Mon ami B-tree

Epitech Summit Lille - 4 février 2025

Agenda

Les enjeux de la performance

HEAP : le mécanisme de stockage avec PostgreSQL

BTREE : structure d’indexation de la donnée

Conclusion

Qui sommes-nous ?

DALIBO

Florent
JARDIN

- Consultants en base de données
- Support, formations, conseils
- Animateurs du Meetup PostgreSQL User Group Lille
- Anciens collègues chez Claranet
- Anciens diplômés SUPINFO

PERCONA

Yoann
LA CANCELLERA

Patientez, S'il vous plaît

Temps d’exécution moyen d’une requête SQL

dans des conditions optimales

100 ms

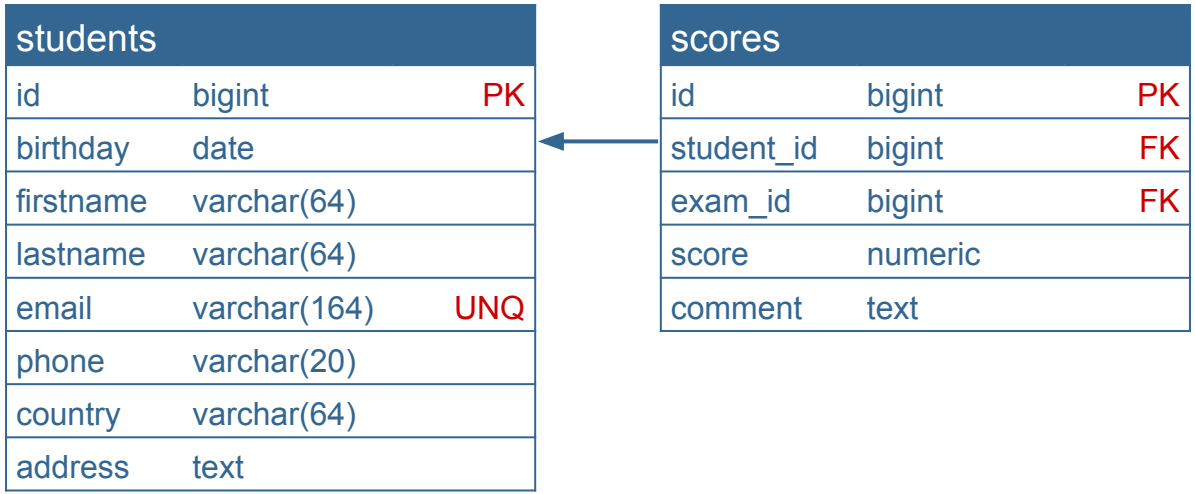
Délai cognitif moyen

pour une expérience utilisateur optimale

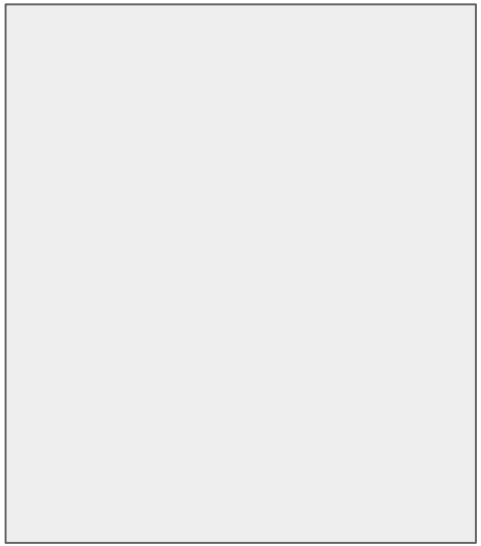
Selon Google, lorsque le temps de chargement d’une page passe d’une à trois secondes, le taux de rebond augmente de 32 %.

HEAP HEAP HEAP

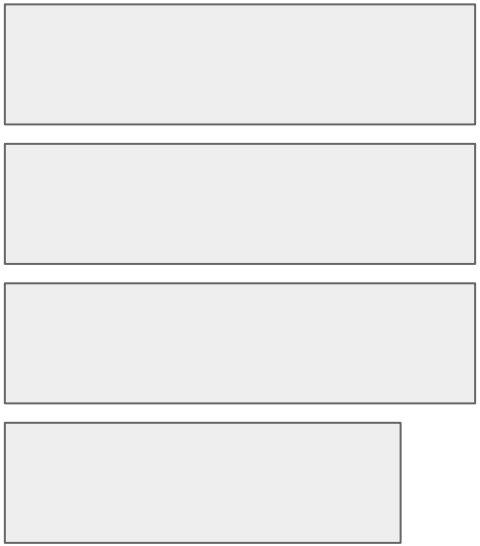
Modélisation du fil rouge



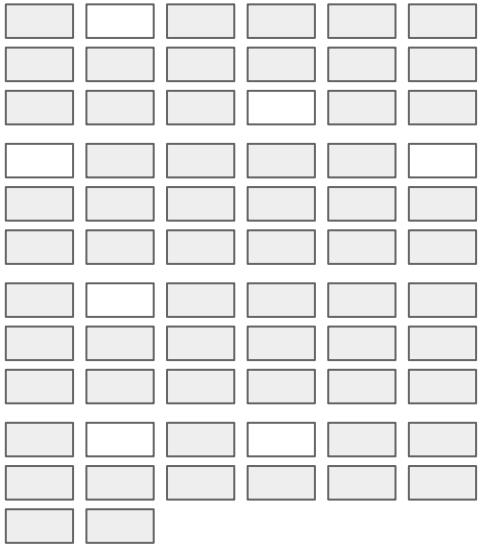
Représentation physique



HEAP Table



Fichiers (1 Go)

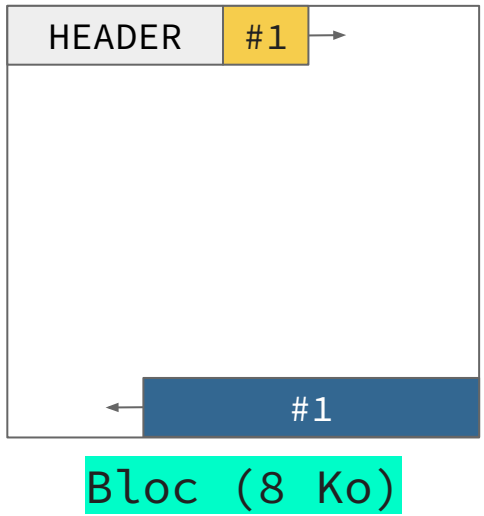


Blocs (8 Ko)

Ajout d'une ligne (tuple)

```
INSERT INTO scores
VALUES (1, 200, 10, .87, NULL);
```

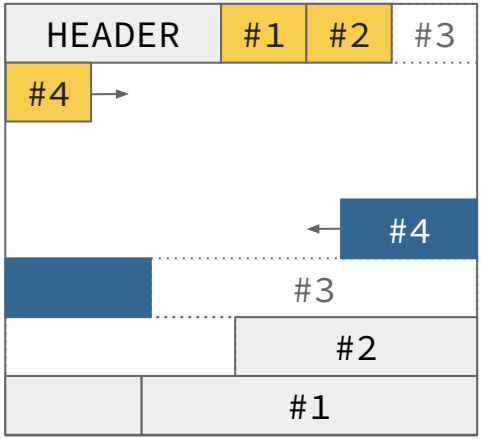
Tuple #1
\x0100000000000000c800000000000000a 0000000000000000b00805700



Mise à jour d'une ligne

```
UPDATE scores SET score = 1.0
 WHERE student_id = 200
   AND exam_id = 30
```

➔ Copy on Write (CoW)
➔ Dead Tuple
➔ Heap Only Tuple (HOT)

