A TEMPORAL BENCHMARK IN BENCHBASE

Paul A. Jungwirth

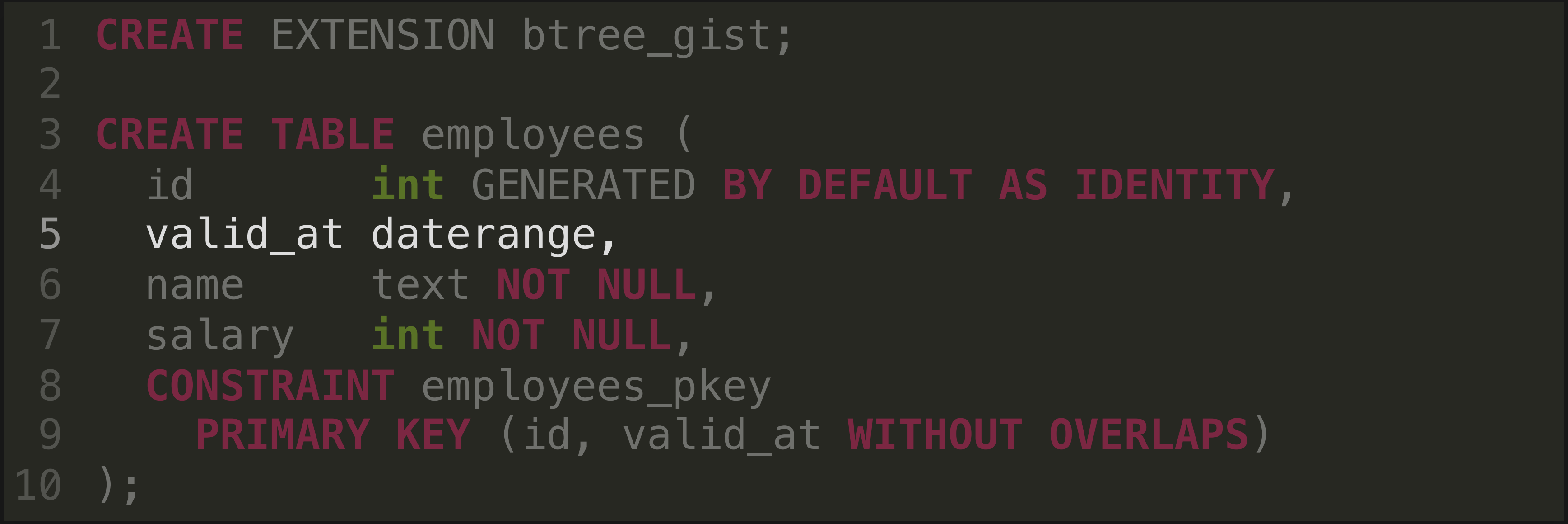
PGConf.dev 2025

14 May 2025

TEMPORAL TABLES

```sql
CREATE EXTENSION btree_gist;

CREATE TABLE employees (
  id       int GENERATED BY DEFAULT AS IDENTITY,
  valid_at daterange,
  name     text NOT NULL,
  salary   int NOT NULL,
  CONSTRAINT employees_pkey
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
);
```

TEMPORAL TABLES

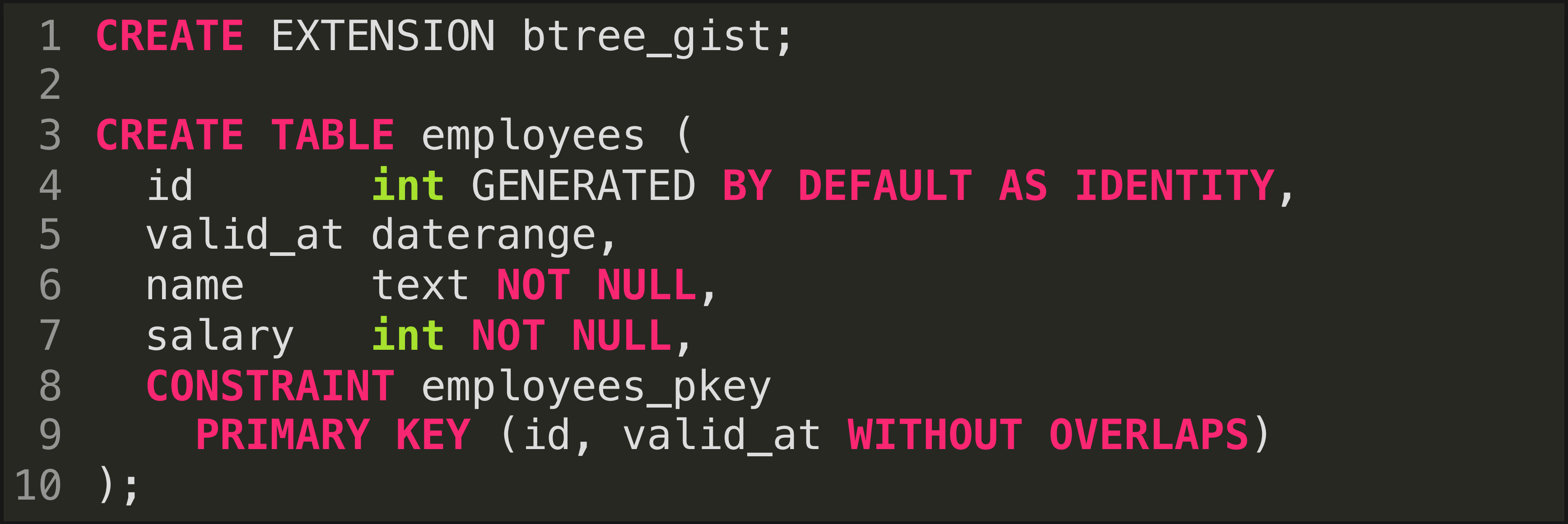


```sql
CREATE EXTENSION btree_gist;

CREATE TABLE employees (
  id        int GENERATED BY DEFAULT AS IDENTITY,
  valid_at daterange,
  name      text NOT NULL,
  salary    int NOT NULL,
  CONSTRAINT employees_pkey
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
);
```

TEMPORAL TABLES

```sql
CREATE EXTENSION btree_gist;

CREATE TABLE employees (
  id       int GENERATED BY DEFAULT AS IDENTITY,
  valid_at daterange,
  name     text NOT NULL,
  salary   int NOT NULL,
  CONSTRAINT employees_pkey
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
);
```

TEMPORAL TABLES



```sql
CREATE EXTENSION btree_gist;

CREATE TABLE employees (
  id       int GENERATED BY DEFAULT AS IDENTITY,
  valid_at daterange,
  name     text NOT NULL,
  salary   int NOT NULL,
  CONSTRAINT employees_pkey
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
);
```

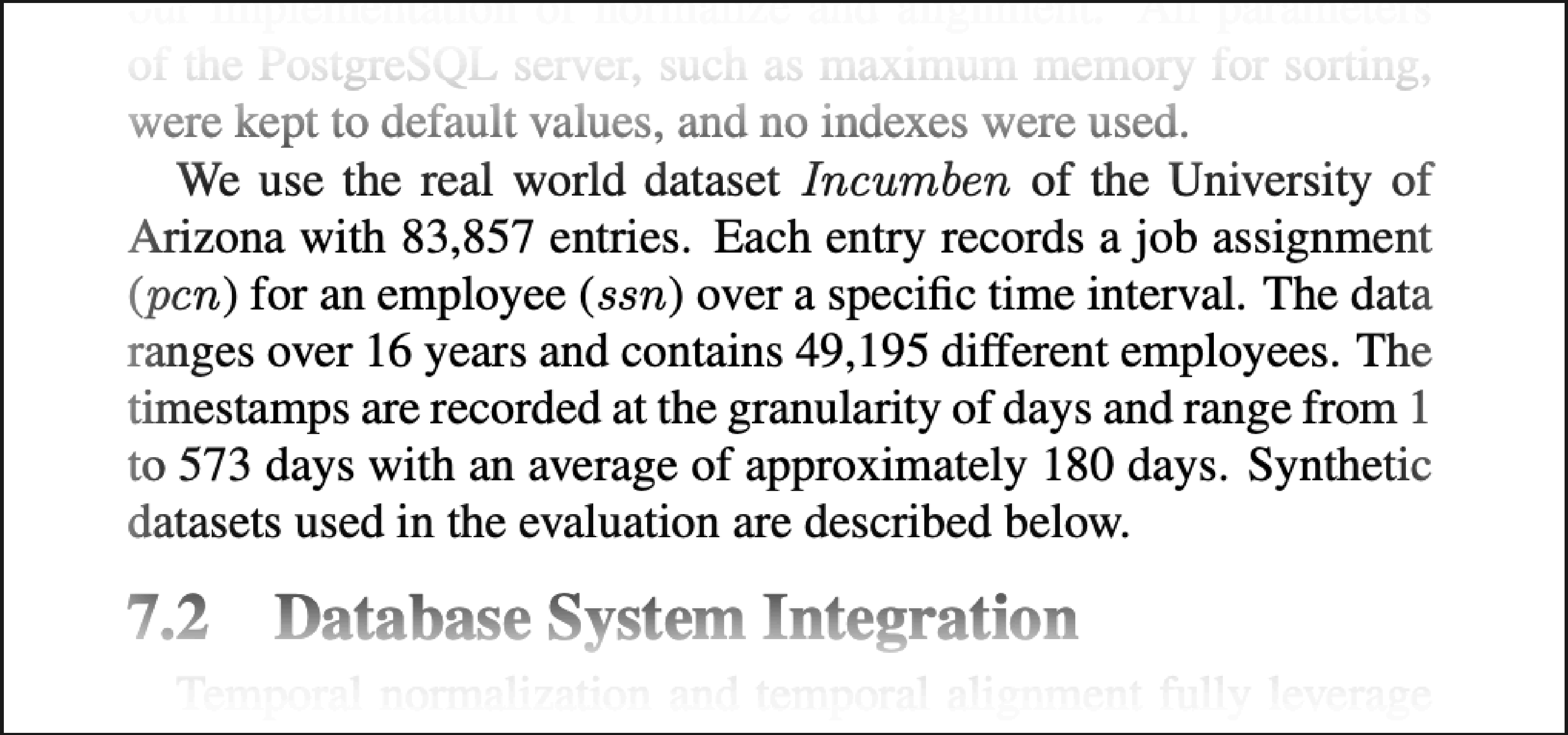
PAST WORK

```
EMPLOYEES(SSN, LAST_NAME, FIRST_NAME, ANNUAL_SALARY)

POSITIONS(PCN, JOB_TITLE_CODE1)

INCUMBENTS(SSN, PCN, START_DATE, END_DATE)

JOB_TITLES(JOB_TITLE_CODE, JOB_TITLE)
```

Richard Snodgrass,
University of Arizona employee database

PAST WORK

PAST WORK



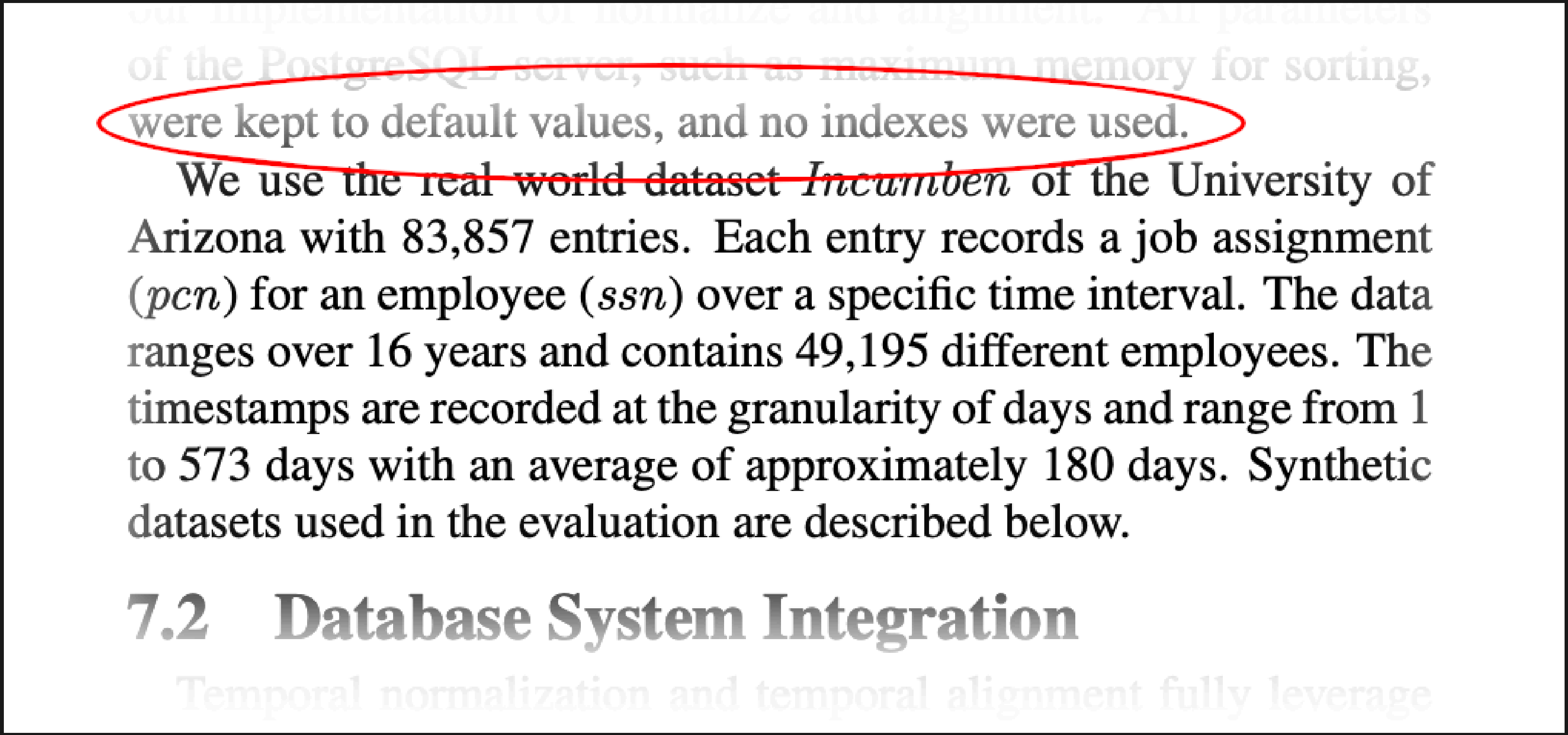
our implementation of normalize and alignment. All parameters of the PostgreSQL server, such as maximum memory for sorting, were kept to default values, and no indexes were used.

We use the real world dataset *Incumben* of the University of Arizona with 83,857 entries. Each entry records a job assignment (pcn) for an employee (ssn) over a specific time interval. The data ranges over 16 years and contains 49,195 different employees. The timestamps are recorded at the granularity of days and range from 1 to 573 days with an average of approximately 180 days. Synthetic datasets used in the evaluation are described below.

7.2 Database System Integration

Temporal normalization and temporal alignment fully leverage

PAST WORK



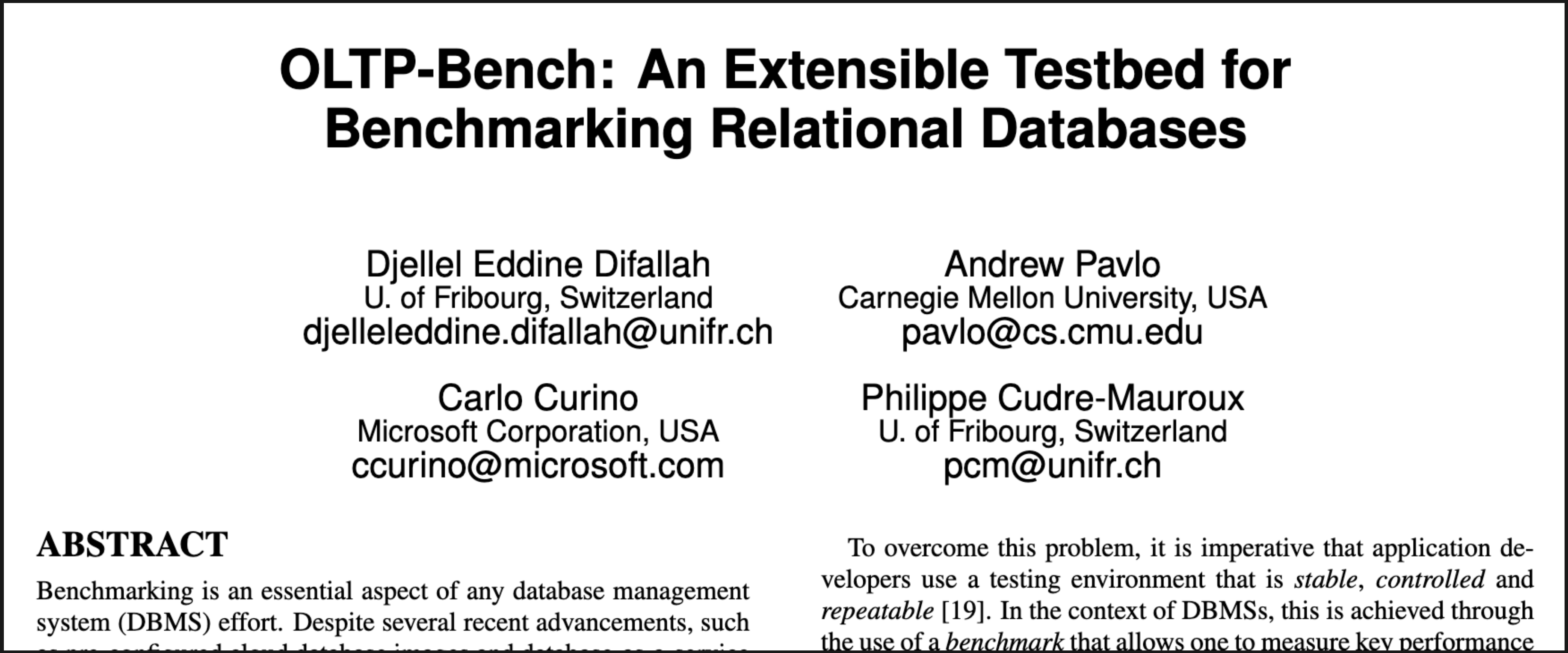
our implementation of normalize and alignment. All parameters of the PostgreSQL server, such as maximum memory for sorting, were kept to default values, and no indexes were used.

