Use o índice!

O essencial para desenvolvedores

A culpa não é do Rails

Usando índices para salvar o dia

(no fundo é uma talk sobre banco de dados)

whoami

Milhouse (Renan Ranelli)

Consultor Independente @ TamingChaos

Senior Software Enginner @ Telnyx LLC

Organizador do meetup de Elixir de São Paulo, o ElugSP

Atualmente: Elixir desde 2015, Python desde 2017 e Ruby antes disso. (E no meio tempo C#, Clojure, JS, Fortran…)

Telnyx LLC

- A primeira plataforma global de comunicações com rede definida por software

  (AWS esta para o datacenter como a Telnyx esta pra telefonia)


AM3
TR1
LD6
FR5
VAN2
CH1
DC2
SV1
MI
HK
SG1
SP
SY
ME
POINT OF PRESENCE
PLANNED POINT OF PRESENCE
FIBER BACKBONE
PLANNED FIBER BACKBONE

Objetivos

- Dar o 1o passo para "subir de nível" na nossa interação com o banco de dados
- Compartilhar a minha experiência a respeito de quais aspectos do banco de dados são mais importantes & menos compreendidos pelos devs
- Tentar transferir um pouco do conhecimento que é meio "lugar comum" na dba-lândia para a developer-lândia
- Não passar vergonha

"X framework é lento"

"X framework é lento"

(Vamo re-escrever em Elixir!)

Metricas & Observabilidade

É fundamental quando se fala de performance. Conseguir observar seu app rodando é "higiene básica", não é opcional.

Não se começa uma discussão sobre performance sem métricas.

Computadores são notoriamente imprevisíveis.

Mas como estamos na Rails-lândia, a vida é feliz. Estamos a um `gem install` de várias soluções maneiras.

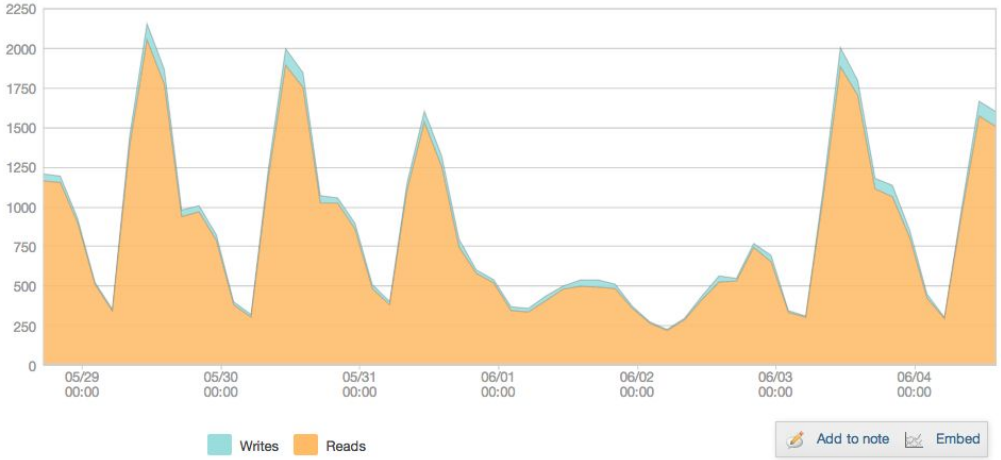
New Relic

Applications
Transactions
Mobile NEW
Servers
Dashboards
F5 LTM NEW
Memcached NEW
MS SQL
MySQL
Nginx NEW
Plugins
Tools

Overview

SQL Volume

This graph shows the total Reads (SELECTs) and Writes (INSERTs,UPDATEs,DELETEs,REPLACEs) executed



PUBLISHED AND SUPPORTED BY

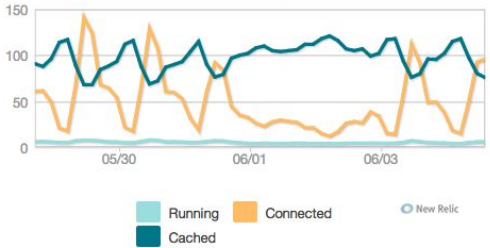
New Relic Inc.
About us | Support site

Key Utilizations

	Percent
Innodb Buffer Pool Hit Ratio	99%
Query Cache Memory In Use	77.5%
Tmp Tables Written To Disk	41.5%
Query Cache Hit Utilization	36%
Connnection Utilization	16.6%

