ANÁLISE DE OPTIMIZADORES SQL

UNIVERSIDADE DO MINHO

UC DE PROJECTO

Grupo 17

Eduardo Pereira | Tomás Meireles | Tiago Fernandes
A70619 A100106 A98983

30/05/2025

OBJETIVOS DO PROJETO

- Perceber o processo de optimização de queries SQL
- Identificar padrões e pontos importantes no processo
- Entender a diferença entre a optimização de diferentes queries
- Apontar possíveis melhorias ao plano de optimização

DESCRIÇÃO

O desenvolvimento do projeto passou pelos seguintes pontos principais:

- Familiarização com as ferramentas e com a base de dados educacional ***Risinglight***
- Inserção de instruções *print* no ficheiro ***optimizer.rs***
- Criação de ficheiros *csv* para guardar a informação necessária
- Geração de representações visuais para mostrar os resultados obtidos

PROCESSO DE OPTIMIZAÇÃO

- De SQL para uma expressão relacional
  - A query é traduzida para uma expressão com vários nodos
- Conjuntos de regras para gerar planos de execução alternativos
  - Várias alternativas significa que se pode escolher a melhor via para a optimização
- Estimar o custo de cada plano
  - O optimizador faz uma estimativa do custo de cada plano disponível
- Escolha do plano com menos custo
  - O plano cujo custo estimado é menor é o plano escolhido para seguir para a próxima etapa

INFORMAÇÃO CAPTURADA

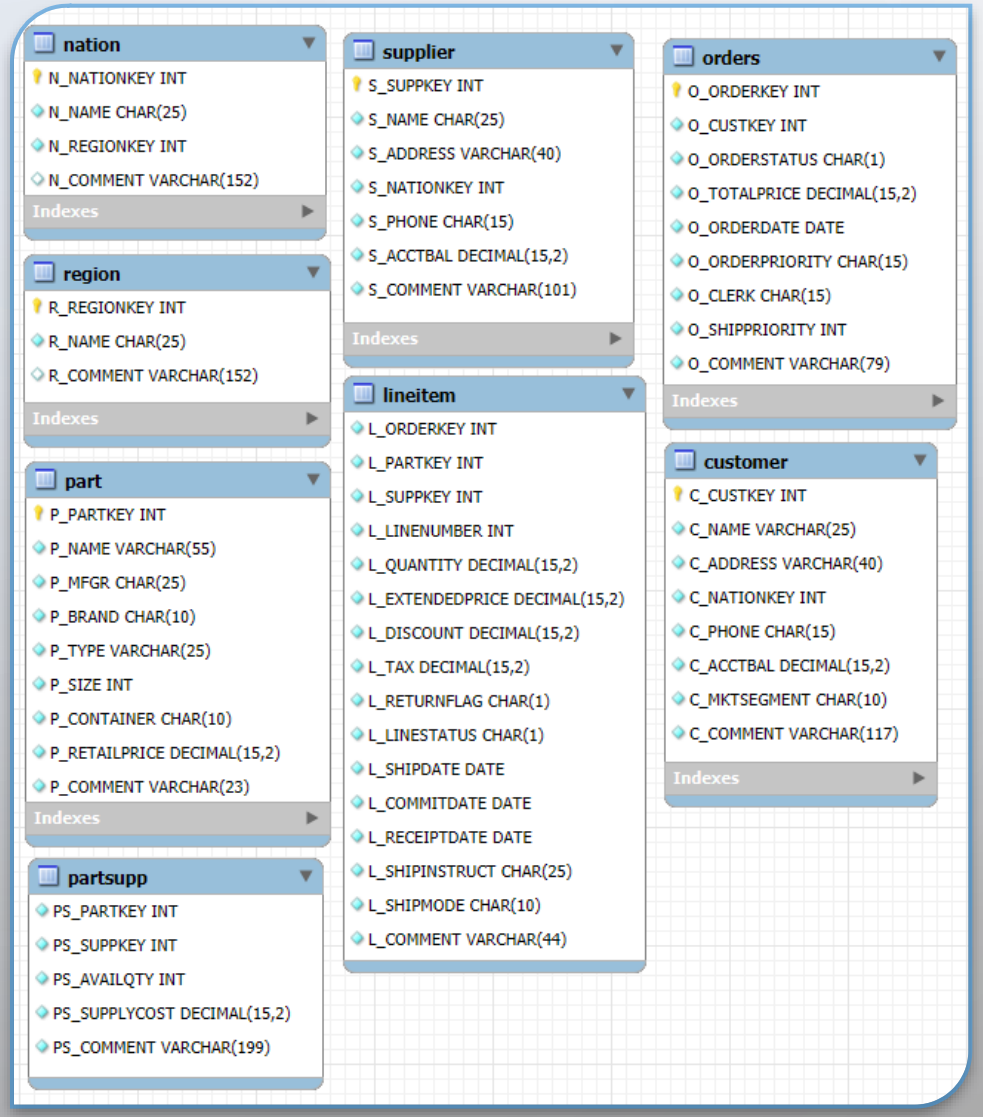
- Variação de custo em todas as queries TPC-H
- Número de nodos na expressão relacional
- Grupos de expressões equivalentes geradas durante a optimização
- Regras mais utilizadas numa certa etapa
- Técnica específica: *merge*

REGRAS DIFERENTES PARA CADA ETAPA

Cada etapa tem um propósito diferente no processo de optimização

- Etapa 1
  - Simplificação de expressões booleanas e transformação de subqueries
- Etapa 2
  - Utilização de índices e filtração de colunas
- Etapa 3
  - Re-ordenação de *joins* e utilização de *hash*
- Aplicação interativa
  - Regras aplicadas iterativamente dentro de cada etapa

