(Re)writing your SQL for optimal performance

Luca Veronese

Freelance consultant

Some projects I have worked on

- Italian Chambers of Commerce:
  - Reporting engine (Visure/Certificati)
  - Replication solution Oracle/Unix to DB2/MVS
- 2000: one of the first browser based business application development tools
- Java ERP development framework
- NebulaERP, italian ERP software
- Database Events notification system
- heterogeneous, over the Internet, data replication solution
- IoT: PostgreSQL based Cloud API server

SQL Performance facets

- Database design
  - Logical, physical (indexing)
- Application design
  - How you interact with the rdbms (ORM?!)
  - Functionality used (e.g. prepared statements, batching...)
- Database tuning and management (e.g. statistics, vacuum), O/S tuning
- Optimizer related issues

SQL is a declarative language

- RDBMS reads your code, parses it and feeds the result to the optimizer
- The optimizer figures out the best plan to execute your query
- SQL optimization is a NP-complete problem
- Optimizers are imperfect so sometimes they return (slow) sub-optimal plans

Manual SQL optimization

- Some RDBMS allow hints to be fed to the planner
- PostgreSQL does not implement hints (proudly)
- BUT, most of the times, we can work around the "issue" by rewriting the SQL
- We will see two real world examples

Reading a PostgreSQL plan

- A plan is a tree shaped structure
- Leaves correspond to data access operations (tables and indexes)
- Intermediate nodes correspond to operators applied to child nodes (e.g. join, sort…)
- To visualize plans I will use PEV from Alex Tatiyants, a handy tool for the job
- Use EXPLAIN (ANALYZE, COSTS, VERBOSE, BUFFERS, FORMAT JSON) to generate JSON plan description
- Paste result into browser based UI

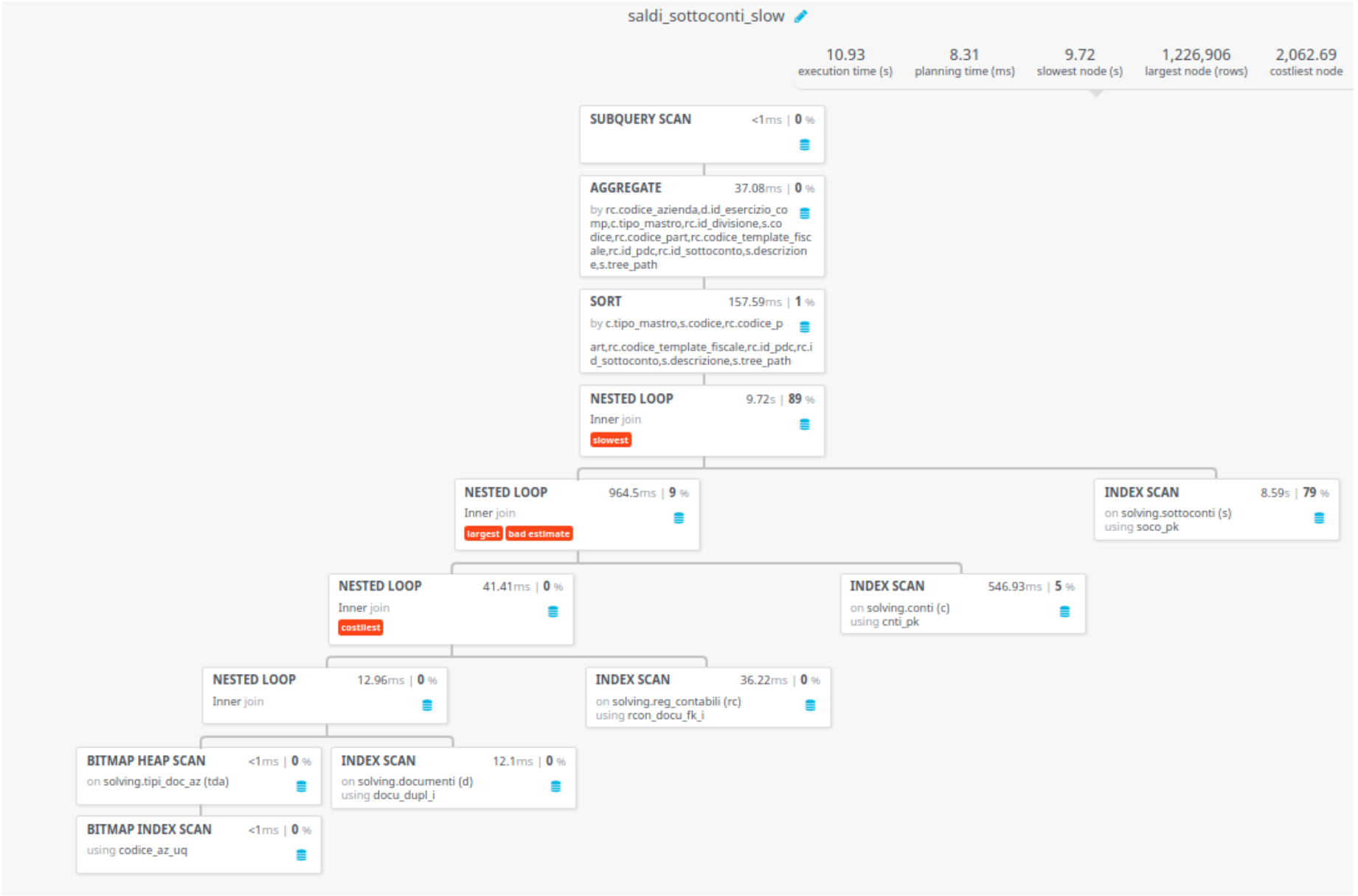
Example 1: The original SQL

```
CREATE or replace VIEW saldi_sottoconti AS
SELECT rc.codice_azienda
    , CASE
        WHEN rc.id_divisione IS NULL
        THEN '-'
        ELSE rc.id_divisione::text
      END || '#' || d.id_esercizio_comp::text || '#' || rc.id_sottoconto::
    , rc.id_divisione
    , d.id_esercizio_comp
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') = 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare_ch
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') = 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere_ch
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '1'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare_ec
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '1'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere_ec
      , c.tipo_mastro
      , rc.codice_part
      , rc.codice_template_fiscale
      , rc.id_pdc
      , rc.id_sottoconto
      , s.tree_path
--      , cee.tree_path || '|' || s.codice as tree_path_cee
FROM documenti d
join tipi_doc_az tda
  on tda.codice_azienda = d.codice_azienda
 and tda.codice_tipo_doc = d.codice_tipo_doc
 and tda.codice_tipo_doc_az = d.codice_tipo_doc_az
JOIN reg_contabili rc
  ON d.codice_azienda = rc.codice_azienda
 AND d.id_documento = rc.id_documento
JOIN sottoconti s
  ON rc.codice_part = s.codice_part
 AND rc.codice_template_fiscale = s.codice_template_fiscale
 AND rc.id_pdc = s.id_pdc
 AND rc.id_sottoconto = s.id_sottoconto
JOIN conti c
  ON s.codice_part = c.codice_part
 AND s.codice_template_fiscale = c.codice_template_fiscale
 AND s.id_pdc = c.id_pdc
 AND s.id_conto = c.id_conto
--left join cee cee
--       on cee.id_cee = s.id_cee
WHERE d.stato = 'C'
GROUP BY rc.codice_azienda
       , d.id_esercizio_comp
       , c.tipo_mastro
       , rc.id_divisione
       , s.codice
       , rc.codice_part
       , rc.codice_template_fiscale
       , rc.id_pdc
       , rc.id_sottoconto
       , s.descrizione
       , s.tree_path
```