Database Connections

This graphs shows current database connection information



Network Traffic

This graph shows the bytes received and set to/from the MySQL instar

Recent events

All

TODAY

13:00 Alert on Example S - Slave 1
Details →

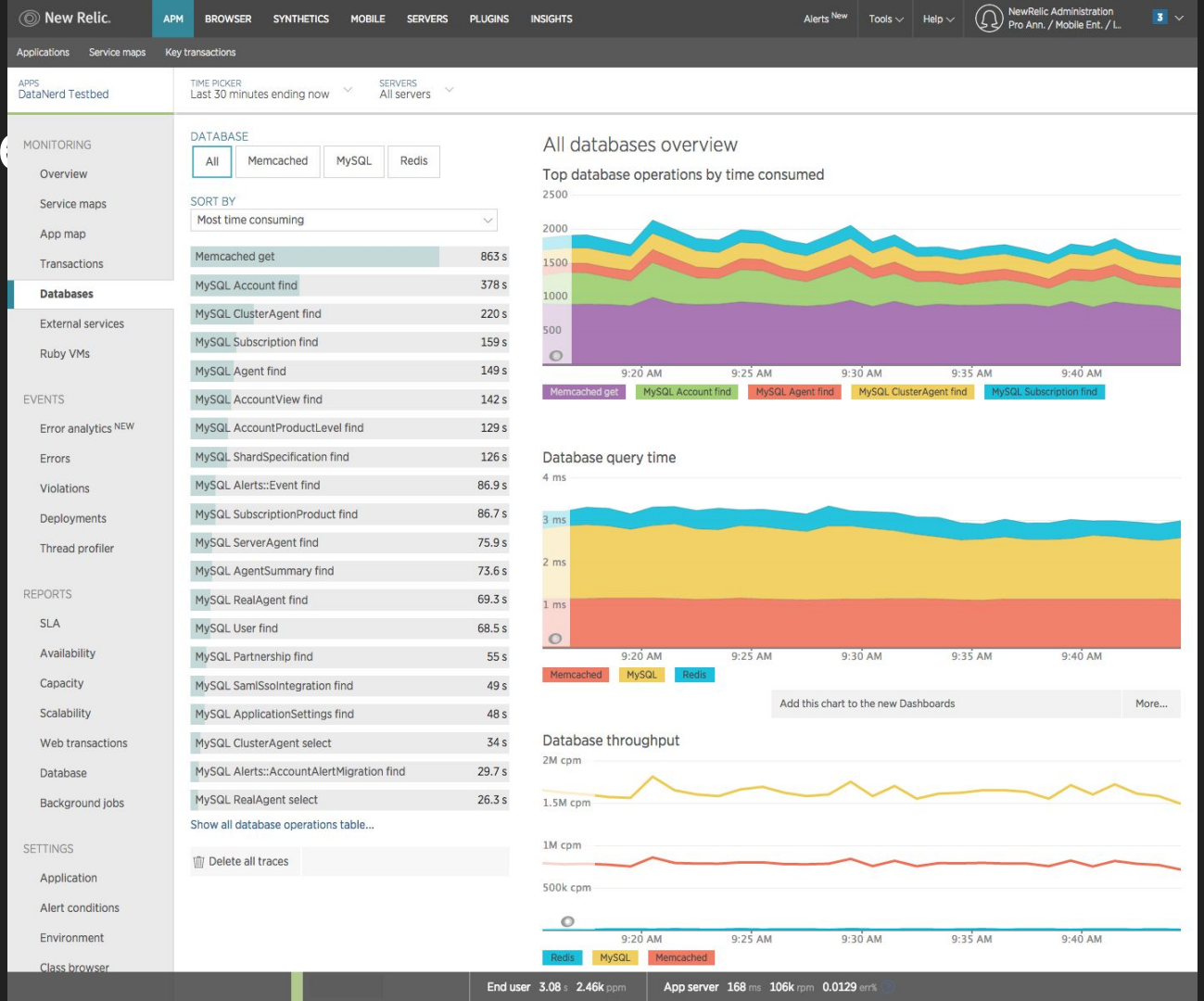
Nev

Summary | Trace details | SQL statements

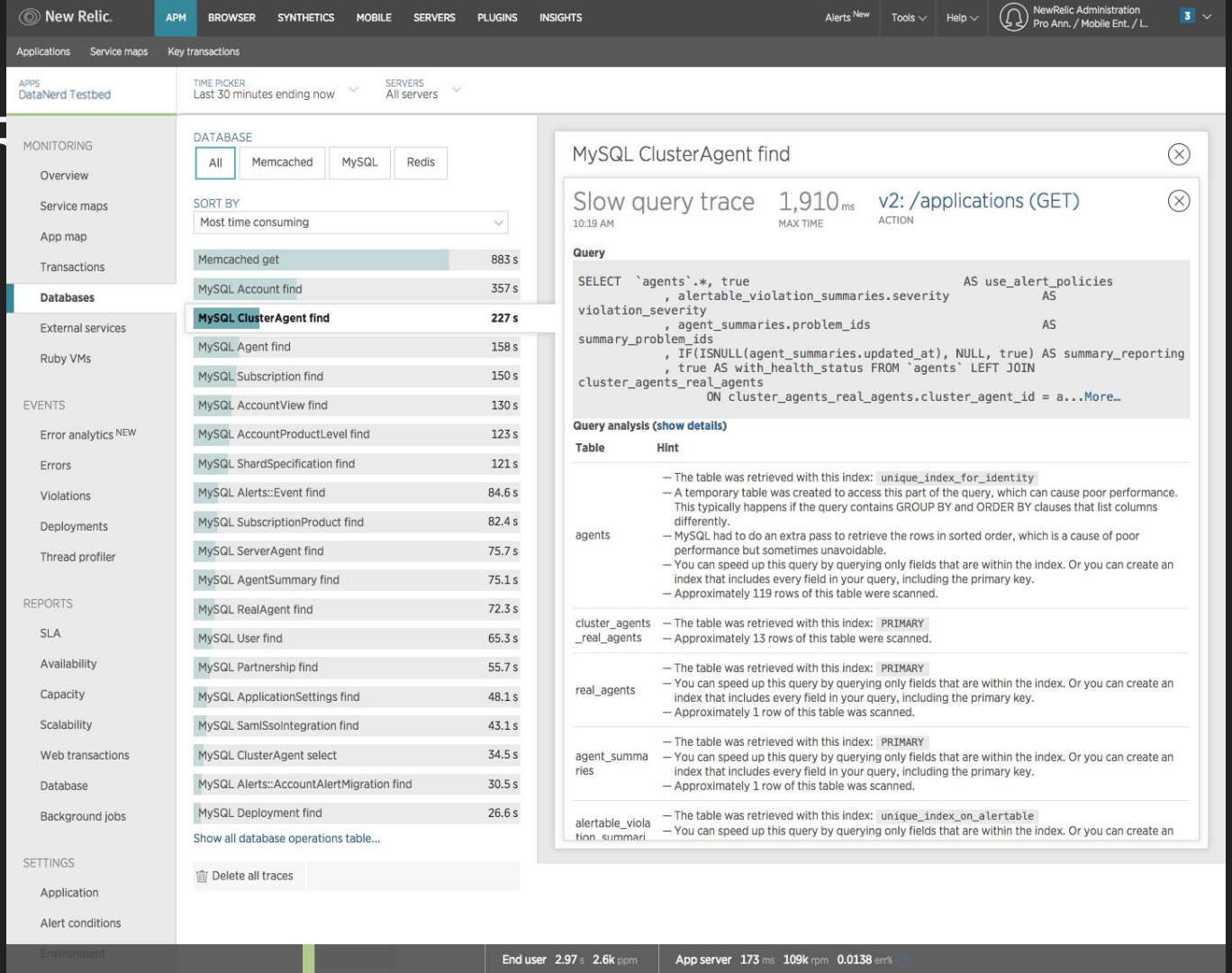
Expand all | Collapse all

Duration (ms)	Duration (%)	Segment	Drilldown	Timestamp
218,126	46.58%	applications/_current_main_charts.html.haml Partial		0.350 s
218,125	46.58%	applications/_current_main_charts.html.haml Template		0.351 s
218,124	46.58%	PageHeadersHelper.show_solr_tab?		0.351 s
198,539	42.40%	find_and_aggregate		0.351 s
160	0.03%	ApplicationMetricAgent#find_by_sql		0.352 s
18,834	4.02%	MetricAgent#find_by_sql		1.300 s
353	0.08%	GC Execution		20.408 s
177,653	37.94%	.do_find_and_aggregate		20.761 s
0.0	0.00%	TimesliceTableInfo#find_by_sql		20.762 s
177,602	37.93%	.fast_time_aggregating_query		20.794 s
170,041	36.31%	.fast_time_aggregating_query_for_shard		20.795 s
169,231	**36.14%**	Timeslice#find		21.060 s
214,102	45.72%	layouts/application/_main_nav.html.haml Partial		242.198 s
214,102	45.72%	layouts/application/_main_nav.html.haml Template		242.198 s
192,832	41.18%	find_and_aggregate		242.199 s
577	0.12%	ApplicationMetricAgent#find_by_sql		242.199 s
19,460	4.16%	MetricAgent#find_by_sql		243.546 s
170,565	36.43%	.do_find_and_aggregate		263.270 s
3.0	0.00%	TimesliceTableInfo#find_by_sql		263.271 s
170,509	36.41%	.fast_time_aggregating_query		263.307 s
162,869	34.78%	.fast_time_aggregating_query_for_shard		263.307 s
543	0.12%	GC Execution		263.598 s
161,412	**34.47%**	Timeslice#find		264.141 s
7,640	1.63%	.fast_time_aggregating_query_for_shard		426.176 s

New Re


New Relic.
APM
BROWSER
SYNTHETICS
MOBILE
SERVERS