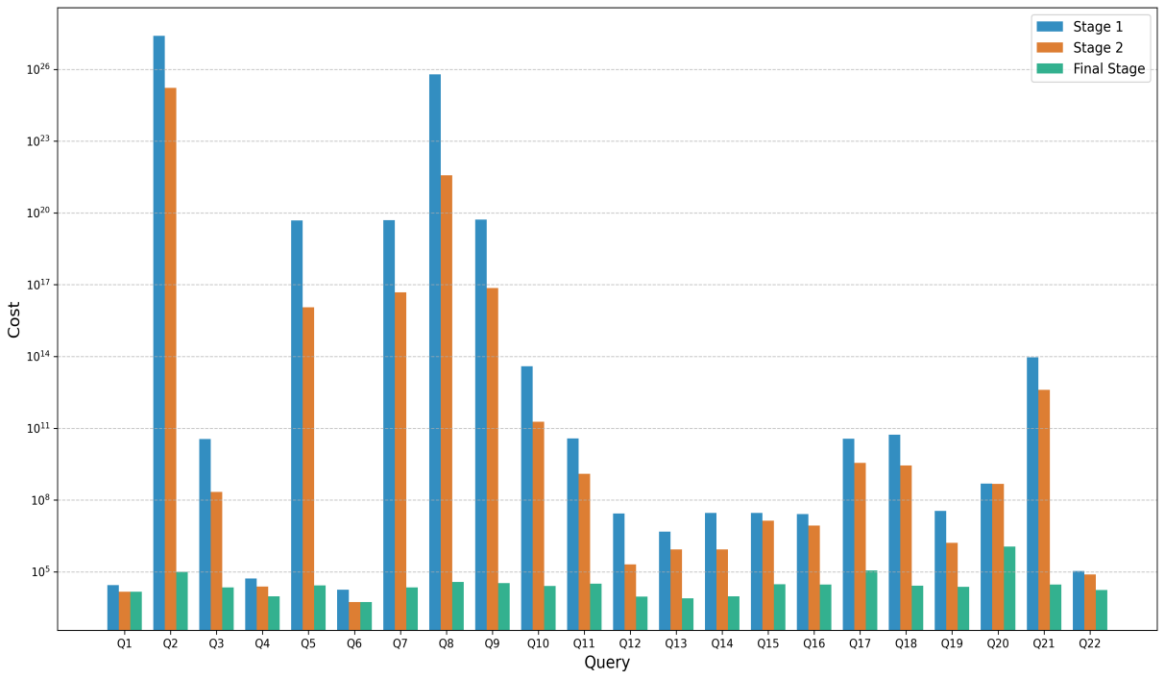
ESTRUTURA DA BASE DE DADOS RISINGLIGHT



EVOLUÇÃO DO CUSTO

Custo inicial $= 3.4 * 10^{38}$

Casos mais extremos: Query 2 e Query 6



Query	Stage 1	Stage 2	Final Stage	Absolute Reduction
Q1	28128.111	14618.112	14618.112	13509.999
Q2	250295800000000091792343040.000	16926520000000000639631360.000	99146.390	250295800000000091792343040.000
Q3	35436200000.000	215360400.000	22157.160	35436177842.840
Q4	52110.470	24255.469	9412.734	42697.736
Q5	4913813000000000000.000	11308230000000000.000	27351.363	49138129999999975424.000
Q6	17958.207	5469.458	5469.458	12488.749
Q7	503149060000000000.000	48091680000000000.000	22097.984	50314905999999975424.000
Q8	62104804999999996569845760.000	3766529499999999885312.000	37279.418	62104804999999996569845760.000
Q9	52074470000000000000.000	72212870000000000.000	33519.906	5207446999999967232.000
Q10	39213236000000.000	189269080000.000	25337.281	39213235974662.719
Q11	37588554000.000	1263876700.000	31861.496	37588522138.504
Q12	27565862.000	201470.450	9076.412	27556785.588
Q13	4840745.500	867070.300	7853.546	4832891.954
Q14	29172502.000	850032.300	9406.975	29163095.025
Q15	29049016.000	13515014.000	29854.697	29019161.303
Q16	26056468.000	8582357.000	29108.352	26027359.648
Q17	36120883000.000	3601246000.000	112654.830	36120770345.170
Q18	54558280000.000	2775188700.000	26530.621	54558253469.379
Q19	34976370.000	1627012.800	23419.270	34952950.730
Q20	491925120.000	475881760.000	1130161.900	490794958.100
Q21	91870570000000.000	4082035900000.000	28986.838	91870569971013.156
Q22	107983.266	77013.290	17342.803	90640.463

QUERY 6: UMA QUERY SIMPLES

```sql
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date '1994-01-01'
    and l_shipdate < date '1994-01-01' + interval '1' year
    and l_discount between 0.08 - 0.01 and 0.08 + 0.01
    and l_quantity < 24;
```

EXPRESSÃO RELACIONAL

Início do processo

```
RecExpr { nodes: [Constant(Bool(true)), Column($7.15), Column($7.14), Column(
    $7.13), Column($7.12), Column($7.11), Column($7.10), Column($7.9), Column
    ($7.8), Column($7.7), Column($7.6), Column($7.5), Column($7.4), Column($7
    .3), Column($7.2), Column($7.1), Column($7.0), List([16, 15, 14, 13, 12,
    11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]), Table($7), Scan([18, 17, 0]),
    Constant(Int32(24)), Lt([12, 20]), Constant(Decimal(0.01)), Constant(
    Decimal(0.08)), Add([23, 22]), LtEq([10, 24]), Sub([23, 22]), GtEq([10,
    26]), And([27, 25]), Constant(Interval(Interval { months: 12, days: 0, ms
    : 0 })), Constant(Date(Date(8766))), Add([30, 29]), Lt([6, 31]), GtEq([6,
     30]), And([33, 32]), And([34, 28]), And([35, 21]), Filter([36, 19]), Mul
    ([11, 10]), Sum(38), List([39]), Agg([40, 37]), Filter([0, 41]), List([])
    , Order([43, 42]), Ref(39), List([45]), Proj([46, 44]), Constant(Int32(0)
    ), Constant(Null), Limit([49, 48, 47])] }
```

