16kB
|
|                                              Planning Time: 0.536 ms
|
|                                  Serialization: time=0.006 ms  output=1kB
```

```
format=text                                          |
|                                                        Execution Time: 0.189 ms
|
+-------------------------------------------------------------------------------
-------------------------------------+
```

Without index the planner performs a sequential scan on the entire account table to check every row's balance value and see if it is 1000. After creating the index on balance, PostgreSQL now can do an index only scan or index scan using the B-tree on balance as allows binary search.

[7]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  MAX(balance)
FROM
  account;
```

[7]:
```
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------+
|                                                                                QUERY
PLAN                                                                             |
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------+
|                                    Result  (cost=0.44..0.45 rows=1 width=32)
(actual time=0.195..0.197 rows=1 loops=1)                                        |
|                                                                  Buffers: shared
hit=2 read=2                                                                     |
|
InitPlan 1
|
|                                      ->  Limit  (cost=0.42..0.44 rows=1 width=4)
(actual time=0.189..0.190 rows=1 loops=1)                                        |
|                                                                        Buffers:
shared hit=2 read=2                                                              |
|           ->  Index Only Scan Backward using balance_idx on account
(cost=0.42..2612.42 rows=100000 width=4) (actual time=0.187..0.188 rows=1
loops=1) |
|
Heap Fetches: 0                                                                  |
|                                                                             Buffers:
shared hit=2 read=2                                                              |
|
Planning:
|
|                                                                  Memory: used=23kB
allocated=32kB                                                                   |
|                                                                     Planning Time:
0.151 ms                                                                         |
```

```
|                                                    Serialization: time=0.003 ms
output=1kB  format=text                                                        |
|                                                                  Execution
Time: 0.233 ms                                                                 |
+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
```

Without index PostgreSQL scans the whole table to find the highest balance but with index on balance it can be used the B-tree structure to go directly to the highest leaf node and find the MAX value instantly.

We drop the index now:

[8]:
```
%%sql
DROP INDEX balance_idx;
```

[8]:
```
++
||
++
++
```

Now we create an hash index for the balance column with the command:

[9]:
```
%%sql
CREATE INDEX balance_idx ON account USING HASH (balance);
```

[9]:
```
++
||
++
++
```

We run the queries again to analize what impact have in the execution

[10]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  account_number
FROM
  account
WHERE
  balance = 1000;
```

[10]:
```
+-------------------------------------------------------------------------------
--------------------------------------+
|                                                  QUERY PLAN
|
+-------------------------------------------------------------------------------
--------------------------------------+
|     Bitmap Heap Scan on account  (cost=4.16..74.12 rows=20 width=10) (actual
```

```
time=0.037..0.106 rows=22 loops=1)      |
|                                       Recheck Cond: (balance =
'1000'::numeric)                                    |
|                                                 Heap Blocks: exact=21
|
|                                                 Buffers: shared hit=23
|
|   ->  Bitmap Index Scan on balance_idx  (cost=0.00..4.15 rows=20 width=0)
(actual time=0.024..0.024 rows=22 loops=1) |
|                                               Index Cond: (balance =
'1000'::numeric)                                      |
|                                                  Buffers: shared hit=2
|
|                                                    Planning:
|
|                                                 Buffers: shared hit=17
|
|                                             Memory: used=11kB  allocated=24kB
|
|                                                 Planning Time: 0.258 ms
|
|                                 Serialization: time=0.006 ms  output=1kB
format=text                                     |
|                                                 Execution Time: 0.135 ms
|
+-------------------------------------------------------------------------------
--------------------------------------+
```

For this query we can see what we already knew, that hash indexes perform good on selective queries or equality conditions and even a little bit better that b-tree indexes.

[11]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  MAX(balance)
FROM
  account;
```

[11]:
```
+-------------------------------------------------------------------------------
-------------------------------------+
|                                                      QUERY PLAN
|
+-------------------------------------------------------------------------------
-------------------------------------+
|           Aggregate  (cost=1931.00..1931.01 rows=1 width=32) (actual
time=16.510..16.511 rows=1 loops=1)           |
|                                                Buffers: shared hit=681
|
```

```
|   ->  Seq Scan on account  (cost=0.00..1681.00 rows=100000 width=4) (actual
time=0.013..5.668 rows=100000 loops=1) |
|                                                    Buffers: shared hit=681
|
|                                                       Planning:
|
|                                              Memory: used=26kB  allocated=32kB
|
|                                                  Planning Time: 0.093 ms
|
|                                Serialization: time=0.006 ms  output=1kB
format=text                                  |
|                                                 Execution Time: 16.553 ms
|
+-------------------------------------------------------------------------------
-----------------------------------+
```

In this query the index is not being used, because hash indexes cannot support ordering, range queries, MAX, or MIN and PostgreSQL cannot traverse a hash index to find max/min values.