We use the real world dataset *Incumben* of the University of Arizona with 83,857 entries. Each entry records a job assignment (pcn) for an employee (ssn) over a specific time interval. The data ranges over 16 years and contains 49,195 different employees. The timestamps are recorded at the granularity of days and range from 1 to 573 days with an average of approximately 180 days. Synthetic datasets used in the evaluation are described below.

7.2 Database System Integration

Temporal normalization and temporal alignment fully leverage

BENCHBASE
NÉE OLTP-BENCH



OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases

Djellel Eddine Difallah
U. of Fribourg, Switzerland
djelleleddine.difallah@unifr.ch

Andrew Pavlo
Carnegie Mellon University, USA
pavlo@cs.cmu.edu

Carlo Curino
Microsoft Corporation, USA
ccurino@microsoft.com

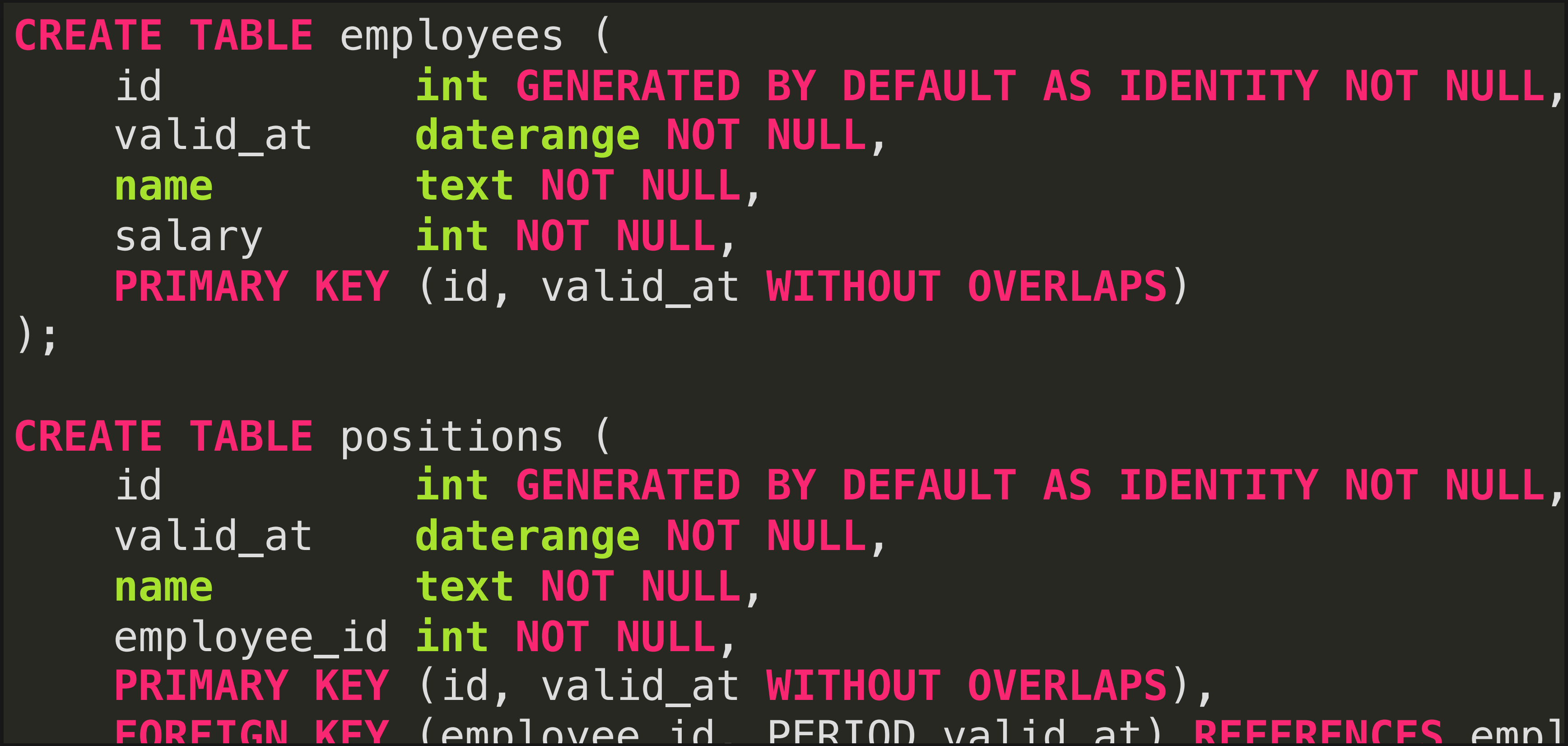
Philippe Cudre-Mauroux
U. of Fribourg, Switzerland
pcm@unifr.ch

ABSTRACT

Benchmarking is an essential aspect of any database management system (DBMS) effort. Despite several recent advancements, such as pre-configured cloud database images and database as a service

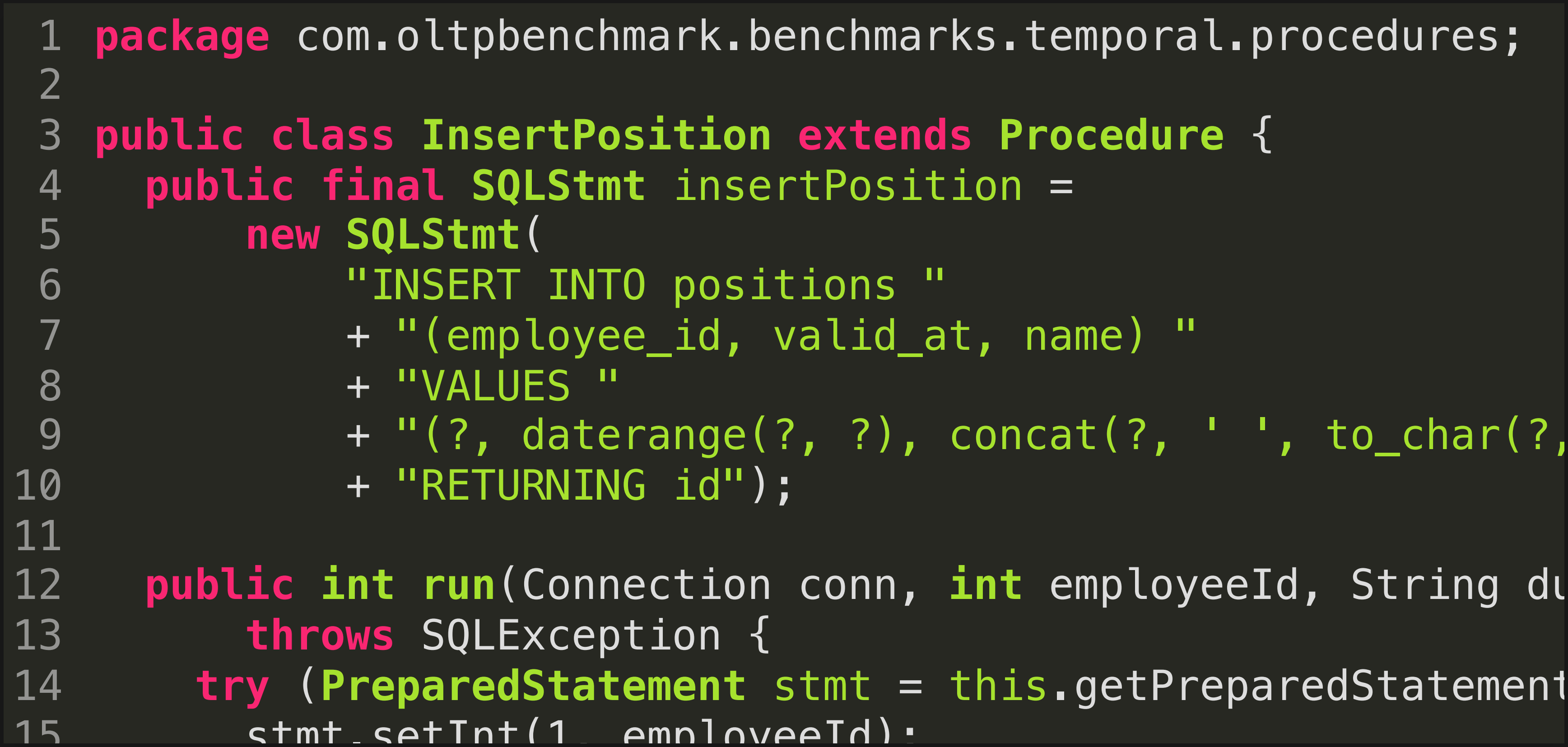
To overcome this problem, it is imperative that application developers use a testing environment that is *stable*, *controlled* and *repeatable* [19]. In the context of DBMSs, this is achieved through the use of a *benchmark* that allows one to measure key performance

DDL



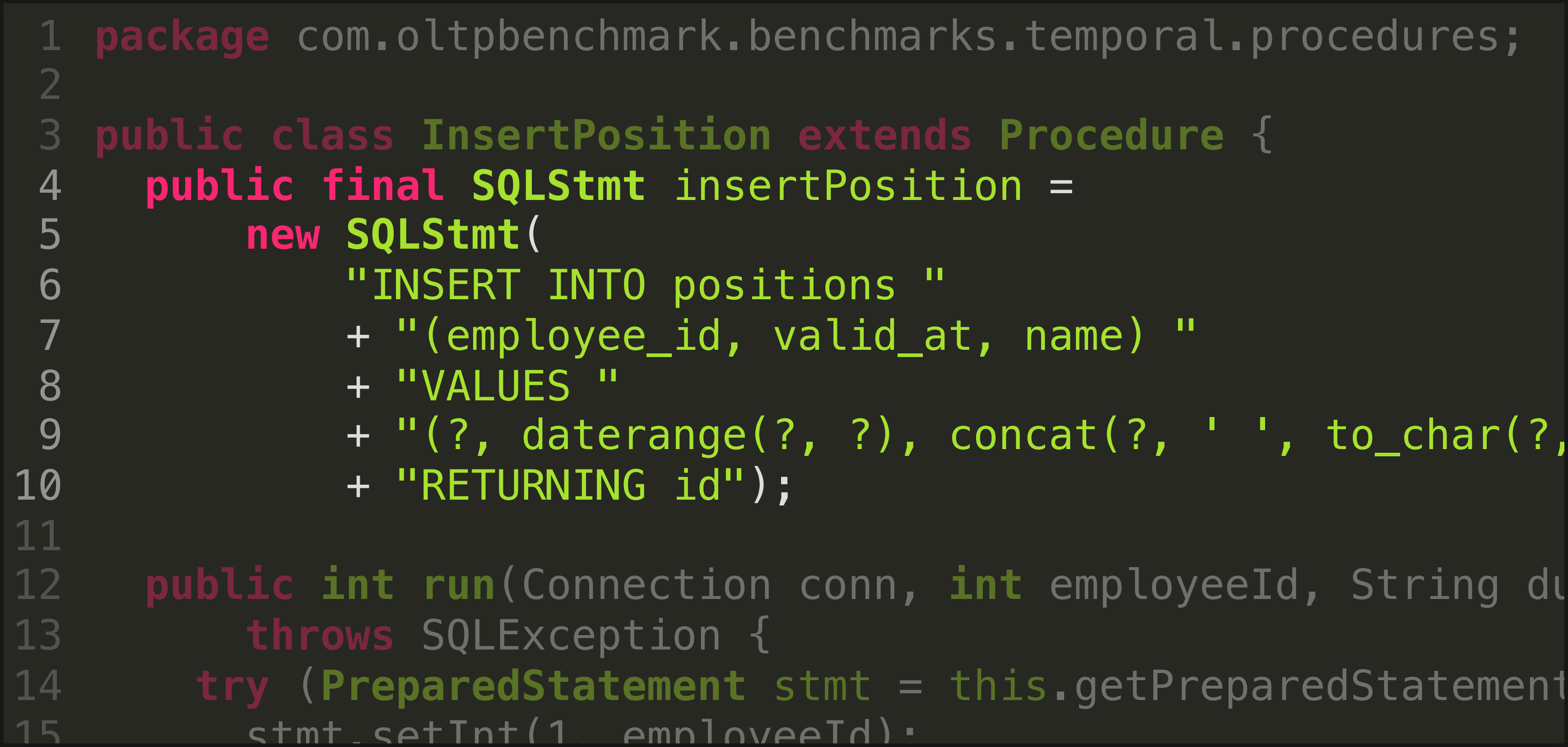
```
CREATE TABLE employees (
    id          int GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    valid_at    daterange NOT NULL,
    name        text NOT NULL,
    salary      int NOT NULL,
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS)
);

CREATE TABLE positions (
    id          int GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    valid_at    daterange NOT NULL,
    name        text NOT NULL,
    employee_id int NOT NULL,
    PRIMARY KEY (id, valid_at WITHOUT OVERLAPS),
    FOREIGN KEY (employee_id, PERIOD valid_at) REFERENCES empl
```

InsertPosition



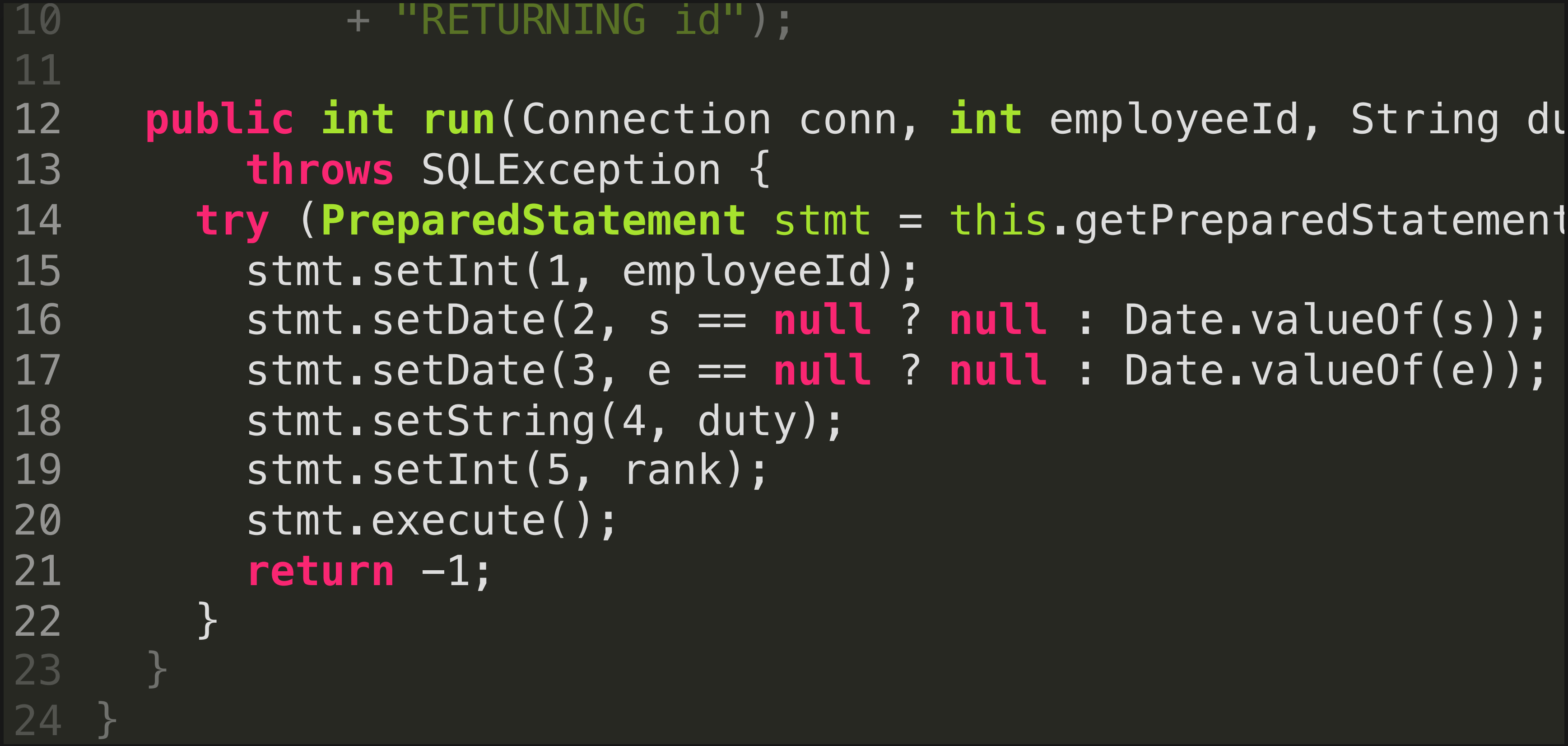
```
package com.oltpbenchmark.benchmarks.temporal.procedures;

public class InsertPosition extends Procedure {
  public final SQLStmt insertPosition =
      new SQLStmt(
          "INSERT INTO positions "
          + "(employee_id, valid_at, name) "
          + "VALUES "
          + "(?, daterange(?, ?), concat(?, ' ', to_char(?,
          + "RETURNING id");

  public int run(Connection conn, int employeeId, String du
      throws SQLException {
    try (PreparedStatement stmt = this.getPreparedStatement
      stmt.setInt(1, employeeId);
```

InsertPosition



```
package com.oltpbenchmark.benchmarks.temporal.procedures;

public class InsertPosition extends Procedure {
  public final SQLStmt insertPosition =
      new SQLStmt(
          "INSERT INTO positions "
          + "(employee_id, valid_at, name) "
          + "VALUES "
          + "(?, daterange(?, ?), concat(?, ' ', to_char(?,
          + "RETURNING id");

  public int run(Connection conn, int employeeId, String du
      throws SQLException {
    try (PreparedStatement stmt = this.getPreparedStatement
      stmt.setInt(1, employeeId);
```