- Ausência de um plano concreto
- Colunas desnecessárias
- 51 nodos na expressão

EXPRESSÃO RELACIONAL

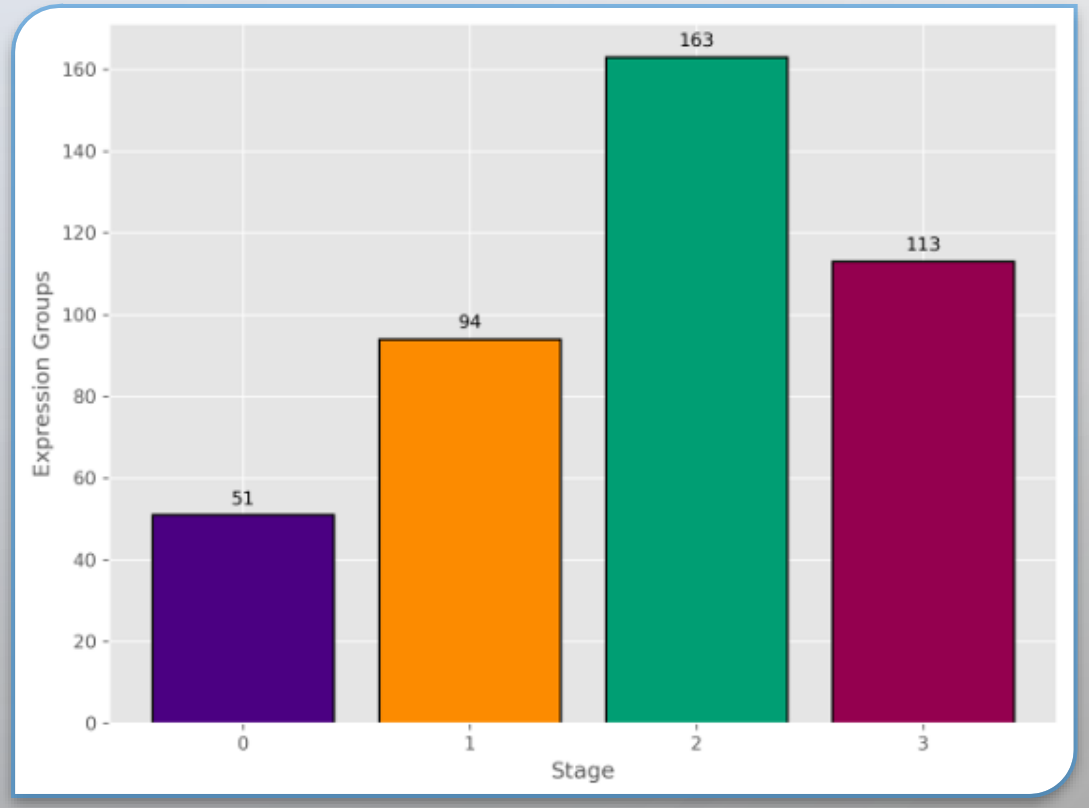
Fim do processo

```
RecExpr { nodes: [Constant(Bool(true)), Column($7.10), Column($7.6), Column($7.5)
    , Column($7.4), List([4, 3, 2, 1]), Table($7), Scan([6, 5, 0]), Constant(Date
    (Date(8766))), GtEq([1, 8]), Constant(Date(Date(9131))), Gt([10, 1]), And
    ([11, 9]), Constant(Int32(24)), Gt([13, 4]), And([14, 12]), Filter([15, 7]),
    List([3, 2]), Proj([17, 16]), Constant(Decimal(0.07)), GtEq([2, 19]),
    Constant(Decimal(0.09)), GtEq([21, 2]), And([22, 20]), Filter([23, 18]), Mul
    ([2, 3]), Sum(25), List([26]), Agg([27, 24]), Ref(26), List([29]), Proj([30,
    28])] }
```

- Melhor plano escolhido baseado no custo estimado
- Plano mais concreto, operações reduzidas
- Colunas desnecessárias removidas
- 31 nodos na expressão

GRUPOS DE EXPRESSÕES EQUIVALENTES

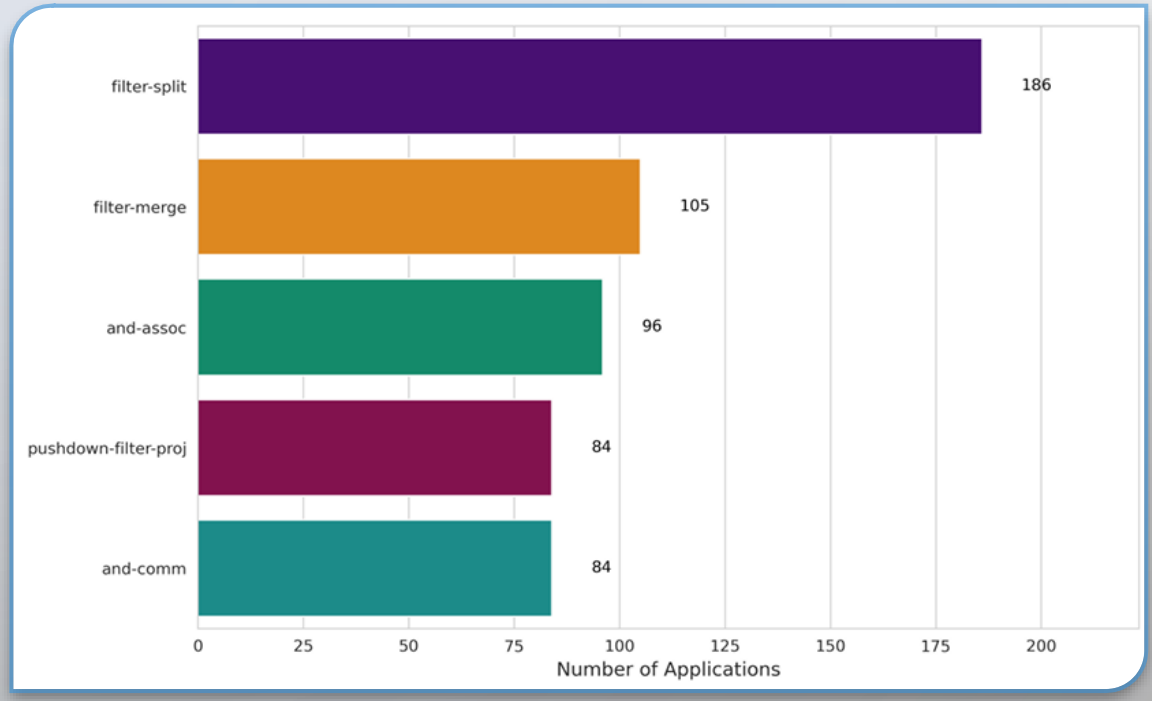
- Cada grupo contém expressões equivalentes entre si
- Expressões podem ter uma representação diferente, mas são semânticamente equivalentes
- Criação de expressões acentuada na etapa 2
- Esta informação coincide com a maior redução de custo



REGRAS MAIS APLICADAS

Etapa 2

- Regra *filter-split* parte condições de filtragem complexas em condições mais simples