We drop the index created.

[12]:
```
%%sql
DROP INDEX balance_idx;
```

[12]:
```
++
||
++
++
```

1.0.3 3. Queries Optimization

We create a table to analize how can we improve queries:

[56]:
```
%%sql
DROP TABLE IF EXISTS employee;

CREATE TABLE
  employee (
    eid INTEGER PRIMARY KEY,
    ename VARCHAR(40) NOT NULL,
    address VARCHAR(255) NOT NULL,
    salary NUMERIC(12, 4) NOT NULL,
    bdate DATE NOT NULL,
    age INTEGER NOT NULL
  );
```

[56]:
```
++
||
++
++
```

We fill the table with random values just to have rows:

[57]:
```
%%sql
INSERT INTO
  employee (eid, ename, address, salary, bdate, age)
VALUES
  (
    1,
    'Alice Johnson',
    '123 Maple St',
    55000.00,
    '1990-04-12',
    35
  ),
  (
    2,
    'Bob Smith',
    '456 Oak Ave',
    72000.50,
    '1985-11-03',
    39
  ),
  (
    3,
    'Charlie Lee',
    '789 Pine Rd',
    48000.25,
    '1992-07-21',
    33
  ),
  (
    4,
    'Diana King',
    '321 Cedar Blvd',
    88000.00,
    '1980-02-15',
    45
  ),
  (
    5,
    'Ethan Wright',
    '654 Birch Ln',
    61000.75,
```

```
  '1995-09-30',
  30
),
(
  6,
  'Renato Davis',
  '987 Walnut Ct',
  93000.00,
  '1983-12-10',
  41
),
(
  7,
  'George Hall',
  '246 Cherry St',
  50000.00,
  '1991-06-25',
  34
),
(
  8,
  'Hannah Scott',
  '135 Spruce Dr',
  47000.00,
  '1998-01-18',
  27
),
(
  9,
  'Ian Turner',
  '579 Poplar Way',
  105000.00,
  '1979-03-02',
  46
),
(
  10,
  'Julia Roberts',
  '864 Palm Cir',
  83000.00,
  '1987-08-14',
  38
),
(
  11,
  'Kevin Brown',
  '753 Elm St',
```

```
  54000.00,
  '1993-10-09',
  32
),
(
  12,
  'Laura Wilson',
  '369 Magnolia Ave',
  76000.00,
  '1986-05-22',
  39
),
(
  13,
  'Michael Green',
  '147 Fir Ln',
  68000.00,
  '1990-09-11',
  34
),
(
  14,
  'Nina Carter',
  '258 Cypress Blvd',
  95000.00,
  '1982-04-29',
  43
),
(
  15,
  'Oscar Perez',
  '951 Redwood St',
  52000.00,
  '1996-11-27',
  28
),
(
  16,
  'Paula Adams',
  '753 Willow Rd',
  47000.00,
  '1997-07-19',
  28
),
(
  17,
  'Quincy Baker',
```

```
  '159 Aspen Dr',
  86000.00,
  '1984-02-08',
  41
),
(
  18,
  'Rachel Evans',
  '852 Dogwood Ln',
  78000.00,
  '1989-12-25',
  35
),
(
  19,
  'Sam Fisher',
  '357 Alder Ct',
  64000.00,
  '1994-03-15',
  31
),
(
  20,
  'Tina Young',
  '951 Beech St',
  72000.00,
  '1988-06-05',
  37
),
(
  21,
  'Colleen Marshall',
  '85264 Smith Glen',
  78267.22,
  '2011-12-28',
  13
),
(
  22,
  'Adam Meyer',
  '41855 Priscilla Oval',
  125640.5,
  '2017-12-31',
  7
),
(
  23,
```

```
  'Brett Gross',
  '7857 Gutierrez Field Apt. 835',
  67339.4,
  '2003-07-04',
  22
),
(
  24,
  'Erica Lewis',
  '96430 Garcia Unions Apt. 379',
  32748.05,
  '1991-04-28',
  34
),
(
  25,
  'Ronald Joseph',
  '028 Wilson Field',
  111600.63,
  '1955-12-26',
  69
),
(
  26,
  'Joanna Snyder',
  '5677 Mccall Turnpike',
  100348.96,
  '1956-06-19',
  69
),
(
  27,
  'Richard Henry',
  '153 Jillian Alley',
  57944.38,
  '1988-10-23',
  36
),
(
  28,
  'Carl Lewis',
  '98179 Ariana Burg Suite 204',
  98284.28,
  '1991-03-04',
  34
),
(
```

```
  29,
  'Holly Torres',
  '43515 Thompson Crescent Apt. 381',
  115521.83,
  '1962-12-31',
  62
),
(
  30,
  'Tiffany Roth',
  '533 Jones Court',
  110760.14,
  '2001-05-20',
  24
),
(
  31,
  'Christina Murphy',
  '88264 Thompson Lake Suite 489',
  61654.57,
  '1956-01-27',
  69
),
(
  32,
  'Brian Norman',
  '07571 Weeks Crossroad Apt. 611',
  44144.09,
  '1960-11-15',
  64
),
(
  33,
  'Melissa Walker',
  '429 Perry Fork Suite 910',
  105536.05,
  '2010-02-20',
  15
),
(
  34,
  'Savannah Parrish',
  '6225 Jefferson Hollow Suite 151',
  35773.74,
  '2015-12-10',
  9
),
```

```
  (
    35,
    'Cheryl Marshall',
    '7790 Kristi Estates Apt. 356',
    33496.11,
    '2005-03-21',
    20
  ),
  (
    36,
    'Michael Smith',
    '5792 Chelsea Groves',
    108597.58,
    '1985-02-05',
    40
  ),
  (
    37,
    'Jasmine Garcia',
    '862 Kimberly Ridge',
    70588.01,
    '1952-04-09',
    73
  ),
  (
    38,
    'Natasha Day',