PLUGINS
INSIGHTS
Alerts New
Tools
Help
NewRelic Administration
Pro Ann. / Mobile Ent. / L...
Applications
Service maps
Key transactions
APPS
DataNerd Testbed
TIME PICKER
Last 30 minutes ending now
SERVERS
All servers
MONITORING
Overview
Service maps
App map
Transactions
Databases
External services
Ruby VMs
EVENTS
Error analytics NEW
Errors
Violations
Deployments
Thread profiler
REPORTS
SLA
Availability
Capacity
Scalability
Web transactions
Database
Background jobs
SETTINGS
Application
Alert conditions
Environment
Class browser
DATABASE
All
Memcached
MySQL
Redis
SORT BY
Most time consuming
Memcached get 863 s
MySQL Account find 378 s
MySQL ClusterAgent find 220 s
MySQL Subscription find 159 s
MySQL Agent find 149 s
MySQL AccountView find 142 s
MySQL AccountProductLevel find 129 s
MySQL ShardSpecification find 126 s
MySQL Alerts::Event find 86.9 s
MySQL SubscriptionProduct find 86.7 s
MySQL ServerAgent find 75.9 s
MySQL AgentSummary find 73.6 s
MySQL RealAgent find 69.3 s
MySQL User find 68.5 s
MySQL Partnership find 55 s
MySQL SamlSsoIntegration find 49 s
MySQL ApplicationSettings find 48 s
MySQL ClusterAgent select 34 s
MySQL Alerts::AccountAlertMigration find 29.7 s
MySQL RealAgent select 26.3 s
Show all database operations table...
Delete all traces
All databases overview
Top database operations by time consumed
2500
2000
1500
1000
500
9:20 AM
9:25 AM
9:30 AM
9:35 AM
9:40 AM
Memcached get
MySQL Account find
MySQL Agent find
MySQL ClusterAgent find
MySQL Subscription find
Database query time
4 ms
3 ms
2 ms
1 ms
9:20 AM
9:25 AM
9:30 AM
9:35 AM
9:40 AM
Memcached
MySQL
Redis
Add this chart to the new Dashboards
More...
Database throughput
2M cpm
1.5M cpm
1M cpm
500k cpm
9:20 AM
9:25 AM
9:30 AM
9:35 AM
9:40 AM
Redis
MySQL
Memcached
End user 3.08 s 2.46k ppm
App server 168 ms 106k rpm 0.0129 err%