REGRAS MAIS APLICADAS

Etapa 2

- Regra *filter-split* parte condições de filtragem complexas em condições mais simples

```
(filter (and (> order_date '2023-01-01') (< total_amount 100)) orders)
```

REGRAS MAIS APLICADAS

Etapa 2

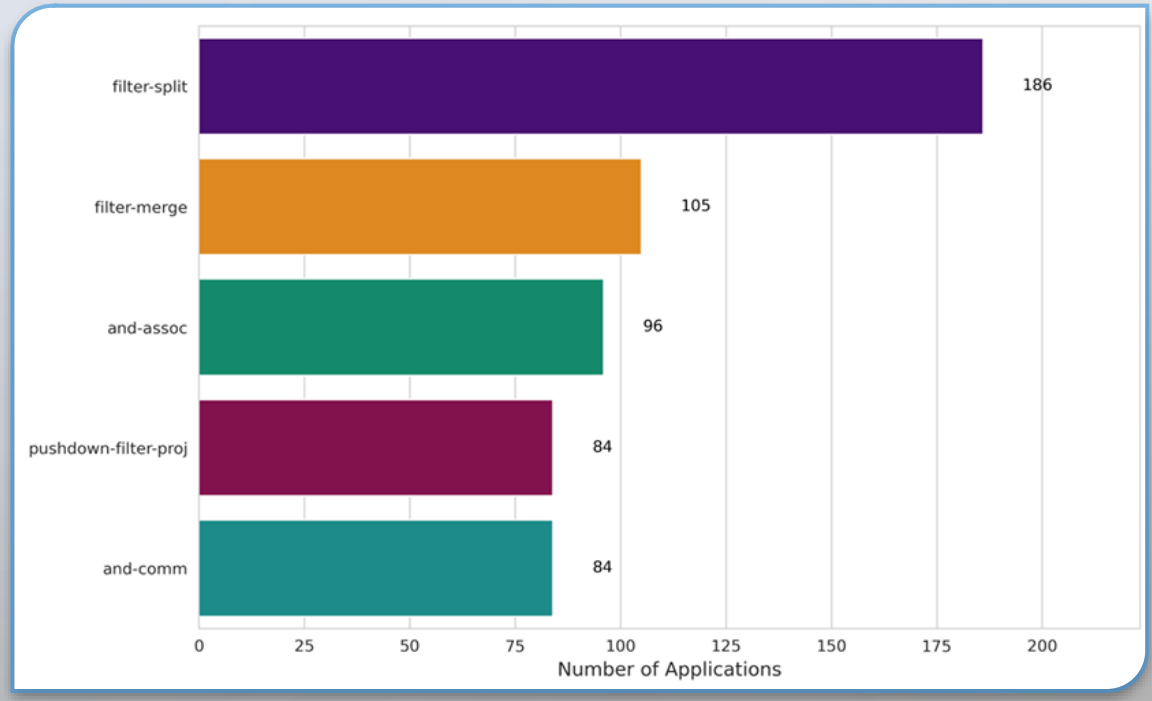
- Regra *filter-split* parte condições de filtragem complexas em condições mais simples
- Cada filtro pode agora ser efetuado mais perto da fonte dos dados
- Filtros independentes podem ser avaliados paralelamente

```
(filter (> order_date '2023-01-01')
    (filter (< total_amount 100) orders))
```

REGRAS MAIS APLICADAS

Etapa 2

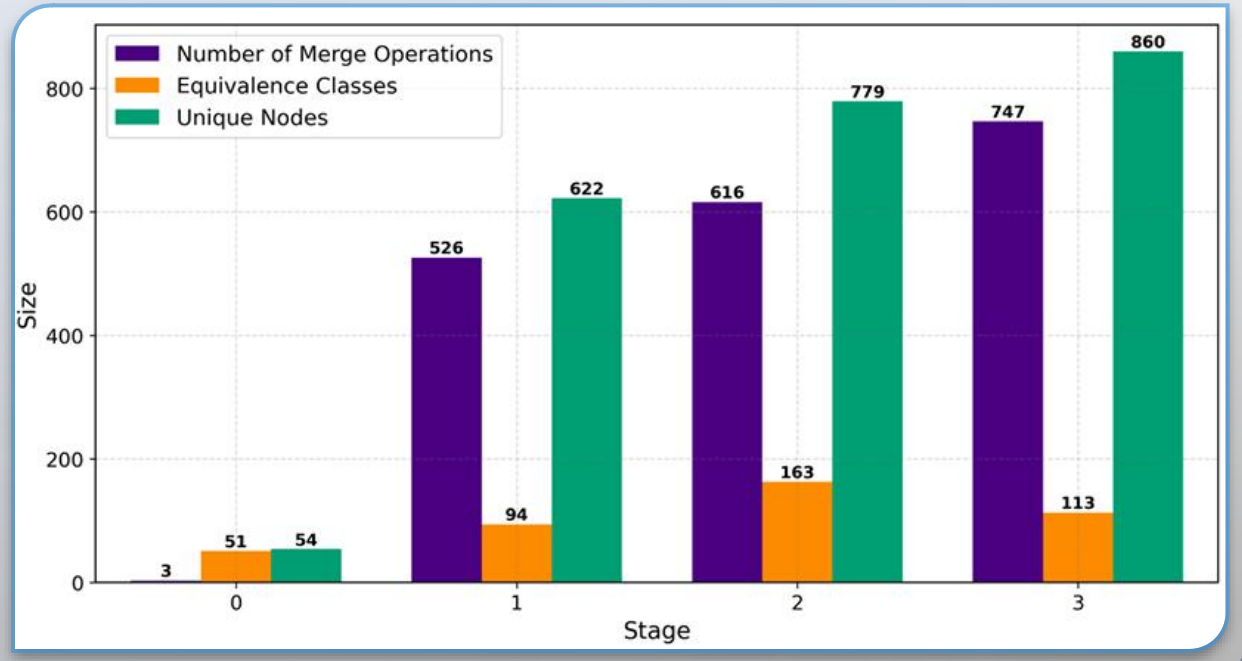
- Regra *filter-merge* efetua o processo inverso da regra anterior
- Aparentemente contra-intuítivo
- Importante relembrar que cada regra pode ser visitada mais que uma vez



TÉCNICA IMPORTANTE

Merge

- Passo importante no processo de optimização
- Cada classe de equivalência representa um grupo cujas expressões são equivalentes
- a+b é um nodo único, diferente de b+a, mas podem ser combinados (*merged*) numa classe de equivalência