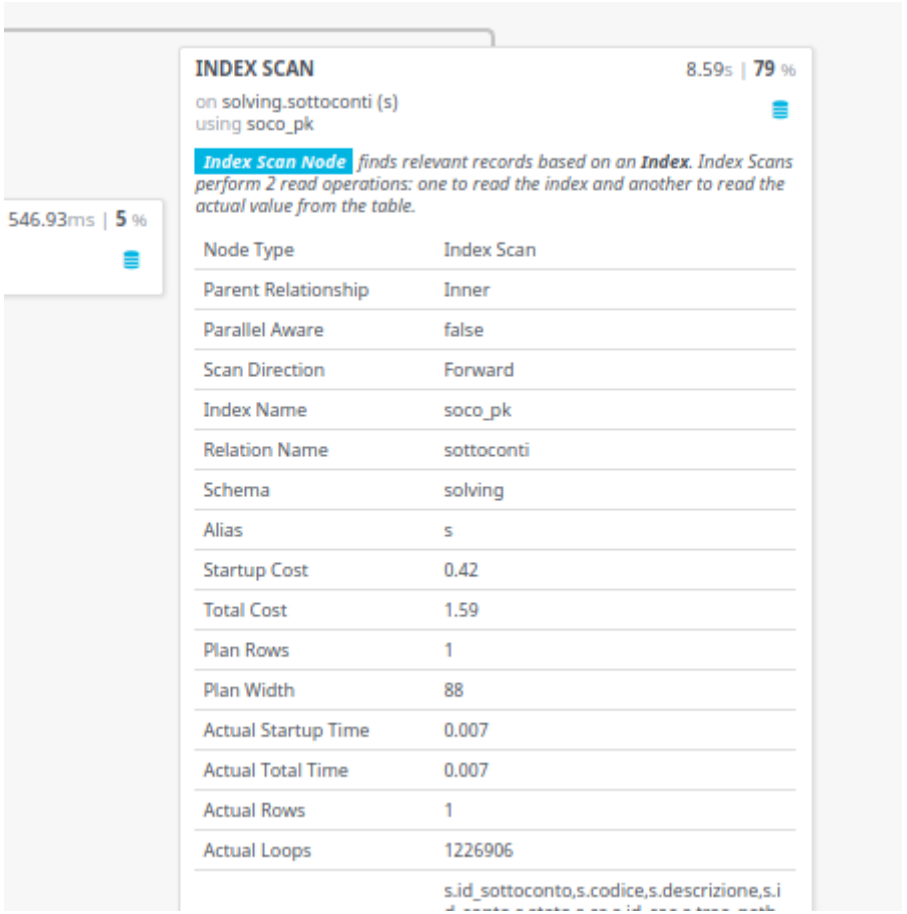
The query

```
SELECT *
FROM saldi_sottoconti
WHERE
    codice_azienda = 'S0040SC'
AND id_esercizio_comp = 2017
AND id_divisione = 2
```

The (slow) plan



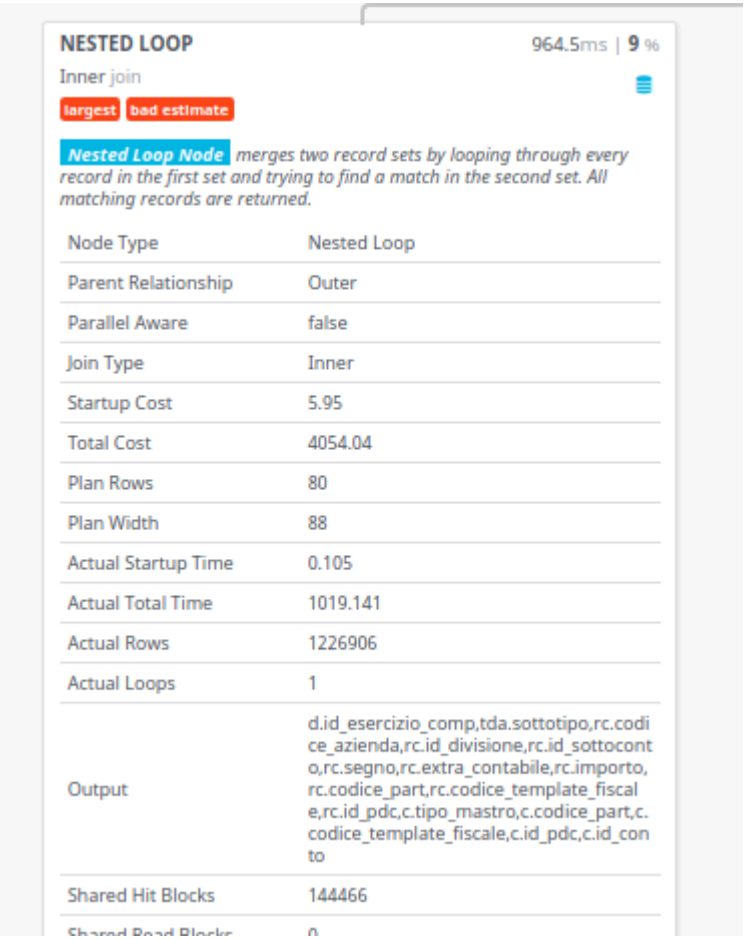
Have a look at the slowest node



Index scan should be fast and it is (0.007 ms)
But this node is executed 1,226,906 times (See Actual Loops)

So the problem is caused by the left node in the join operation that is feeding too much data into this node