New R


New Relic.
APM
BROWSER
SYNTHETICS
MOBILE
SERVERS
PLUGINS
INSIGHTS
Alerts New
Tools
Help
NewRelic Administration
Pro Ann. / Mobile Ent. / L...
Applications
Service maps
Key transactions
APPS
DataNerd Testbed
TIME PICKER
Last 30 minutes ending now
SERVERS
All servers
MONITORING
Overview
Service maps
App map
Transactions
Databases
External services
Ruby VMs
EVENTS
Error analytics NEW
Errors
Violations
Deployments
Thread profiler
REPORTS
SLA
Availability
Capacity
Scalability
Web transactions
Database
Background jobs
SETTINGS
Application
Alert conditions
Environment
DATABASE
All
Memcached
MySQL
Redis
SORT BY
Most time consuming
Memcached get 883 s
MySQL Account find 357 s
MySQL ClusterAgent find 227 s
MySQL Agent find 158 s
MySQL Subscription find 150 s
MySQL AccountView find 130 s
MySQL AccountProductLevel find 123 s
MySQL ShardSpecification find 121 s
MySQL Alerts::Event find 84.6 s
MySQL SubscriptionProduct find 82.4 s
MySQL ServerAgent find 75.7 s
MySQL AgentSummary find 75.1 s
MySQL RealAgent find 72.3 s
MySQL User find 65.3 s
MySQL Partnership find 55.7 s
MySQL ApplicationSettings find 48.1 s
MySQL SamlSsoIntegration find 43.1 s
MySQL ClusterAgent select 34.5 s
MySQL Alerts::AccountAlertMigration find 30.5 s
MySQL Deployment find 26.6 s
Show all database operations table...
Delete all traces
MySQL ClusterAgent find
Slow query trace
10:19 AM
1,910 ms
MAX TIME
v2: /applications (GET)
ACTION
Query
SELECT `agents`.*, true AS use_alert_policies
, alertable_violation_summaries.severity AS violation_severity
, agent_summaries.problem_ids AS summary_problem_ids
, IF(ISNULL(agent_summaries.updated_at), NULL, true) AS summary_reporting
, true AS with_health_status FROM `agents` LEFT JOIN cluster_agents_real_agents
ON cluster_agents_real_agents.cluster_agent_id = a...More…
Query analysis (show details)
Table
Hint
agents
— The table was retrieved with this index: unique_index_for_identity
— A temporary table was created to access this part of the query, which can cause poor performance. This typically happens if the query contains GROUP BY and ORDER BY clauses that list columns differently.
— MySQL had to do an extra pass to retrieve the rows in sorted order, which is a cause of poor performance but sometimes unavoidable.
— You can speed up this query by querying only fields that are within the index. Or you can create an index that includes every field in your query, including the primary key.
— Approximately 119 rows of this table were scanned.
cluster_agents_real_agents
— The table was retrieved with this index: PRIMARY
— Approximately 13 rows of this table were scanned.
real_agents
— The table was retrieved with this index: PRIMARY
— You can speed up this query by querying only fields that are within the index. Or you can create an index that includes every field in your query, including the primary key.
— Approximately 1 row of this table was scanned.
agent_summaries
— The table was retrieved with this index: PRIMARY
— You can speed up this query by querying only fields that are within the index. Or you can create an index that includes every field in your query, including the primary key.
— Approximately 1 row of this table was scanned.
alertable_violation_summari
— The table was retrieved with this index: unique_index_on_alertable
— You can speed up this query by querying only fields that are within the index. Or you can create an
End user 2.97 s 2.6k ppm
App server 173 ms 109k rpm 0.0138 err%

Transaction trace

Track as Key Transaction

Delete this trace

Dec 9, '16 8:38 pm	4,890 ms	301 ms (6.17%)	77.3 ms (1.58%)
TRACE TIME	RESP. TIME	USER CPU BURN	SYSTEM CPU BURN

Summary | Trace details | **Database queries**

Show fast queries (< 5ms)

Total duration	Call count	Query
4,340 ms	7	`SELECT fc.*FROM field_config fcWHERE (fc.field_name = :db_condition_placeholder_?) AND (fc.active = :db_condition_placeholder_?) AND (fc.storage_active = :db_condition_placeholder_?) AND (fc.deleted = :db_condition_placeholder_?)`
98 ms	3	`DELETE FROM variable WHERE (name = :db_condition_placeholder_?)`
10 ms	1	`INSERT INTO semaphore (name, value, expire) VALUES (:db_insert_placeholder_?, :db_insert_placeholder_?, :db_insert_placeholder_?)`

Query details

Duration	4,344 ms

Stack trace

```
in PDOStatement::execute called at /mnt/www/html/newrelic/docroot/includes/database/database.inc (2171)
in DatabaseStatementBase::execute called at /mnt/www/html/newrelic/docroot/includes/database/database.inc (683)
```

Transaction trace

Track as Key Transaction

Delete this trace

Dec 9, '16 8:38 pm	4,890 ms	301 ms (6.17%)	77.3 ms (1.58%)
TRACE TIME	RESP. TIME	USER CPU BURN	SYSTEM CPU BURN

Summary | Trace details | **Database queries**

Show fast queries (< 5ms)

Total duration	Call count	Query
4,340 ms	7	SELECT fc.*FROM field_config fcWHERE (fc.field_name = :db_condition_placeholder_?) AND (fc.active = :db_condition_placeholder_?) AND (fc.storage_active = :db_condition_placeholder_?) AND (fc.deleted = :db_condition_placeholder_?)
98 ms	3	DELETE FROM variable WHERE (name = :db_condition_placeholder_?)
10 ms	1	INSERT INTO semaphore (name, value, expire) VALUES (:db_insert_placeholder_?, :db_insert_placeholder_?, :db_insert_placeholder_?)

Query details

Duration	4,344 ms

Stack trace

```
in PDOStatement::execute called at /mnt/www/html/newrelic/docroot/includes/database/database.inc (2171)
in DatabaseStatementBase::execute called at /mnt/www/html/newrelic/docroot/includes/database/database.inc (683)
```

(Lembrando que existem alternativas ao NR)

- AppSignal
- AppOptics
- Pingdom Server Monitor (formerly Scout App)
- Stackify Retrace
- DynaTrace (formerly Ruxit)
- Atatus
- Datadog
- LogicMonitor
- AppDynamics
- Cloudwatch & todas as parafernalhas AWS
- .... etc etc etc etc

Ok. Mas agora como eu resolvo a query lenta?