    '5568 Kirsten Vista',
    44373.3,
    '2008-06-04',
    17
  ),
  (
    39,
    'Jessica Cooley',
    '813 Tonya Canyon Suite 355',
    129274.96,
    '1988-05-05',
    37
  ),
  (
    40,
    'Brian Terrell',
    '57153 Kenneth Court',
    92821.14,
    '1982-02-22',
    43
```

```
),
(
  41,
  'Jean Cline',
  '771 Cobb Knoll Apt. 728',
  58981.94,
  '2017-09-19',
  7
),
(
  42,
  'Luke Dougherty',
  '207 Harris Meadow',
  77321.18,
  '1977-05-17',
  48
),
(
  43,
  'Victor Clark',
  '997 Joseph Crescent',
  117791.45,
  '1983-12-03',
  41
),
(
  44,
  'Ana Wright',
  '09837 Roy Mission Suite 904',
  113345.1,
  '1999-01-14',
  26
),
(
  45,
  'Steven Sutton',
  '050 Edwin Plain Suite 031',
  69303.38,
  '1952-11-16',
  72
),
(
  46,
  'Ms. Tracy Cole',
  '5928 Montoya Junction',
  68043.53,
  '2016-07-27',
```

```
  9
),
(
  47,
  'Christina Johnson',
  '13972 Larson Port',
  96651.41,
  '1985-04-11',
  40
),
(
  48,
  'Adrian Ball',
  '454 Lopez Shores Apt. 774',
  101757.62,
  '1968-09-08',
  56
),
(
  49,
  'Mrs. Jaclyn Evans',
  '1551 Michael Springs',
  31969.55,
  '1966-01-06',
  59
),
(
  50,
  'Brandon Brooks',
  '47652 Howe Streets',
  106717.92,
  '1997-09-28',
  27
),
(
  51,
  'David White',
  '15843 Avery Loop Suite 632',
  93717.68,
  '1981-08-18',
  44
),
(
  52,
  'Isaiah Harper',
  '31710 Anne Groves',
  53128.2,
```

```
  '1985-03-11',
  40
),
(
  53,
  'Alexander Lee',
  '626 Kiara Stravenue Suite 987',
  40665.18,
  '1975-08-26',
  49
),
(
  54,
  'Judy Ramsey',
  '21291 Steven Haven Suite 281',
  114466.41,
  '2015-11-16',
  9
),
(
  55,
  'Michelle Nguyen',
  '2923 Mark Ranch Suite 357',
  64443.56,
  '1967-08-18',
  58
),
(
  56,
  'Kimberly Thomas',
  '6717 Graves Ville Suite 720',
  74754.95,
  '1959-03-15',
  66
),
(
  57,
  'Stacie Castro',
  '893 Benjamin Unions',
  79783.96,
  '1983-02-08',
  42
),
(
  58,
  'Stephanie Davis',
  '2585 Curtis Flats Apt. 884',
```

```
  46501.34,
  '1951-03-07',
  74
),
(
  59,
  'Jason Williams',
  '68392 Lindsey Flats',
  51586.92,
  '2016-04-02',
  9
),
(
  60,
  'Mr. Christopher Brennan',
  '8636 Michael Via Apt. 975',
  97890.28,
  '1973-03-30',
  52
),
(
  61,
  'Joshua Thomas',
  '9594 Olson Walks',
  48692.1,
  '1976-02-08',
  49
),
(
  62,
  'Amber Hahn',
  '234 Brown Pines',
  42111.11,
  '1979-05-21',
  46
),
(
  63,
  'Anthony Bass',
  '96633 Daniel Square',
  46625.77,
  '1971-10-28',
  53
),
(
  64,
  'Joshua Williams',
```

```
  '4404 Alyssa Square',
  83126.2,
  '1969-08-11',
  56
),
(
  65,
  'Gina Greer',
  '79808 Joseph Lakes Apt. 379',
  112155.33,
  '1996-01-29',
  29
),
(
  66,
  'Christopher Hart',
  '8236 Hart Flat',
  79396.39,
  '2004-09-18',
  20
),
(
  67,
  'Savannah Roberts',
  '597 Richard Viaduct',
  118000.06,
  '1960-01-15',
  65
),
(
  68,
  'Jill Dudley',
  '20224 Whitney Bridge Suite 017',
  111135.84,
  '1968-05-17',
  57
),
(
  69,
  'Elizabeth Sullivan',
  '580 Adam Walks Apt. 697',
  112822.24,
  '2017-10-16',
  7
),
(
  70,
```

```
  'Patrick Wilson',
  '783 Julie Course Apt. 101',
  80433.6,
  '1956-06-25',
  69
),
(
  71,
  'Julie Wilson',
  '38583 Jackson Mews',
  104141.5,
  '2011-02-26',
  14
),
(
  72,
  'Mark Holt',
  '3057 Nicole Trail Suite 844',
  47214.83,
  '1974-09-10',
  50
),
(
  73,
  'Terri Jordan',
  '812 Thompson Shore',
  72765.29,
  '2000-09-23',
  24
),
(
  74,
  'Robert Ball',
  '42884 Benjamin Ridge',
  105828.16,
  '2000-04-13',
  25
),
(
  75,
  'Catherine Newman',
  '909 Kevin Mills',
  92948.95,
  '1988-01-24',
  37
),
(
```

```
  76,
  'Kathleen Ray',
  '152 Clifford Extensions',
  32667.36,
  '1974-02-10',
  51
),
(
  77,
  'Mark Delacruz',
  '698 Schwartz Highway Suite 732',
  115129.69,
  '1955-01-10',
  70
),
(
  78,
  'Thomas Bell',
  '925 Paul Mission',
  96205.58,
  '1950-03-27',
  75
),
(
  79,
  'Amanda Barnett',
  '2172 Angela Points Apt. 994',
  51611.17,
  '1959-06-22',
  66
),
(
  80,
  'Devon Lewis',
  '2216 Vega Brook Suite 848',
  32478.06,
  '2012-02-01',
  13
),
(
  81,