InsertPosition

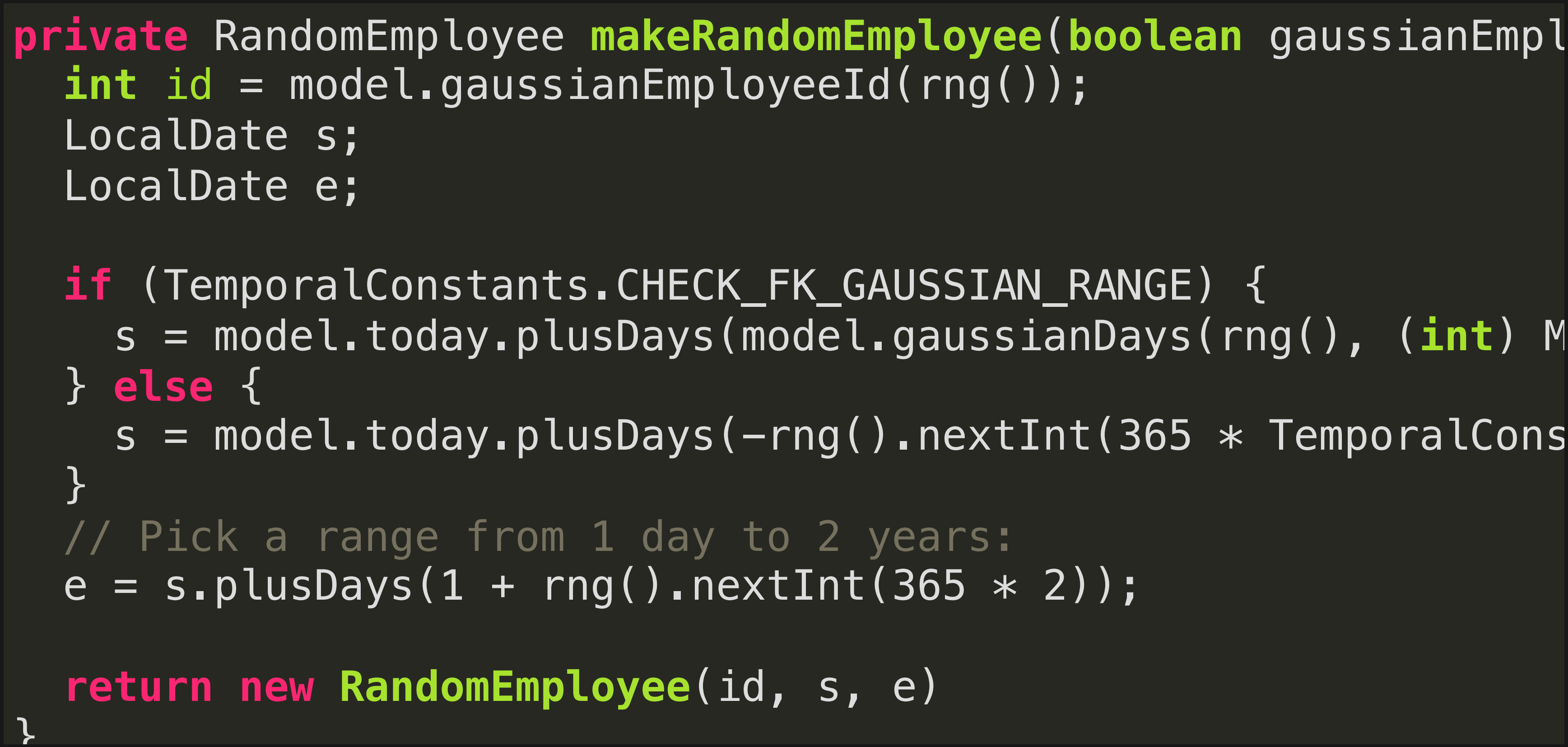


```
          + "RETURNING id");

  public int run(Connection conn, int employeeId, String du
      throws SQLException {
    try (PreparedStatement stmt = this.getPreparedStatement
      stmt.setInt(1, employeeId);
      stmt.setDate(2, s == null ? null : Date.valueOf(s));
      stmt.setDate(3, e == null ? null : Date.valueOf(e));
      stmt.setString(4, duty);
      stmt.setInt(5, rank);
      stmt.execute();
      return -1;
    }
  }
}
```

PROCEDURES

```
DeleteEmployee.java
InsertPosition.java
Noop.java
SelectAllEmployees.java
SelectAllEmployeesWithOptionalPositions.java
SelectEmployeesWithoutPosition.java
SelectEmployeesWithPosition.java
SelectOneEmployee.java
SelectOneEmployeePositions.java
SelectOneEmployeeWithOptionalPositions.java
UpdateEmployee.java
UpdatePosition.java
```

TemporalWorker



```
private RandomEmployee makeRandomEmployee(boolean gaussianEmpl
  int id = model.gaussianEmployeeId(rng());
  LocalDate s;
  LocalDate e;

  if (TemporalConstants.CHECK_FK_GAUSSIAN_RANGE) {
    s = model.today.plusDays(model.gaussianDays(rng(), (int) M
  } else {
    s = model.today.plusDays(-rng().nextInt(365 * TemporalCons
  }
  // Pick a range from 1 day to 2 years:
  e = s.plusDays(1 + rng().nextInt(365 * 2));

  return new RandomEmployee(id, s, e)
}
```

FOREIGN KEYS

- `EXISTS`
- `lag`
- `range_agg`

EXISTS IMPL

```
SELECT 1
-- There was a PK when the FK started:
WHERE EXISTS
  SELECT  1
  FROM    [ONLY] <pktable>
  WHERE   pkatt1 = $1 [AND ...]
  AND     COALESCE(lower(pkperiodatt), '-Infinity')
       <= COALESCE(lower($n), '-Infinity')
  AND     COALESCE(lower($n), '-Infinity')
       <  COALESCE(upper(pkperiodatt), 'Infinity')
  FOR KEY SHARE OF x
)
...
```

EXISTS IMPL

```
-- There was a PK when the FK ended:
AND EXISTS (
  SELECT  1
  FROM    [ONLY] <pktable>
  WHERE   pkatt1 = $1 [AND ...]
  AND     COALESCE(lower(pkperiodatt), '-Infinity')
       <  COALESCE(upper($n), 'Infinity')
  AND     COALESCE(upper($n), 'Infinity')
       <= COALESCE(upper(pkperiodatt), 'Infinity')
  FOR KEY SHARE OF x
)
...
```

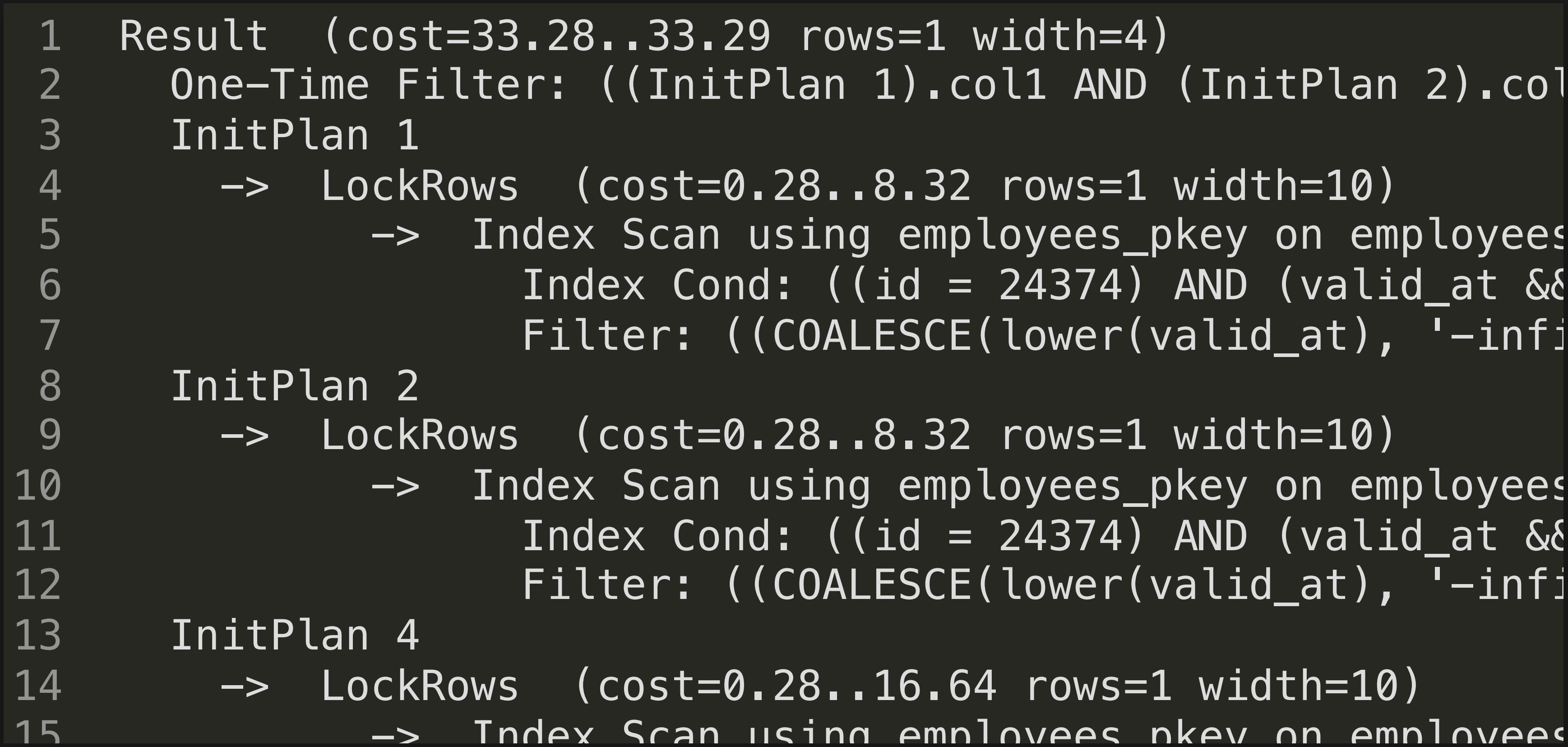
EXISTS IMPL

```
-- There are no gaps in the PK:
-- (i.e. there is no PK that ends early,
-- unless a matching PK record starts right away)
AND NOT EXISTS (
  SELECT  1
  FROM    [ONLY] <pktable> AS pk1
  WHERE   pkatt1 = $1 [AND ...]
  AND     COALESCE(lower($n), '-Infinity')
       <  COALESCE(upper(pkperiodatt), 'Infinity')
  AND     COALESCE(upper(pkperiodatt), 'Infinity')
       <  COALESCE(upper($n), 'Infinity')
  ...
```

EXISTS IMPL

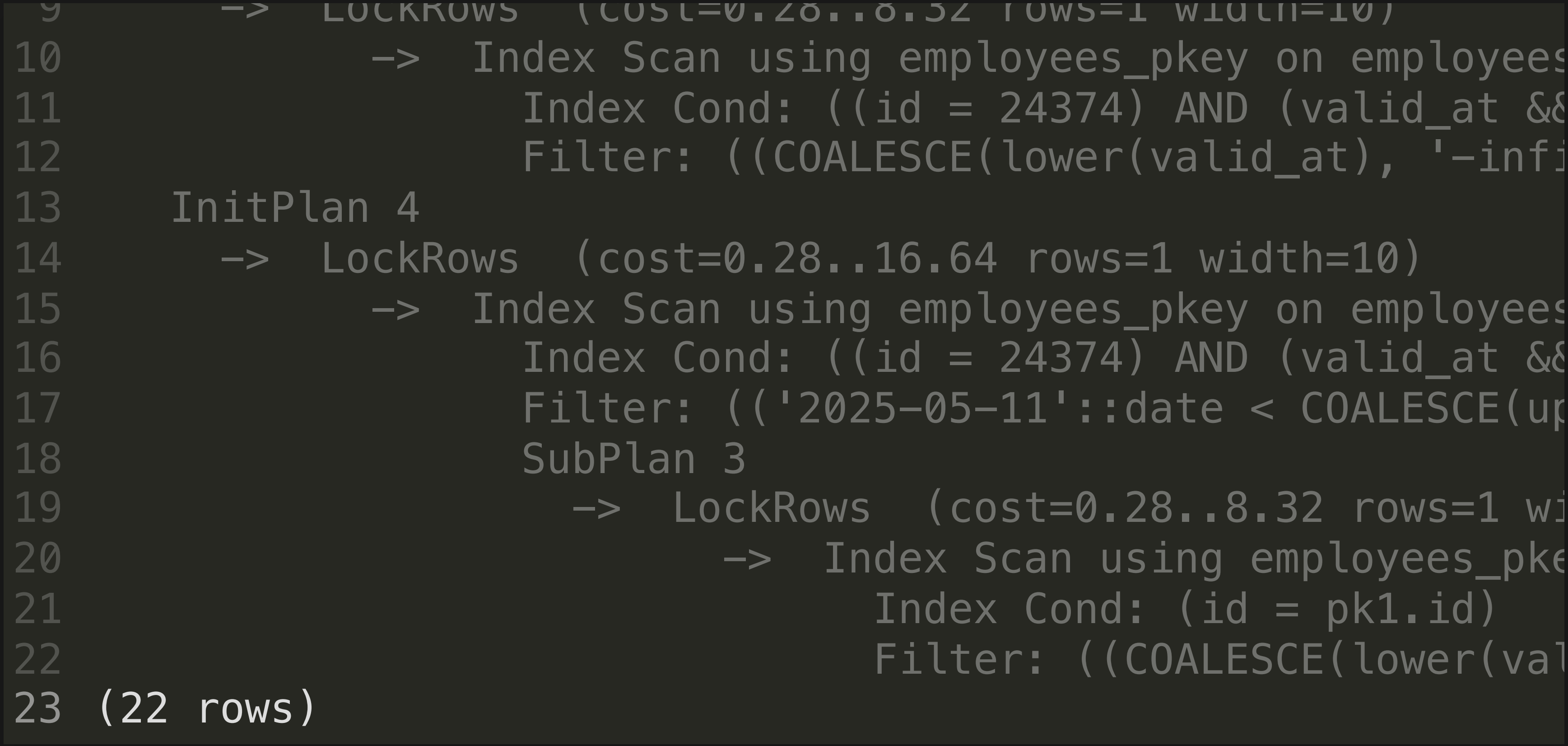
```
  ...
  AND     NOT EXISTS (
    SELECT  1
    FROM    [ONLY] <pktable> AS pk2
    WHERE   pk1.pkatt1 = pk2.pkatt1 [AND ...]
            -- but skip pk1.pkperiodatt && pk2.pkperiodatt
    AND     COALESCE(lower(pk2.pkperiodatt), '-Infinity')
         <= COALESCE(upper(pk1.pkperiodatt), 'Infinity')
            COALESCE(upper(pk1.pkperiodatt), 'Infinity')
         <  COALESCE(upper(pk2.pkperiodatt), 'Infinity')
    FOR KEY SHARE OF pk2
  )
  FOR KEY SHARE OF pk1
)
```

EXISTS IMPL



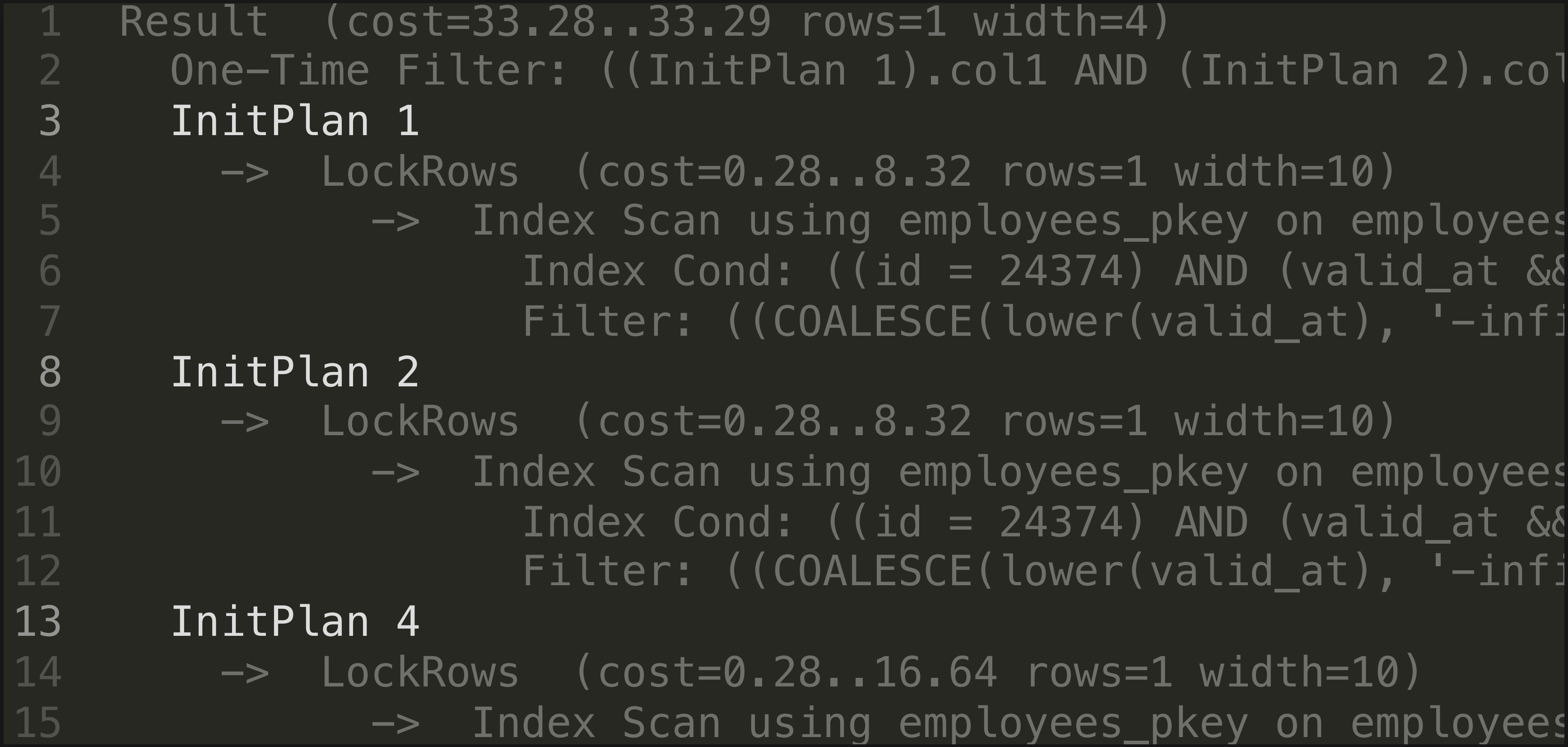
```
Result  (cost=33.28..33.29 rows=1 width=4)
  One-Time Filter: ((InitPlan 1).col1 AND (InitPlan 2).col
  InitPlan 1
    ->  LockRows  (cost=0.28..8.32 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374) AND (valid_at &&
                Filter: ((COALESCE(lower(valid_at), '-infi
  InitPlan 2
    ->  LockRows  (cost=0.28..8.32 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374) AND (valid_at &&
                Filter: ((COALESCE(lower(valid_at), '-infi
  InitPlan 4
    ->  LockRows  (cost=0.28..16.64 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
```