This node generates 1.2 million rows



NESTED LOOP 964.5ms | 9 %

Inner join

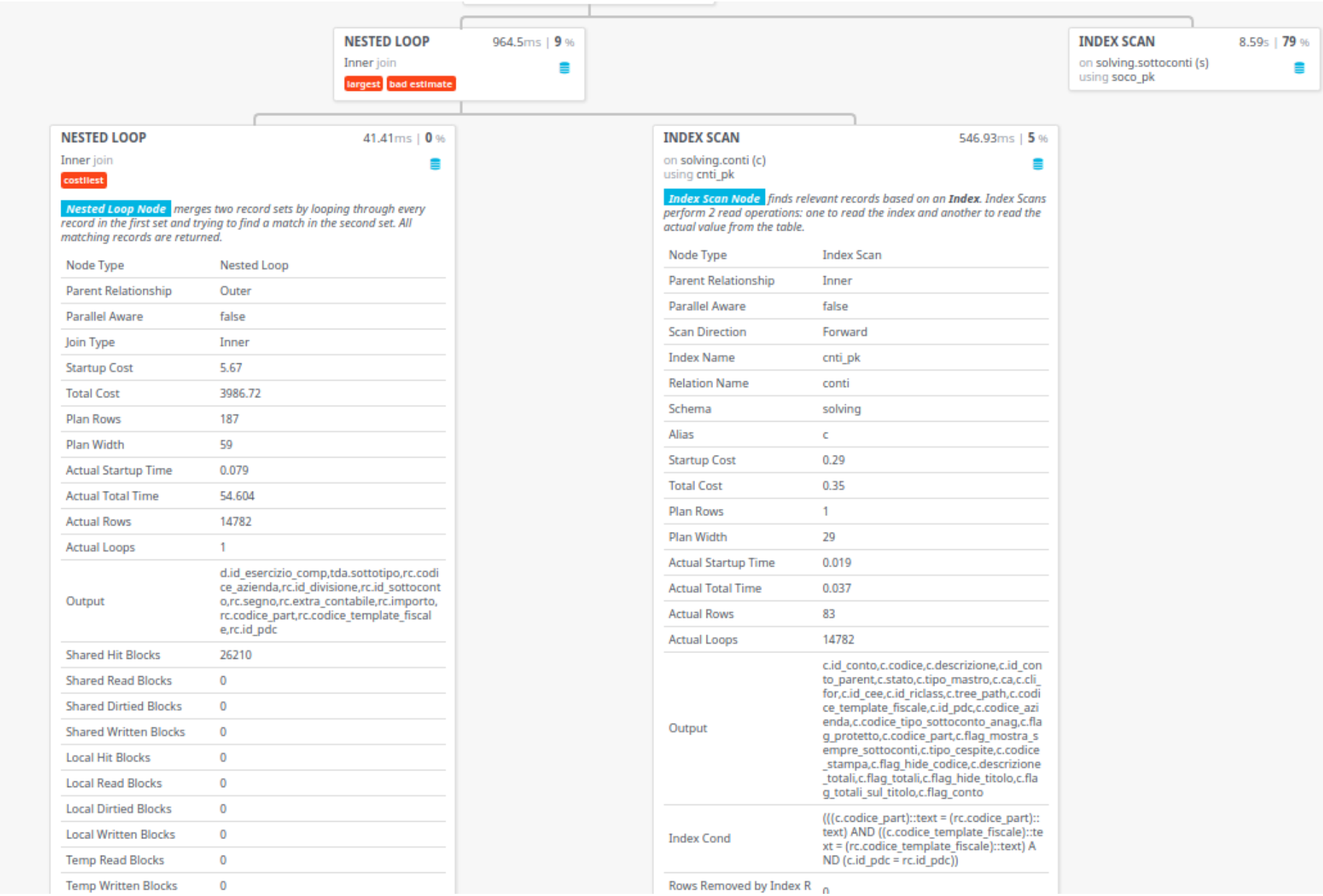
largest | bad estimate

Nested Loop Node merges two record sets by looping through every record in the first set and trying to find a match in the second set. All matching records are returned.

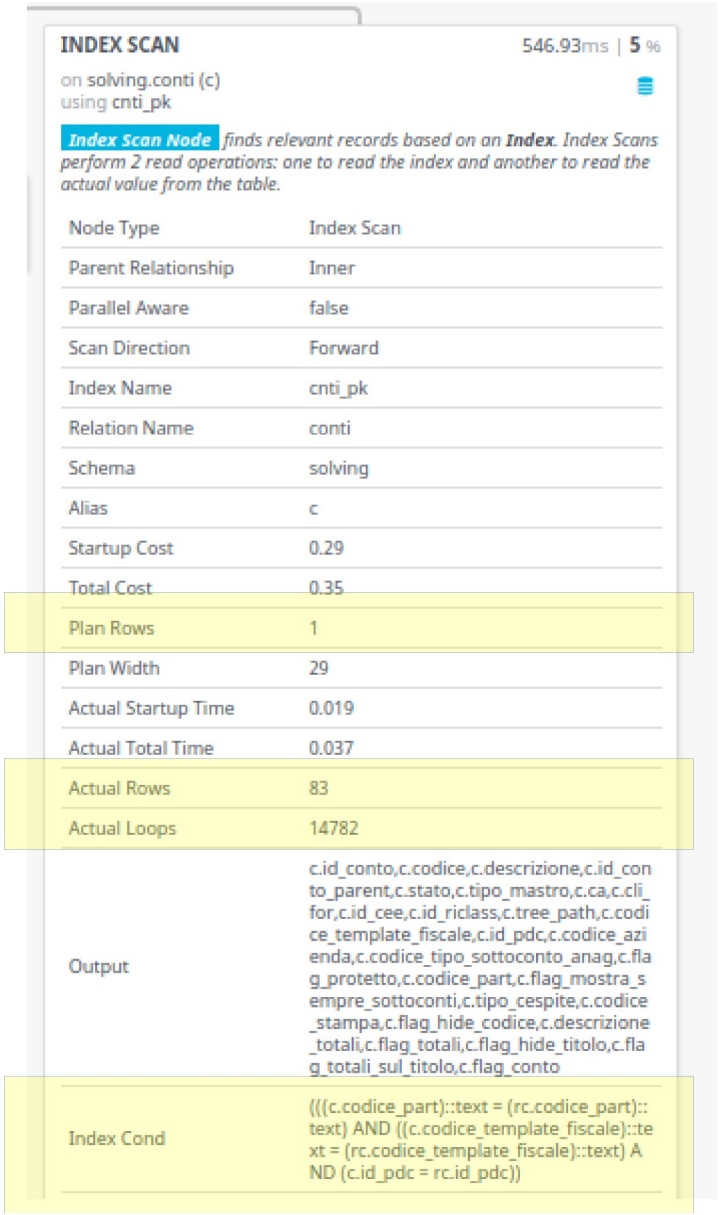
Node Type	Nested Loop
Parent Relationship	Outer
Parallel Aware	false
Join Type	Inner
Startup Cost	5.95
Total Cost	4054.04
Plan Rows	80
Plan Width	88
Actual Startup Time	0.105
Actual Total Time	1019.141
Actual Rows	1226906
Actual Loops	1
Output	d.id_esercizio_comp,tda.sottotipo,rc.codice_azienda,rc.id_divisione,rc.id_sottoconto,rc.segno,rc.extra_contabile,rc.importo,rc.codice_part,rc.codice_template_fiscale,rc.id_pdc,c.tipo_mastro,c.codice_part,c.codice_template_fiscale,c.id_pdc,c.id_conto
Shared Hit Blocks	144466
Shared Read Blocks	0

So we descend to the children nodes to further investigate

Here are the children



The right node is the culprit



INDEX SCAN 546.93ms | 5 %

on solving.conti (c)
using cnti_pk

Index Scan Node *finds relevant records based on an* ***Index****. Index Scans perform 2 read operations: one to read the index and another to read the actual value from the table.*

Node Type	Index Scan
Parent Relationship	Inner
Parallel Aware	false
Scan Direction	Forward
Index Name	cnti_pk
Relation Name	conti
Schema	solving
Alias	c
Startup Cost	0.29
Total Cost	0.35
Plan Rows	1
Plan Width	29
Actual Startup Time	0.019
Actual Total Time	0.037
Actual Rows	83
Actual Loops	14782
Output	c.id_conto,c.codice,c.descrizione,c.id_conto_parent,c.stato,c.tipo_mastro,c.ca,c.cli_for,c.id_cee,c.id_riclass,c.tree_path,c.codice_template_fiscale,c.id_pdc,c.codice_azienda,c.codice_tipo_sottoconto_anag,c.flag_protetto,c.codice_part,c.flag_mostra_sempre_sottoconti,c.tipo_cespite,c.codice_stampa,c.flag_hide_codice,c.descrizione_totali,c.flag_totali,c.flag_hide_titolo,c.flag_totali_sul_titolo,c.flag_conto
Index Cond	(((c.codice_part)::text = (rc.codice_part)::text) AND ((c.codice_template_fiscale)::text = (rc.codice_template_fiscale)::text) AND (c.id_pdc = rc.id_pdc))

From the SQL query we expect the JOIN to the conti table to be performed after the join to the sottoconti table

BUT the optimizer decides to join conti to the result of the preceding join operations before joining to sottoconti

The node uses cnti_pk (PK index) but since it does not have the sottoconti row data it can only use three of the four primary key columns

The optimizer "thinks" this will return 1 row (Plan rows) but this is NOT true (83 rows returned)

This is a BAD result I can only explain by the planner assuming statistical independence on the index columns. In this case these 3 columns are perfectly correlated!