  'Tammy Jenkins',
  '377 Katherine Stream',
  55495.82,
  '1965-12-02',
  59
),
```

```
  (
    82,
    'Heather Pierce',
    '479 Kelley Walk',
    93090.16,
    '1997-05-30',
    28
  ),
  (
    83,
    'Fiona Davis',
    '076 Perkins Mountain',
    87986.07,
    '1979-03-02',
    46
  ),
  (
    84,
    'Jessica White',
    '2520 Hicks Ranch Suite 294',
    36527.7,
    '1954-08-21',
    71
  ),
  (
    85,
    'Courtney White',
    '1019 John Coves Suite 202',
    79320.54,
    '1978-01-17',
    47
  ),
  (
    86,
    'Janice Benitez',
    '135 Raymond Turnpike',
    40638.14,
    '1959-05-14',
    66
  ),
  (
    87,
    'Jason Hamilton',
    '4540 Lee Mews Apt. 550',
    31383.22,
    '1998-02-12',
    27
```

```
),
(
  88,
  'Ryan Evans',
  '38796 Jackson Islands',
  98172.69,
  '1950-04-06',
  75
),
(
  89,
  'Lori Cruz',
  '9852 Mark Wall Suite 349',
  40194.44,
  '2005-03-01',
  20
),
(
  90,
  'Andrew Compton',
  '45670 Raymond Vista',
  113844.3,
  '1956-02-06',
  69
),
(
  91,
  'Christina Mcclain',
  '032 Robert Meadows Suite 335',
  79824.48,
  '1988-01-28',
  37
),
(
  92,
  'Dr. Derek Chavez Jr.',
  '046 Peggy Springs',
  32342.45,
  '1964-12-02',
  60
),
(
  93,
  'Peter Moon',
  '7540 Murphy Valleys',
  100124.86,
  '2007-04-27',
```

```
  18
),
(
  94,
  'Tammy Kennedy',
  '6427 Christopher Mission Suite 330',
  70128.23,
  '2014-06-03',
  11
),
(
  95,
  'Wendy Clay',
  '60304 Robert Valley Suite 482',
  48277.17,
  '1976-01-07',
  49
),
(
  96,
  'Mason Tran',
  '80859 Matthew Terrace',
  93272.2,
  '1994-12-22',
  30
),
(
  97,
  'Marissa Jackson',
  '377 Jeffrey Drive Suite 959',
  45337.76,
  '1985-11-17',
  39
),
(
  98,
  'James Nicholson',
  '136 Stacy Isle Suite 575',
  110715.34,
  '2002-12-03',
  22
),
(
  99,
  'Rose Wright',
  '2404 Wesley Village Suite 564',
  128608.25,
```

```
  '1968-08-28',
  56
),
(
  100,
  'Vanessa Hill',
  '72852 Karen Point',
  126959.75,
  '2005-05-04',
  20
),
(
  101,
  'Brandy Fisher',
  '841 Mary Extension',
  38467.09,
  '1980-07-04',
  45
),
(
  102,
  'Stacey Miller',
  '9631 Mary Square Apt. 250',
  80351.77,
  '1986-12-09',
  38
),
(
  103,
  'Barbara Brown',
  '6569 Gilmore Canyon',
  107808.63,
  '2011-03-17',
  14
),
(
  104,
  'Paul Jones',
  '62738 Scott Lights',
  70418.07,
  '1969-12-07',
  55
),
(
  105,
  'Kimberly Wells',
  '288 April Shoals Suite 781',
```

```
  78486.55,
  '1994-09-18',
  30
),
(
  106,
  'Charles Ward',
  '34799 West Meadows',
  109898.97,
  '1991-12-26',
  33
),
(
  107,
  'Katherine Luna',
  '641 Vincent Avenue Suite 012',
  90845.16,
  '1962-09-25',
  62
),
(
  108,
  'Todd Morgan',
  '77232 Pamela Corners Apt. 527',
  82355.97,
  '2010-08-15',
  15
),
(
  109,
  'Austin Watts',
  '53280 Carr Forest',
  31797.01,
  '2002-11-15',
  22
),
(
  110,
  'Kristy Campbell',
  '623 Fernandez River Apt. 392',
  97041.86,
  '2007-07-31',
  18
),
(
  111,
  'Kimberly Mills',
```

```
    '00376 Gomez Extension Apt. 988',
    52135.68,
    '2016-03-22',
    9
  ),
  (
    112,
    'Sara Paul',
    '2786 Rodriguez Garden Suite 340',
    99942.16,
    '1968-07-07',
    57
  ),
  (
    113,
    'Raymond Wall',
    '426 Kristin Row',
    113214.98,
    '2008-08-02',
    17
  ),
  (
    114,
    'Sarah Thornton',
    '418 Cline Locks',
    78420.38,
    '1957-01-29',
    68
  ),
  (
    115,
    'Jill Frazier',
    '36236 Melissa Meadow',
    87983.22,
    '1994-03-26',
    31
  ),
  (
    116,
    'Kelly Hernandez',
    '6260 Jennifer Summit',
    123025.54,
    '2009-12-18',
    15
  ),
  (
    117,
```

```
    'Jessica Williams',
    '580 Shannon Turnpike Apt. 004',
    109859.51,
    '1951-09-16',
    73
  ),
  (
    118,
    'Thomas Leonard',
    '4036 Gonzalez Drive Suite 774',
    122939.94,
    '1998-07-30',
    27
  ),
  (
    119,
    'Isaac Kirby',
    '88254 Hernandez Neck Apt. 493',
    72352.11,
    '1986-02-13',
    39
  ),
  (
    120,
    'Kayla Wang',
    '04501 Karen Ways',
    91390.66,
    '1957-03-23',
    68
  );
```