QUERY 2: UMA QUERY COMPLEXA

```sql
select
    s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from
    part, supplier, partsupp, nation, region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp, supplier, nation, region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'EUROPE'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
limit 100;
```

EXPRESSÃO RELACIONAL

Início do processo

```
RecExpr { nodes: [Constant(Bool(true)), Column($1.2(1)), Column($1.1(1)),
    Column($1.0(1)), List([3, 2, 1]), Table($1), Scan([5, 4, 0]), Column($0
    .3(1)), Column($0.2(1)), Column($0.1(1)), Column($0.0(1)), List([10, 9,
    8, 7]), Table($0), Scan([12, 11, 0]), Column($3.6(1)), Column($3.5(1)),
    Column($3.4(1)), Column($3.3(1)), Column($3.2(1)), Column($3.1(1)),
    Column($3.0(1)), List([20, 19, 18, 17, 16, 15, 14]), Table($3), Scan([22,
     21, 0]), Column($4.4(1)), Column($4.3(1)), Column($4.2(1)), Column($4
    .1(1)), Column($4.0(1)), List([28, 27, 26, 25, 24]), Table($4), Scan([30,
     29, 0]), Inner, Join([32, 0, 31, 23]), Join([32, 0, 33, 13]), Join([32,
    0, 34, 6]), Constant(String(""EUROPE"")), Eq([2, 36]), Eq([8, 3]), Eq
    ([17, 10]), Eq([20, 27]), Column($2.0), Eq([41, 28]), And([42, 40]), And
    ([43, 39]), And([44, 38]), And([45, 37]), Filter([46, 35]), Min(25), List
    ([48]), Agg([49, 47]), Filter([0, 50]), List([]), Order([52, 51]), Ref
    (48), List([54]), Proj([55, 53]), Constant(Int32(0)), Constant(Null),
    Limit([58, 57, 56]), Column($1.2), Column($1.1), Column($1.0), List([62,
    61, 60]), Scan([5, 63, 0]), Column($0.3), Column($0.2), Column($0.1),
    Column($0.0), List([68, 67, 66, 65]), Scan([12, 69, 0]), Column($4.4),
    Column($4.3), Column($4.2), Column($4.1), Column($4.0), List([75, 74, 73,
     72, 71]), Scan([30, 76, 0]), Column($3.6), Column($3.5), Column($3.4),
    Column($3.3), Column($3.2), Column($3.1), Column($3.0), List([84, 83, 82,
     81, 80, 79, 78]), Scan([22, 85, 0]), Column($2.8), Column($2.7), Column(
    $2.6), Column($2.5), Column($2.4), Column($2.3), Column($2.2), Column($2
    .1), List([41, 94, 93, 92, 91, 90, 89, 88, 87]), Table($2), Scan([96, 95,
     0]), Join([32, 0, 97, 86]), Join([32, 0, 98, 77]), Join([32, 0, 99, 70])
    , Join([32, 0, 100, 64]), LeftOuter, Apply([102, 101, 59]), Eq([72, 54]),
     Eq([61, 36]), Eq([66, 62]), Eq([81, 68]), Constant(String(""\%BRASS"")),
     Like([91, 108]), Constant(Int32(15)), Eq([90, 110]), Eq([84, 74]), Eq
    ([41, 75]), And([113, 112]), And([114, 111]), And([115, 109]), And([116,
    107]), And([117, 106]), And([118, 105]), And([119, 104]), Filter([120,
    103]), Filter([0, 121]), Desc(79), List([123, 67, 83, 41]), Order([124,
    122]), List([79, 83, 67, 41, 93, 82, 80, 78]), Proj([126, 125]), Constant
    (Int32(100)), Limit([128, 57, 127])] }
```

- Expressão bem mais complexa
- Plano pouco concreto: Uso de *joins* de forma geral
- Um *join* apenas indica que duas tabelas são combinadas
- 129 nodos na expressão