This causes the join to produce a cartesian product between the 14,782 rows of the left node and the 83 rows of this node (14,782 * 83 = 1,226,906)

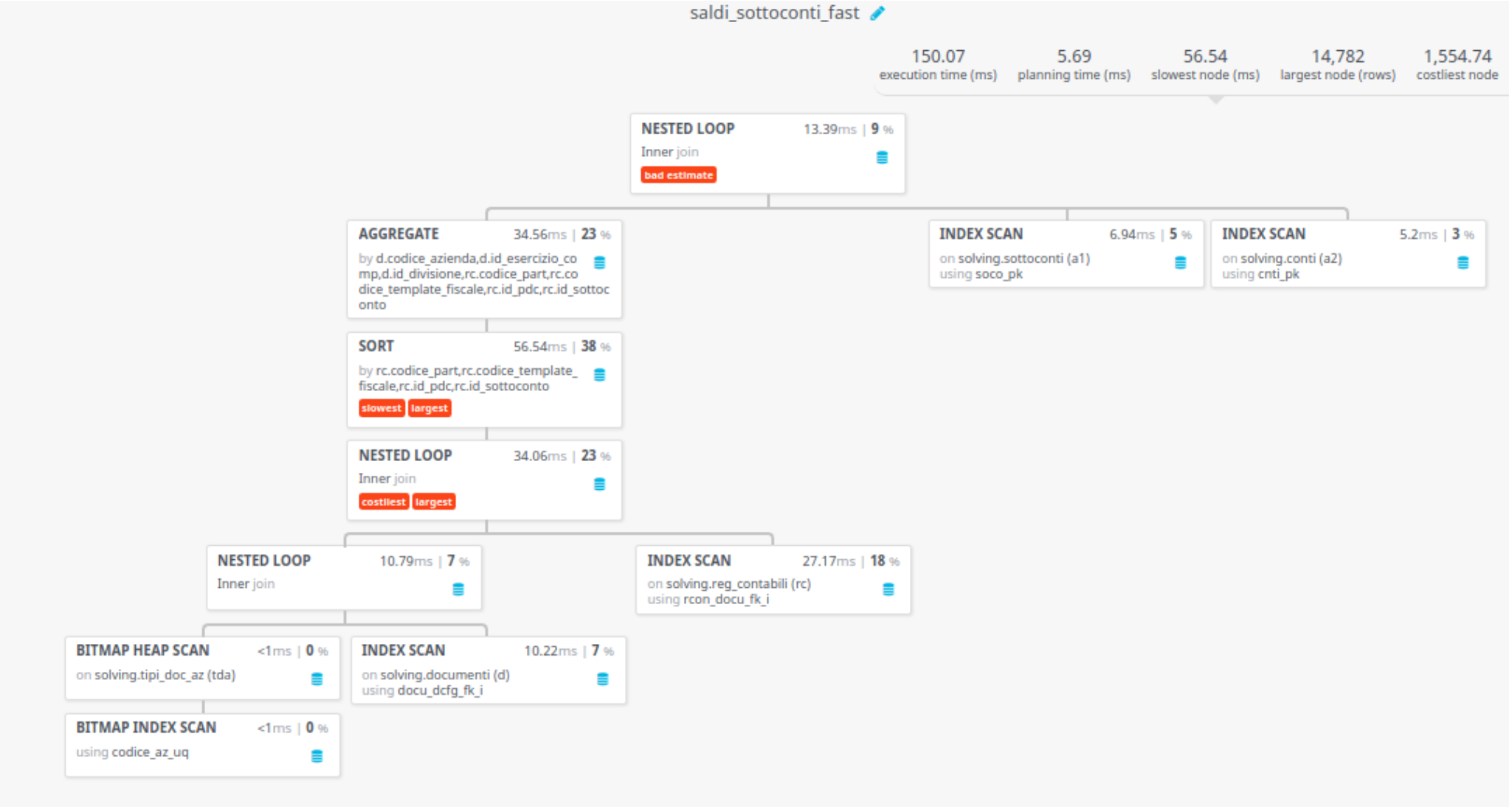
How can we rewrite the SQL?

- Observe: conti and sottoconti are not relevant for filter and aggregation
- They are only used for projection
- We can commute the aggregation with the conti and sottoconti joins
  - So we will join to a smaller cardinality set
- This can be done introducing a subquery in the FROM clause (nesting)

The rewritten SQL

```sql
CREATE or replace VIEW saldi_sottoconti AS
SELECT A0.*, A1.descrizione, A1.codice, A1.tree_path,
(SELECT A2.tipo_mastro FROM CONTI A2 WHERE A2.CODICE_PART = A1.CODICE_PART
AND A2.CODICE_TEMPLATE_FISCALE = A1.CODICE_TEMPLATE_FISCALE
AND A2.ID_PDC = A1.ID_PDC
AND A2.ID_CONTO = A1.ID_CONTO) AS tipo_mastro
FROM (
  SELECT d.codice_azienda
    , CASE
        WHEN d.id_divisione IS NULL
        THEN '-'
        ELSE d.id_divisione::text
      END || '#' || d.id_esercizio_comp || '#' || rc.id_sottoconto AS record_key
    , d.id_divisione
    , d.id_esercizio_comp
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') = 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare_ch
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '0'
              AND coalesce(tda.sottotipo, '-') = 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere_ch
    , sum(CASE
            WHEN rc.segno = 1
              AND rc.extra_contabile = '1'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_dare_ec
    , sum(CASE
            WHEN rc.segno = (-1)
              AND rc.extra_contabile = '1'
              AND coalesce(tda.sottotipo, '-') <> 'CH'
            THEN rc.importo
            ELSE 0
          END) AS totale_avere_ec
    , rc.codice_part
    , rc.codice_template_fiscale
    , rc.id_pdc
    , rc.id_sottoconto
  FROM documenti d
  join tipi_doc_az tda
    on tda.codice_azienda = d.codice_azienda
   and tda.codice_tipo_doc = d.codice_tipo_doc
   and tda.codice_tipo_doc_az = d.codice_tipo_doc_az
  JOIN reg_contabili rc
    ON d.codice_azienda = rc.codice_azienda
   AND d.id_documento = rc.id_documento
  WHERE d.stato = 'C'
  GROUP BY d.codice_azienda
         , d.id_esercizio_comp
         , d.id_divisione
         , rc.codice_part
         , rc.codice_template_fiscale
         , rc.id_pdc
         , rc.id_sottoconto
) AS A0
JOIN SOTTOCONTI A1
ON A1.CODICE_PART = A0.CODICE_PART
AND A1.CODICE_TEMPLATE_FISCALE = A0.CODICE_TEMPLATE_FISCALE
AND A1.ID_PDC = A0.ID_PDC
AND A1.ID_SOTTOCONTO = A0.ID_SOTTOCONTO
```

The optimal plan



This plan is more than 76x faster than the original

The pattern I used

- Move joins used only for projection outside the aggregation
  - need to check the rewrite preserves semantics!
- Eventually force join order using subqueries in the SELECT clause
  - Only if 1 column selected
  - Otherwise introduce additional subquery level

What about WITH?