[57]:
```
++
||
++
++
```

a) What is the identifier, ename, and address of employees born within a certain range of dates?

[58]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  eid,
  ename,
  address,
  age (bdate)
FROM
  employee
WHERE
```

```
  bdate BETWEEN '1988-01-01' AND '1993-12-31';
```

[58]:
```
+-------------------------------------------------------------------------------
----------------------+
|                                                 QUERY PLAN
|
+-------------------------------------------------------------------------------
----------------------+
| Seq Scan on employee  (cost=0.00..11.81 rows=1 width=634) (actual
time=0.020..0.036 rows=14 loops=1) |
|               Filter: ((bdate >= '1988-01-01'::date) AND (bdate <=
'1993-12-31'::date))              |
|                                       Rows Removed by Filter: 106
|
|                                          Buffers: shared hit=2
|
|                                               Planning:
|
|                                          Buffers: shared hit=2
|
|                                    Memory: used=11kB  allocated=16kB
|
|                                        Planning Time: 0.090 ms
|
|                          Serialization: time=0.013 ms  output=1kB  format=text
|
|                                        Execution Time: 0.065 ms
|
+-------------------------------------------------------------------------------
----------------------+
```

[59]:
```
%%sql
SELECT
  eid,
  ename,
  address,
  age (bdate)
FROM
  employee
WHERE
  bdate BETWEEN '1988-01-01' AND '1993-12-31';
```

[59]:
```
+-----+------------------+-----------------------------+---------------------+
| eid |       ename      |           address           |         age         |
+-----+------------------+-----------------------------+---------------------+
|  1  |   Alice Johnson  |         123 Maple St        | 12904 days, 0:00:00 |
|  3  |    Charlie Lee   |         789 Pine Rd         | 12075 days, 0:00:00 |
```