EXISTS IMPL



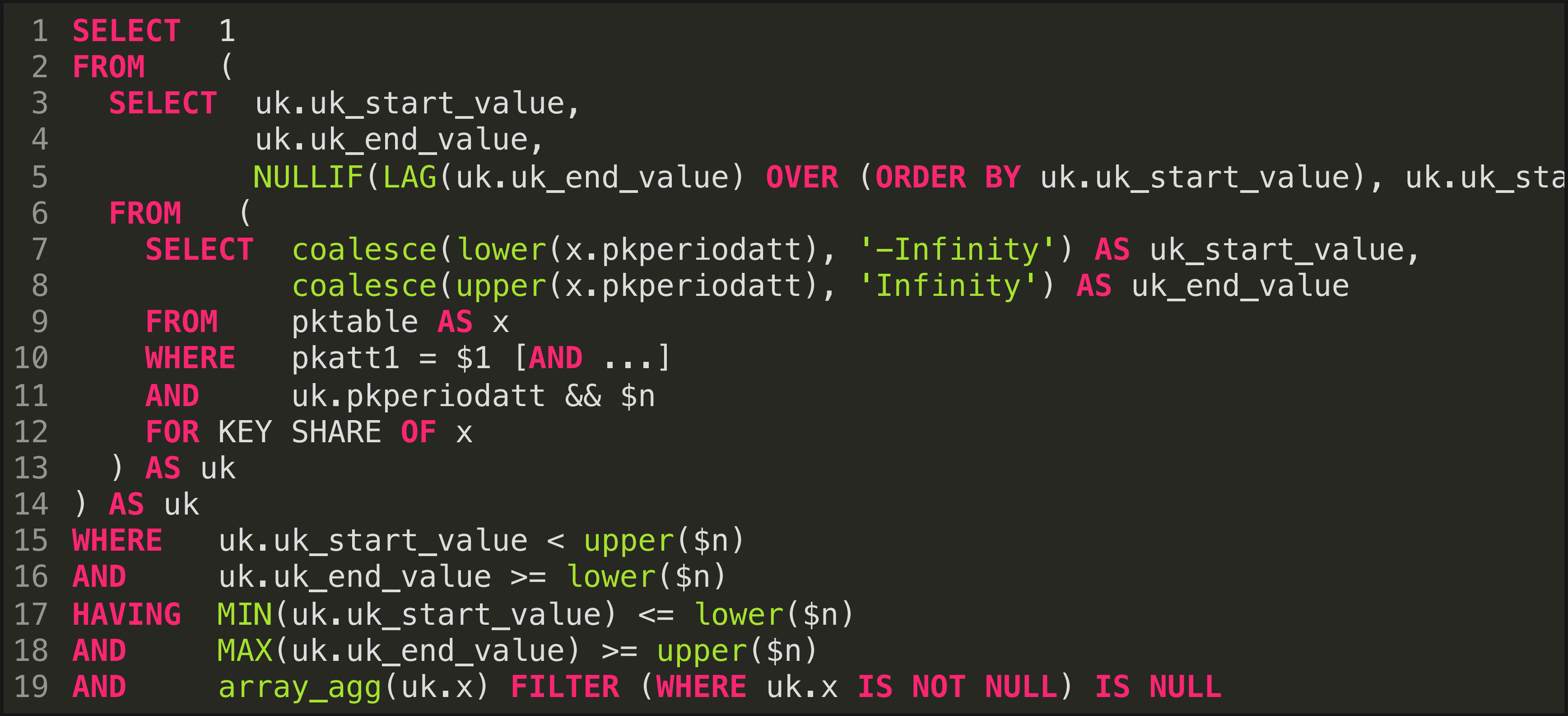
```
     ->  LockRows  (cost=0.28..8.32 rows=1 width=10)
           ->  Index Scan using employees_pkey on employees
                 Index Cond: ((id = 24374) AND (valid_at &&
                 Filter: ((COALESCE(lower(valid_at), '-infi
   InitPlan 4
     ->  LockRows  (cost=0.28..16.64 rows=1 width=10)
           ->  Index Scan using employees_pkey on employees
                 Index Cond: ((id = 24374) AND (valid_at &&
                 Filter: (('2025-05-11'::date < COALESCE(up
                 SubPlan 3
                   ->  LockRows  (cost=0.28..8.32 rows=1 wi
                         ->  Index Scan using employees_pke
                               Index Cond: (id = pk1.id)
                               Filter: ((COALESCE(lower(val
(22 rows)
```

EXISTS IMPL



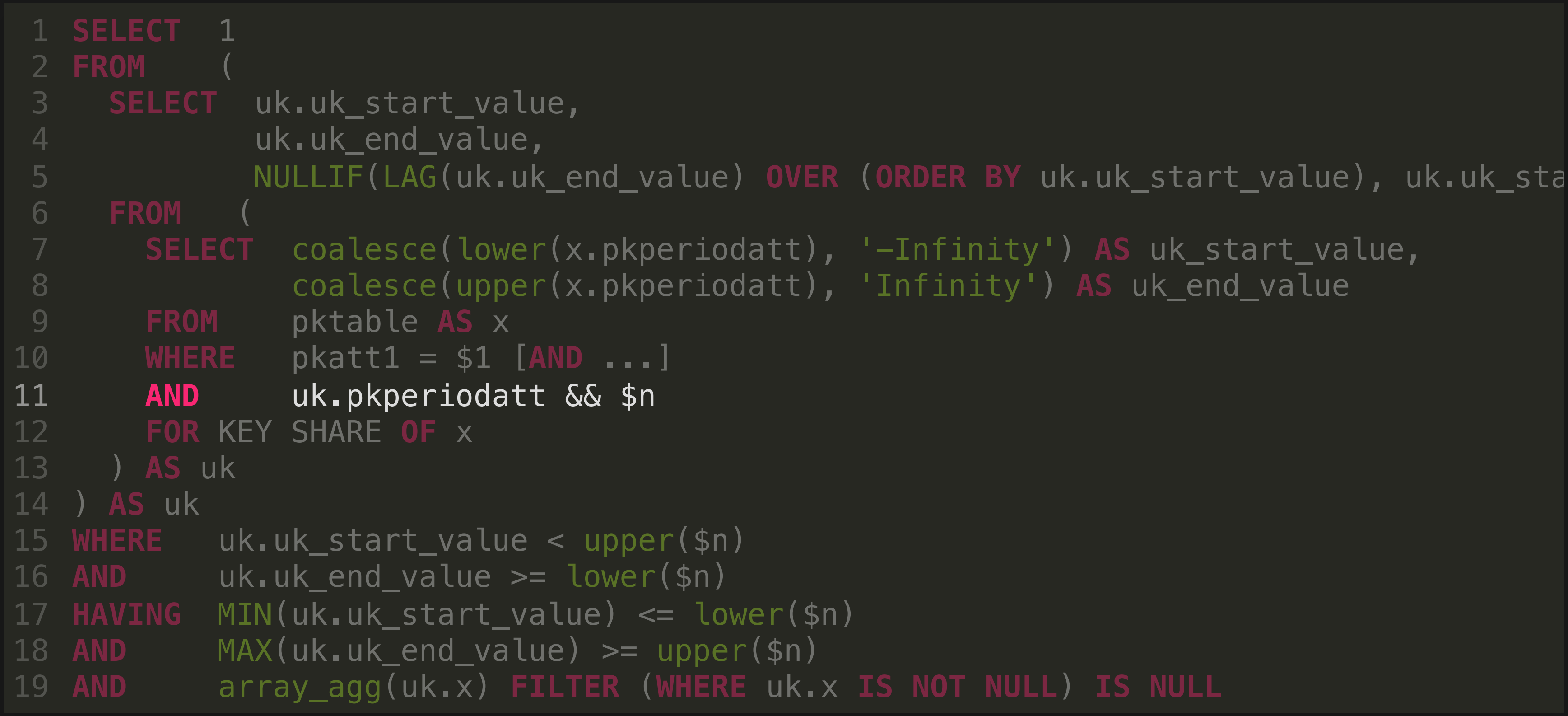
```
Result  (cost=33.28..33.29 rows=1 width=4)
  One-Time Filter: ((InitPlan 1).col1 AND (InitPlan 2).col
  InitPlan 1
    ->  LockRows  (cost=0.28..8.32 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374) AND (valid_at &&
                Filter: ((COALESCE(lower(valid_at), '-infi
  InitPlan 2
    ->  LockRows  (cost=0.28..8.32 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374) AND (valid_at &&
                Filter: ((COALESCE(lower(valid_at), '-infi
  InitPlan 4
    ->  LockRows  (cost=0.28..16.64 rows=1 width=10)
          ->  Index Scan using employees_pkey on employees
```

LAG IMPL



```sql
SELECT  1
FROM    (
  SELECT  uk.uk_start_value,
          uk.uk_end_value,
          NULLIF(LAG(uk.uk_end_value) OVER (ORDER BY uk.uk_start_value), uk.uk_sta
  FROM    (
    SELECT  coalesce(lower(x.pkperiodatt), '-Infinity') AS uk_start_value,
            coalesce(upper(x.pkperiodatt), 'Infinity') AS uk_end_value
    FROM    pktable AS x
    WHERE   pkatt1 = $1 [AND ...]
    AND     uk.pkperiodatt && $n
    FOR KEY SHARE OF x
  ) AS uk
) AS uk
WHERE   uk.uk_start_value < upper($n)
AND     uk.uk_end_value >= lower($n)
HAVING  MIN(uk.uk_start_value) <= lower($n)
AND     MAX(uk.uk_end_value) >= upper($n)
AND     array_agg(uk.x) FILTER (WHERE uk.x IS NOT NULL) IS NULL
```

LAG IMPL

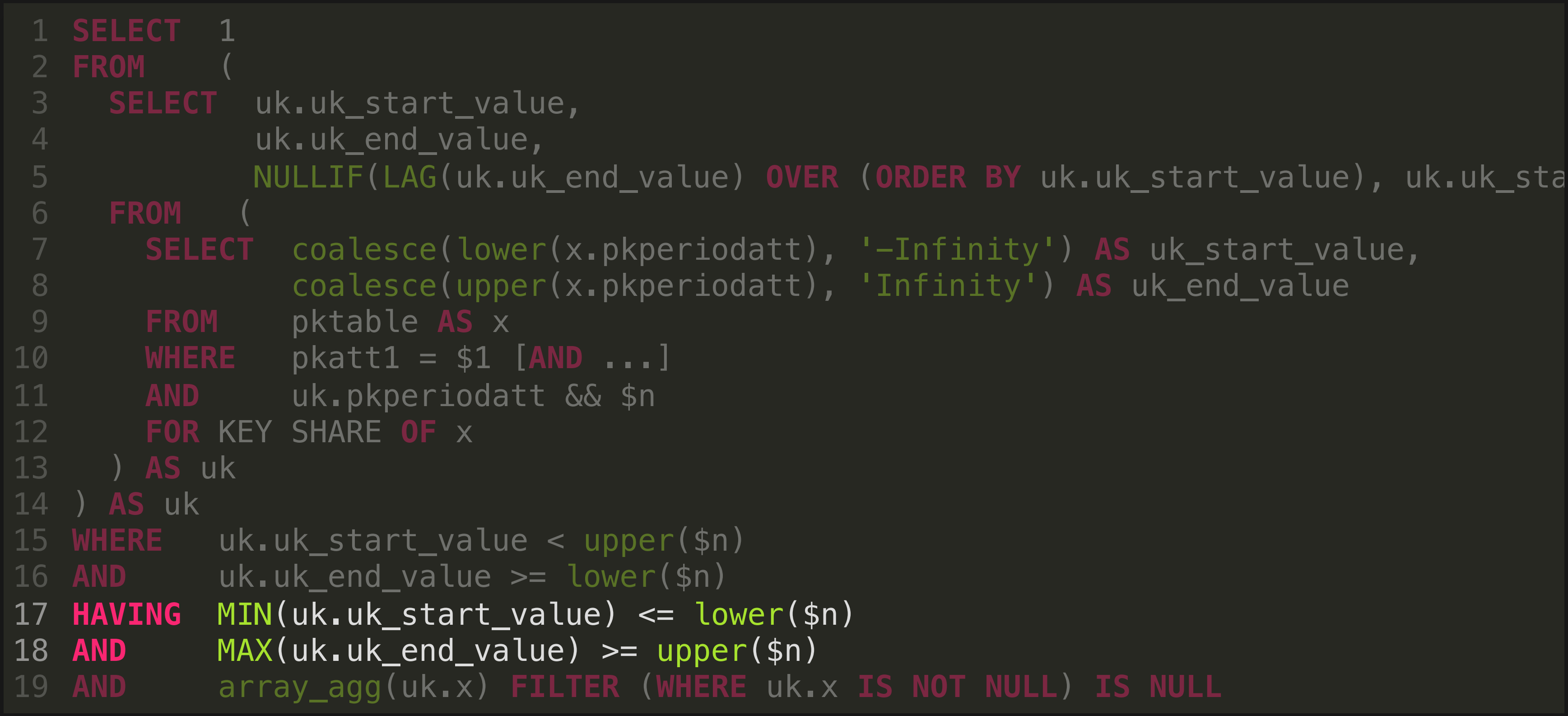


```
SELECT  1
FROM    (
  SELECT  uk.uk_start_value,
          uk.uk_end_value,
          NULLIF(LAG(uk.uk_end_value) OVER (ORDER BY uk.uk_start_value), uk.uk_sta
  FROM   (
    SELECT  coalesce(lower(x.pkperiodatt), '-Infinity') AS uk_start_value,
            coalesce(upper(x.pkperiodatt), 'Infinity') AS uk_end_value
    FROM    pktable AS x
    WHERE   pkatt1 = $1 [AND ...]
    AND     uk.pkperiodatt && $n
    FOR KEY SHARE OF x
  ) AS uk
) AS uk
WHERE   uk.uk_start_value < upper($n)
AND     uk.uk_end_value >= lower($n)
HAVING  MIN(uk.uk_start_value) <= lower($n)
AND     MAX(uk.uk_end_value) >= upper($n)
AND     array_agg(uk.x) FILTER (WHERE uk.x IS NOT NULL) IS NULL
```

LAG IMPL

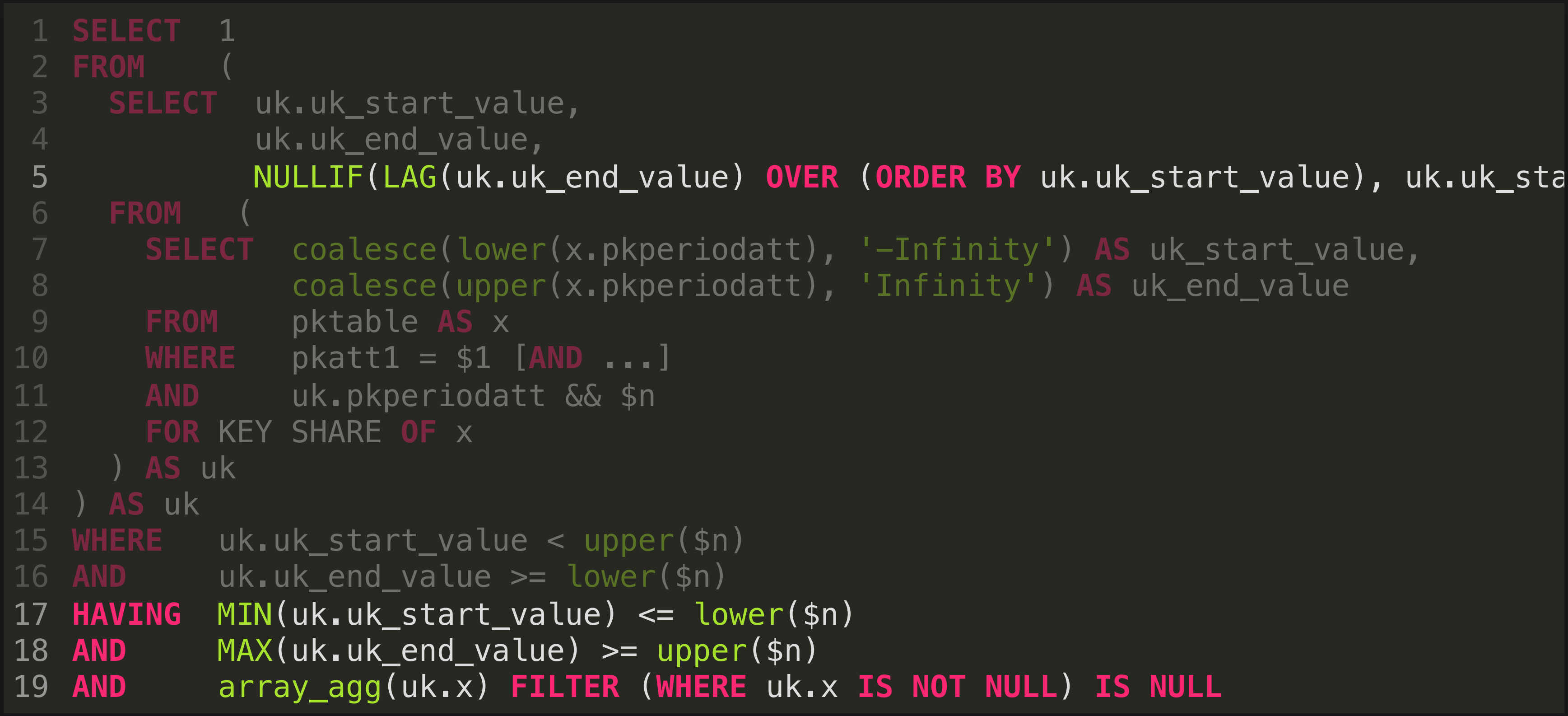
```
SELECT  1
FROM    (
  SELECT  uk.uk_start_value,
          uk.uk_end_value,
          NULLIF(LAG(uk.uk_end_value) OVER (ORDER BY uk.uk_start_value), uk.uk_sta
  FROM   (
    SELECT  coalesce(lower(x.pkperiodatt), '-Infinity') AS uk_start_value,
            coalesce(upper(x.pkperiodatt), 'Infinity') AS uk_end_value
    FROM    pktable AS x
    WHERE   pkatt1 = $1 [AND ...]
    AND     uk.pkperiodatt && $n
    FOR KEY SHARE OF x
  ) AS uk
) AS uk
WHERE   uk.uk_start_value < upper($n)
AND     uk.uk_end_value >= lower($n)
HAVING  MIN(uk.uk_start_value) <= lower($n)
AND     MAX(uk.uk_end_value) >= upper($n)
AND     array_agg(uk.x) FILTER (WHERE uk.x IS NOT NULL) IS NULL
```