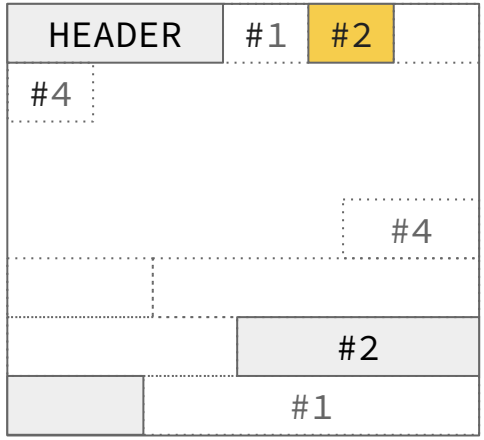


Bloc (8 Ko)

Suppression d'une ligne

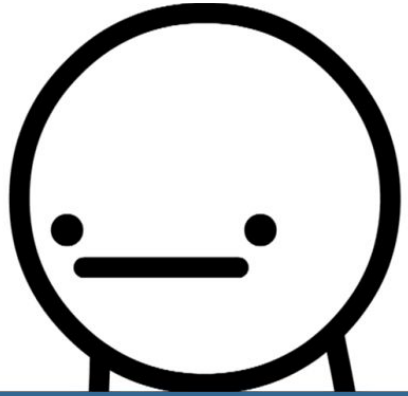
```
DELETE FROM scores
 WHERE student_id = 200
```

- ➔ Fragmentation (Bloat)
- ➔ VACUUM
- ➔ VACUUM FULL



Bloc (8 Ko)

```
SELECT * FROM scores
 WHERE student_id = 200
```



Structured Query Language (SQL)

➔ Langage déclaratif

Requête SQL

```
SELECT …
  FROM …
  JOIN …
 WHERE …
```

Résultat

1	@email.com
2	@email.com
3	@email.com
4	@email.com
5	@email.com

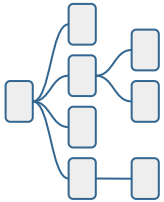
Structured Query Language (SQL)

➔ Langage déclaratif

Requête SQL

```
SELECT …
  FROM …
  JOIN …
 WHERE …
```

Arbre



Parser

Résultat

1	@email.com
2	@email.com
3	@email.com
4	@email.com
5	@email.com

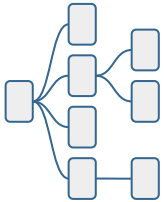
Structured Query Language (SQL)

➔ Langage déclaratif

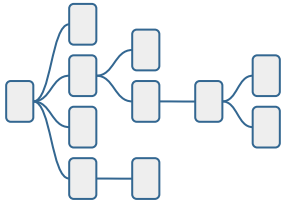
Requête SQL

```
SELECT …
  FROM …
  JOIN …
 WHERE …
```

Arbre



Arbre étendu



Résultat

1	@email.com
2	@email.com
3	@email.com
4	@email.com
5	@email.com

Parser → Rewriter

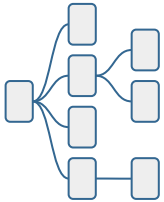
Structured Query Language (SQL)

➔ Langage déclaratif

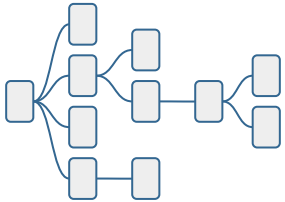
Requête SQL

```
SELECT …
  FROM …
  JOIN …
 WHERE …
```

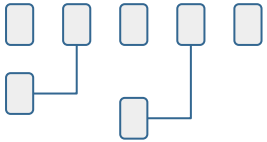
Arbre



Arbre étendu



Plan d'exécution



Résultat

1	@email.com
2	@email.com
3	@email.com
4	@email.com
5	@email.com

Parser → Rewriter → Planner

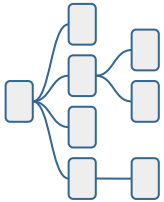
Structured Query Language (SQL)

➔ Langage déclaratif

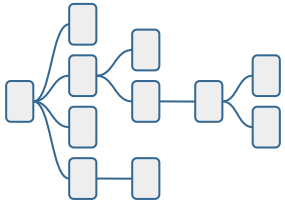
Requête SQL

```
SELECT …
FROM …
JOIN …
WHERE …
```

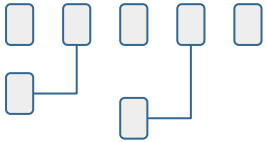
Arbre



Arbre étendu



Plan d'exécution



Résultat

1	@email.com
2	@email.com
3	@email.com
4	@email.com
5	@email.com

Parser → Rewriter → Planner → Executor

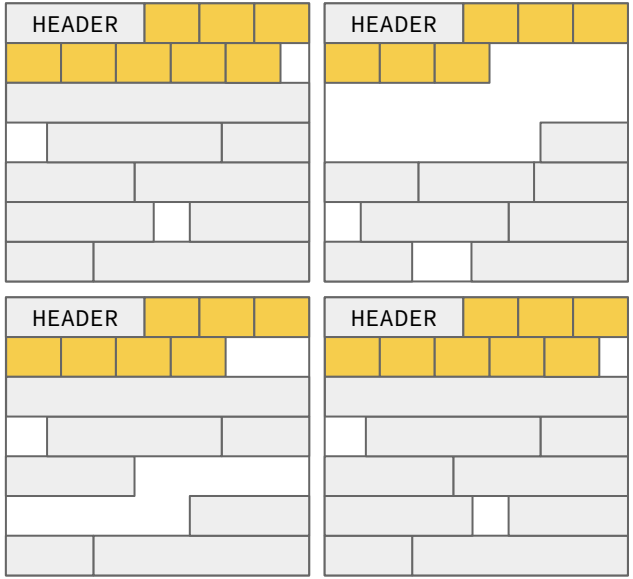
Planificateur & Plans d'exécution

- ➔ Identifie le chemin le moins coûteux
- ➔ Statistiques de données
- ➔ Modélisation (les contraintes, les index, etc.)
- ➔ Paramètres d'instance

Lecture de lignes

```
SELECT * FROM scores
 WHERE student_id = 200
```

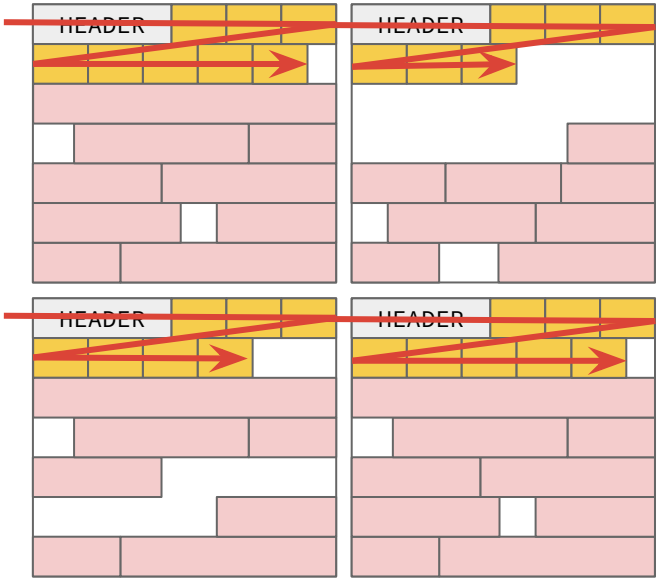
➔ Quel plan d'exécution ?