- While WITH could be used to bring the original query to 1.6s
- It can’t be used in this VIEW since predicate pushdown does not happen for WITH (at the moment)
- The codice_azienda restriction cant’ be hard coded!
- Without the restriction the query takes >16s which is worse than the original

Another example

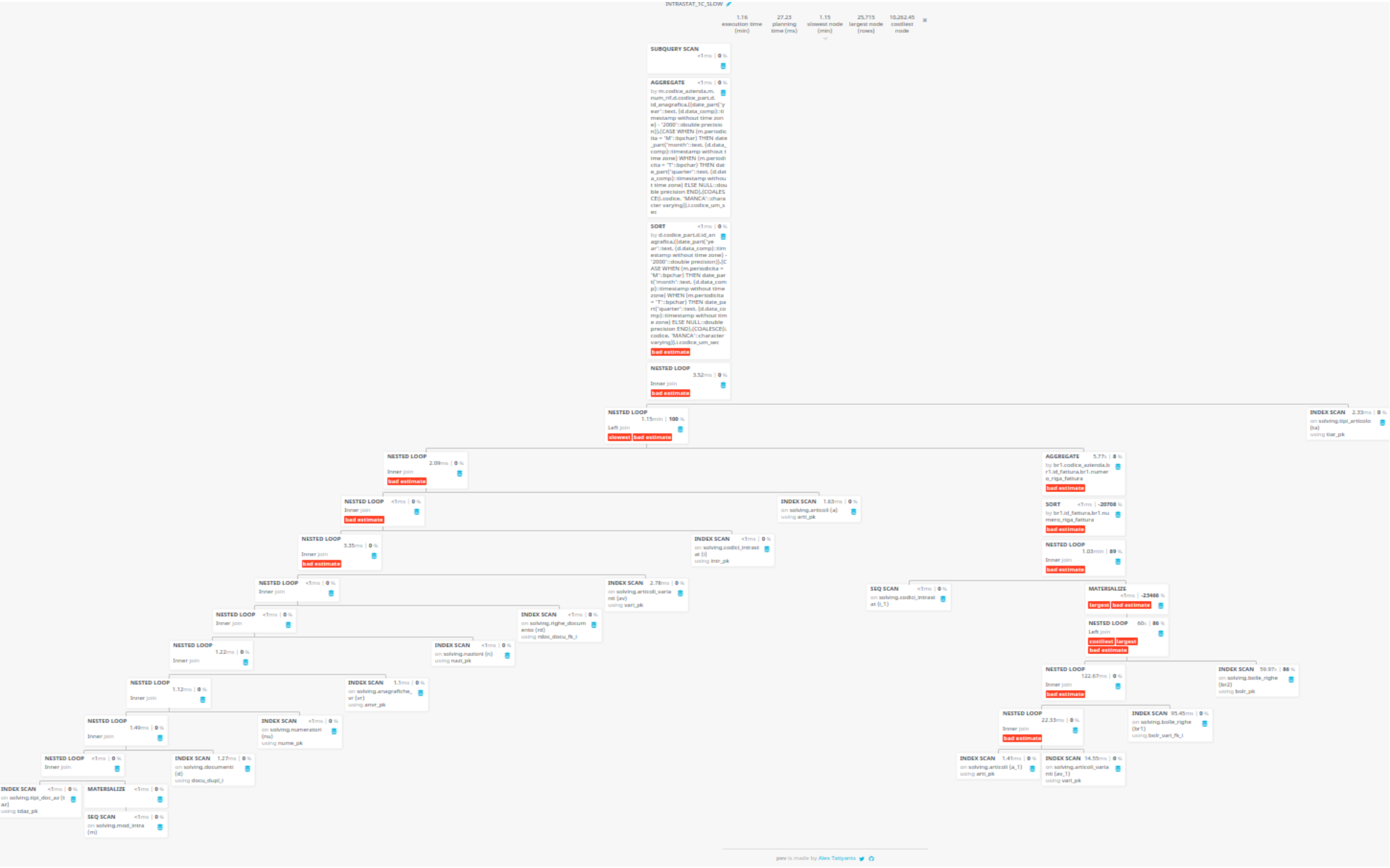
```
CREATE OR REPLACE VIEW intrastat_1c AS
SELECT m.codice_azienda,
  m.num_rif,
  d.codice_part,
  d.id_anagrafica,
  date_part('year', d.data_comp) - 2000 AS anno_com
  CASE
    WHEN m.periodicita = 'M' THEN date_part('month'
    WHEN m.periodicita = 'T' THEN date_part('quarte
  END AS periodo_comp,
  COALESCE(i.codice, 'MANCA') AS codice_intrastat,
  round(sum((rd.importo + rd.costo_trasporto_rip) *
  round(sum(br.mn1 + br.mn2), 0) AS massa_netta,
  round(sum(br.qta_um_sec), 0) AS qta_um_sec
FROM mod_intra m
JOIN documenti d ON m.codice_azienda = d.codice_az
  CASE
    WHEN m.periodicita = 'M' THEN (('01-' || m.per
    WHEN m.periodicita = 'T' THEN (('01-' || ((m.p
  END
  AND d.data_doc <
  CASE
    WHEN m.periodicita = 'M' THEN (('01-' || (mod(
    CASE
      WHEN m.periodo = 12 THEN m.anno + 1
      ELSE m.anno
    END
    WHEN m.periodicita = 'T' THEN (('01-' || (mod(
    CASE
      WHEN m.periodo = 4 THEN m.anno + 1
      ELSE m.anno
    END
  END
JOIN tipi_doc_az taz
  ON taz.codice_azienda = d.codice_azienda
  AND taz.codice_tipo_doc = d.codice_tipo_doc
  AND taz.codice_tipo_doc_az = d.codice_tipo_doc_a
  AND taz.codice_tipo_doc = 'VEN'
JOIN numeratori nu
  ON nu.codice_azienda = d.codice_azienda
  AND nu.anno = d.anno
  AND nu.id_numeratore = d.id_numeratore
  AND nu.tipo_protocollo <> 2
JOIN anagrafiche_vr vr
  ON vr.codice_part = d.codice_part
  AND vr.id_anagrafica = d.id_anagrafica
  AND vr.prog_vr = d.prog_vr_anag
JOIN nazioni n
  ON n.codice_nazione = vr.codice_nazione
  AND n.cee = '1'
  AND n.codice_nazione <> 'ITA'
JOIN righe_documento rd
  ON rd.codice_azienda = d.codice_azienda
  AND rd.id_documento = d.id_documento
  AND rd.tipo_omaggio IS NULL
JOIN articoli_varianti av
  ON av.codice_azienda = rd.codice_azienda
  AND av.id_articolo = rd.id_articolo
  AND av.codice_variante = rd.codice_variante
JOIN codici_intrastat i
  ON av.intra_id = i.intra_id
JOIN articoli a
  ON a.codice_azienda = av.codice_azienda
  AND a.id_articolo = av.id_articolo
JOIN tipi_articolo ta
  ON ta.codice_azienda = a.codice_azienda
  AND ta.codice_tipo_articolo = a.codice_tipo
  AND ta.tipo = 'M'
LEFT JOIN (
  SELECT br1.codice_azienda,
         br1.id_fattura,
         br1.numero_riga_fattura,
         COALESCE(sum(
           CASE
             WHEN br1.peso > 0 THEN br1.peso
             WHEN av_1.peso_unitario IS NOT NULL AND av_1.um_peso =
             WHEN a_1.codice_um = 'KG' THEN br1.qta
             WHEN br1.codice_um_acqven = 'KG' THEN br1.qta_acquisto
             WHEN a_1.codice_um_acquisto = 'KG' THEN br1.qta * av_1.
           END), 0) AS mn1,
         COALESCE(sum(
           CASE
             WHEN br2.peso > 0 THEN br2.peso
             WHEN br2.codice_um_acqven = 'KG' THEN br2.qta_acquisto
           END), 0) AS mn2,
         sum(
           CASE
             WHEN a_1.codice_um = i_1.codice_um_sec THEN br1.qta
             WHEN br1.codice_um_acqven = i_1.codice_um_sec THEN br1.
             WHEN a_1.codice_um_acquisto = i_1.codice_um_sec THEN br
           END) AS qta_um_sec
  FROM bolle_righe br1
  JOIN articoli_varianti av_1
    ON av_1.codice_azienda = br1.codice_azienda
    AND av_1.id_articolo = br1.id_articolo
    AND av_1.codice_variante = br1.codice_variante
  JOIN articoli a_1
    ON a_1.codice_azienda = av_1.codice_azienda
    AND a_1.id_articolo = av_1.id_articolo
  JOIN codici_intrastat i_1
    ON av_1.intra_id = i_1.intra_id
  LEFT JOIN bolle_righe br2
    ON br2.codice_azienda = br1.codice_azienda
    AND br2.numero_bolla = br1.numero_bolla
    AND br2.numero_riga_padre = br1.numero_riga
    AND br2.numero_riga_padre IS NOT NULL
  WHERE br1.numero_riga_padre IS NULL
  GROUP BY br1.codice_azienda, br1.id_fattura, br1.numero_riga_
) br
ON br.codice_azienda = rd.codice_azienda
AND br.id_fattura = rd.id_documento
AND br.numero_riga_fattura = rd.id_riga_documento
WHERE m.tipo_riepilogo = 'C'
GROUP BY m.codice_azienda, m.num_rif, d.codice_part, d.id_anagra
  date_part('year', d.data_comp) - 2000,
  CASE
    WHEN m.periodicita = 'M' THEN date_part('month', d.data_comp
    WHEN m.periodicita = 'T' THEN date_part('quarter', d.data_co
  END, COALESCE(i.codice, 'MANCA'), i.codice_um_sec;
```