LAG IMPL



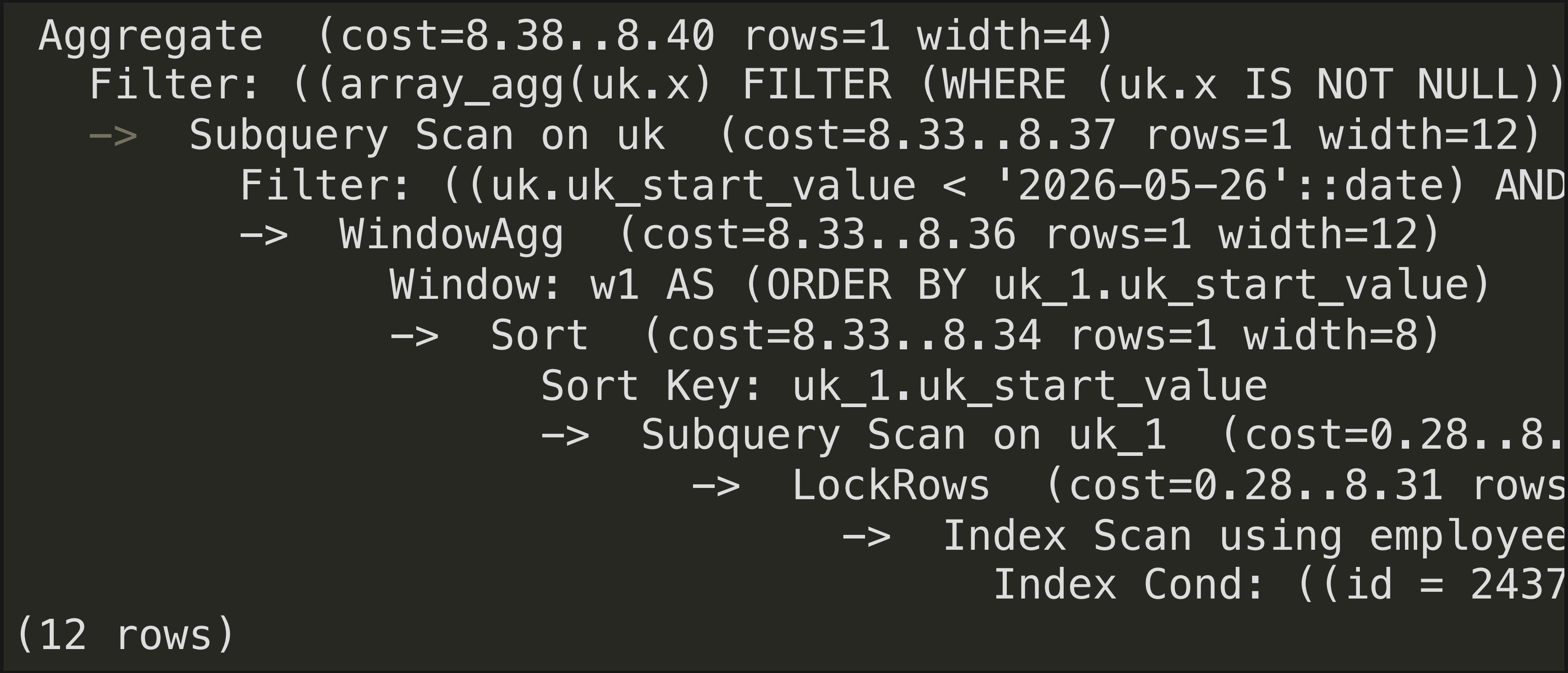
```
SELECT  1
FROM    (
  SELECT  uk.uk_start_value,
          uk.uk_end_value,
          NULLIF(LAG(uk.uk_end_value) OVER (ORDER BY uk.uk_start_value), uk.uk_sta
  FROM   (
    SELECT  coalesce(lower(x.pkperiodatt), '-Infinity') AS uk_start_value,
            coalesce(upper(x.pkperiodatt), 'Infinity') AS uk_end_value
    FROM    pktable AS x
    WHERE   pkatt1 = $1 [AND ...]
    AND     uk.pkperiodatt && $n
    FOR KEY SHARE OF x
  ) AS uk
) AS uk
WHERE   uk.uk_start_value < upper($n)
AND     uk.uk_end_value >= lower($n)
HAVING  MIN(uk.uk_start_value) <= lower($n)
AND     MAX(uk.uk_end_value) >= upper($n)
AND     array_agg(uk.x) FILTER (WHERE uk.x IS NOT NULL) IS NULL
```

LAG IMPL



```
SELECT  1
FROM    (
  SELECT  uk.uk_start_value,
          uk.uk_end_value,
          NULLIF(LAG(uk.uk_end_value) OVER (ORDER BY uk.uk_start_value), uk.uk_sta
  FROM   (
    SELECT  coalesce(lower(x.pkperiodatt), '-Infinity') AS uk_start_value,
            coalesce(upper(x.pkperiodatt), 'Infinity') AS uk_end_value
    FROM    pktable AS x
    WHERE   pkatt1 = $1 [AND ...]
    AND     uk.pkperiodatt && $n
    FOR KEY SHARE OF x
  ) AS uk
) AS uk
WHERE   uk.uk_start_value < upper($n)
AND     uk.uk_end_value >= lower($n)
HAVING  MIN(uk.uk_start_value) <= lower($n)
AND     MAX(uk.uk_end_value) >= upper($n)
AND     array_agg(uk.x) FILTER (WHERE uk.x IS NOT NULL) IS NULL
```

LAG IMPL

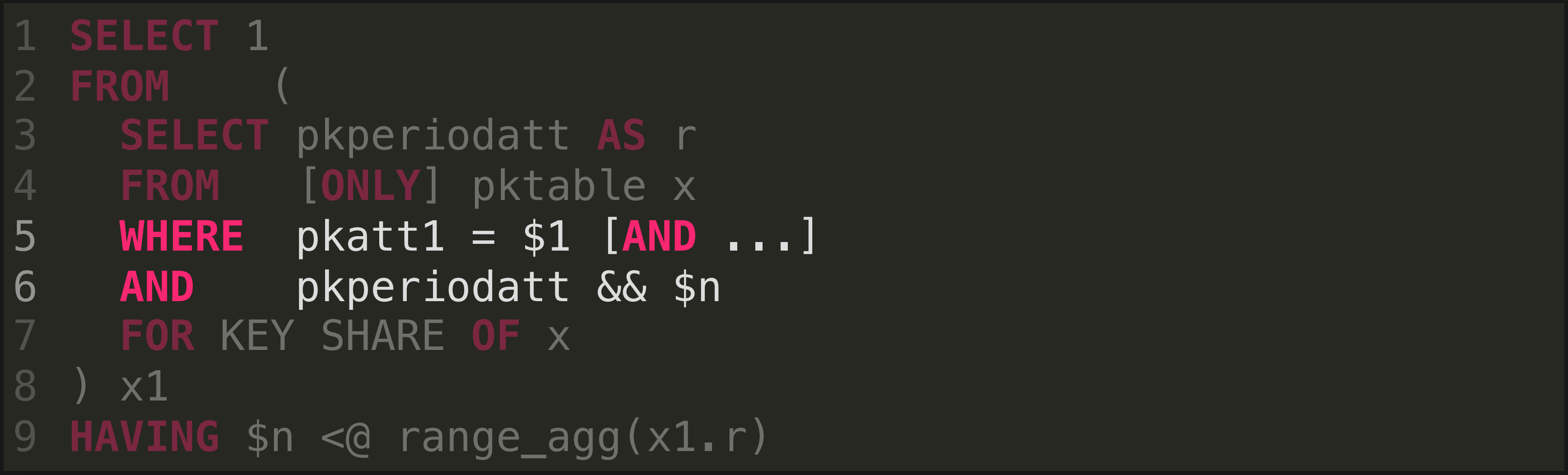


```
 Aggregate  (cost=8.38..8.40 rows=1 width=4)
   Filter: ((array_agg(uk.x) FILTER (WHERE (uk.x IS NOT NULL))
   ->  Subquery Scan on uk  (cost=8.33..8.37 rows=1 width=12)
         Filter: ((uk.uk_start_value < '2026-05-26'::date) AND
         ->  WindowAgg  (cost=8.33..8.36 rows=1 width=12)
               Window: w1 AS (ORDER BY uk_1.uk_start_value)
               ->  Sort  (cost=8.33..8.34 rows=1 width=8)
                     Sort Key: uk_1.uk_start_value
                     ->  Subquery Scan on uk_1  (cost=0.28..8.
                           ->  LockRows  (cost=0.28..8.31 rows
                                 ->  Index Scan using employee
                                       Index Cond: ((id = 2437
(12 rows)
```

range_agg IMPL

```
SELECT 1
FROM    (
  SELECT pkperiodatt AS r
  FROM   [ONLY] pktable x
  WHERE  pkatt1 = $1 [AND ...]
  AND    pkperiodatt && $n
  FOR KEY SHARE OF x
) x1
HAVING $n <@ range_agg(x1.r)
```

range_agg IMPL



```
SELECT 1
FROM    (
  SELECT pkperiodatt AS r
  FROM   [ONLY] pktable x
  WHERE  pkatt1 = $1 [AND ...]
  AND    pkperiodatt && $n
  FOR KEY SHARE OF x
) x1
HAVING $n <@ range_agg(x1.r)
```

range_agg IMPL

```sql
SELECT 1
FROM    (
  SELECT pkperiodatt AS r
  FROM   [ONLY] pktable x
  WHERE  pkatt1 = $1 [AND ...]
  AND    pkperiodatt && $n
  FOR KEY SHARE OF x
) x1
HAVING $n <@ range_agg(x1.r)
```

range_agg IMPL

```sql
SELECT 1
FROM    (
  SELECT pkperiodatt AS r
  FROM   [ONLY] pktable x
  WHERE  pkatt1 = $1 [AND ...]
  AND    pkperiodatt && $n
  FOR KEY SHARE OF x
) x1
HAVING $n <@ range_agg(x1.r)
```

range_agg IMPL

```
 Aggregate  (cost=8.32..8.34 rows=1 width=4)
   Filter: ('[2025-05-11,2026-05-26)'::daterange <@ range_agg(
   ->  Subquery Scan on x1  (cost=0.28..8.32 rows=1 width=10)
         ->  LockRows  (cost=0.28..8.31 rows=1 width=16)
               ->  Index Scan using employees_pkey on employee
                     Index Cond: ((id = 24374) AND (valid_at &
(6 rows)
```

HYPOTHESIS

- `range_agg` fastest
- `lag` nearly as fast
- `EXISTS` much slower

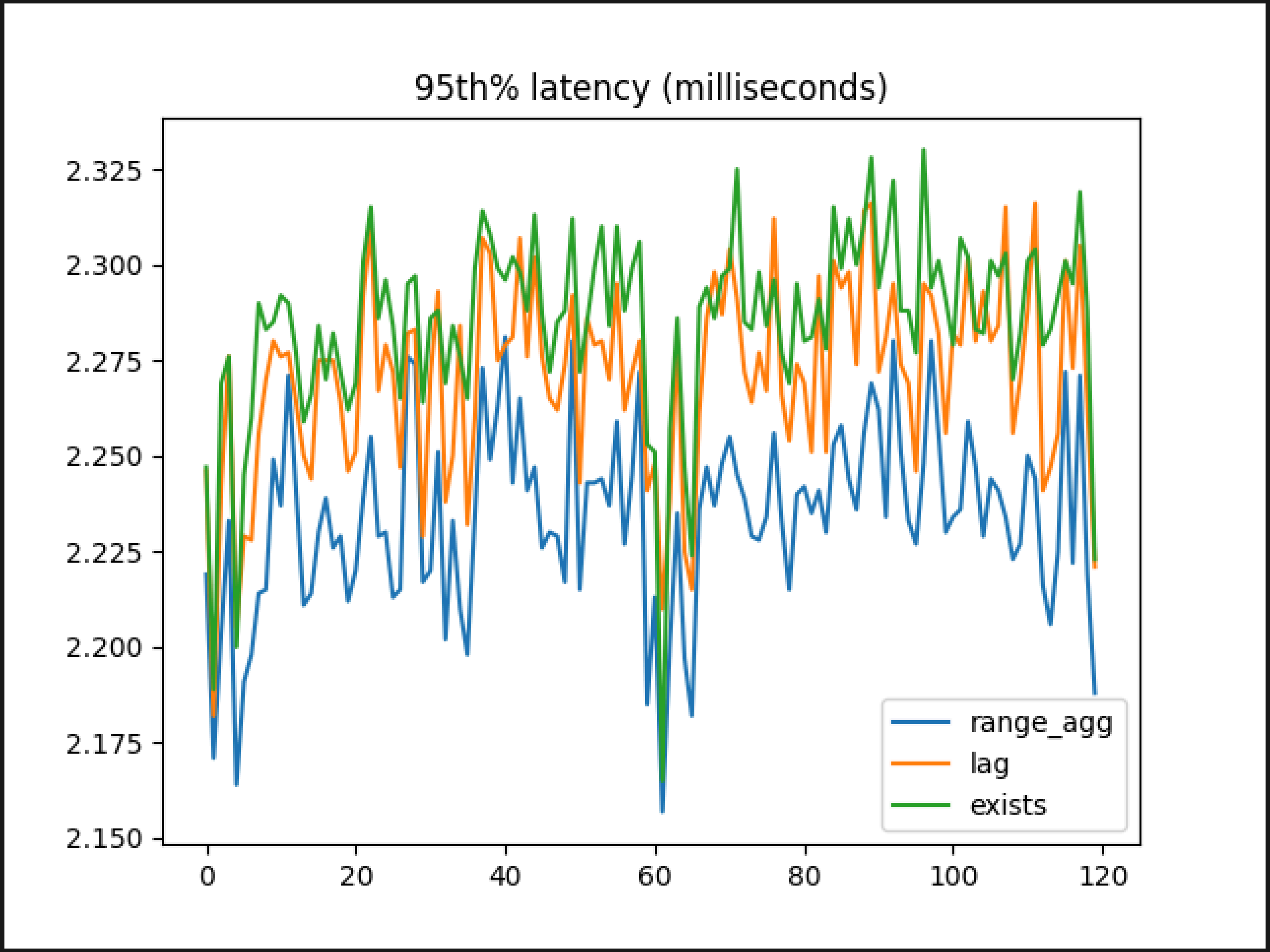
WORKLOAD

```xml
<transactiontypes>
    <transactiontype>
        <name>InsertPosition</name>
    </transactiontype>
    <transactiontype>
        <name>UpdatePosition</name>
    </transactiontype>
    <transactiontype>
        <name>UpdateEmployee</name>
    </transactiontype>
    <transactiontype>
        <name>DeleteEmployee</name>
    </transactiontype>
</transactiontypes>
```

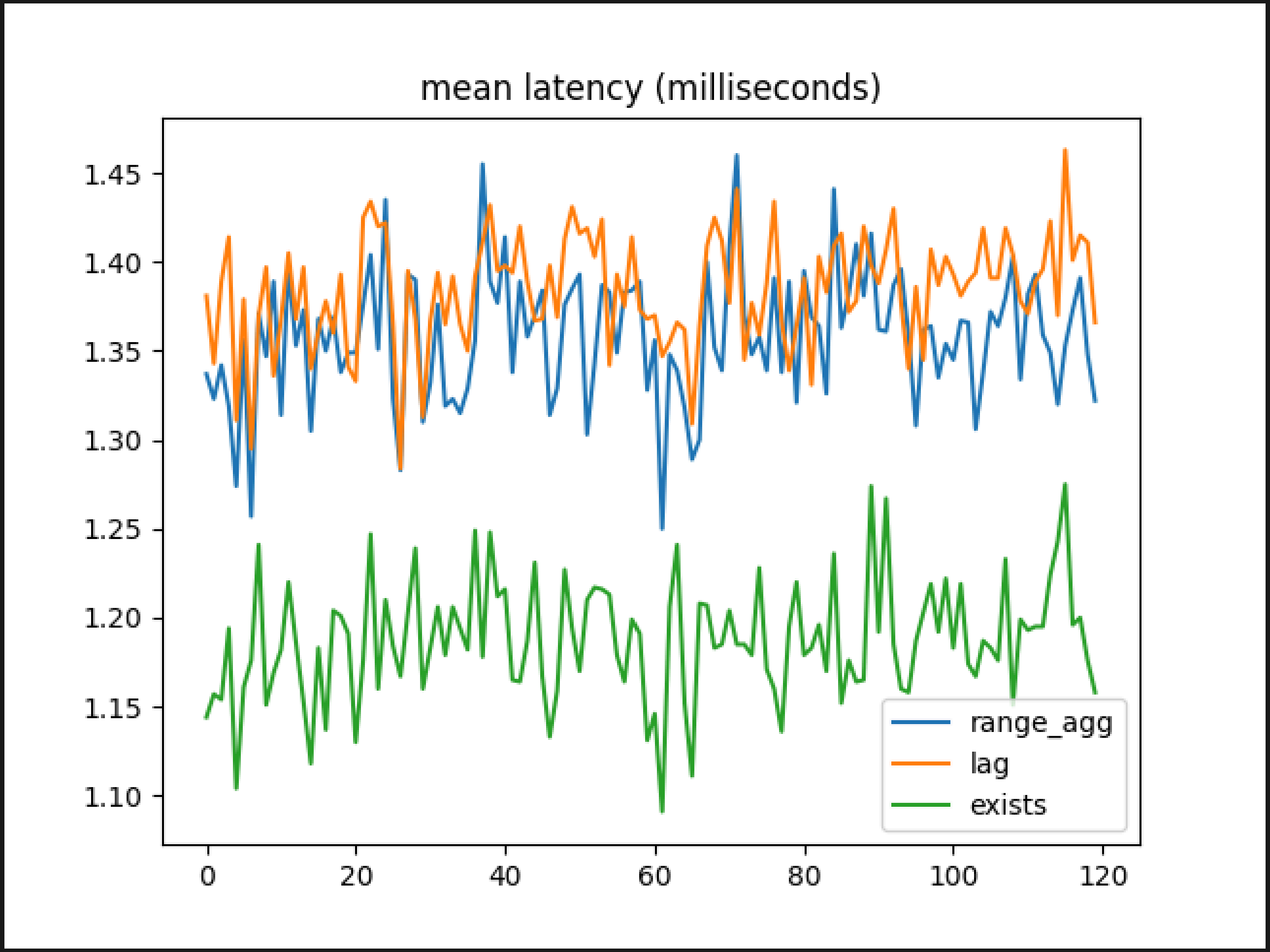
PROCEDURES

```
CheckForeignKeyExists.java
CheckForeignKeyLag.java
CheckForeignKeyRangeAgg.java
```