Lecture de lignes

```
SELECT * FROM scores
 WHERE student_id = 200
```

➔ Sequential Scan
➔ Mise en cache des blocs
➔ Filtre en mémoire



EXPLAIN

```
scores=# EXPLAIN (analyze, buffers, costs off)
         SELECT * FROM scores WHERE student_id = 200;

                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on scores (actual time=0.136..345.266 rows=27 loops=1)
   Filter: (student_id = 200)
   Rows Removed by Filter: 5810606
   Buffers: shared hit=386 read=36625
 Planning Time: 0.120 ms
 Execution Time: 345.301 ms
```

EXPLAIN

```
scores=# EXPLAIN (analyze, buffers, costs off)
         SELECT * FROM scores WHERE student_id = 200;

                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on scores (actual time=0.136..345.266 rows=27 loops=1)
   Filter: (student_id = 200)
   Rows Removed by Filter: 5810606
   Buffers: shared hit=386 read=36625
 Planning Time: 0.120 ms
 Execution Time: 345.301 ms
```

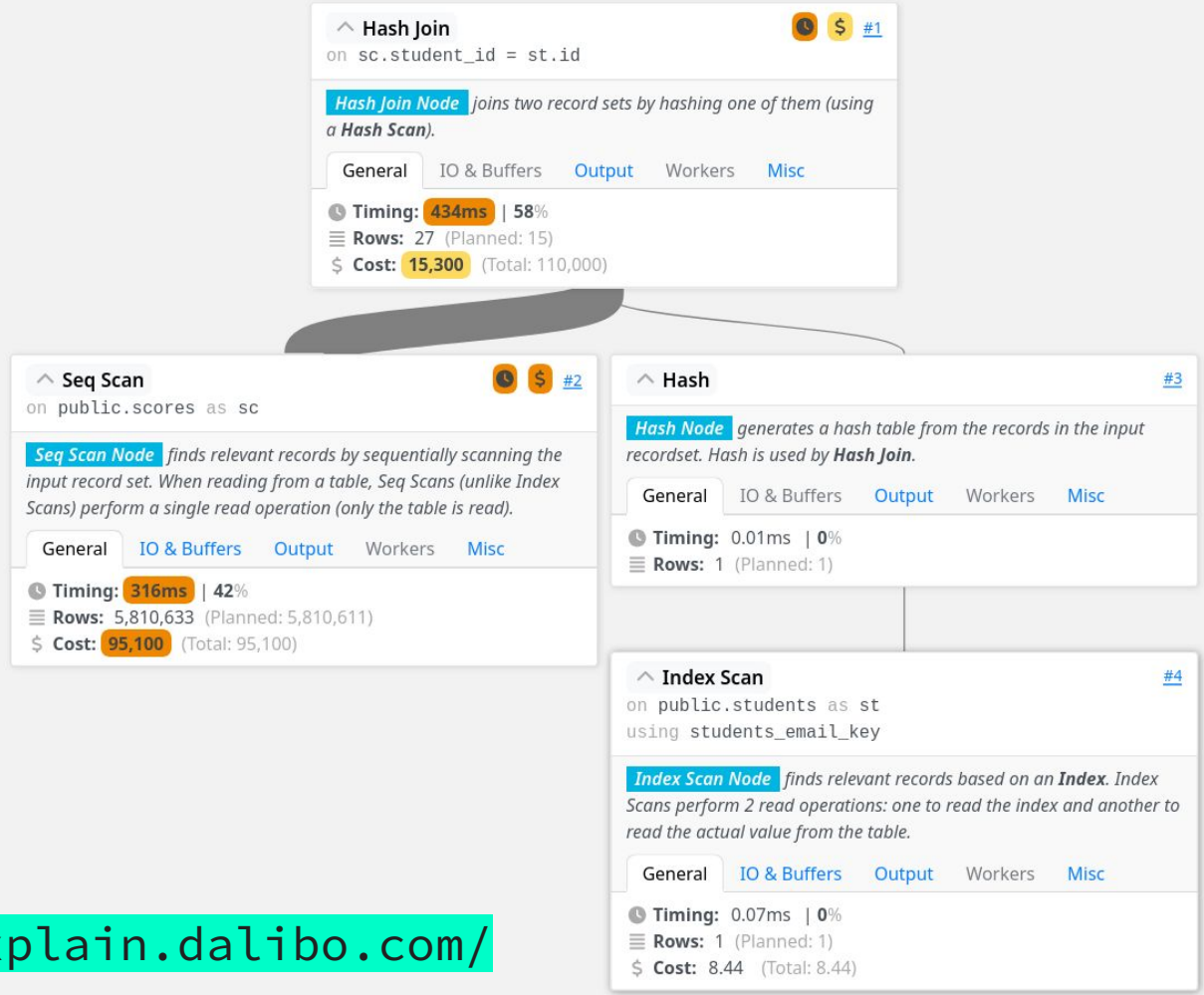
EXPLAIN

```
scores=# EXPLAIN (analyze, buffers, costs off)
         SELECT * FROM scores WHERE student_id = 200;

                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on scores (actual time=0.136..345.266 rows=27 loops=1)
   Filter: (student_id = 200)
   Rows Removed by Filter: 5810606
   Buffers: shared hit=386 read=36625
 Planning Time: 0.120 ms
 Execution Time: 345.301 ms
```

= 289 Mo

```
scores=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
         SELECT * FROM students st
           JOIN scores sc ON st.id = sc.student_id
          WHERE email = 'johndoe@email.com';

                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join (actual time=0.176..809.010 rows=27 loops=1)
   Hash Cond: (sc.student_id = st.id)
   Buffers: shared hit=358 read=36657
   ->  Seq Scan on scores sc
         (actual time=0.055..334.512 rows=5810633 loops=1)
       Buffers: shared hit=354 read=36657
   ->  Hash (actual time=0.033..0.034 rows=1 loops=1)
       Buckets: 1024  Batches: 1  Memory Usage: 9kB
       Buffers: shared hit=4
       ->  Index Scan using students_email_key on students st
             (actual time=0.027..0.028 rows=1 loops=1)
           Index Cond: ((email)::text = 'johndoe@email.com'::text)
           Buffers: shared hit=4
 Planning:
   Buffers: shared hit=8
 Planning Time: 0.310 ms
 Execution Time: 809.064 ms
```

