Carnegie Mellon University
Department of Computer Science
15-415/615 - Database Applications
C. Faloutsos, A. Pavlo , Spring 2015

Homework 5 (by Elomar Souza) - Solutions

Due: **hard and e-copy**, at **1:30pm, Mar. 17th, 2015**

IMPORTANT – what to hand in:

1. **Hard copy** of your answers (including SQL statements and database output, whenever stated by the question), in **class** at 1:30pm, on Thursday, Mar. 17th, 2015.
2. **tar-file**, with your queries, via *Blackboard*, before the due date (Mar. 17th, 2015, 1:30pm). See page 2 ('*What to deliver*'), for details.

Reminders

- *Plagiarism*: Homework is to be done **individually**.
- *Typeset* all of your answers. Handwritten work will get zero points.
- *Late homeworks*: Standard policy: email (a) to all TAs, (b) with the subject line exactly `15-415 Homework Submission (HW 5)`, and (c) the count of slip-days you are using.

For your information:

- Graded out of **100** points
- **4** questions total
- Expected effort: ≈ 4-9h (30min-1h to set up PostgreSQL; 1-2h per question).

Revision : 2015/03/23 10:11

Question	Points	Score
EXPLAIN and ANALYZE	25	
Using indexes	25	
Joins	25	
Optimizing Joins	25	
Total:	100	

Preliminaries

Database set-up

Before starting the homework, follow the instructions for importing the data you'll work with, available at `http://www.cs.cmu.edu/~christos/courses/dbms.S15/hws/HW5/postgresql-setup.html`.

For your convenience, a file with all the queries used in the homework is available at `http://www.cs.cmu.edu/~christos/courses/dbms.S15/hws/HW5/hw5-queries.sql`. You can copy the queries from the file to your `psql` session, to avoid copying-from-pdf errors.

What to deliver: Check-list

The two items, as mentioned on the front page:

1. **Hard copy**:
   - What: hard copy of your answers (including **SQL queries**, and **database output**, whenever stated by the question)
   - When: Mar. 17th, 2015, 1:30pm
   - Where: in class

   As before, please separate your answers for each question, providing on all answers the usual information (`course#`, `Homework#`, `Question#`, `Andrew ID`, `name`).
2. **tar-file**:
   - What: A `tar` file (`<your-andrew-id>.tar`) with all your code - no need to include the answers.
   - When: Mar. 17th, 2015, 1:30pm
   - Where: on *Blackboard*, under 'Assignments'/'Homework #5'

   Please use the template provided at `www.cs.cmu.edu/~christos/courses/dbms.S15/hws/HW5/hw5.tar` – just insert your SQL query to each place-holder `.sql` file.

Introduction

The purpose of this homework is to make you familiar with the query execution engine of PostgreSQL. In particular, you will have to analyze a few queries, and answer questions regarding their performance when turning different knobs of the execution engine.

In order to answer the questions, you might find the following documentation links useful:

- Documentation of `EXPLAIN ANALYZE`:
  http://www.postgresql.org/docs/8.4/static/sql-explain.html.
- Making sense of the `EXPLAIN ANALYZE` output:
  http://www.postgresql.org/docs/8.4/static/performance-tips.html.
- PostgreSQL query planner documentation:
  http://www.postgresql.org/docs/8.4/static/runtime-config-query.html.

Homework 5 continues…

- How to create an index:
  http://www.postgresql.org/docs/8.4/static/sql-createindex.html.
- The system table `pg_class`:
  http://www.postgresql.org/docs/8.4/static/catalog-pg-class.html.

When creating an index, do not alter PostgreSQL default options (type B-Tree, unclustered).

Database schema

The database you'll use in this exercise tracks users behavior on an ecommerce website. As typical for both large (e.g. Amazon) and small e-commerce portals, it tracks **user sessions**, **clicks**, and **purchases**.

The `sessions` table has data on each particular visit to the website. Its fields are: `sessid` (int, primary key), `zipcode` (int), `sex` (char), `agegroup` (varchar), and `browser` (varchar). For example, the tuple

(1, 90755, 'M', "5-19", "Chrome")

means that the #1 session happened at zipcode 90755, and was a visit from a Male, age between 5 and 19, using Chrome.