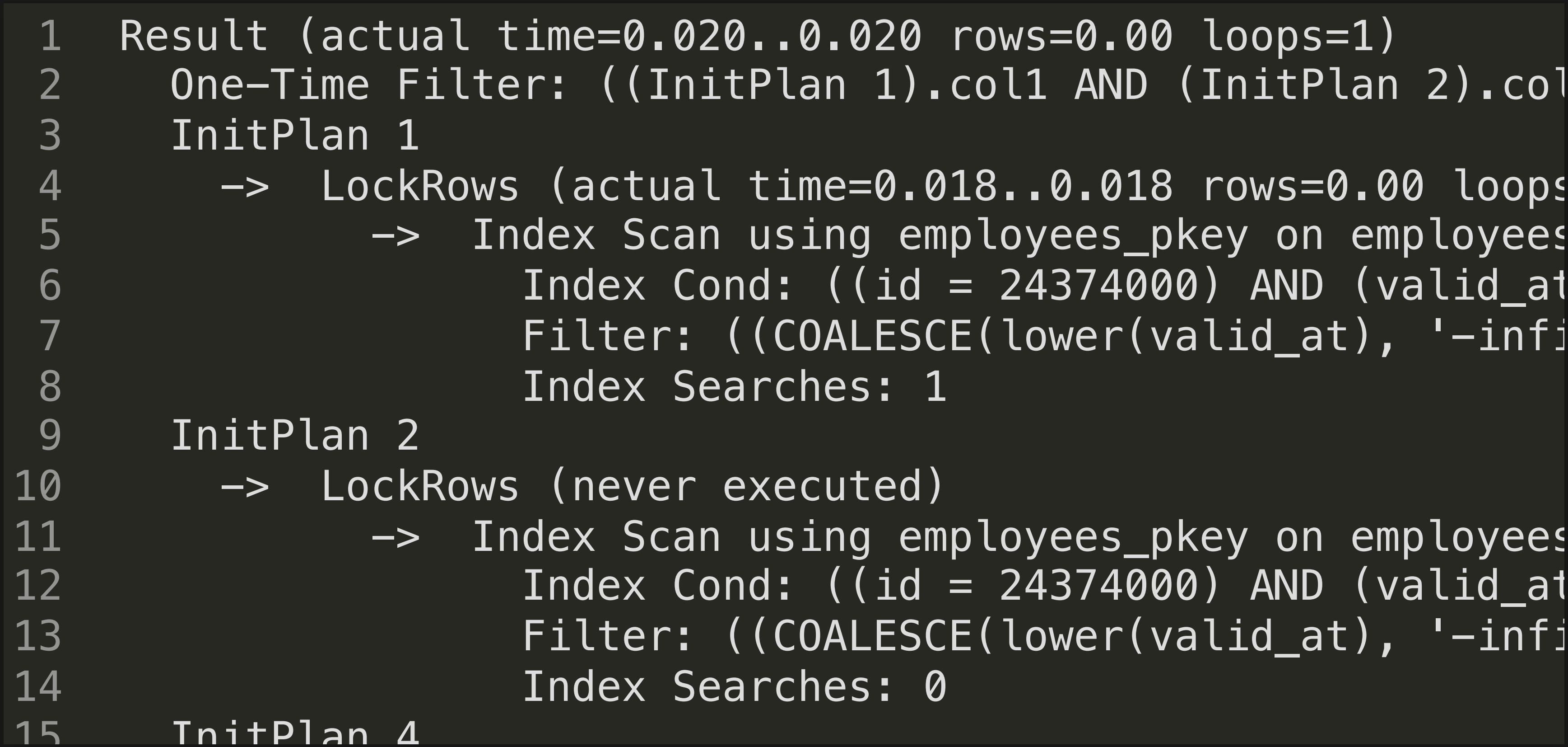
95% LATENCY


95th% latency (milliseconds)
2.325
2.300
2.275
2.250
2.225
2.200
2.175
2.150
0
20
40
60
80
100
120
range_agg
lag
exists

MEAN LATENCY

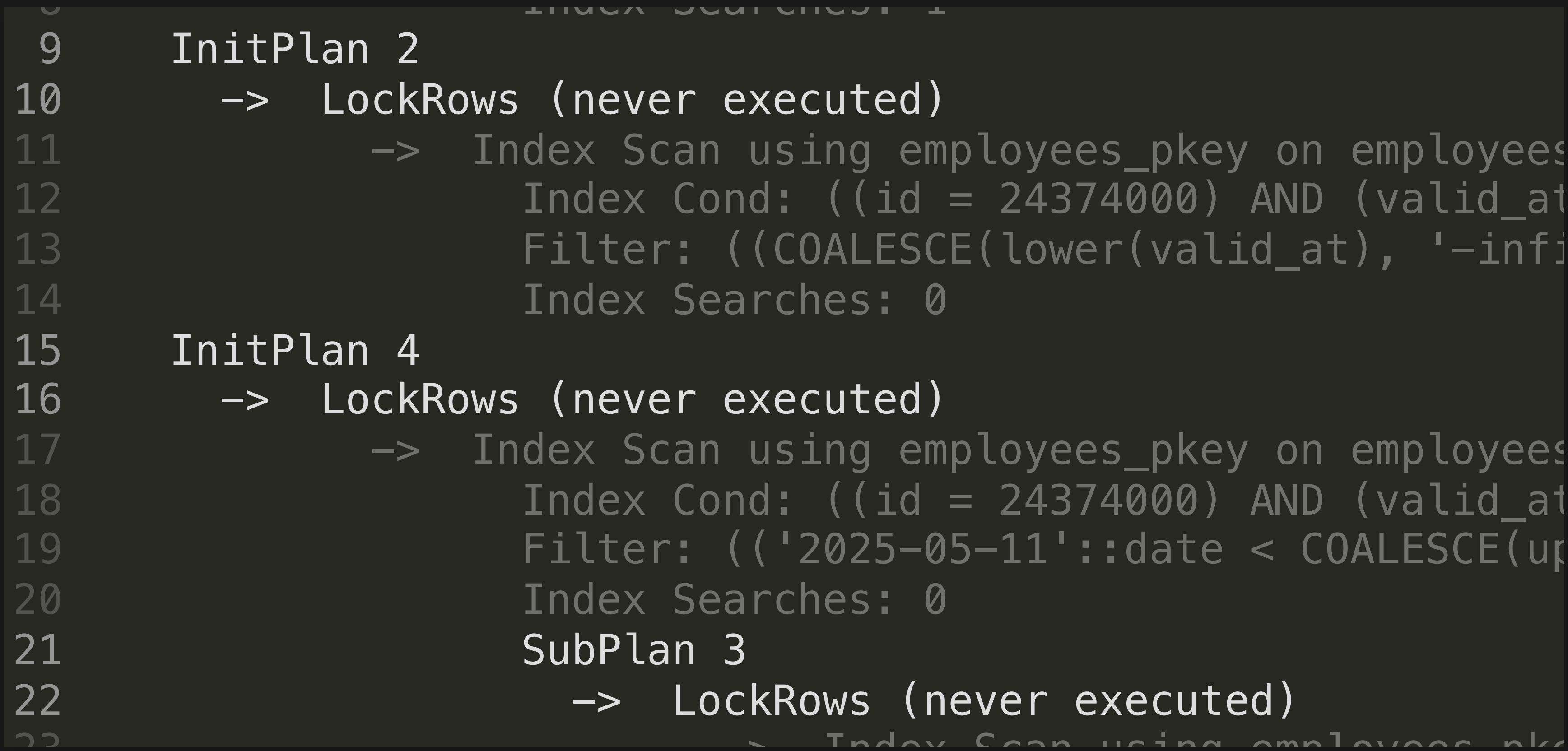


EXISTS IMPL



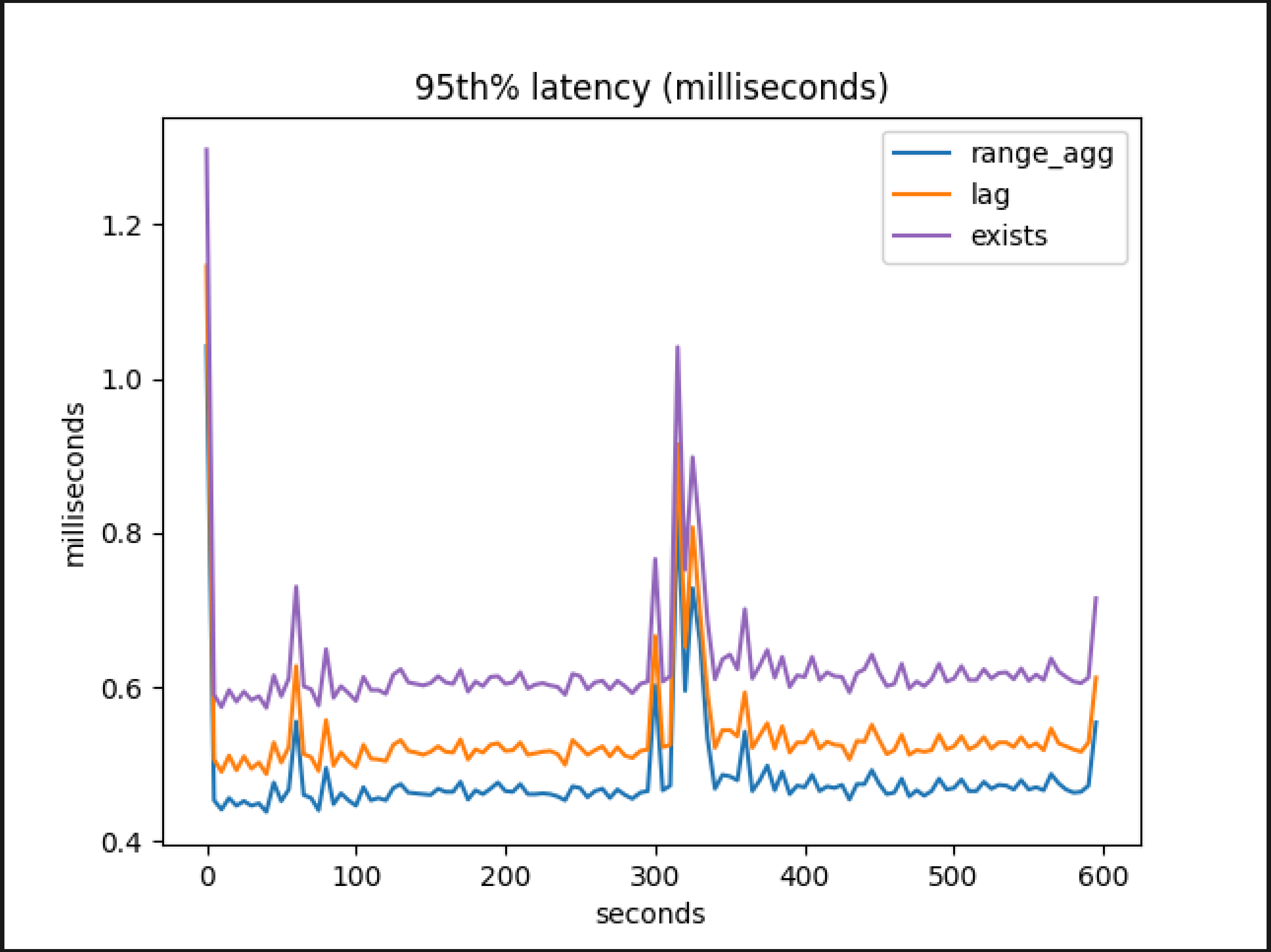
```
Result (actual time=0.020..0.020 rows=0.00 loops=1)
  One-Time Filter: ((InitPlan 1).col1 AND (InitPlan 2).col
  InitPlan 1
    ->  LockRows (actual time=0.018..0.018 rows=0.00 loops
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374000) AND (valid_at
                Filter: ((COALESCE(lower(valid_at), '-infi
                Index Searches: 1
  InitPlan 2
    ->  LockRows (never executed)
          ->  Index Scan using employees_pkey on employees
                Index Cond: ((id = 24374000) AND (valid_at
                Filter: ((COALESCE(lower(valid_at), '-infi
                Index Searches: 0
  InitPlan 4
```

EXISTS IMPL



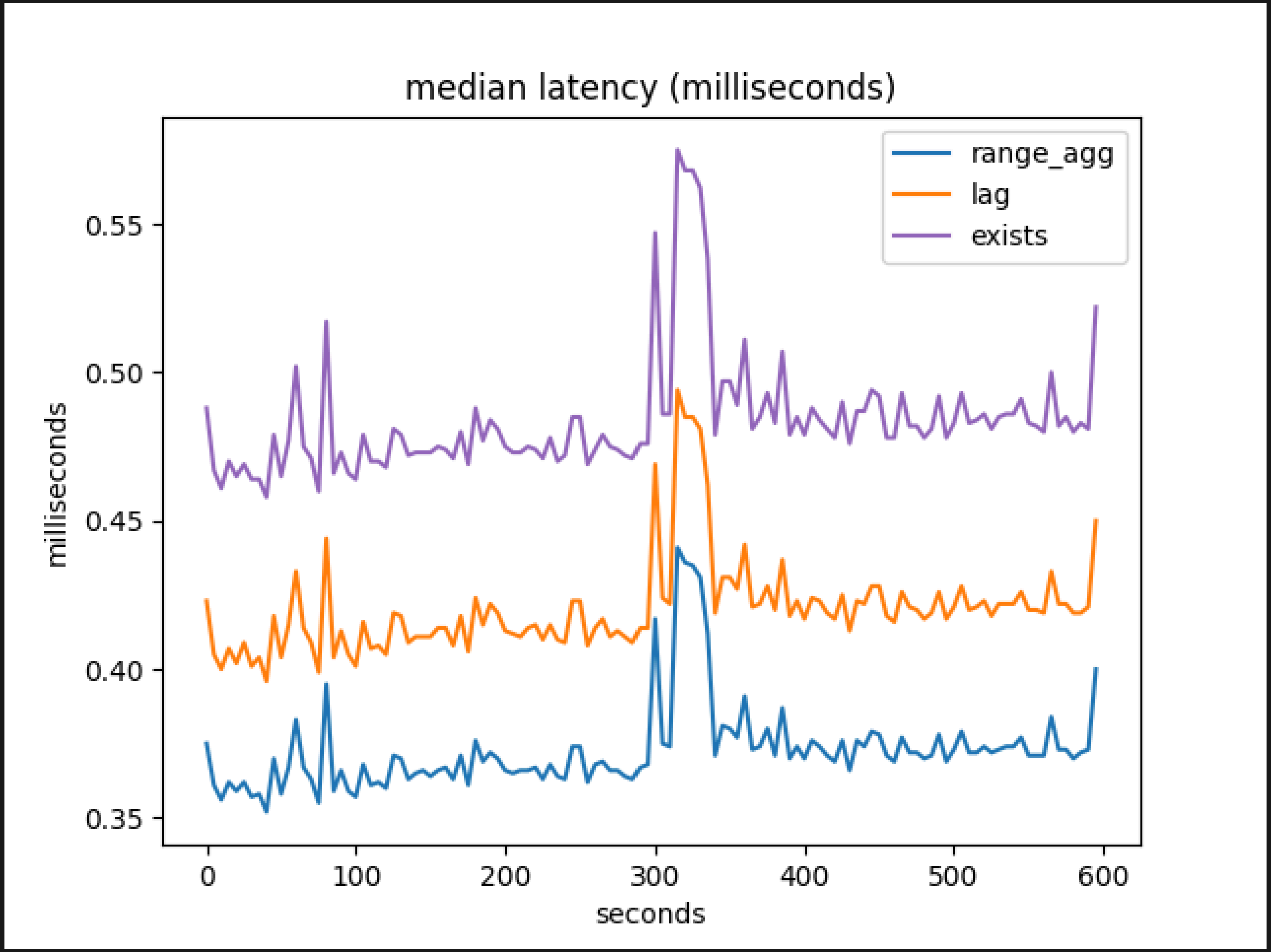
```
                  Index Searches: 1
    InitPlan 2
      ->  LockRows (never executed)
            ->  Index Scan using employees_pkey on employees
                  Index Cond: ((id = 24374000) AND (valid_at
                  Filter: ((COALESCE(lower(valid_at), '-infi
                  Index Searches: 0
    InitPlan 4
      ->  LockRows (never executed)
            ->  Index Scan using employees_pkey on employees
                  Index Cond: ((id = 24374000) AND (valid_at
                  Filter: (('2025-05-11'::date < COALESCE(up
                  Index Searches: 0
                  SubPlan 3
                    ->  LockRows (never executed)
                          ->  Index Scan using employees_pke
```