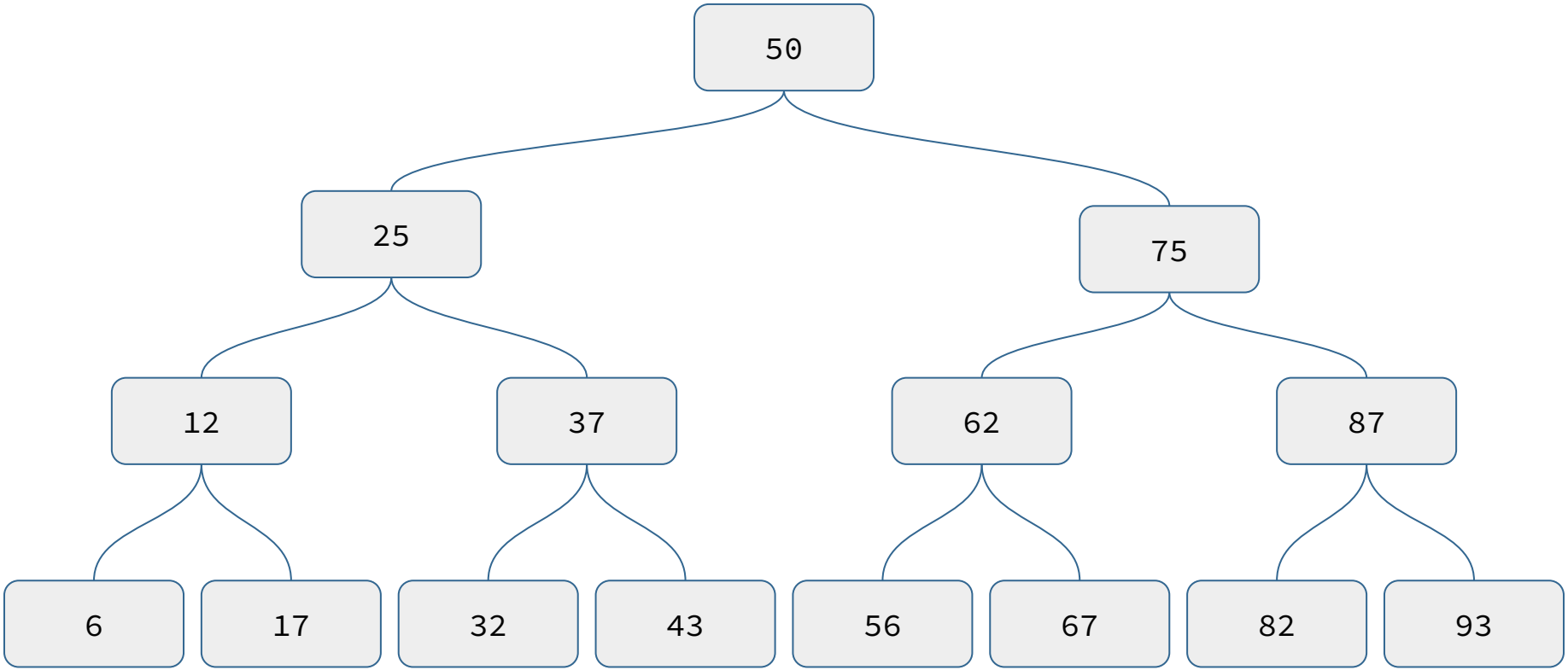
```
scores=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
         SELECT * FROM students st
           JOIN scores sc ON st.id = sc.student_id
          WHERE email = 'johndoe@email.com';

                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join (actual time=0.176..809.010 rows=27 loops=1)
   Hash Cond: (sc.student_id = st.id)
   Buffers: shared hit=358 read=36657
   ->  Seq Scan on scores sc
         (actual time=0.055..334.512 rows=5810633 loops=1)
         Buffers: shared hit=354 read=36657
   ->  Hash (actual time=0.033..0.034 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         Buffers: shared hit=4
         ->  Index Scan using students_email_key on students st
               (actual time=0.027..0.028 rows=1 loops=1)
             Index Cond: ((email)::text = 'johndoe@email.com'::text)
             Buffers: shared hit=4
 Planning:
   Buffers: shared hit=8
 Planning Time: 0.310 ms
 Execution Time: 809.064 ms
```