The `clicks` table tracks the time, item, and category of each click made during a particular session. Its fields are: `sessid` (int), `created` (timestamp), `itemid` (int), and `catid` (int).

The `purchases` table tracks the item, price, and quantity of each purchase made during a particular session. Its fields are: `sessid` (int), `created` (timestamp), `itemid` (int), `price` (numeric), and `quantity` (int).

Homework 5 continues. . .

Question 1: EXPLAIN and ANALYZE .... [25 points]

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

Graded by: Vinay

In this question, you'll learn how to use `EXPLAIN` and `ANALYZE` to understand the impact of indexes on simple queries.

Answer the questions based on the query below:

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999;
```

(a) [**3 points**] Provide the execution plan of the query. Provide the SQL statement you used and its output.

Solution:

```
EXPLAIN ANALYZE SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999;
```

QUERY PLAN

```
Seq Scan on clicks (cost=0.00..139276.61 rows=181486 width=32)
(actual time=13.490..3467.981 rows=155493 loops=1)
Filter: ((itemid >= 214800000) AND (itemid <= 214819999))
Total runtime: 3476.465 ms
```

(b) Based on the execution plan:

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 139276.61 *the actual number may change, correct answer just has to match cost provided in the execution plan.*

ii. [**1 point**] What was the total runtime? (in ms)

Solution: 3476.465 *the actual number may change, correct answer just has to match total runtime provided in the execution plan.*

(c) [**3 points**] Create an index on the attribute `itemid` on the table `clicks`.[1] Provide the SQL statement.

Solution:

```
CREATE INDEX clicks_itemid ON clicks(itemid);
```

(d) [**3 points**] Provide the new execution plan of the query, with the index in place.

Solution:

```
EXPLAIN ANALYZE SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999;
```

[1]Using the default PostgreSQL options.

Question 1 continues. . .

[redacted]

(e) Based on the new execution plan:

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 52390.45

ii. [**1 point**] What was the total runtime? (in ms)

Solution: 666.751

iii. [**1 point**] What was the estimated number of tuples to be output?

Solution: 18146

iv. [**1 point**] What was the actual number of tuples to be output?

Solution: 155493

(f) Use the table `pg_class` to answer the following questions:

i. [**2 points**] How many pages are used to store the table `clicks`?

Solution:

```
SELECT relpages FROM pg_class
WHERE relname = 'clicks';
```

relpages
45815

ii. [**2 points**] How many tuples are in the table `clicks`, according to `pg_class`?

Solution:

```
SELECT reltuples FROM pg_class
WHERE relname = 'clicks';
```

reltuples
6.23079e+06

iii. [**2 points**] Is that number always equal to the result of running `SELECT COUNT(*) FROM clicks`?

Question 1 continues…

Solution: No.

iv. [**2 points**] How many pages are used to store the index you created?

Solution:

```
SELECT relpages FROM pg_class
WHERE relname = 'clicks_itemid';
```

relpages
17087

v. [**2 points**] How many tuples are in the index?

Solution:

```
SELECT reltuples FROM pg_class
WHERE relname = 'clicks_itemid';
```

reltuples
6.23079e+06

Question 2: Using indexes . . . . [25 points]

Graded by: Jiayu Liu

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

In this question, you'll learn the conditions under which indexes may or may not be used by the query optimizer.

Make sure you have an index on the column `clicks.itemid`, created in Q1-(c).

(a) For each of those queries, answer (yes) if the index you created for Q1, item (c) was used or (not) if it wasn't:

i. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999;
```

Solution: yes

ii. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid > 214800000;
```

Solution: no

iii. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
AND created > '2014-04-01';
```

Solution: yes

iv. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
AND created IS NULL;
```

Solution: yes

v. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
OR created > '2014-04-01';
```

Solution: no

vi. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid = 214507226;
```

Solution: yes

vii. [**1 point**]

Question 2 continues. . .