Pra conseguir raciocinar sobre o que vimos, precisamos entender sobre como o banco de dados funciona.

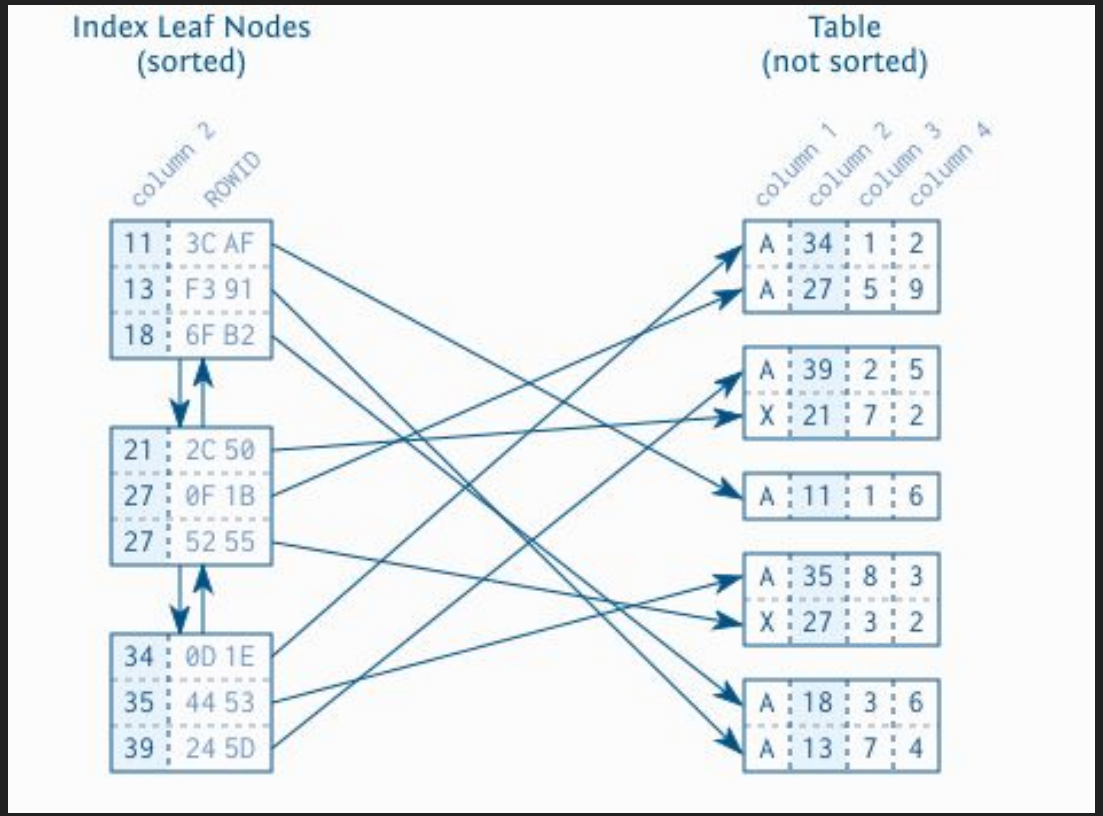
Vamos focar em apenas duas coisas no banco de dados: **Indices** e **Tabelas**.

- Vamos imaginar a "**tabela**" como um arquivo em que sempre escrevemos os dados sequencialmente. (pensa no `>>` do shell)
- Índices são um pouco mais envolvidos e vamos falar em seguida

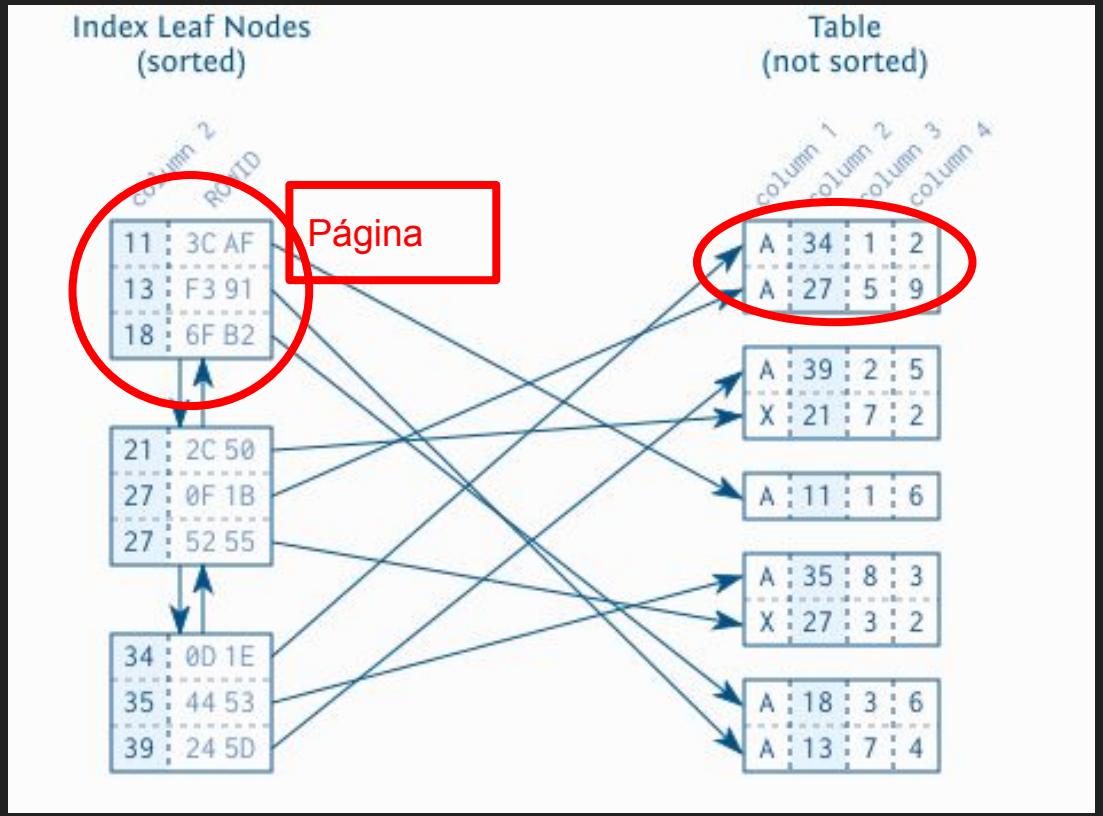
Use o indice, luke!