EXPRESSÃO RELACIONAL

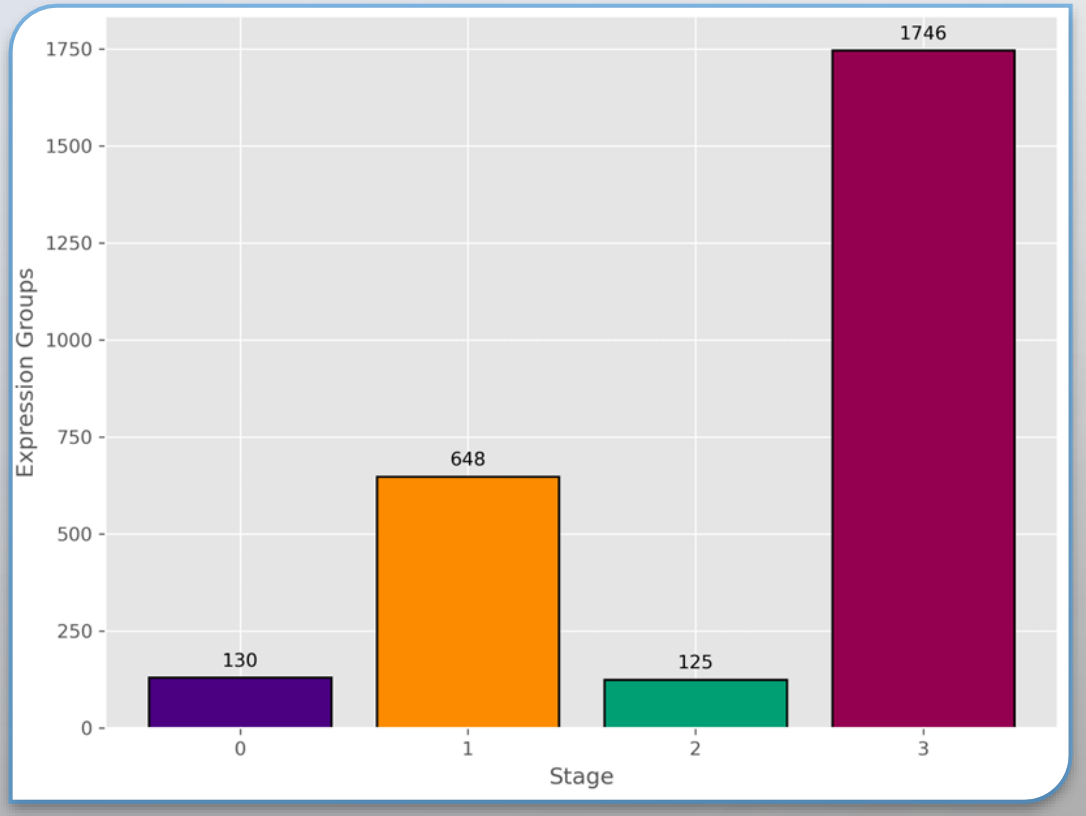
Fim do processo

```
RecExpr { nodes: [Constant(Bool(true)), Column($4.3(1)), Column($4.1(1)), Column(
    $4.0(1)), List([3, 2, 1]), Table($4), Scan([5, 4, 0]), Column($3.3(1)),
    Column($3.0(1)), List([8, 7]), Table($3), Scan([10, 9, 0]), Column($0.2(1)),
    Column($0.0(1)), List([13, 12]), Table($0), Scan([15, 14, 0]), Column($1.1(1)
    ), Column($1.0(1)), List([18, 17]), Table($1), Scan([20, 19, 0]), Constant(
    String(""EUROPE"")), Eq([17, 22]), Filter([23, 21]), List([18]), Proj([25,
    24]), List([12]), Inner, HashJoin([28, 0, 25, 27, 26, 16]), List([13]), Proj
    ([30, 29]), List([7]), HashJoin([28, 0, 30, 32, 31, 11]), List([8]), Proj
    ([34, 33]), List([2]), HashJoin([28, 0, 34, 36, 35, 6]), List([3, 1]), Proj
    ([38, 37]), Column($4.4), Column($4.3), Column($4.2), Column($4.1), Column($4
    .0), List([44, 43, 42, 41, 40]), Scan([5, 45, 0]), Column($3.6), Column($3.5)
    , Column($3.4), Column($3.3), Column($3.2), Column($3.1), Column($3.0), List
    ([53, 52, 51, 50, 49, 48, 47]), Scan([10, 54, 0]), Column($1.2), Column($1.1)
    , Column($1.0), List([58, 57, 56]), Scan([20, 59, 0]), Eq([22, 57]), Filter
    ([61, 60]), Column($0.3), Column($0.2), Column($0.1), Column($0.0), List([66,
     65, 64, 63]), Scan([15, 67, 0]), List([58]), List([64]), HashJoin([28, 0,
    70, 69, 68, 62]), List([50]), List([66]), HashJoin([28, 0, 73, 72, 71, 55]),
    List([43]), List([53]), HashJoin([28, 0, 76, 75, 74, 46]), List([53, 52, 51,
    50, 49, 48, 47, 44, 43, 42, 41, 40, 66, 65, 64, 63, 58, 57, 56]), Proj([78,
    77]), Column($2.8), Column($2.7), Column($2.6), Column($2.5), Column($2.4),
    Column($2.3), Column($2.2), Column($2.1), Column($2.0), List([88, 87, 86, 85,
     84, 83, 82, 81, 80]), Table($2), Scan([90, 89, 0]), Constant(Int32(15)), Eq
    ([83, 92]), Constant(String(""\%BRASS"")), Like([84, 94]), And([95, 93]),
    Filter([96, 91]), List([44]), List([88]), HashJoin([28, 0, 99, 98, 97, 79]),
    List([3]), LeftOuter, HashJoin([102, 0, 99, 101, 100, 39]), List([88, 87, 86,
     85, 84, 83, 82, 81, 80, 53, 52, 51, 50, 49, 48, 47, 44, 43, 42, 41, 40, 66,
    65, 64, 63, 58, 57, 56, 1]), Proj([104, 103]), Min(1), List([106]), List([88,
     87, 86, 85, 84, 83, 82, 81, 80, 53, 52, 51, 50, 49, 48, 47, 44, 43, 42, 41,
    40, 66, 65, 64, 63, 58, 57, 56]), HashAgg([108, 107, 105]), Ref(106), List
    ([88, 86, 52, 51, 49, 48, 47, 41, 65, 110]), Proj([111, 109]), Eq([41, 110]),
     Filter([113, 112]), List([88, 86, 52, 51, 49, 48, 47, 65]), Proj([115, 114])
    , Desc(48), List([117, 65, 52, 88]), Constant(Int32(0)), Constant(Int32(100))
    , TopN([120, 119, 118, 116]), List([48, 52, 65, 88, 86, 51, 49, 47]), Proj
    ([122, 121])] }
```

- Plano mais consolidado:
  - De *join* para *hashjoin*
  - Uso de tipos de *join* concretos
- 109 nodos na expressão

GRUPOS DE EXPRESSÕES EQUIVALENTES

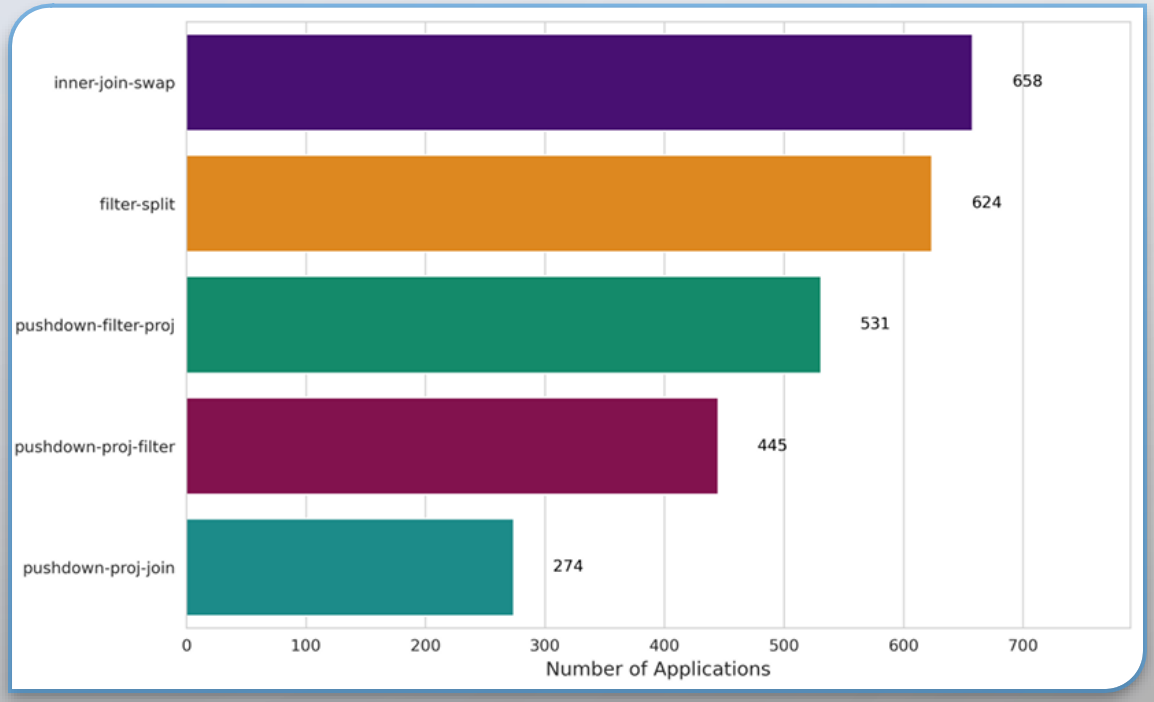
- Criação de expressões acentuada na etapa 3
- Custo menor apesar do número elevado de novos grupos de expressões equivalentes
- Regras de *join reordering* e *hash join*
- Esta informação coincide com a maior redução de custo