```
SELECT * FROM clicks
WHERE itemid != 214507226;
```

Solution: no

(b) [**1 point**] Create an index on the column `created` on the table `clicks`.[2] Provide the SQL command.

Solution:

```
CREATE INDEX clicks_created ON clicks(created);
```

(c) For each of those queries, answer (1) if only the index on `itemid` was used, (2) if only the index on `created` was used, (3) if both were used, or (4) if neither one of the indexes were used:

i. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
AND created > '2014-04-01';
```

Solution: (1) only clicks_itemid

ii. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
AND created BETWEEN '2014-04-01' AND '2014-04-02';
```

Solution: (3) both

iii. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid BETWEEN 214800000 AND 214819999
OR created BETWEEN '2014-04-01' AND '2014-04-02';
```

Solution: (3) both

iv. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid < 214819999
OR created BETWEEN '2014-04-01' AND '2014-04-02';
```

Solution: (4) neither

v. [**1 point**]

```
SELECT * FROM clicks
WHERE itemid < 214819999
AND created BETWEEN '2014-04-01' AND '2014-04-02';
```

[2]Using the default PostgreSQL options.

Question 2 continues...

Solution: (2) only clicks_created

(d) For the query `SELECT * FROM clicks WHERE created BETWEEN '2014-04-01' AND '2014-04-02';` answer the following questions:

i. [**1 point**] Was the index on `created` used?

Solution: yes

ii. [**1 point**] What percentage of the total records in the table `clicks` was returned?

Solution: ~3% (220872 out of 6230791 records)

(e) For the query `SELECT * FROM clicks WHERE created BETWEEN '2014-04-01' AND '2015-12-31';` answer the following questions:

i. [**1 point**] Was the index on `created` used?

Solution: no

ii. [**1 point**] What percentage of the total records in the table `clicks` was returned?

Solution: 100%

(f) For the query `SELECT * FROM clicks WHERE itemid BETWEEN 214800000 AND 214819999 ORDER BY itemid;`, answer the following questions:

i. [**1 point**] Which method was used for sorting?

Solution: external merge

ii. [**1 point**] Where did the sorting happen – memory or disk?

Solution: disk

iii. [**1 point**] How much space was used for sorting?

Solution: 6384kB

iv. [**1 point**] What was the total runtime? (in ms)

Solution: 939.996 ms

(g) Increase PostgreSQL working memory with the command `SET work_mem = '25MB';`. For the same query as above, answer the following questions:

i. [**1 point**] Which method was used for sorting?

Solution: quicksort

ii. [**1 point**] Where did the sorting happen – memory or disk?

Solution: memory

iii. [**1 point**] How much space was used for sorting?

Question 2 continues. . .

Solution: 18292kB

iv. [**1 point**] What was the total runtime? (in ms)

Solution: 268.737 ms

(h) [**0 points**] Execute the command `RESET work_mem;` to get PostgreSQL working memory back to the default value (or your answers for the next questions will turn out wrong).

Question 3: Joins ....[25 points]

Graded by: Hong Bin Shim

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

In this question, you'll learn more about the different methods used by PostgreSQL for executing joins.

Make sure you reset `work_mem` to its default value, as per Q2-(h).

Answer the questions based on the query below:

```
SELECT sessions.*, purchases.*
FROM sessions,  purchases
WHERE sessions.sessid = purchases.sessid;
```

(a) Provide the query plan for the query above, and answer the following questions:

Solution:

```
QUERY PLAN
Hash Join (cost=58767.00..79813.48 rows=187316 width=60)
(actual time=623.732..2785.586 rows=187316 loops=1)
Hash Cond: (purchases.sessid = sessions.sessid)
-> Seq Scan on purchases (cost=0.00..3434.16 rows=187316 width=34)
(actual time=0.017..18.695 rows=187316 loops=1)
-> Hash (cost=27829.00..27829.00 rows=1600000 width=26)
(actual time=609.821..609.821 rows=1600000 loops=1)
-> Seq Scan on sessions (cost=0.00..27829.00 rows=1600000 width=26)
(actual time=0.008..152.561 rows=1600000 loops=1)
Total runtime: 2792.943 ms
```

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

Solution: hash join

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 79813.48

iii. [**1 point**] What was the total runtime? (in ms)

Solution: 2792.943ms

(b) [**5 points**] Create an index that improves the total runtime of this query.[3]. Provide the SQL statement.

Solution:

```
CREATE INDEX purchases_sessid on purchases(sessid);
```

[3]Using the default PostgreSQL options.

Question 3 continues. . .

(c) Provide the new query plan with the index in place, and answer the following questions:

Solution:

QUERY PLAN

```
Merge Join (cost=5.74..66636.65 rows=187316 width=60)
(actual time=0.049..432.424 rows=187316 loops=1)
Merge Cond: (sessions.sessid = purchases.sessid)
-> Index Scan using sessions_pkey on sessions
(cost=0.00..53392.43 rows=1600000 width=26)
(actual time=0.027..220.930 rows=1599994 loops=1)
-> Index Scan using purchases_sessid on purchases
(cost=0.00..6908.52 rows=187316 width=34)
(actual time=0.017..42.720 rows=187316 loops=1)
Total runtime: 440.720 ms
```

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

Solution: merge

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 66636.65

iii. [**1 point**] What was the total runtime? (in ms)

Solution: 440.720ms

(d) Execute the command `SET enable_mergejoin = false;` to disable merge joins. Provide the new query plan, and answer the following questions:

Solution:

QUERY PLAN

```
Hash Join (cost=58767.00..79813.48 rows=187316 width=60)
(actual time=578.585..2804.627 rows=187316 loops=1)
Hash Cond: (purchases.sessid = sessions.sessid)
-> Seq Scan on purchases (cost=0.00..3434.16 rows=187316 width=34)
(actual time=0.013..17.356 rows=187316 loops=1)
-> Hash (cost=27829.00..27829.00 rows=1600000 width=26)
(actual time=563.244..563.244 rows=1600000 loops=1)
-> Seq Scan on sessions (cost=0.00..27829.00 rows=1600000 width=26)
(actual time=0.005..145.207 rows=1600000 loops=1)
Total runtime: 2811.931 ms
```

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

Solution: hash

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 79813.48

iii. [**1 point**] What was the total runtime? (in ms)

Question 3 continues. . .

Solution: 2811.931ms

(e) Execute the command `SET enable_hashjoin = false;` to disable hash joins. Provide the new query plan, and answer the following questions:

Solution:

QUERY PLAN

```
Nested Loop (cost=0.00..257367.33 rows=187316 width=60)
(actual time=0.041..404.355 rows=187316 loops=1)
-> Seq Scan on purchases (cost=0.00..3434.16 rows=187316 width=34)
(actual time=0.013..18.294 rows=187316 loops=1)
-> Index Scan using sessions_pkey on sessions (cost=0.00..1.34 rows=1 width=26)
(actual time=0.002..0.002 rows=1 loops=187316)
Index Cond: (sessions.sessid = purchases.sessid)
Total runtime: 414.467 ms
```

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

Solution: nested loop

ii. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 257367.33

iii. [**1 point**] What was the total runtime? (in ms)

Solution: 414.467ms

(f) Execute the command `SET enable_indexscan = false; SET enable_bitmapscan = false;` to disable index scans. Provide the new query plan, and answer the following questions:

Solution:

QUERY PLAN

```
Nested Loop (cost=40367.00..8792282209.16 rows=187316 width=60)
Join Filter: (sessions.sessid = purchases.sessid)
-> Seq Scan on purchases (cost=0.00..3434.16 rows=187316 width=34)
-> Materialize (cost=40367.00..67305.00 rows=1600000 width=26)
-> Seq Scan on sessions (cost=0.00..27829.00 rows=1600000 width=26)
```

i. [**2 points**] Which join method was used – nested loop, merge, or hash?

Solution: nested loop

ii. [**2 points**] What was the estimated cost of the query? (Feel free to find out the actual time, but be aware that it takes a while.)

Solution: 8792282209.16

(g) [**0 points**] Execute these commands to re-enable the different joins (or your answers for the next questions will turn out wrong):

```
RESET enable_mergejoin;
RESET enable_hashjoin;
```

Question 3 continues…

```
    RESET enable_indexscan;
    RESET enable_bitmapscan;
```

Homework 5 continues. . .

Question 4: Optimizing Joins . . . . [25 points]

Graded by: Elomar

On separate page, with '[course-id] [hw#] [question#] [andrew-id] [your-name]'

In this question you will optimize a query with multiple joins.

Make sure you reset the optimizer to the default settings, as per Q3-(g).

Answer the questions based on the query below:

```
SELECT c.itemid, s.browser, s.agegroup,
MAX(c.created) - MIN(c.created) as date_span
FROM sessions s, clicks c
WHERE s.sessid = c.sessid
AND (s.zipcode BETWEEN 15000 AND 16000
  OR s.agegroup = (SELECT MAX(agegroup) FROM sessions))
GROUP BY c.itemid, s.browser, s.agegroup
ORDER BY c.itemid, s.browser, s.agegroup;
```

(a) [**1 point**] Destroy any indexes created on the previous questions. Provide the SQL commands.

Solution:

```
DROP INDEX clicks_created;
DROP INDEX clicks_itemid;
DROP INDEX purchases_sessid;
```

(b) Provide the query plan for the query above and answer the following questions:

Question 4 continues. . .

Solution:

```
QUERY PLAN
GroupAggregate (cost=519638.19..541908.75 rows=173162 width=28)

(actual time=12086.282..14076.026 rows=79761 loops=1)
InitPlan 1 (returns $0)
-> Aggregate (cost=31829.00..31829.01 rows=1 width=5)

(actual time=422.703..422.703 rows=1 loops=1)
-> Seq Scan on sessions (cost=0.00..27829.00 rows=1600000 width=5)

(actual time=0.002..120.622 rows=1600000 loops=1)
-> Sort (cost=487809.17..491015.88 rows=1282682 width=28)

(actual time=12086.251..13860.541 rows=821542 loops=1)
Sort Key: c.itemid, s.browser, s.agegroup
Sort Method: external merge Disk: 34216kB
-> Hash Join (cost=45876.24..296293.39 rows=1282682 width=28)

(actual time=848.701..7947.015 rows=821542 loops=1)
Hash Cond: (c.sessid = s.sessid)
-> Seq Scan on clicks c (cost=0.00..108122.91 rows=6230791 width=24)

(actual time=0.008..760.595 rows=6230791 loops=1)
-> Hash (cost=39829.00..39829.00 rows=329379 width=20)

(actual time=832.929..832.929 rows=210346 loops=1)
-> Seq Scan on sessions s (cost=0.00..39829.00 rows=329379 width=20)

(actual time=422.726..760.814 rows=210346 loops=1)
Filter: (((zipcode ¿= 15000) AND (zipcode ¡= 16000)) OR ((agegroup)::text = $0)
Total runtime: 14630.339 ms
```

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 541908.75

ii. [**1 point**] What was the total runtime? (in ms)

Solution: 14630.339ms

Grading info: -1 for not providing query plan.

(c) [**20 points**] Optimize the total runtime of the query. You're allowed to (i) use any techniques discussed on this homework, and (ii) tweak the query itself, without modifying the results. The query will be ran against the **same database dump** provided. Provide a list of the optimizations you made, all the SQL statements to implement them, and the final query used. *Hint:* You should see performance gains

Question 4 continues. . .

of ˜10 times on your assigned GHC machine.

Solution: Two solutions were accepted for full points: one that optimized cost, and one that optimized runtime.
Solution #1:

```
CREATE INDEX sessions_agegroup ON sessions(agegroup);
CREATE INDEX sessions_zipcode ON sessions(zipcode);
SET work_mem = '25MB';
```

Solution: Solution #2:

```
CREATE INDEX sessions_agegroup ON sessions(agegroup);
CREATE INDEX clicks_sessid ON clicks(sessid);
SET work_mem = '25MB';
SET enable_hashjoin = false; SET enable_mergejoin = false;
```

Grading info: -5 points for missing one of the steps.

Grading info: -5 points for creating indexes that did not get used.

(d) Provide the new query plan with your optimizations in place and answer the following questions:

i. [**1 point**] What was the estimated cost of the query? (in arbitrary units)

Solution: 275178.36 *or* 2782503.64

ii. [**1 point**] What was the total runtime? (in ms)

Solution: 2294.062ms *or* 1550.938 ms
Grading info: -1 for not providing query plan.

Solution:

```
QUERY PLAN
Sort (cost=274756.29..275178.36 rows=168828 width=28)
(actual time=2285.589..2289.289 rows=61613 loops=1)
Sort Key: c.itemid, s.browser, s.agegroup
Sort Method: quicksort Memory: 6396kB
InitPlan 2 (returns $1)
-> Result (cost=0.05..0.06 rows=1 width=0)
(actual time=0.020..0.020 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Limit (cost=0.00..0.05 rows=1 width=5)
(actual time=0.017..0.018 rows=1 loops=1)
-> Index Scan Backward using sessions_agegroup on sessions (cost=0.00..86646.17 rows=1600000 width=5)
(actual time=0.017..0.017 rows=1 loops=1)
Filter: ((agegroup)::text IS NOT NULL)
```

Question 4 continues. . .

```
-> HashAggregate (cost=257143.08..260097.57 rows=168828 width=28)
(actual time=2225.525..2245.117 rows=61613 loops=1)
-> Hash Join (cost=27510.40..241561.35 rows=1246538 width=28)
(actual time=126.618..1847.209 rows=777514 loops=1)
Hash Cond: (c.sessid = s.sessid)
-> Seq Scan on clicks c (cost=0.00..108123.23 rows=6230823 width=24)
(actual time=0.013..443.655 rows=6230791 loops=1)
-> Hash (cost=23509.20..23509.20 rows=320096 width=20)
(actual time=126.503..126.503 rows=199074 loops=1)
-> Bitmap Heap Scan on sessions s (cost=6078.10..23509.20 rows=320096 width=20)
(actual time=19.852..88.199 rows=199074 loops=1)
Recheck Cond: (((zipcode >= 15000) AND (zipcode ¡= 15010)) OR ((agegroup)::text = $1))
-> BitmapOr (cost=6078.10..6078.10 rows=320120 width=0)
(actual time=18.110..18.110 rows=0 loops=1)
-> Bitmap Index Scan on sessions_zipcode (cost=0.00..5.63 rows=120 width=0)
(actual time=0.081..0.081 rows=474 loops=1)
Index Cond: ((zipcode >= 15000) AND (zipcode ¡= 15010))
-> Bitmap Index Scan on sessions_agegroup (cost=0.00..5912.43 rows=320000 width=0)
(actual time=18.027..18.027 rows=198658 loops=1)
Index Cond: ((agegroup)::text = $1)
Total runtime: 2294.062 ms
```

QUERY PLAN

```
Sort (cost=2782081.57..2782503.64 rows=168828 width=28)
(actual time=1543.806..1547.571 rows=61613 loops=1)
Sort Key: c.itemid, s.browser, s.agegroup
Sort Method: quicksort Memory: 6396kB
InitPlan 2 (returns $1)
-> Result (cost=0.05..0.06 rows=1 width=0)
(actual time=0.021..0.021 rows=1 loops=1)
InitPlan 1 (returns $0)
-> Limit (cost=0.00..0.05 rows=1 width=5)
(actual time=0.018..0.018 rows=1 loops=1)
-> Index Scan Backward using sessions_agegroup on sessions (cost=0.00..86646.17 rows=1600000 width=5)
(actual time=0.018..0.018 rows=1 loops=1)
Filter: ((agegroup)::text IS NOT NULL)
-> HashAggregate (cost=2764468.36..2767422.85 rows=168828 width=28)
(actual time=1483.940..1503.523 rows=61613 loops=1)
-> Nested Loop (cost=0.00..2748886.64 rows=1246538 width=28)
(actual time=0.061..1084.167 rows=777514 loops=1)
-> Seq Scan on sessions s (cost=0.00..39829.00 rows=320096 width=20)
(actual time=0.044..346.021 rows=199074 loops=1)
Filter: (((zipcode >= 15000) AND (zipcode ¡= 15010)) OR ((agegroup)::text = $1))
-> Index Scan using clicks_sessid on clicks c (cost=0.00..8.33 rows=11 width=24)
(actual time=0.002..0.003 rows=4 loops=199074)
Index Cond: (c.sessid = s.sessid)
Total runtime: 1550.938 ms
```