https://use-the-index-luke.com/

A anatomia do índice



A anatomia do índice



A principal coisa mal-compreendida

Quando vamos recuperar dados do banco, só podemos recuperar *páginas* inteiras, e não registros.

Se você precisa de um registro, você precisa ler toda a *página* que o contém. Não há escapatória.

PÁGINAS LIDAS >>> NUMERO DE REGISTROS

A anaton

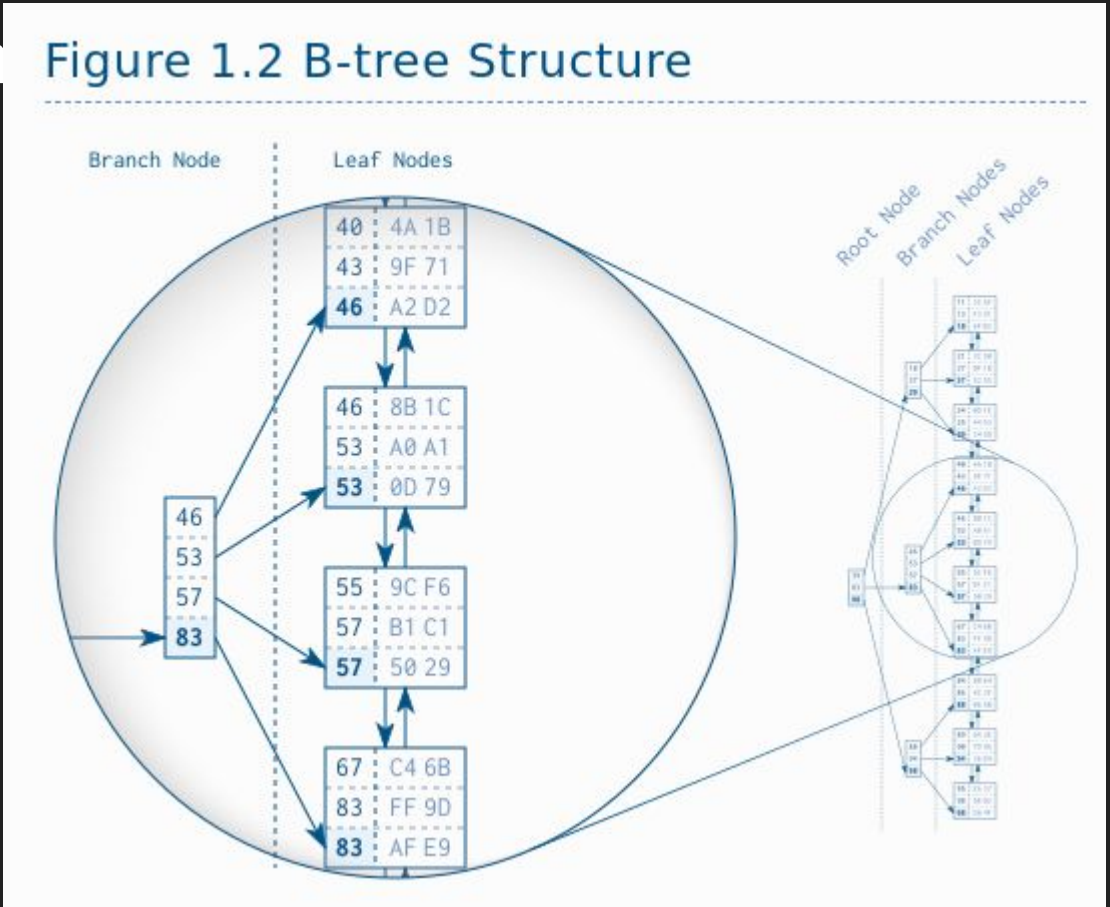


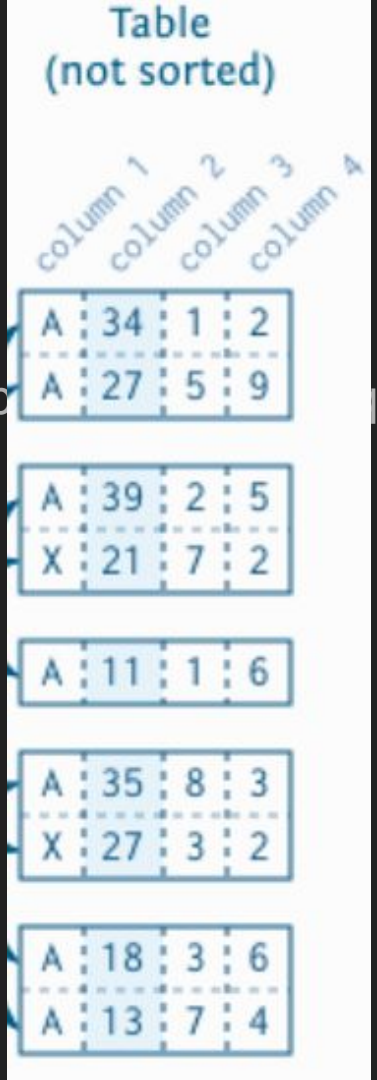
Figure 1.2 B-tree Structure
Branch Node
Leaf Nodes
40 4A 1B
43 9F 71
46 A2 D2
46 8B 1C
53 A0 A1
53 0D 79
55 9C F6
57 B1 C1
57 50 29
67 C4 6B
83 FF 9D
83 AF E9
46
53
57
83
Root Node
Branch Nodes
Leaf Nodes