https://explain.dalibo.com/

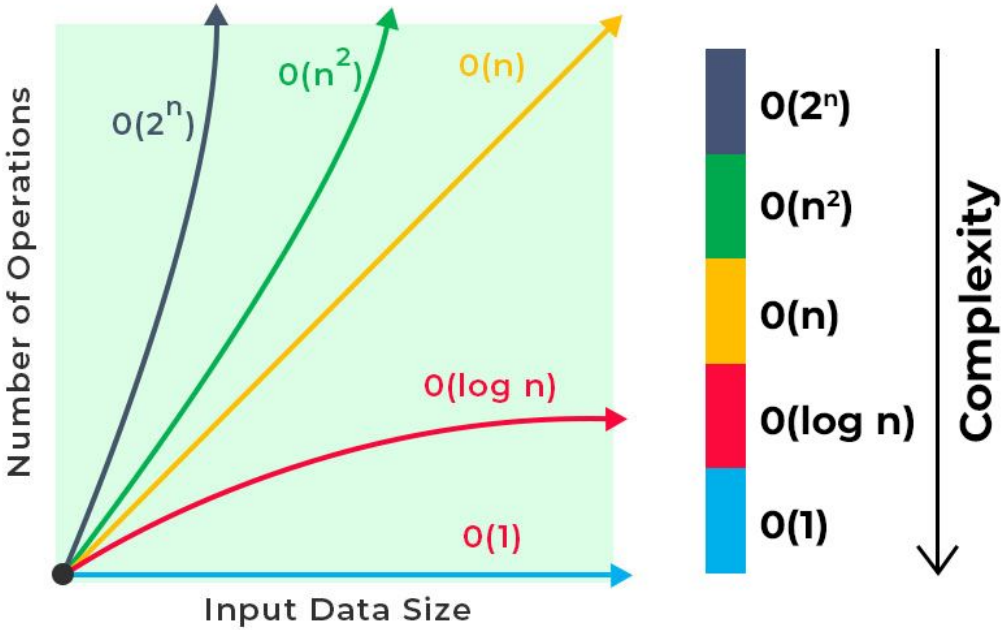
B(RO)-TREE

Arbre binaire basique



BTREE

- log2(100)= 6.64
- log2(1000)= 9.96
- **log2(10^9)= 29.9**
- log2(10^12)= 39.86

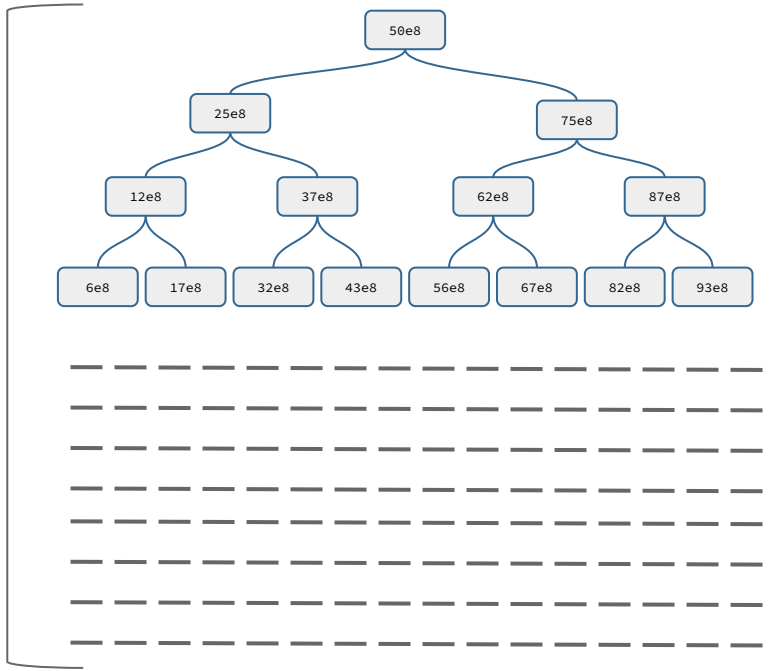


Binary tree != B+tree

- ➔ 29.9 “nodes” internes
- ➔ 29.9 opérations

log(10^9) =

29.9 niveaux

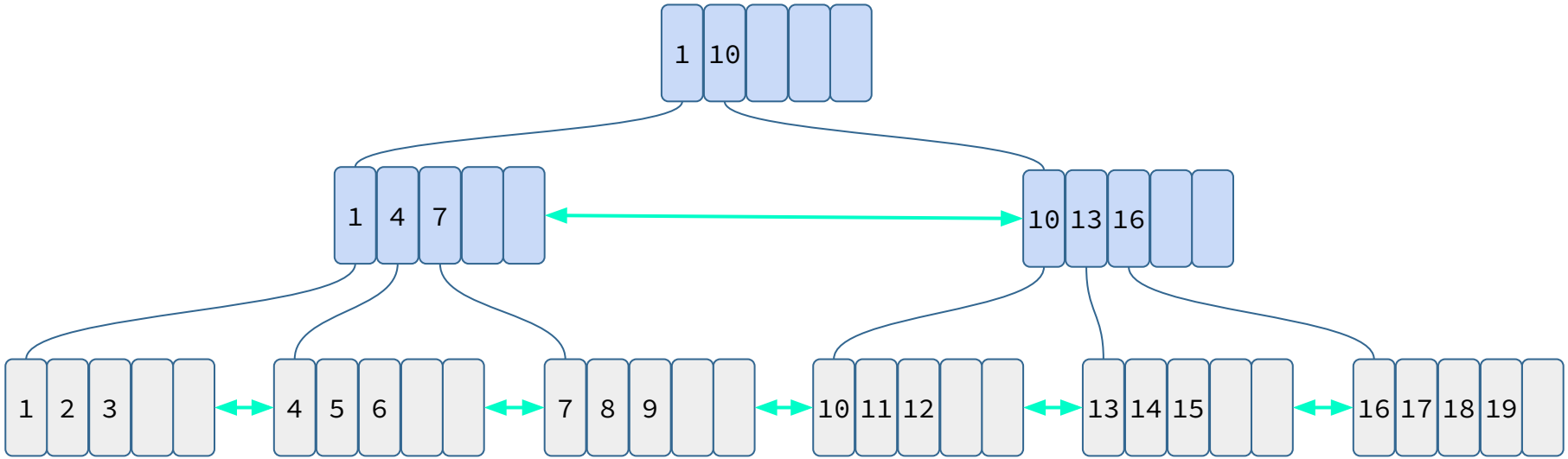


29.9 niveaux **si BTREE de facteur 1**

B+TREE

node “interne”

node “feuille”

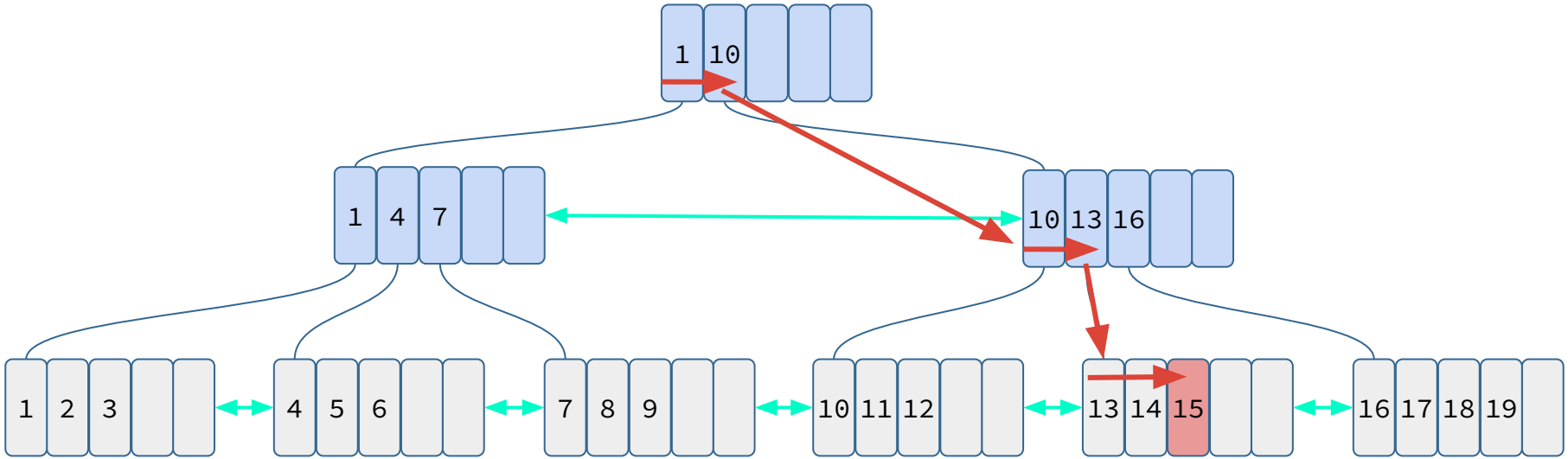


Recherche

node “interne”