```
|  7  |    George Hall    |        246 Cherry St         | 12466 days, 0:00:00 |
|  11 |    Kevin Brown    |          753 Elm St          | 11627 days, 0:00:00 |
|  13 |   Michael Green   |          147 Fir Ln          | 12750 days, 0:00:00 |
|  18 |    Rachel Evans   |        852 Dogwood Ln        | 13012 days, 0:00:00 |
|  20 |     Tina Young    |         951 Beech St         | 13581 days, 0:00:00 |
|  24 |    Erica Lewis    | 96430 Garcia Unions Apt. 379 | 12523 days, 0:00:00 |
|  27 |   Richard Henry   |      153 Jillian Alley       | 13439 days, 0:00:00 |
|  28 |     Carl Lewis    | 98179 Ariana Burg Suite 204  | 12577 days, 0:00:00 |
|  39 |   Jessica Cooley  |  813 Tonya Canyon Suite 355  | 13611 days, 0:00:00 |
|  75 |  Catherine Newman |       909 Kevin Mills        | 13713 days, 0:00:00 |
|  91 | Christina Mcclain | 032 Robert Meadows Suite 335 | 13709 days, 0:00:00 |
| 106 |    Charles Ward   |      34799 West Meadows      | 12281 days, 0:00:00 |
+-----+-------------------+------------------------------+---------------------+
```

[60]:
```
%%sql
CREATE INDEX bdate_idx ON employee USING BTREE (bdate);
```

[60]:
```
++
||
++
++
```

[64]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  eid,
  ename,
  address,
  age (bdate)
FROM
  employee
WHERE
  bdate BETWEEN '1988-01-01' AND '1993-12-31';
```

[64]:
```
+-------------------------------------------------------------------------------
----------------------+
|                                                QUERY PLAN
|
+-------------------------------------------------------------------------------
----------------------+
| Seq Scan on employee  (cost=0.00..3.95 rows=15 width=54) (actual
time=0.015..0.031 rows=14 loops=1) |
|               Filter: ((bdate >= '1988-01-01'::date) AND (bdate <=
'1993-12-31'::date))             |
|                                     Rows Removed by Filter: 106
|
|                                        Buffers: shared hit=2
```

```
|
|                                                  Planning:
|
|                                            Buffers: shared hit=21
|
|                                       Memory: used=11kB  allocated=32kB
|
|                                           Planning Time: 0.165 ms
|
|                              Serialization: time=0.013 ms  output=1kB  format=text
|
|                                           Execution Time: 0.060 ms
|
+-------------------------------------------------------------------------------
----------------------+
```

In this case we can see the same performance for this query with and without an index. We believe a B-tree on bdate is the right index as this is a range query and we sort by bdate. The reason we see the same performance could be because the table is small in this case, so even if the index could narrow down rows, it's cheaper overall to read the whole table in one go or the other one that the filter is not very selective, that in proportion it is not the case but as the table is small it is cheap anyways.

Now we drop the index to continue with the other question:

[62]:
```
%%sql
DROP INDEX bdate_idx;
```

[62]:
```
++
||
++
++
```

b) What is the identifier and address of employees with a given name?

[66]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  eid,
  address
FROM
  employee
WHERE
  ename = 'Fiona Davis';
```

[66]:
```
+-------------------------------------------------------------------------------
--------------------+
|                                                 QUERY PLAN
|
```

```
+-------------------------------------------------------------------------------
--------------------+
| Seq Scan on employee  (cost=0.00..3.50 rows=1 width=25) (actual
time=0.017..0.026 rows=1 loops=1) |
|                            Filter: ((ename)::text = 'Fiona Davis'::text)
|
|                                     Rows Removed by Filter: 119
|
|                                        Buffers: shared hit=2
|
|                                             Planning:
|
|                                   Memory: used=9kB  allocated=16kB
|
|                                      Planning Time: 0.050 ms
|
|                        Serialization: time=0.002 ms  output=1kB  format=text
|
|                                      Execution Time: 0.043 ms
|
+-------------------------------------------------------------------------------
--------------------+
```

[70]:
```
%%sql
CREATE INDEX name_idx ON employee USING HASH (ename);
```

[70]:
```
++
||
++
++
```

[71]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  eid,
  address
FROM
  employee
WHERE
  ename = 'Fiona Davis';
```