The query

```
SELECT *
FROM intrastat_1c
WHERE codice_azienda = 'S0032IC'
AND num_rif = 451
```

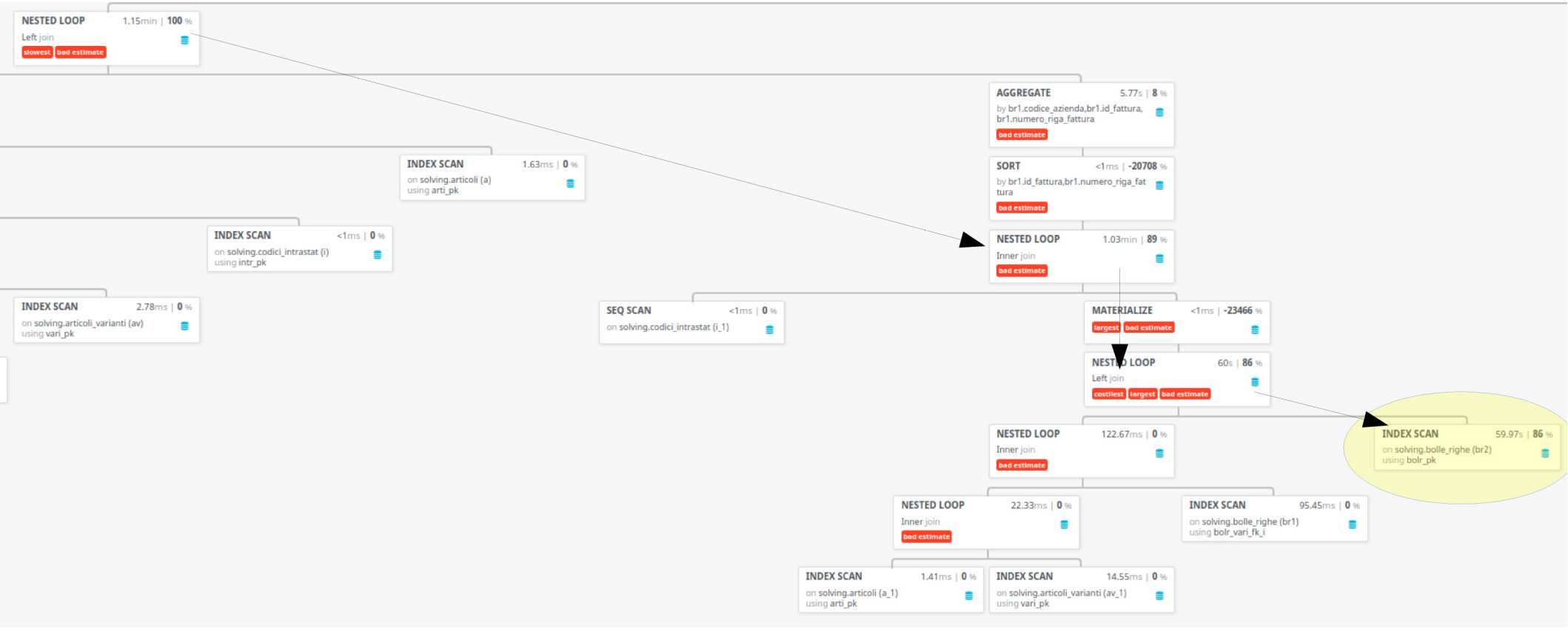
This is the PRIMARY KEY of the mod_intra table

The plan



This is the slow plan. Takes 1 min and 16 seconds.

Let’s zoom in


NESTED LOOP 1.15min | 100 %
Left join
slowest bad estimate
INDEX SCAN 1.63ms | 0 %
on solving.articoli (a)
using arti_pk
INDEX SCAN <1ms | 0 %
on solving.codici_intrastat (i)
using intr_pk
INDEX SCAN 2.78ms | 0 %
on solving.articoli_varianti (av)
using vari_pk
AGGREGATE 5.77s | 8 %
by br1.codice_azienda,br1.id_fattura, br1.numero_riga_fattura
bad estimate
SORT <1ms | -20708 %
by br1.id_fattura,br1.numero_riga_fattura
bad estimate
NESTED LOOP 1.03min | 89 %
Inner join
bad estimate
SEQ SCAN <1ms | 0 %
on solving.codici_intrastat (i_1)
MATERIALIZE <1ms | -23466 %
largest bad estimate
NESTED LOOP 60s | 86 %
Left join
costliest largest bad estimate
NESTED LOOP 122.67ms | 0 %
Inner join
bad estimate
INDEX SCAN 59.97s | 86 %
on solving.bolle_righe (br2)
using bolr_pk
NESTED LOOP 22.33ms | 0 %
Inner join
bad estimate
INDEX SCAN 95.45ms | 0 %
on solving.bolle_righe (br1)
using bolr_vari_fk_i
INDEX SCAN 1.41ms | 0 %
on solving.articoli (a_1)
using arti_pk
INDEX SCAN 14.55ms | 0 %
on solving.articoli_varianti (av_1)
using vari_pk

Continue zooming...