Ta, mas como usa

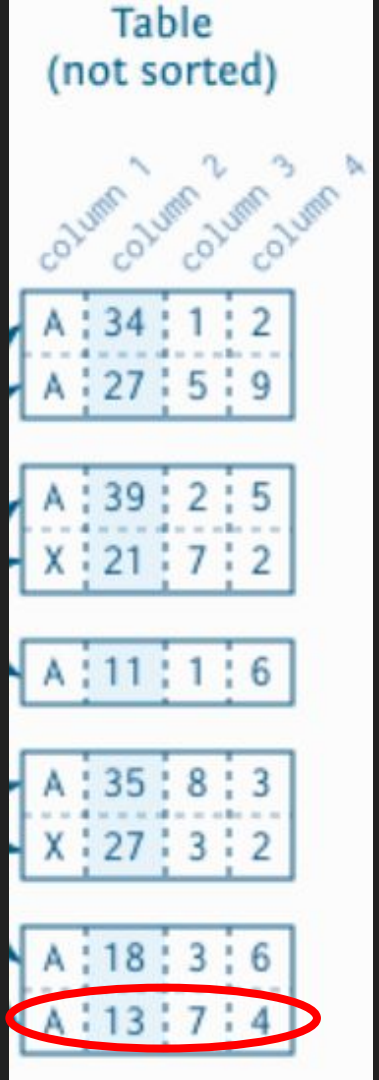
Temos essa estrutura doida de ponteiros. Como que isso ajuda?

Ta, mas como usa

Temos essa estrutura doida de p ue isso ajuda?