95% LATENCY



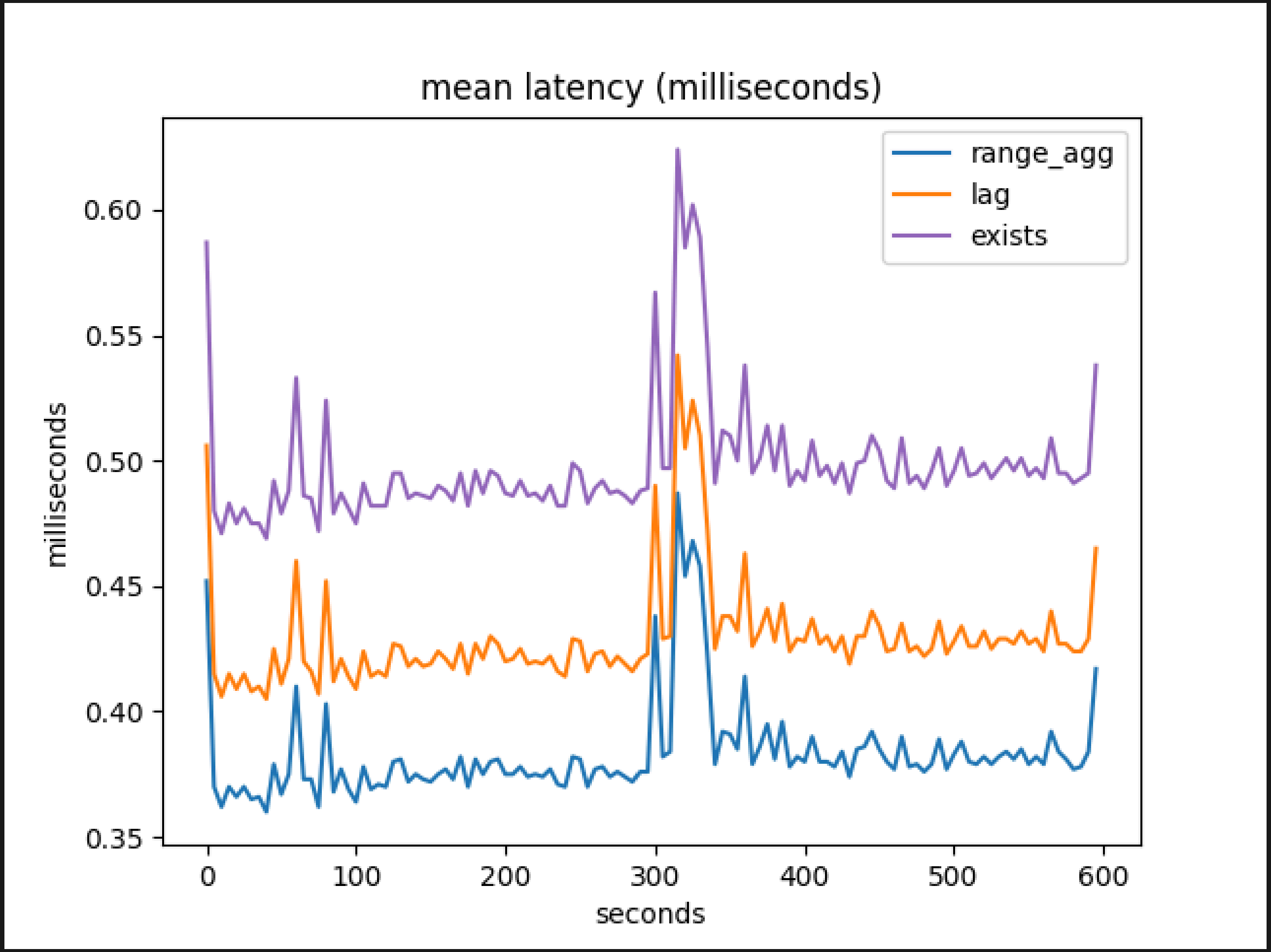
100k employees

MEDIAN LATENCY



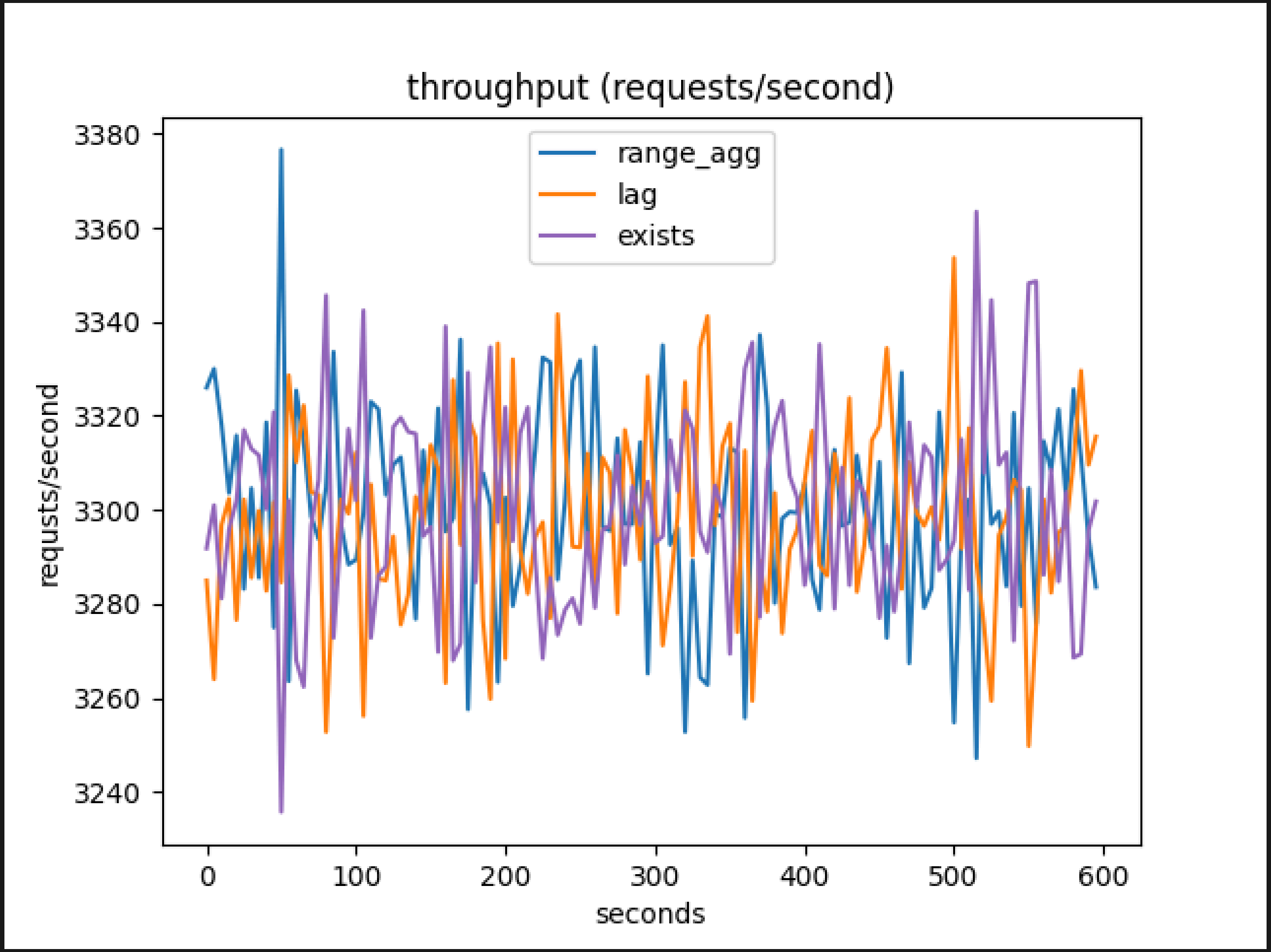
100k employees

MEAN LATENCY



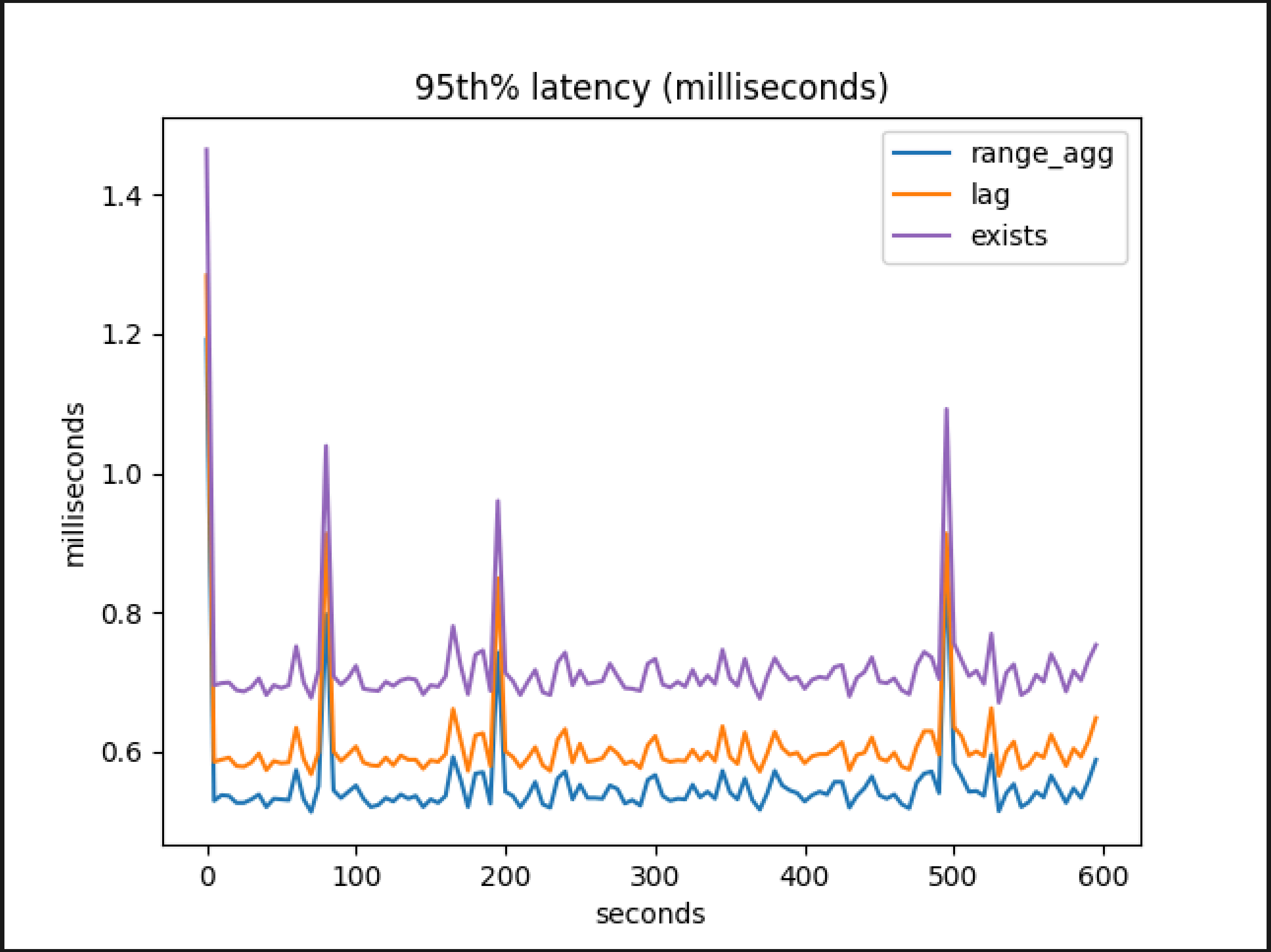
100k employees

THROUGHPUT



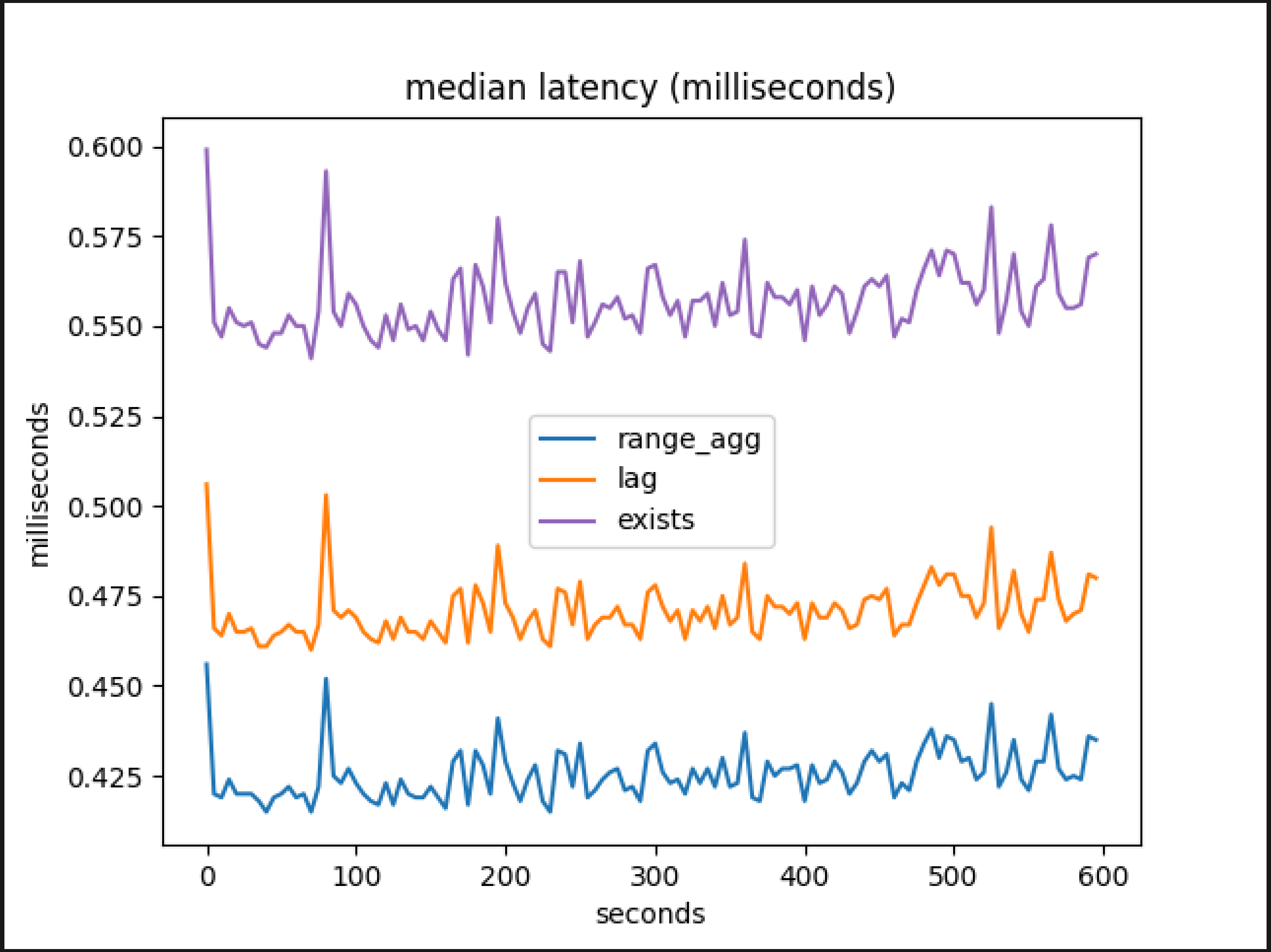
100k employees

95% LATENCY



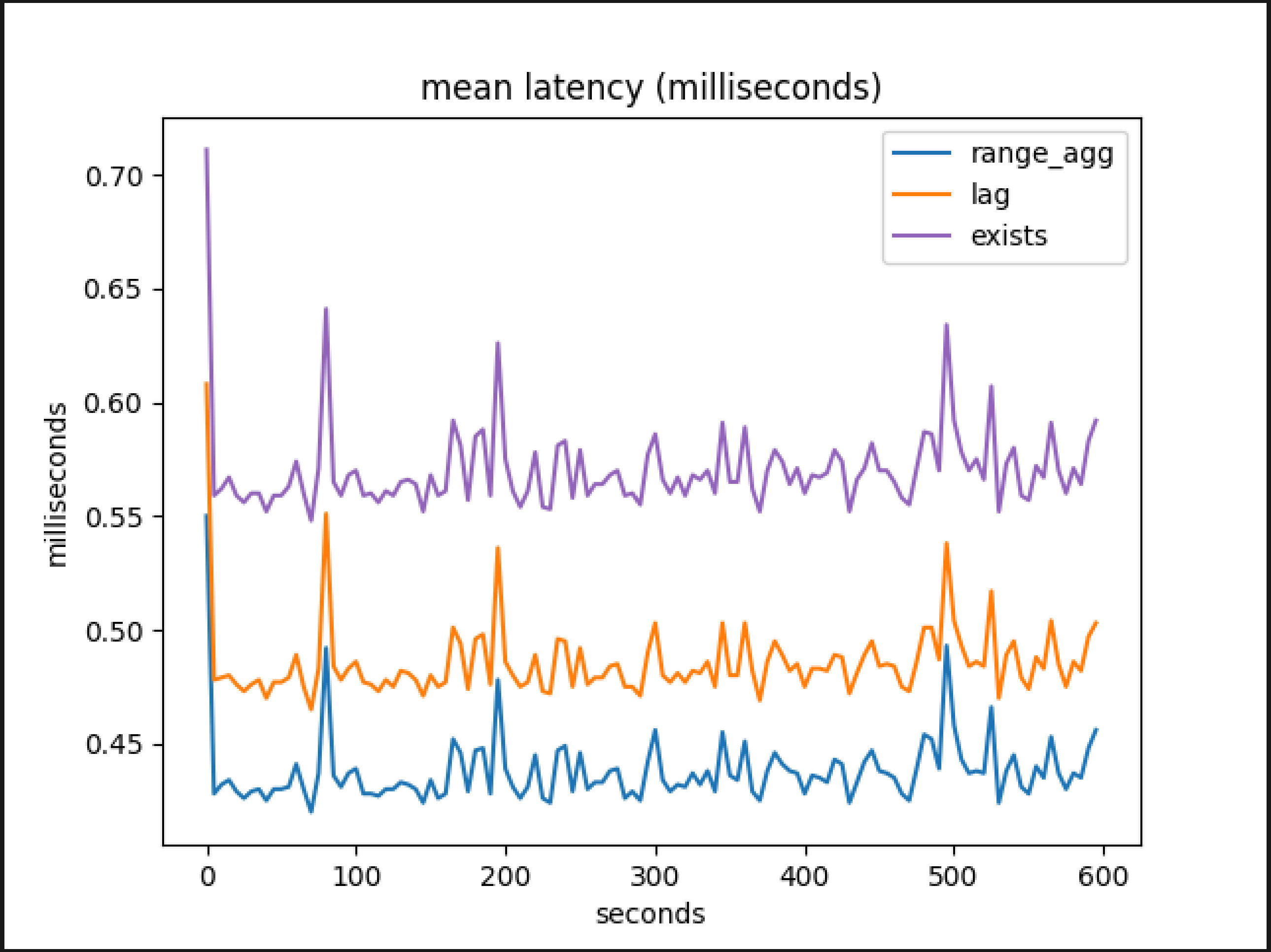
10M employees

MEDIAN LATENCY



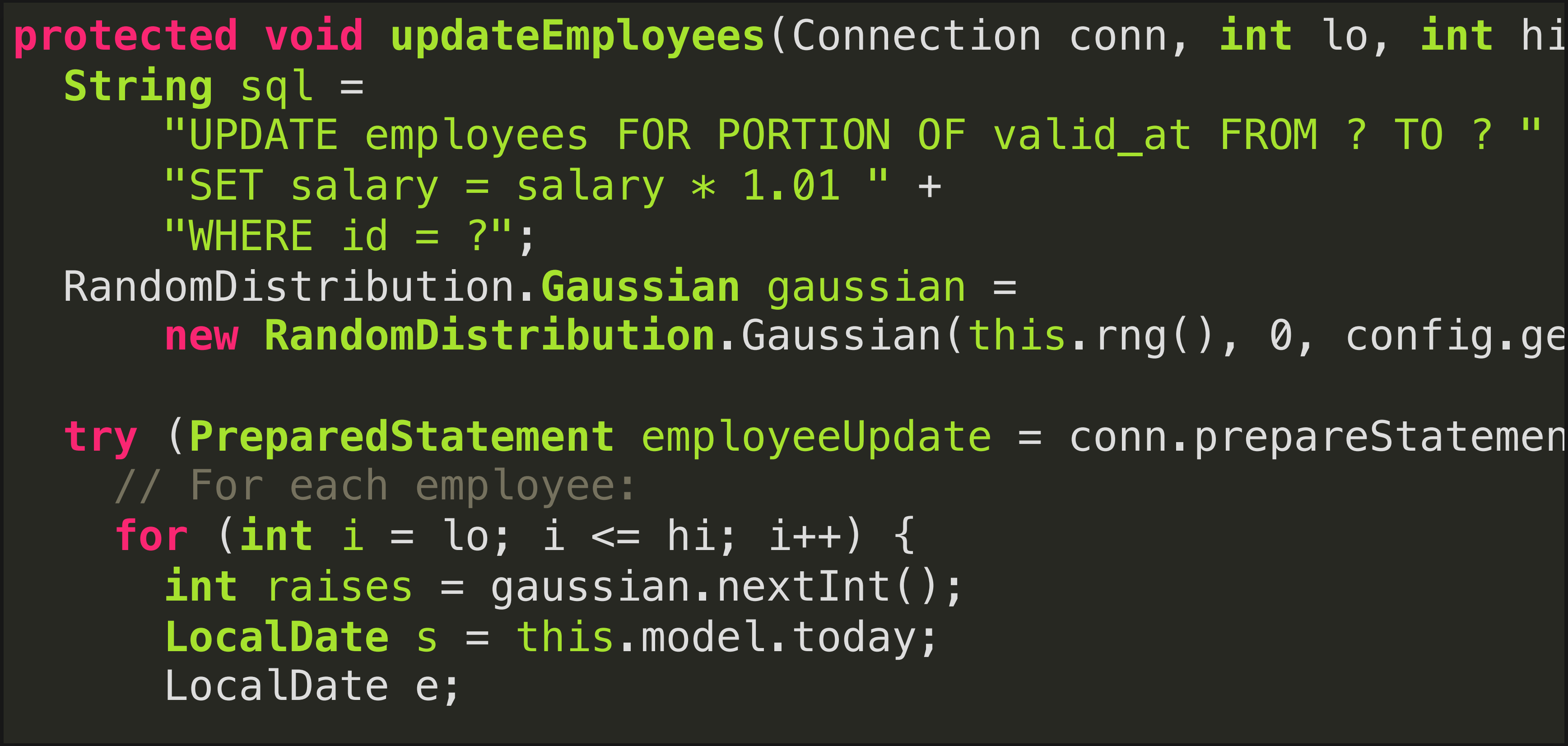
10M employees

MEAN LATENCY



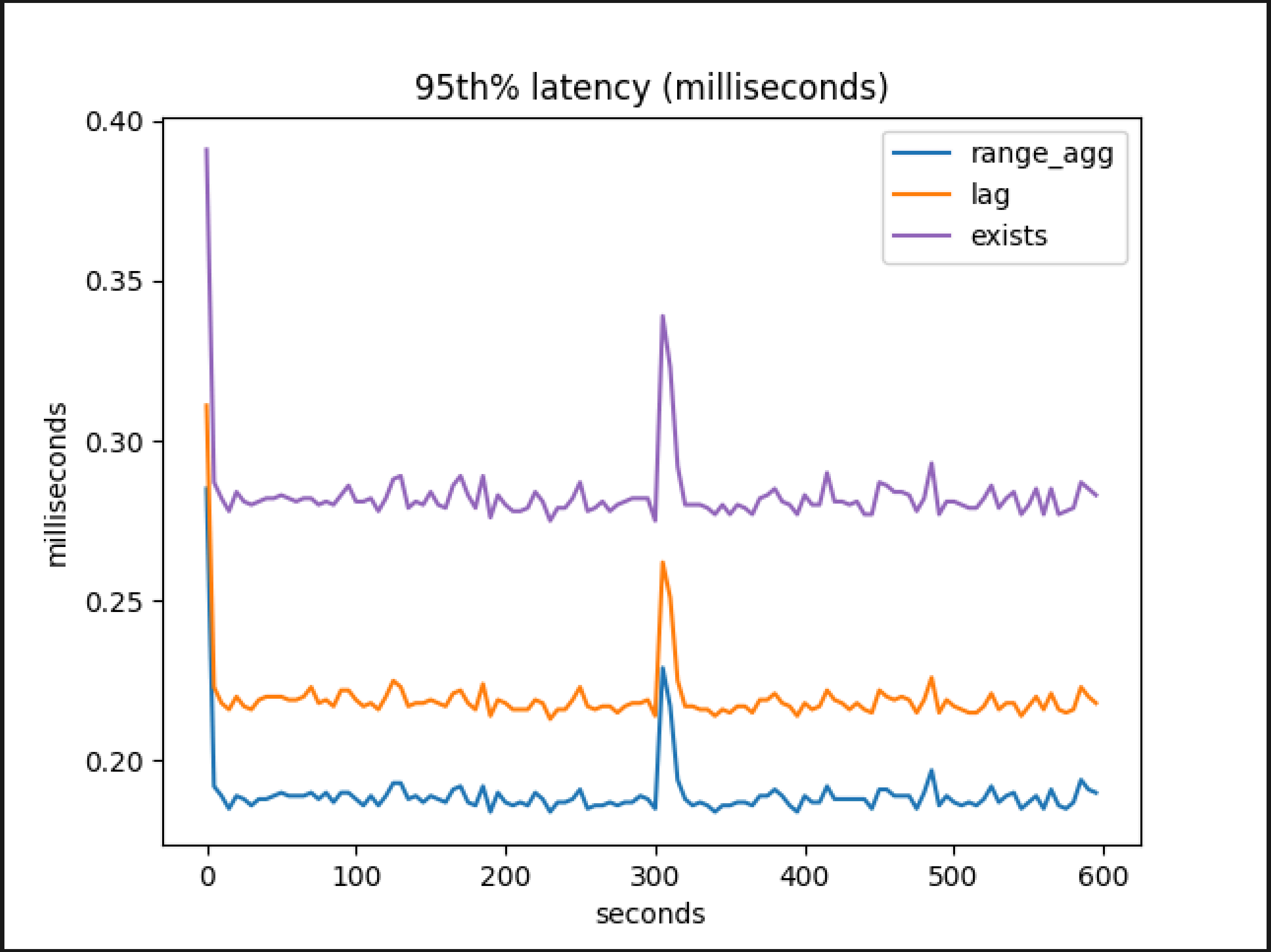
10M employees

WITH HISTORY



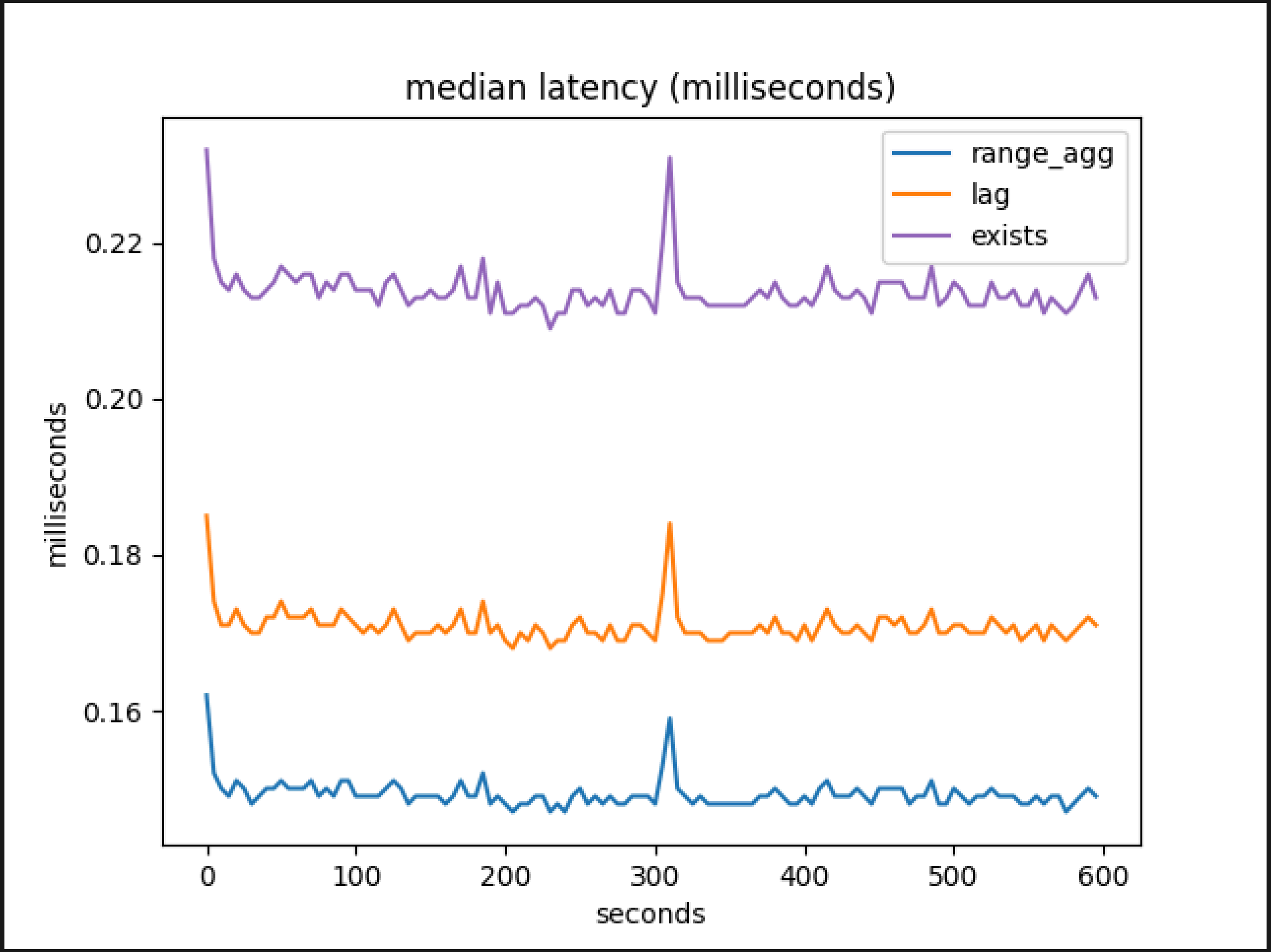
```
protected void updateEmployees(Connection conn, int lo, int hi
  String sql =
      "UPDATE employees FOR PORTION OF valid_at FROM ? TO ? "
      "SET salary = salary * 1.01 " +
      "WHERE id = ?";
  RandomDistribution.Gaussian gaussian =
      new RandomDistribution.Gaussian(this.rng(), 0, config.ge

  try (PreparedStatement employeeUpdate = conn.prepareStatemen
    // For each employee:
    for (int i = lo; i <= hi; i++) {