[71]:
```
+-------------------------------------------------------------------------------
--------------------+
|                                                QUERY PLAN
|
+-------------------------------------------------------------------------------
--------------------+
```

```
| Seq Scan on employee  (cost=0.00..3.50 rows=1 width=25) (actual
time=0.022..0.030 rows=1 loops=1) |
|                              Filter: ((ename)::text = 'Fiona Davis'::text)
|
|                                       Rows Removed by Filter: 119
|
|                                          Buffers: shared hit=2
|
|                                               Planning:
|
|                                          Buffers: shared hit=16
|
|                                    Memory: used=10kB  allocated=24kB
|
|                                       Planning Time: 0.204 ms
|
|                          Serialization: time=0.003 ms  output=1kB  format=text
|
|                                       Execution Time: 0.058 ms
|
+-------------------------------------------------------------------------------
--------------------+
```

Here we have the same case in terms of execution time, as this question is more for reasoning and analize which indexes would fit better. Here the obvious index it is on (ename) and it could be b-tree or hash.

Now we drop the index to continue with the other question:

[69]:
```
%%sql
DROP INDEX name_idx;
```

[69]:
```
++
||
++
++
```

c) What is the maximum salary for employees?

[72]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  MAX(salary)
FROM
  employee;
```

[72]:
```
+-------------------------------------------------------------------------------
----------------------------+
|                                                  QUERY PLAN
```

```
|
+-------------------------------------------------------------------------------
----------------------------+
|           Aggregate  (cost=3.50..3.51 rows=1 width=32) (actual
time=0.032..0.033 rows=1 loops=1)           |
|                                              Buffers: shared hit=2
|
|   ->  Seq Scan on employee  (cost=0.00..3.20 rows=120 width=8) (actual
time=0.007..0.014 rows=120 loops=1) |
|                                                 Buffers: shared hit=2
|
|                                                   Planning:
|
|                                               Buffers: shared hit=8
|
|                                        Memory: used=27kB  allocated=32kB
|
|                                            Planning Time: 0.101 ms
|
|                              Serialization: time=0.001 ms  output=1kB
format=text                                |
|                                            Execution Time: 0.055 ms
|
+-------------------------------------------------------------------------------
----------------------------+
```

To answer this query it is clear that have salaries ordered would be the best option, so we create a b-tree index for salary.

[73]:
```
%%sql
CREATE INDEX salary_idx ON employee USING BTREE (salary);
```

[73]:
```
++
||
++
++
```

[74]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  MAX(salary)
FROM
  employee;
```

[74]:
```
+-------------------------------------------------------------------------------
-------------------------------------------------------------------+
|                                                                                QUERY PLAN
|
```

```
+-------------------------------------------------------------------------------
-------------------------------------------------------------------+
|                                Result  (cost=0.29..0.30 rows=1 width=32)
(actual time=0.024..0.024 rows=1 loops=1)                                    |
|                                                              Buffers: shared
hit=1 read=1                                                                 |
|                                                                InitPlan
1                                                                           |
|                                 ->  Limit  (cost=0.14..0.29 rows=1 width=8)
(actual time=0.020..0.021 rows=1 loops=1)                                    |
|                                                                 Buffers:
shared hit=1 read=1                                                          |
|           ->  Index Only Scan Backward using salary_idx on employee
(cost=0.14..17.89 rows=120 width=8) (actual time=0.019..0.019 rows=1 loops=1) |
|                                                                              Heap
Fetches: 1                                                                  |
|                                                                           Buffers:
shared hit=1 read=1                                                           |
|                                                                            Planning:
|
|                                                                 Buffers: shared
hit=15 read=1                                                                 |
|                                                              Memory: used=23kB
allocated=32kB                                                               |
|                                                                 Planning Time:
0.192 ms                                                                     |
|                                                  Serialization: time=0.002 ms
output=1kB  format=text                                                      |
|                                                                 Execution Time:
0.043 ms                                                                     |
+-------------------------------------------------------------------------------
-------------------------------------------------------------------+
```

We can see that now the query only does an index only scan as we only need the value of salary as the answer and the query benefits from values being ordered.

Now we drop the index to continue with the other question:

[80]:
```
%%sql
DROP INDEX salary_idx;
```

[80]:
```
++
||
++
++
```

d) What is the average salary of employees by age?

[82]:
```
%%sql
SELECT
  age,
  AVG(salary) AS average_salary
FROM
  employee
GROUP BY
  age;
```