node “feuille”

```
SELECT * FROM scores
 WHERE student_id = 15
```

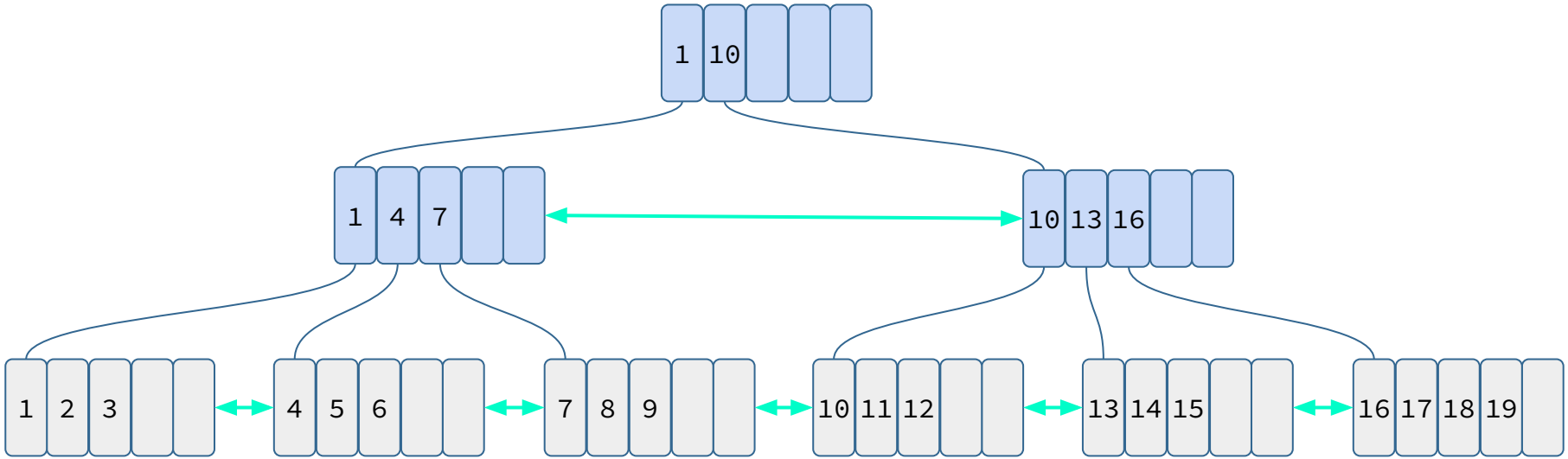


➔ Nombre de valeurs par nodes pour des int8

8ko / 8 ~= 1024 valeurs par nodes

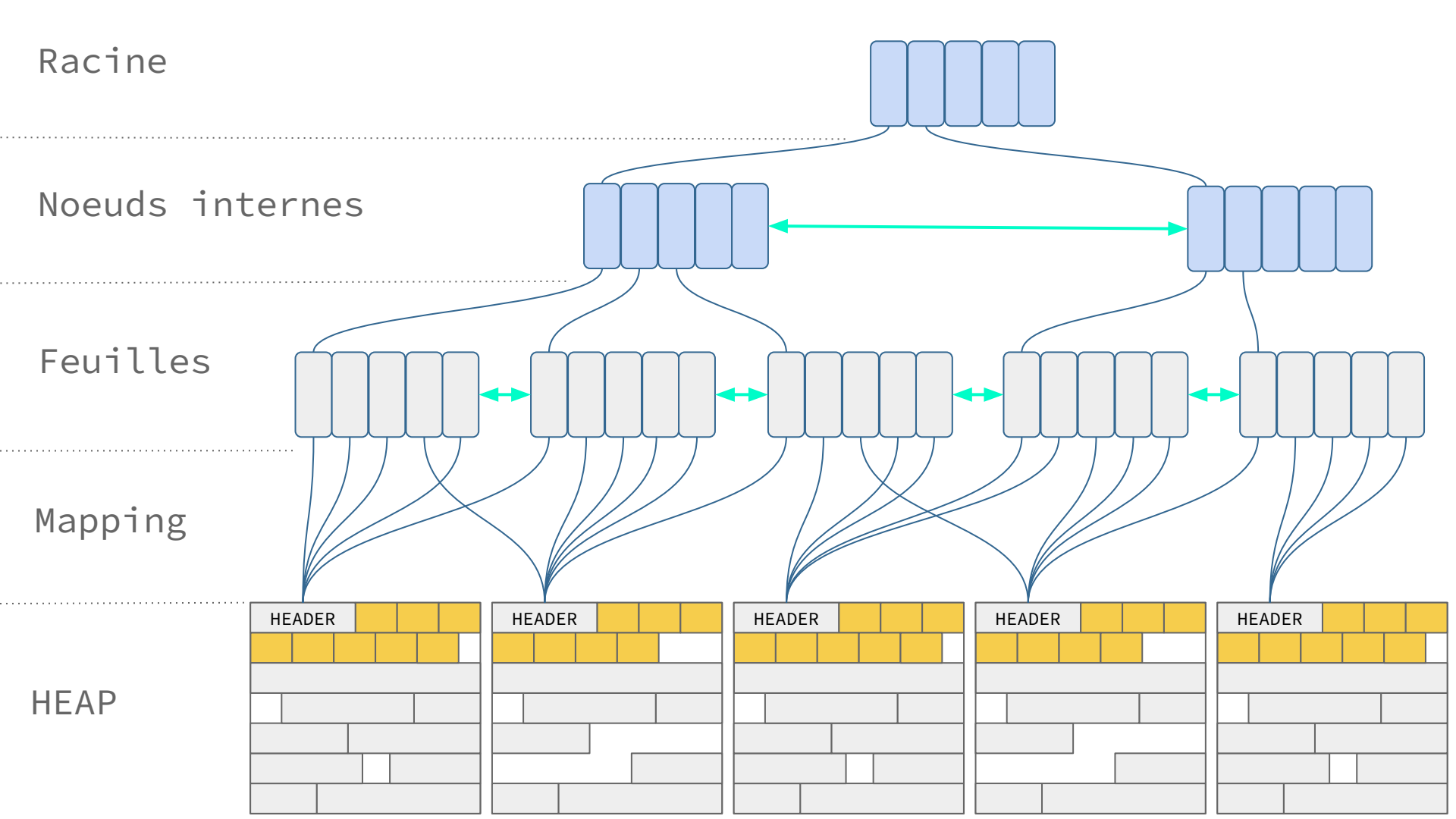
node “interne”

node “feuille”



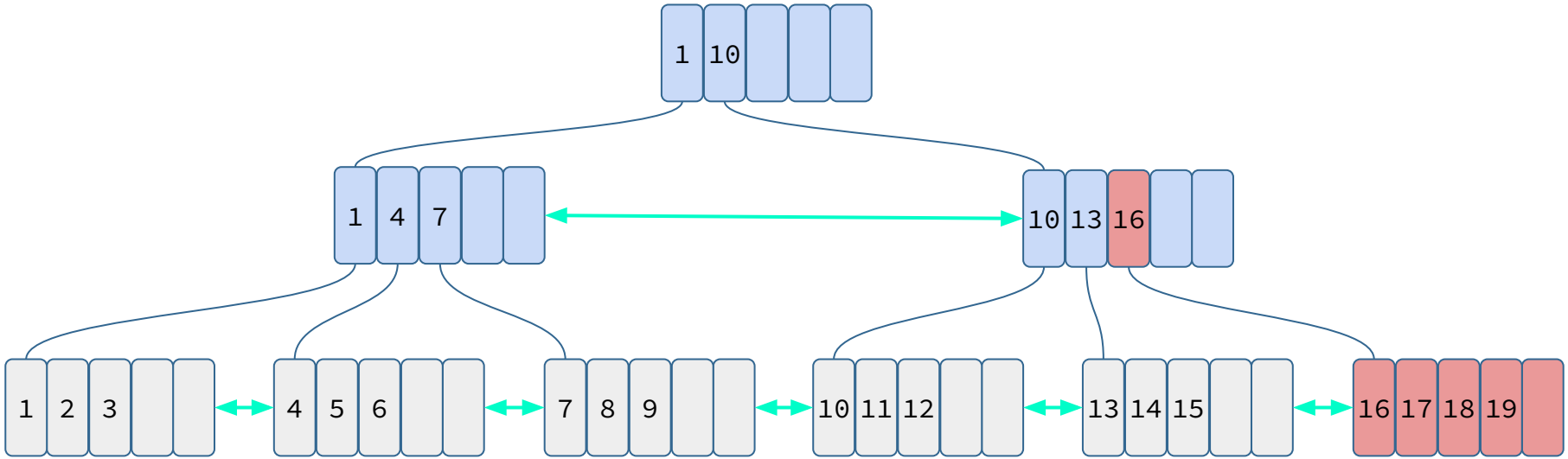
log1024(10^9) = 2.98

Pour un index B+TREE de entier 8 octet, parcourir un milliard de valeurs ne prendra que 3 blocs d’index 8kb.


Racine
Noeuds internes
Feuilles
Mapping
HEAP
HEADER
HEADER
HEADER
HEADER
HEADER

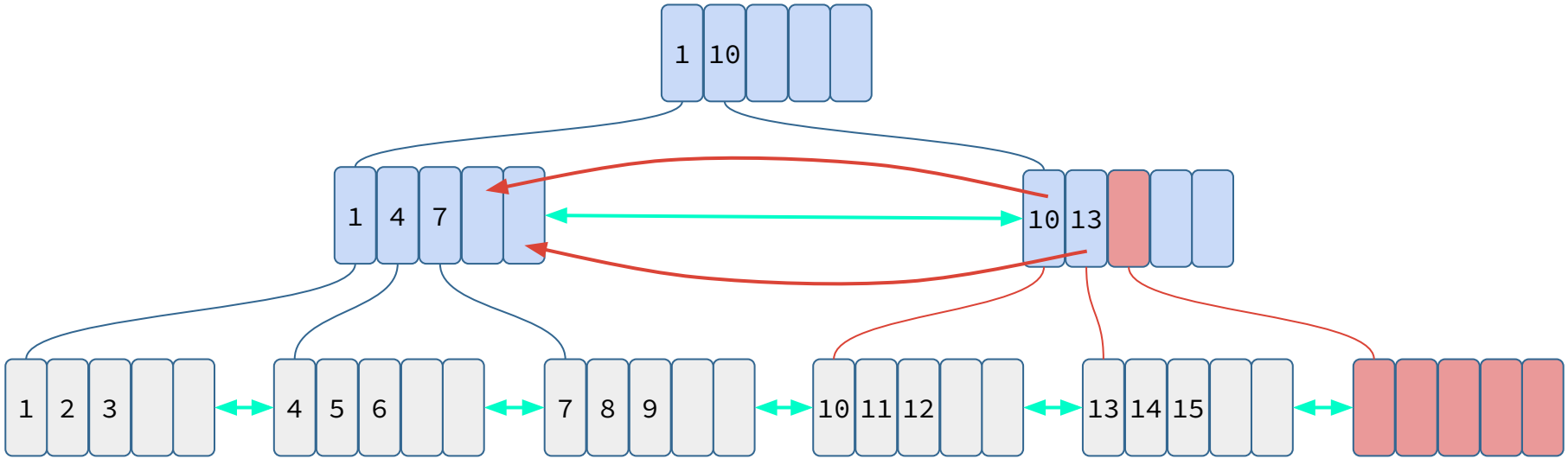
Suppression de plusieurs lignes

```
DELETE FROM scores
 WHERE student_id > 15
```