REGRAS MAIS APLICADAS

Etapa 3

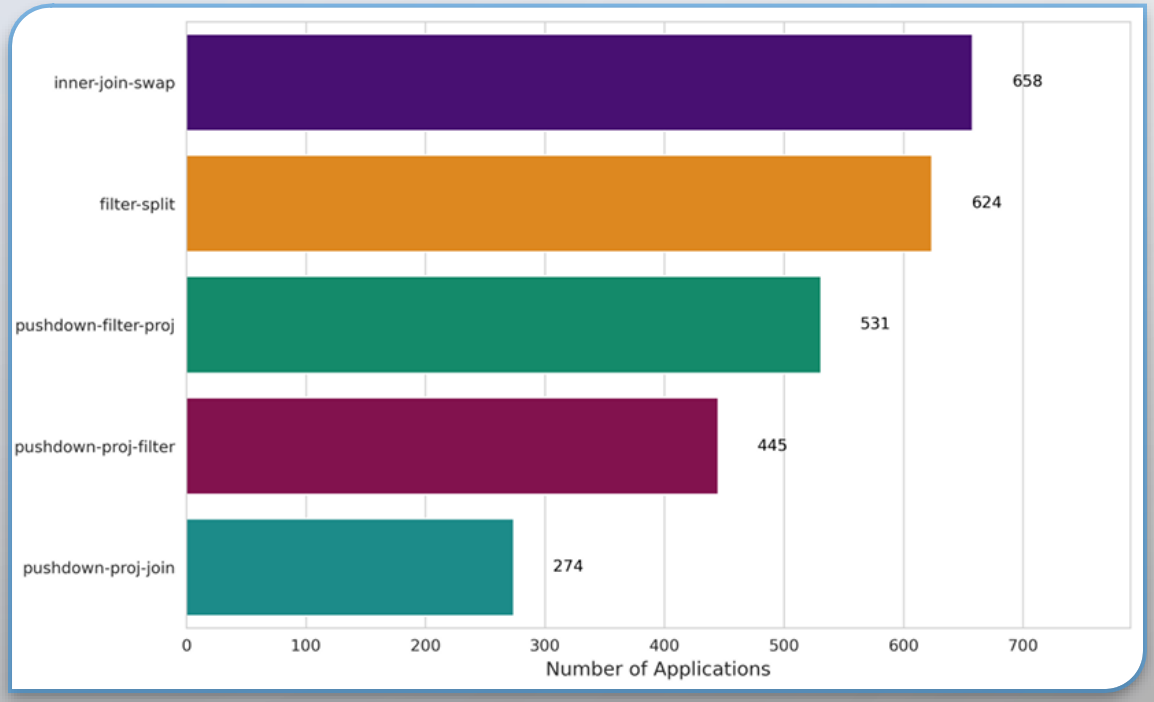
- Operações de *join* são frequentemente as mais custosas, devido a lidarem com várias tabelas
- *Hashjoins* funcionam melhor com estruturas de dados maiores



REGRAS MAIS APLICADAS

Etapa 3

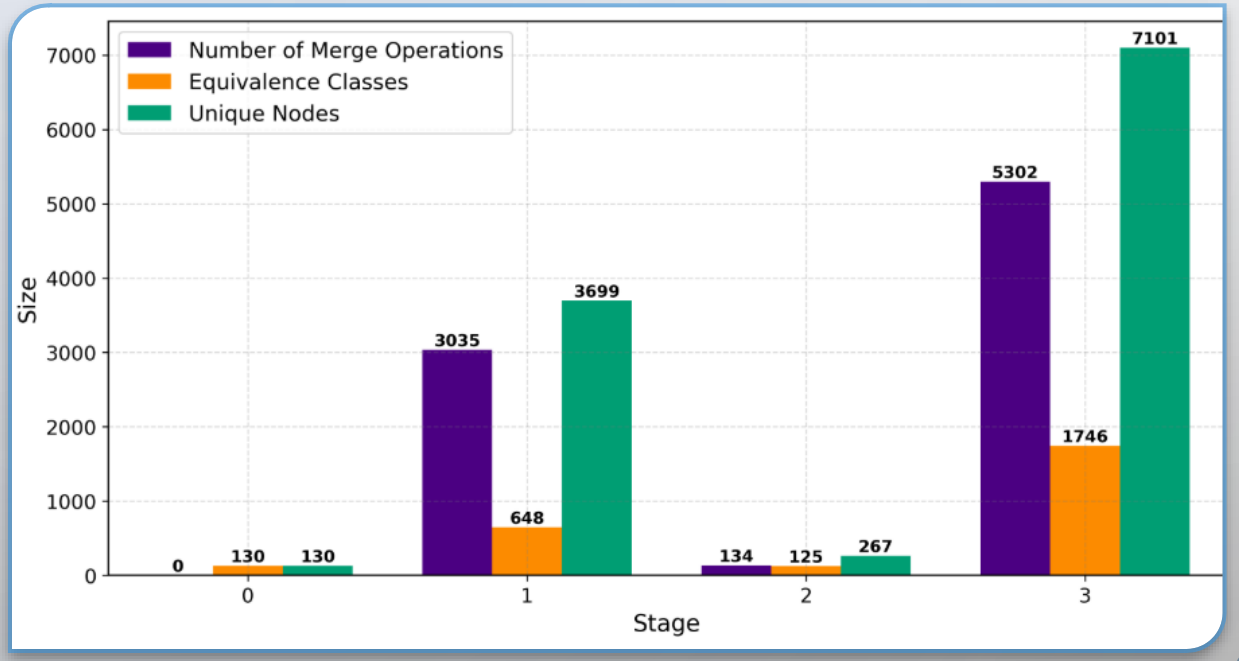
- Regra *inner-join-swap* foi aplicada 658 vezes
- Considere-se que ▷◁ é o símbolo de *join*
- (A ▷◁ B) ▷◁ C ⇒ A ▷◁ (B ▷◁ C)
- Se a tabela (B ▷◁ C) produz um resultado intermédio melhor que (A ▷◁ B), o lado direito da implicação é menos custosa