INDEX SCAN 59.97s | 86 %

on solving.bolle_righe (br2)
using bolr_pk

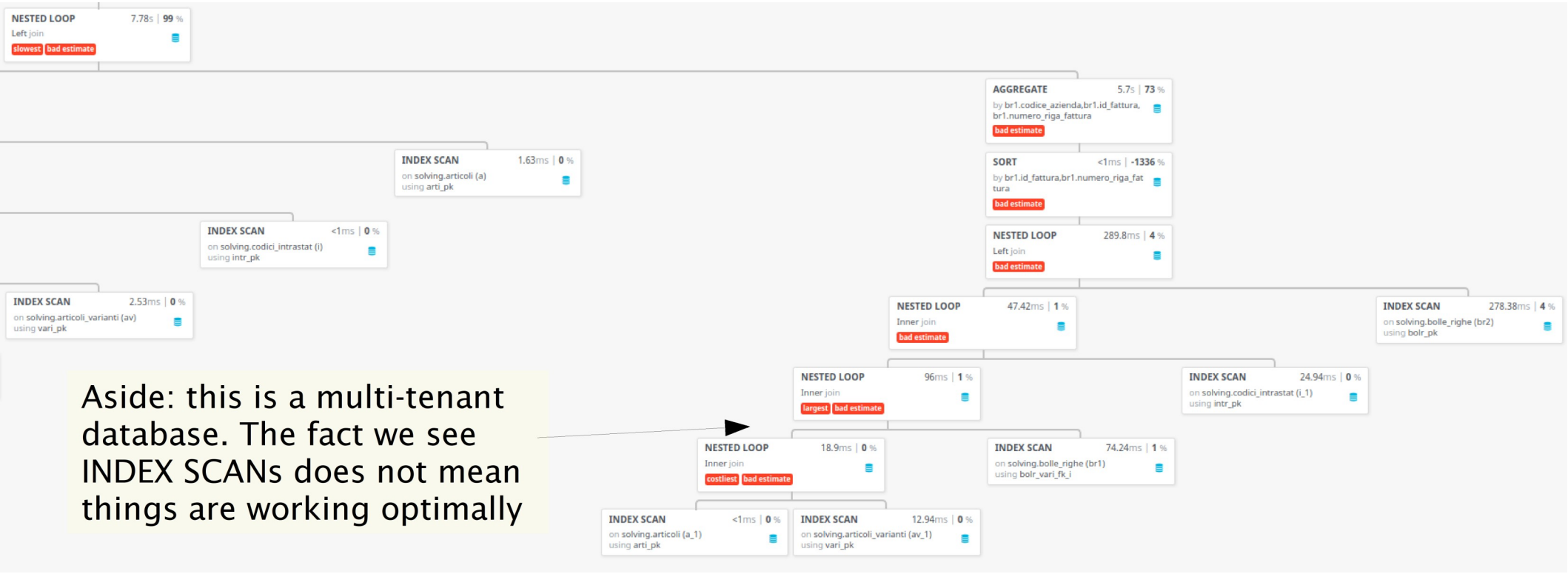
Index Scan Node *finds relevant records based on an* ***Index****. Index Scans perform 2 read operations: one to read the index and another to read the actual value from the table.*

Node Type	Index Scan
Parent Relationship	Inner
Parallel Aware	false
Scan Direction	Forward
Index Name	bolr_pk
Relation Name	bolle_righe
Schema	solving
Alias	br2
Startup Cost	0.42
Total Cost	331.04
Plan Rows	1
Plan Width	34
Actual Startup Time	2.369
Actual Total Time	2.405
Actual Rows	0
Actual Loops	24936
Index Cond	(((br2.codice_azienda)::text = (br1.codice_azienda)::text) AND ((br2.codice_azienda)::text = 'S0032IC'::text) AND ((br2.numero_bolla)::text = (br1.numero_bolla)::text))
Rows Removed by Index Recheck	0
Filter	((br2.numero_riga_padre IS NOT NULL) AND (br2.numero_riga_padre = br1.numero_riga))
Rows Removed by Filter	10
Shared Hit Blocks	4242705

Here the developer had the join wrong and forgot to add a condition. This causes the index to be used inefficiently (see large value of Shared Hit Blocks).

So we fix the query adding the missing clause.

This is MUCH better



Aside: this is a multi-tenant database. The fact we see INDEX SCANs does not mean things are working optimally

This fixed version talkes 7.83 seconds. Are we done?
We can observe that the query has two main branches and the right one depicted here actually does not depend on the restriction criteria we have in the query. PostgreSQL does its best to optimize this subquery but here we have a case of linear complexity where the query time is going to grow steadily as more data is added to the bolle_righe table. This is going to be a problem in the future assuming the current performance result is acceptable.

We should rewrite the query so that the right branch can be bound in size depending on the results from the left branch which actually depends on the query restriction.

The strategy

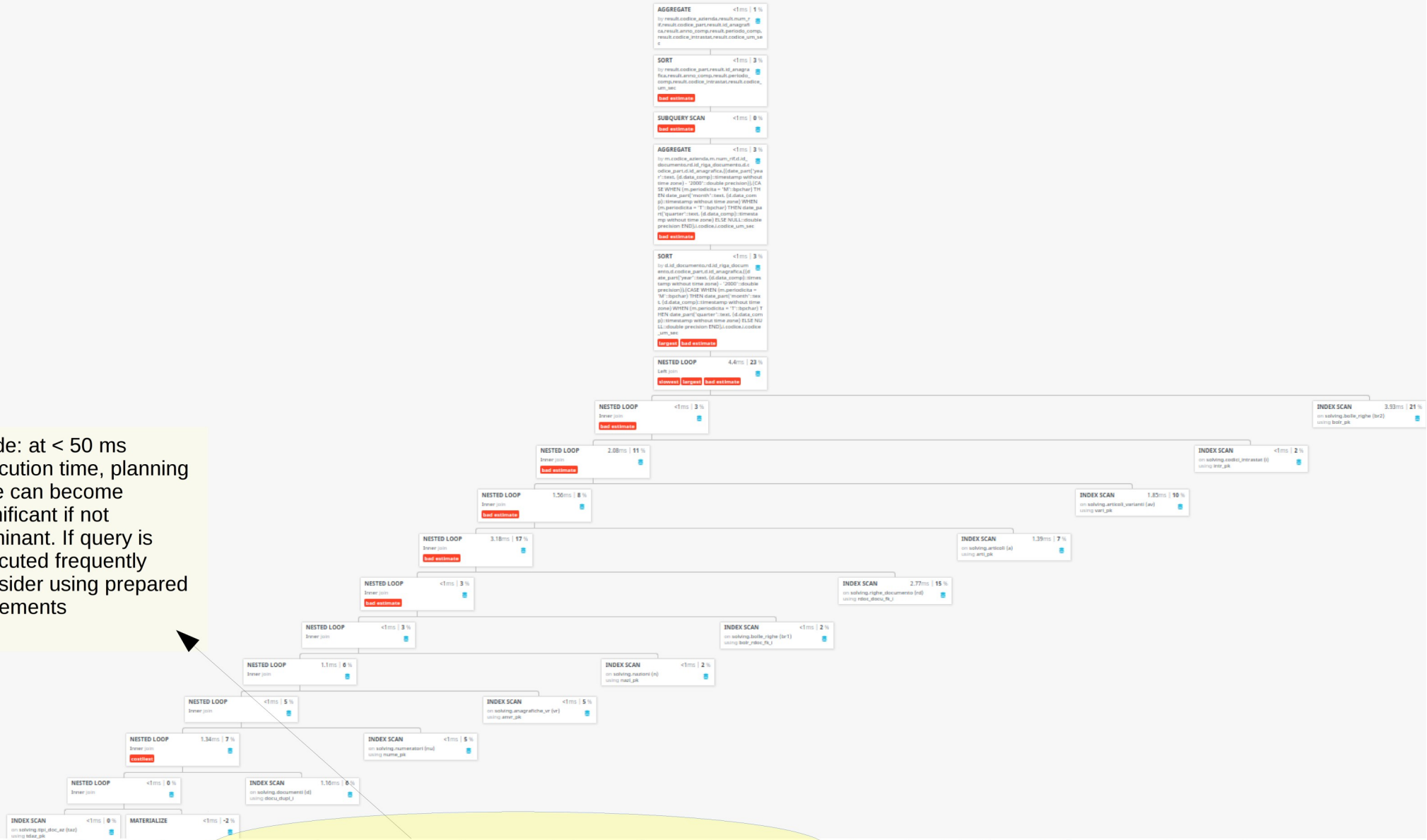
- Rewrite the query to force a particular join order
- This can be done using the nesting trick, using subqueries in the FROM clause to our advantage
- The rewritten query is more complex sintactically and less readable, but is much faster, as we will see