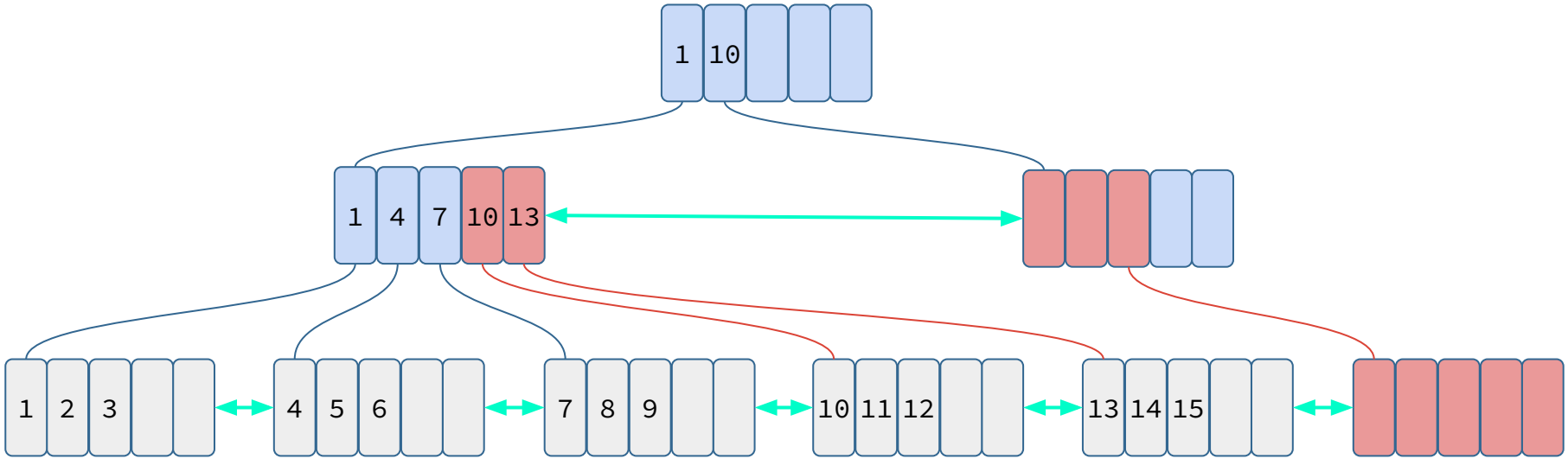
Suppression de plusieurs lignes

```
DELETE FROM scores
 WHERE student_id > 15
```



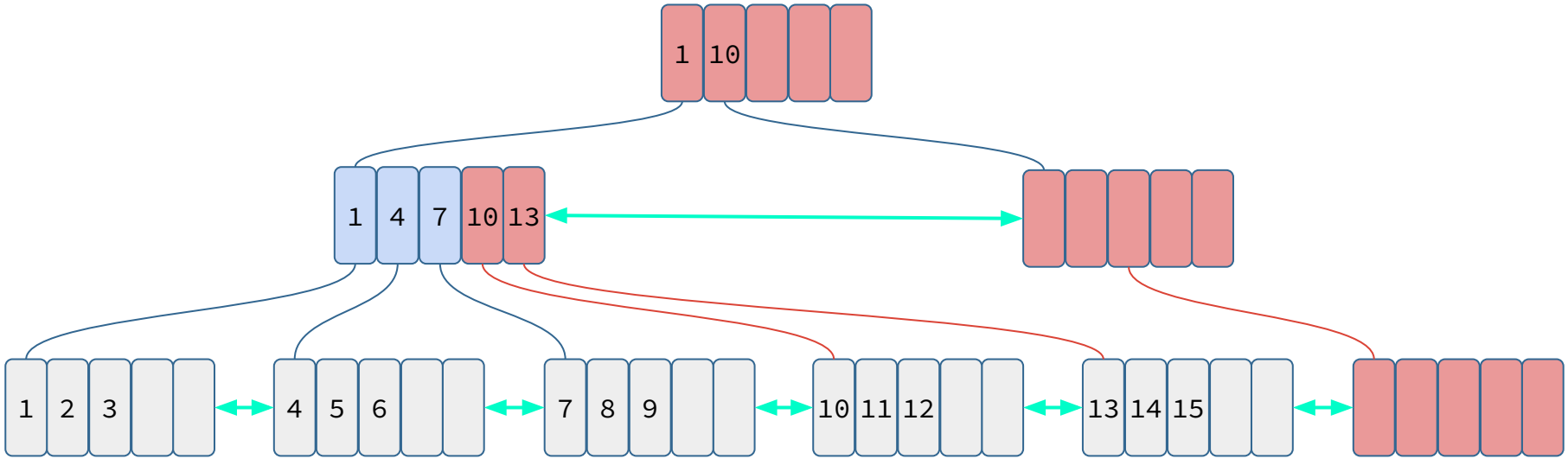
Suppression de plusieurs lignes

```
DELETE FROM scores
 WHERE student_id > 15
```



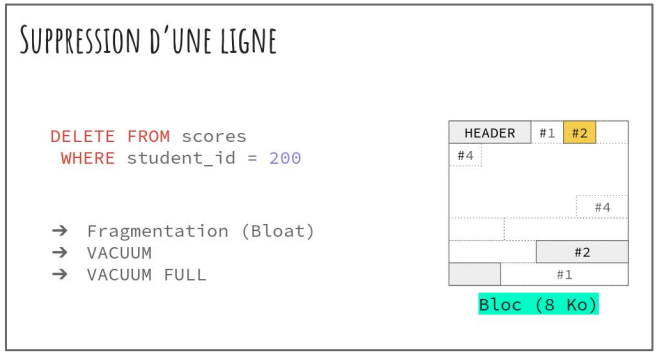
Suppression de plusieurs lignes

```
DELETE FROM scores
 WHERE student_id > 15
```

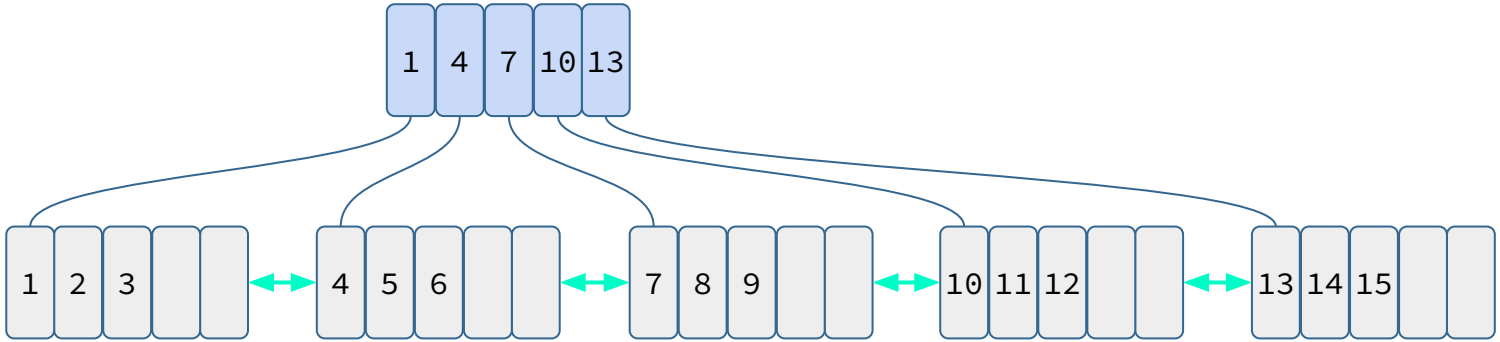


Suppression de plusieurs lignes

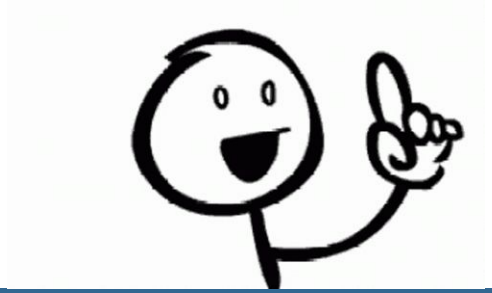
```
DELETE FROM scores
 WHERE student_id > 15
```