[82]:
```
+-----+--------------------+
| age |    average_salary  |
+-----+--------------------+
|  74 |  46501.340000000000 |
|  29 | 112155.330000000000 |
|  71 |  36527.700000000000 |
|  68 |  84905.520000000000 |
|  34 |  62258.082500000000 |
|  51 |  32667.360000000000 |
|  70 | 115129.690000000000 |
|  52 |  97890.280000000000 |
|  35 |  66500.000000000000 |
|  45 |  63233.545000000000 |
|  39 |  66422.592500000000 |
|  36 |  57944.380000000000 |
|  69 |  93576.412000000000 |
|  31 |  75991.610000000000 |
|  50 |  47214.830000000000 |
|  60 |  32342.450000000000 |
|  14 | 105975.065000000000 |
|  66 |  55668.086666666667 |
|  22 |  69950.583333333333 |
|  59 |  43732.685000000000 |
|  13 |  55372.640000000000 |
|  65 | 118000.060000000000 |
|  62 | 103183.495000000000 |
|  75 |  97189.135000000000 |
|  73 |  90223.760000000000 |
|  11 |  70128.230000000000 |
|  44 |  93717.680000000000 |
|  42 |  79783.960000000000 |
|  41 |  98930.483333333333 |
|  40 |  86125.730000000000 |
|  46 |  78365.726666666667 |
|  43 |  93910.570000000000 |
|  53 |  46625.770000000000 |
|  32 |  54000.000000000000 |
|  9  |  64401.256000000000 |
```

```
|  7  |  99148.226666666667 |
|  38 |  81675.885000000000 |
|  15 | 103639.186666666667 |
|  26 | 113345.100000000000 |
|  48 |  77321.180000000000 |
|  72 |  69303.380000000000 |
|  57 | 105539.000000000000 |
|  24 |  91762.715000000000 |
|  25 | 105828.160000000000 |
|  30 |  77586.500000000000 |
|  49 |  45878.150000000000 |
|  47 |  79320.540000000000 |
|  17 |  78794.140000000000 |
|  20 |  70011.672500000000 |
|  37 |  93512.097500000000 |
|  28 |  64030.053333333333 |
|  33 |  78949.610000000000 |
|  18 |  98583.360000000000 |
|  64 |  44144.090000000000 |
|  55 |  70418.070000000000 |
|  27 |  77010.270000000000 |
|  56 | 104497.356666666667 |
|  58 |  64443.560000000000 |
+-----+---------------------+
```

[93]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  age,
  AVG(salary) AS average_salary
FROM
  employee
GROUP BY
  age;
```

[93]:
```
+-------------------------------------------------------------------------------
------------------------------+
|                                                    QUERY PLAN
|
+-------------------------------------------------------------------------------
------------------------------+
|         HashAggregate  (cost=3.80..4.52 rows=58 width=36) (actual
time=0.069..0.092 rows=58 loops=1)        |
|                                                Group Key: age
|
|                                           Batches: 1  Memory Usage: 56kB
|
```

```
|                                                  Buffers: shared hit=2
|
|   ->  Seq Scan on employee  (cost=0.00..3.20 rows=120 width=12) (actual
time=0.009..0.017 rows=120 loops=1) |
|                                                    Buffers: shared hit=2
|
|                                                      Planning:
|
|                                          Buffers: shared hit=5 dirtied=2
|
|                                         Memory: used=14kB  allocated=40kB
|
|                                             Planning Time: 0.108 ms
|
|                               Serialization: time=0.011 ms  output=2kB
format=text                                 |
|                                             Execution Time: 0.136 ms
|
+-------------------------------------------------------------------------
------------------------------+
```

For this query in order to try to improve the performance the options are an index on age as we try to avoid the need of sorting or a composite index for (age, salary) to go for a index only scan.

[97]:
```
%%sql

CREATE INDEX age_salary_idx ON employee USING BTREE (age, salary);
```

[97]:
```
++
||
++
++
```

[98]:
```
%%sql
EXPLAIN (ANALYZE, BUFFERS, MEMORY, SERIALIZE)
SELECT
  age,
  AVG(salary) AS average_salary
FROM
  employee
GROUP BY
  age;
```

[98]:
```
+-------------------------------------------------------------------------
------------------------------+
|                                                    QUERY PLAN
|
+-------------------------------------------------------------------------
```

```
------------------------------+
|         HashAggregate  (cost=3.80..4.52 rows=58 width=36) (actual
time=0.070..0.093 rows=58 loops=1)        |
|                                                  Group Key: age
|
|                                           Batches: 1  Memory Usage: 56kB
|
|                                               Buffers: shared hit=2
|
|   ->  Seq Scan on employee  (cost=0.00..3.20 rows=120 width=12) (actual
time=0.009..0.018 rows=120 loops=1) |
|                                                    Buffers: shared hit=2
|
|                                                      Planning:
|
|                                           Buffers: shared hit=19 read=1
|
|                                         Memory: used=16kB  allocated=40kB
|
|                                            Planning Time: 0.218 ms
|
|                              Serialization: time=0.011 ms  output=2kB
format=text                                 |
|                                            Execution Time: 0.139 ms
|
+-------------------------------------------------------------------------------
------------------------------+
```

Here we can observe there is no impact on performance, but could be for the small size of our table.

We drop the index:

[96]:
```
%%sql
DROP INDEX age_idx;
```

[96]:
```
++
||
++
++
```