TÉCNICA IMPORTANTE

Merge

- Maior ocorrências na etapa 3
- Grande número de *merges* para ajudar o optimizador a lidar com os grupos de expressões gerados nesta etapa
- Comportamento semelhante à query 6



RESULTADOS E MELHORIAS

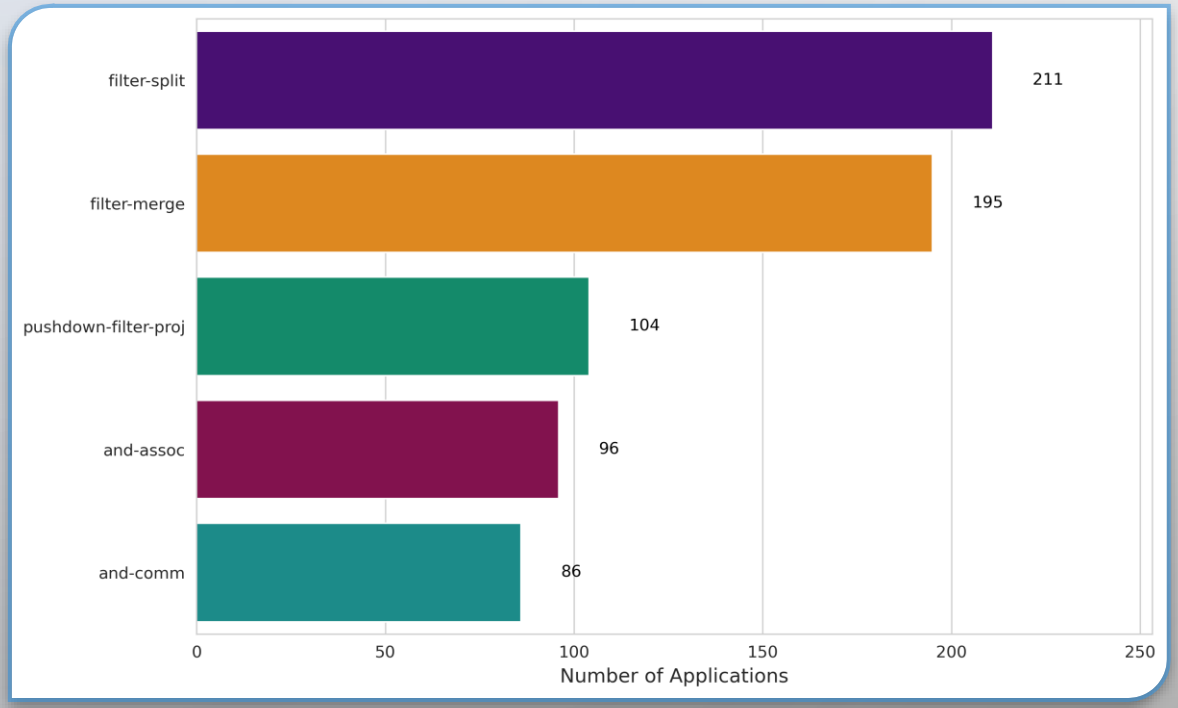
- Queries muito diferentes: o optimizador fez muito mais progresso na query 2
- Aspetos em comum: Nodos desnecessários e informação redundante
- Utilização de *joins* em vez de um plano concreto
- Colunas incluídas nas expressões desnecessárias ao plano final
- Possíveis melhorias:
  - Previsão de comportamento baseado na complexidade da query
  - Alteração da expressão relacional inicial

RESULTADOS E MELHORIAS

- A complexidade de uma query influencia diretamente a etapa mais importante na optimização:
  - Na query 2, a etapa mais importante é a 3, ao passo que na query 6, vê-se maior actividade na etapa 2
  - Possíveis benefícios de estratégias de optimização diferentes
- Optimização por iterações:
  - A possibilidade de aplicar o mesmo conjunto de regras um número indefinido de vezes antes de avançar para a próxima etapa maximiza a eficiência
- Regras mais utilizadas:
  - Apesar de serem aplicadas as mesmas regras a todas as queries, observam-se resultados muito diferentes
  - Pudemos ver isso na query 2 e 6

RESULTADOS E MELHORIAS

- Apesar da última etapa de optimização ser mais focada em optimização de *joins,* a query 6 continua a ter a regra *filter-split* como mais utilizada
- Possíveis melhorias:
  - Modificar as regras utilizadas nestes casos
  - Saltar esta etapa



CONCLUSÃO

- Sistemas de bases de dados são mais complexos do que inicialmente pensavamos
- Inicialmente tivemos alguns problemas em avançar com o projeto
- Este projeto permitiu extender o nosso conhecimento de bases de dados e ganhar a noção da importância de optimização
- Ferramentas como **Git** e **GitHub** foram indispensáveis
- Possíveis melhorias através de *machine learning*
- Seria interessante conseguir integrar em sistemas reais, por exemplo, sensores

LINKS IMPORTANTES

- Risinglight
  - https://github.com/risinglightdb/risinglight
- Biblioteca Egg
  - https://github.com/egraphs-good/egg
- Repositório de GitHub
  - https://github.com/Blackparkd/risinglight

ANÁLISE DE OPTIMIZADORES SQL

UNIVERSIDADE DO MINHO

UC DE PROJECTO

Grupo 17

Eduardo Pereira | Tomás Meireles | Tiago Fernandes
A70619 A100106 A98983

30/05/2025