× 4 lignes



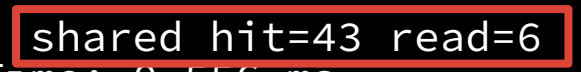
```
CREATE INDEX
 ON scores(student_id)
```



```
scores=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
         SELECT * FROM students st
           JOIN scores sc ON st.id = sc.student_id
          WHERE email = 'johndoe@email.com';

                          QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (actual time=0.067..0.251 rows=27 loops=1)
   ->  Index Scan using students_email_key on students st
         (actual time=0.019..0.020 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
   ->  Bitmap Heap Scan on scores sc
         (actual time=0.043..0.217 rows=27 loops=1)
       Recheck Cond: (st.id = student_id)
       Heap Blocks: exact=27
       Buffers: shared hit=6 read=24
       ->  Bitmap Index Scan on scores_student_id_idx
             (actual time=0.029..0.029 rows=27 loops=1)
           Index Cond: (student_id = st.id)
           Buffers: shared hit=2 read=1
 Planning:
   Buffers: shared hit=43 read=6
 Planning Time: 0.556 ms
 Execution Time: 0.285 ms
```

```
scores=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
         SELECT * FROM students st
           JOIN scores sc ON st.id = sc.student_id
          WHERE email = 'johndoe@email.com';

                          QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (actual time=0.067..0.251 rows=27 loops=1)
   ->  Index Scan using students_email_key on students st
         (actual time=0.019..0.020 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
   ->  Bitmap Heap Scan on scores sc
         (actual time=0.043..0.217 rows=27 loops=1)
       Recheck Cond: (st.id = student_id)
       Heap Blocks: exact=27
       Buffers: shared hit=6 read=24
       ->  Bitmap Index Scan on scores_student_id_idx
             (actual time=0.029..0.029 rows=27 loops=1)
           Index Cond: (student_id = st.id)
           Buffers: shared hit=2 read=1
 Planning:
   Buffers: shared hit=43 read=6
 Planning Time: 0.556 ms
 Execution Time: 0.285 ms
```

```
scores=# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
         SELECT * FROM students st
           JOIN scores sc ON st.id = sc.student_id
          WHERE email = 'johndoe@email.com';

                          QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (actual time=0.067..0.251 rows=27 loops=1)
   ->  Index Scan using students_email_key on students st
         (actual time=0.019..0.020 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
   ->  Bitmap Heap Scan on scores sc
         (actual time=0.043..0.217 rows=27 loops=1)
       Recheck Cond: (st.id = student_id)
       Heap Blocks: exact=27
       Buffers: shared hit=6 read=24
       ->  Bitmap Index Scan on scores_student_id_idx
             (actual time=0.029..0.029 rows=27 loops=1)
           Index Cond: (student_id = st.id)
           Buffers: shared hit=2 read=1
 Planning:
   Buffers: shared hit=43 read=6
 Planning Time: 0.556 ms
 Execution Time: 0.285 ms
```



CASCADE FOREIGN KEYS

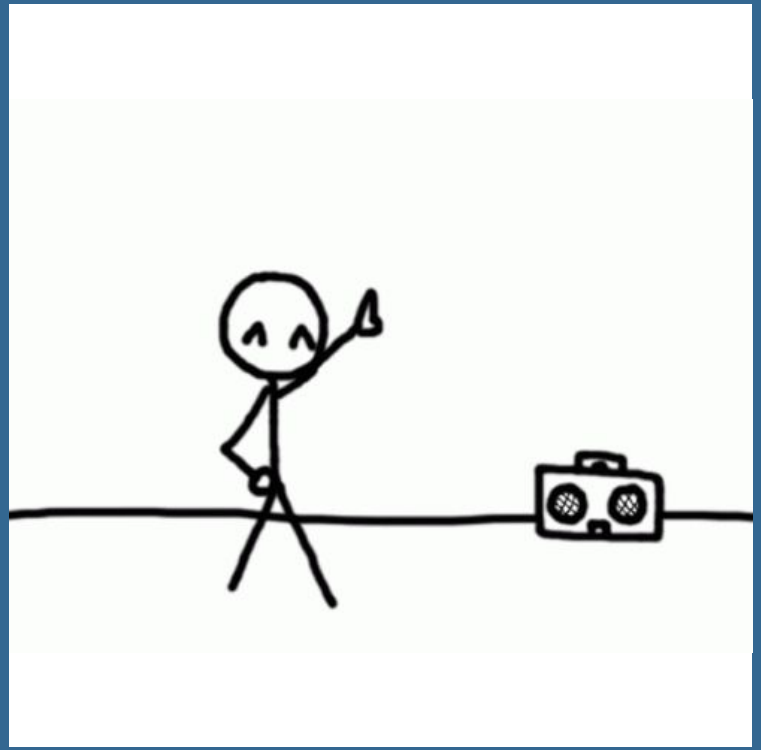
```
scores=*# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
          DELETE FROM students WHERE email = 'johndoe@email.com';

                                QUERY PLAN
-------------------------------------------------------------------------
 Delete on students (actual time=0.082..0.083 rows=0 loops=1)
   Buffers: shared hit=9 dirtied=2
   ->  Index Scan using students_email_key on students
         (actual time=0.029..0.032 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
 Planning Time: 0.089 ms
 Trigger for constraint scores_student_id_fkey: time=562.464 calls=1
 Execution Time: 562.635 ms
(8 rows)
```

CASCADE FOREIGN KEYS

```
scores=*# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
          DELETE FROM students WHERE email = 'johndoe@email.com';

                                QUERY PLAN
-------------------------------------------------------------------------
 Delete on students (actual time=0.082..0.083 rows=0 loops=1)
   Buffers: shared hit=9 dirtied=2
   ->  Index Scan using students_email_key on students
         (actual time=0.029..0.032 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
 Planning Time: 0.089 ms
 Trigger for constraint scores_student_id_fkey: time=562.464 calls=1
 Execution Time: 562.635 ms
(8 rows)
```

CASCADE FOREIGN KEYS - AVEC INDEX

```
scores=*# EXPLAIN (ANALYZE, BUFFERS, COSTS OFF)
          DELETE FROM students WHERE email = 'johndoe@email.com';

                                QUERY PLAN
-------------------------------------------------------------------------
 Delete on students (actual time=0.076..0.077 rows=0 loops=1)
   Buffers: shared hit=6
   ->  Index Scan using students_email_key on students
         (actual time=0.051..0.053 rows=1 loops=1)
       Index Cond: ((email)::text = 'johndoe@email.com'::text)
       Buffers: shared hit=4
 Planning Time: 0.120 ms
 Trigger for constraint scores_student_id_fkey: time=0.650 calls=1
 Execution Time: 0.757 ms
(8 rows)
```



Conclusion

- ➔ N’indexez que ce qui est nécessaire
- ➔ Indexez vos colonnes de clés étrangères
- ➔ Surveillez vos plans d’exécution avec EXPLAIN

Envie de participer, de partager votre expérience ou de sponsoriser les prochains événements ?

Contactez-nous sur les pages Meetup et Linkedin du groupe !