      int raises = gaussian.nextInt();
      LocalDate s = this.model.today;
      LocalDate e;
```

95% LATENCY



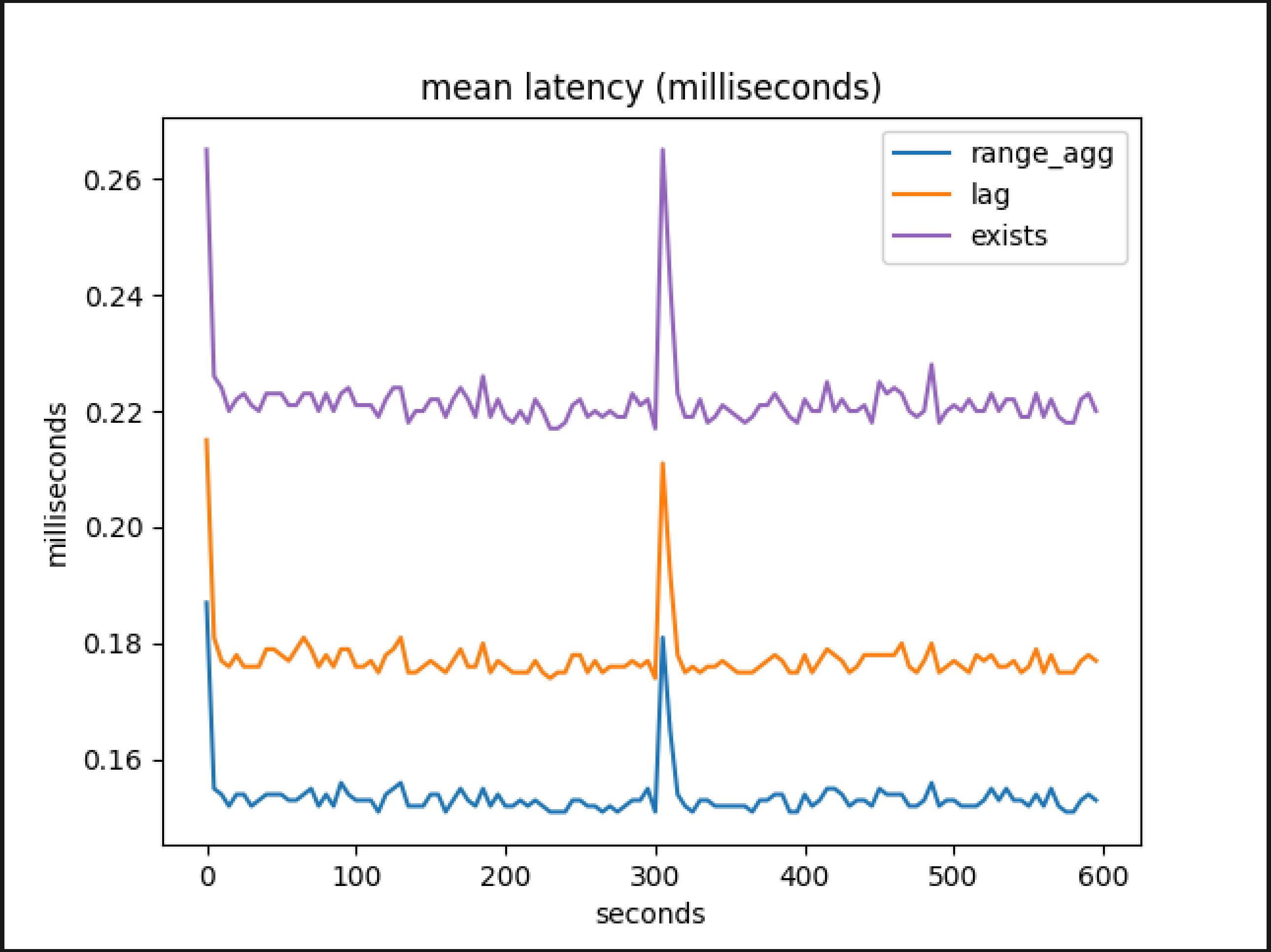
100k employees, salary history

MEDIAN LATENCY



100k employees, salary history

MEAN LATENCY



100k employees, salary history

FUTURE WORK

- More experiments
- More system stats
- `PERIOD`s
- Compare to other RDBMSes

REFERENCES

- https://github.com/pjungwir/introducing-a-temporal-benchmark-in-benchbase
- https://dl.acm.org/doi/10.1145/2213836.2213886
- https://www.vldb.org/pvldb/vol7/p277-difallah.pdf
- https://github.com/cmu-db/benchbase
- https://github.com/pjungwir/benchbase/tree/temporal
- https://github.com/pjungwir/temporal_ops
- https://illuminatedcomputing.com/pages/pdxpug2024-benchbase-and-temporal-foreign-keys/
- https://github.com/pjungwir/postgresql/tree/valid-time
- https://github.com/pjungwir/benchmarking-temporal-tables

THANK YOU

https://github.com/pjungwir/introducing-a-temporal-benchmark-in-benchbase