The new VIEW definition

```
CREATE OR REPLACE VIEW intrastat_1c AS
SELECT codice_azienda,
       num_rif,
       codice_part,
       id_anagrafica,
       anno_comp,
       periodo_comp,
       codice_intrastat,
       codice_um_sec,
       ROUND(SUM(ammontare), 0) AS ammontare,
       ROUND(SUM(peso), 0) AS massa_netta,
       ROUND(SUM(qta_um_sec), 0) AS qta_um_sec
FROM (
  SELECT t1.codice_azienda,
         t1.num_rif,
         t1.id_documento,
         t1.id_riga_documento,
         t1.codice_part,
         t1.id_anagrafica,
         t1.anno_comp,
         t1.periodo_comp,
         i.codice AS codice_intrastat,
         i.codice_um_sec,
         MIN(t1.ammontare) AS ammontare,
         sum(COALESCE(CASE
           WHEN br1.peso > 0 THEN br1.peso
           WHEN av.peso_unitario IS NOT NULL
             AND av.um_peso = 'KG' THEN av.peso_unitario *
           WHEN a.codice_um = 'KG' THEN br1.qta
           WHEN br1.codice_um_acqven = 'KG' THEN br1.qta_a
           WHEN a.codice_um_acquisto = 'KG' THEN br1.qta *
         END,
         CASE
           WHEN br2.peso > 0 THEN br2.peso
           WHEN br2.codice_um_acqven = 'KG' THEN br2.qta_a
         END, 0)) AS peso,
         sum(COALESCE(CASE
           WHEN a.codice_um = i.codice_um_sec THEN br1.qta
           WHEN br1.codice_um_acqven = i.codice_um_sec THE
           WHEN a.codice_um_acquisto = i.codice_um_sec THE
       END, 0)) AS qta_um_sec
  FROM(
    SELECT m.codice_azienda,
           m.num_rif,
           d.id_documento,
           rd.id_riga_documento,
           d.codice_part,
           d.id_anagrafica,
           date_part('year', d.data_comp) - 2000::double
           CASE
             WHEN m.periodicita = 'M' THEN date_part('mor
             WHEN m.periodicita = 'T' THEN date_part('qua
           END AS periodo_comp,
           rd.importo + rd.costo_trasporto_rip * taz.segno
    FROM mod_intra m
    JOIN documenti d ON m.codice_azienda = d.codice_azienda
    AND d.data_doc >= CASE
        WHEN m.periodicita = 'M' THEN (('01-' || m.periodo)
        WHEN m.periodicita = 'T' THEN (('01-' || ((m.periodc
      END::date
    AND d.data_doc < CASE
      WHEN m.periodicita = 'M' THEN (('01-' || (mod(m.periodc
        CASE
          WHEN m.periodo = 12 THEN m.anno + 1
          ELSE m.anno
        END
      WHEN m.periodicita = 'T' THEN (('01-' || (mod(m.periodc
        CASE
          WHEN m.periodo = 4 THEN m.anno + 1
          ELSE m.anno
        END
     END::date
    JOIN tipi_doc_az taz
    ON taz.codice_azienda = d.codice_azienda
    AND taz.codice_tipo_doc = d.codice_tipo_doc
    AND taz.codice_tipo_doc_az = d.codice_tipo_doc_az
    AND taz.codice_tipo_doc = 'VEN'
    JOIN numeratori nu
    ON nu.codice_azienda = d.codice_azienda
    AND nu.anno = d.anno
    AND nu.id_numeratore = d.id_numeratore
    AND nu.tipo_protocollo <> 2
    JOIN anagrafiche_vr vr
    ON vr.codice_part = d.codice_part
    AND vr.id_anagrafica = d.id_anagrafica
    AND vr.prog_vr = d.prog_vr_anag
    JOIN nazioni n
    ON n.codice_nazione = vr.codice_nazione
    AND n.cee = '1'
    AND n.codice_nazione <> 'ITA'
    JOIN righe_documento rd
    ON rd.codice_azienda = d.codice_azienda
    AND rd.id_documento = d.id_documento
    AND rd.tipo_omaggio IS NULL
    WHERE m.tipo_riepilogo = 'C'
  ) AS t1
  JOIN bolle_righe br1
  ON br1.codice_azienda = t1.codice_azienda
  AND br1.id_fattura = t1.id_documento
  AND br1.numero_riga_fattura = t1.id_riga_documento
  JOIN articoli_varianti av
  ON av.codice_azienda = br1.codice_azienda
  AND av.id_articolo = br1.id_articolo
  AND av.codice_variante = br1.codice_variante
  JOIN articoli a
  ON a.codice_azienda = av.codice_azienda
  AND a.id_articolo = av.id_articolo
  JOIN codici_intrastat i
  ON av.intra_id = i.intra_id
  LEFT JOIN bolle_righe br2
  ON br2.codice_azienda = br1.codice_azienda
  AND br2.codice_magazzino = br1.codice_magazzino
  AND br2.numero_bolla = br1.numero_bolla
  AND br2.numero_riga_padre = br1.numero_riga
  WHERE br1.numero_riga_padre IS NULL
  AND br1.id_fattura IS NOT NULL
  GROUP BY t1.codice_azienda,
           t1.num_rif,
           t1.id_documento,
           t1.id_riga_documento,
           t1.codice_part,
           t1.id_anagrafica,
           t1.anno_comp,
           t1.periodo_comp,
           i.codice,
           i.codice_um_sec
) AS RESULT
GROUP BY codice_azienda,
         num_rif,
         codice_part,
         id_anagrafica,
         anno_comp,
         periodo_comp,
         codice_intrastat,
         codice_um_sec
;
```

We have two GROUP BY clauses in order to get amounts correct. We have one amount but multiple weights coming from bolle_righe. A single GROUP BY would sum amounts multiple times.

The OPTIMAL plan



Aside: at < 50 ms execution time, planning time can become significant if not dominant. If query is executed frequently consider using prepared statements

This plan takes 19ms for execution + 24ms for planning. This is 178x ;-) faster than the previous solution and, even more importantly, it maintains constant performance as the data size grows.

So, to RDBMS deterrents we can say that a 12-way join can take a few milliseconds… ;-)

Summary and thoughts

- Perform these optimizations only when required
- Optimal SQL sometimes needs to be written in a less declarative way
- For views, beware of multiple use cases. You may optimize one but worsen others
- Think about expected data flows and check if the plan matches. Many times the planner does better than you, but sometimes it gets things wrong
- Don’t be afraid of using nesting when it can be useful. PostgreSQL is good at pushing down predicates (WITH is currently an exception)
- Think about branch complexity: avoid linear complexity (branches that don’t take advantage of query restrictions)
- Could PostgreSQL do these rewrites by himself?