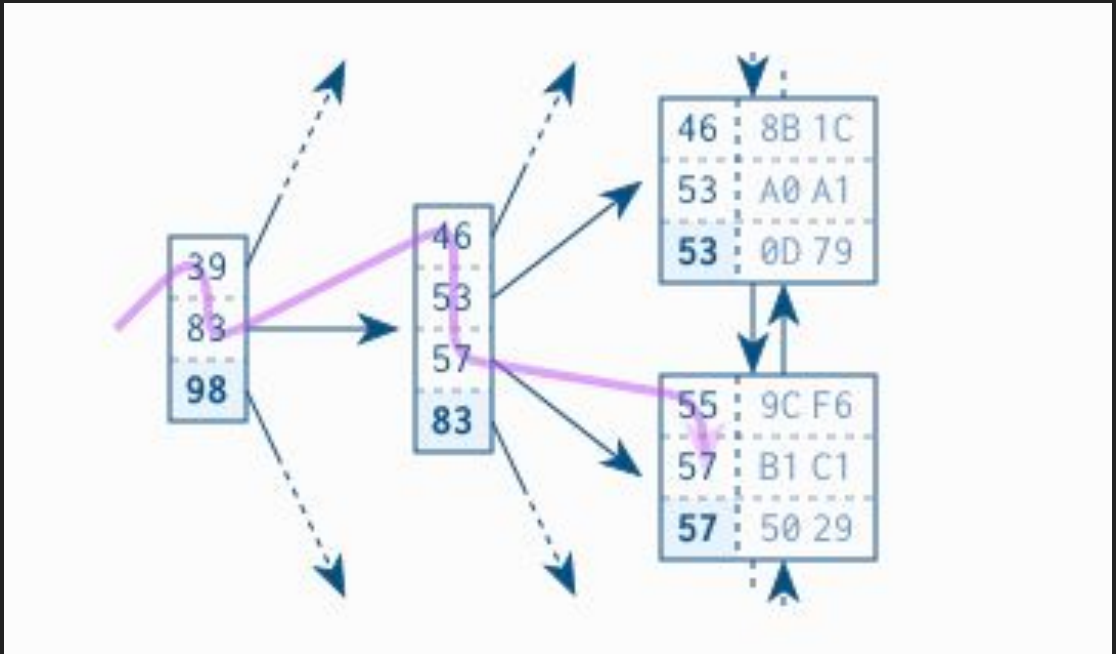


Ta, mas como usa



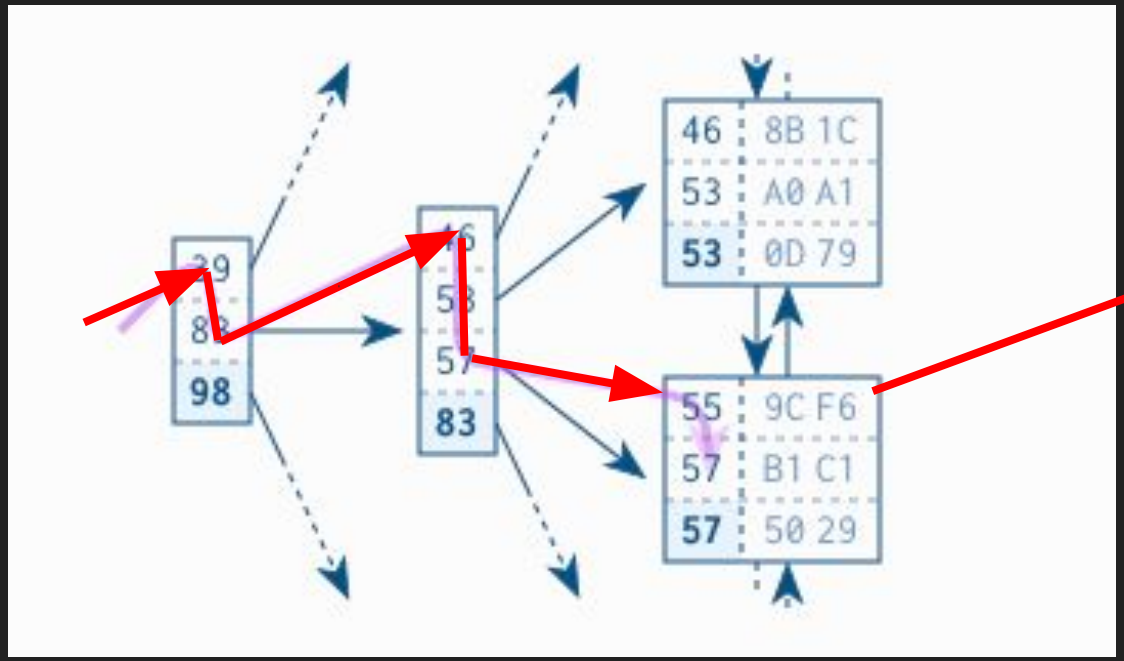
A anatomia do índice

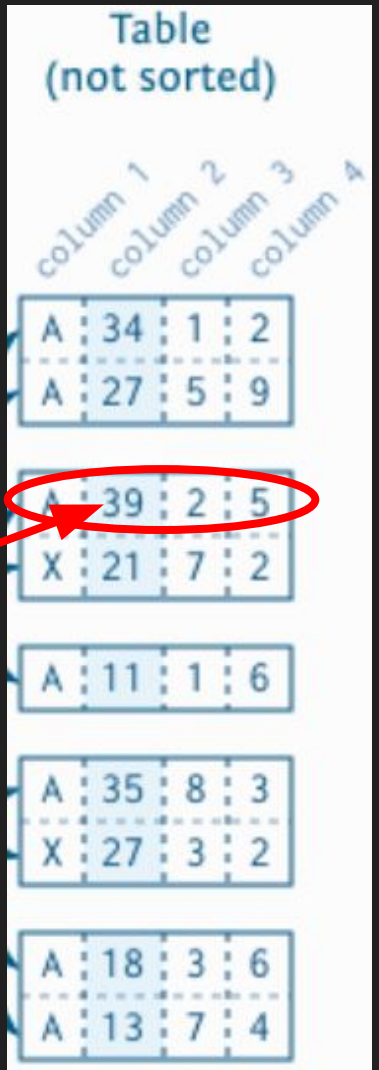
Buscando: id==57



A anatomia do índice

Buscando: id==55





Animações pra quem gosta:

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Range Scan

```
#+begin_src sql :results code
SELECT first_name, last_name, subsidiary_id, phone_number
  FROM employees
 WHERE subsidiary_id >= 40 and subsidiary_id <= 55
#+end_src
```

Range Scan

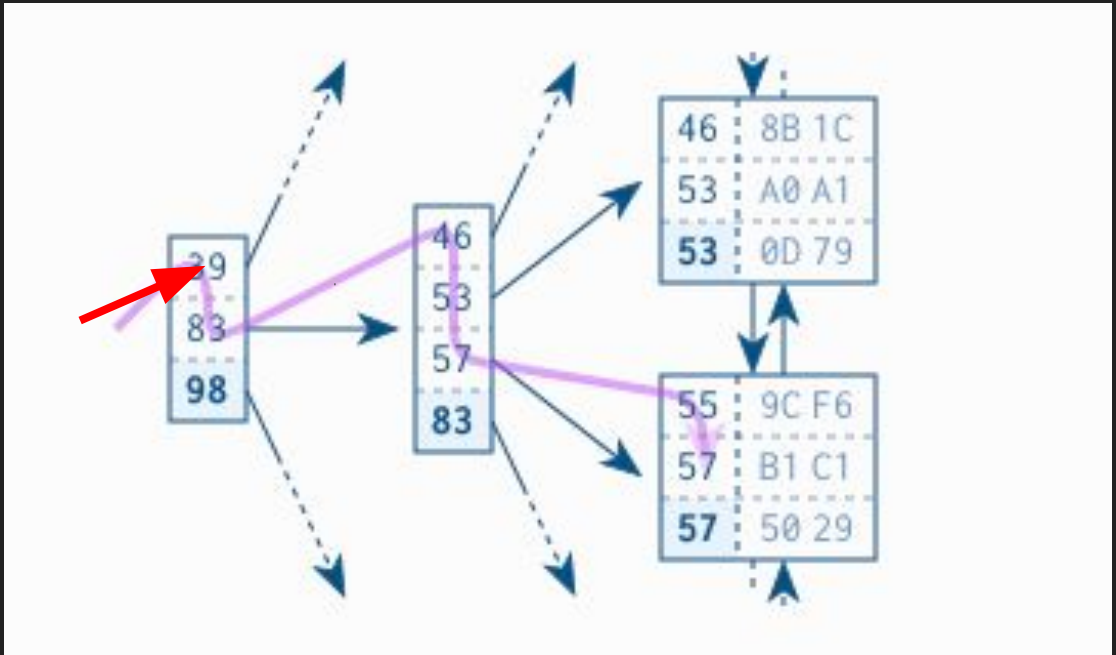


Table
(not sorted)

	column 1	column 2	column 3	column 4
	A	34	1	2
	A	27	5	9
	A	39	2	5
	X	21	7	2
	A	11	1	6
	A	35	8	3
	X	27	3	2
	A	18	3	6
	A	13	7	4

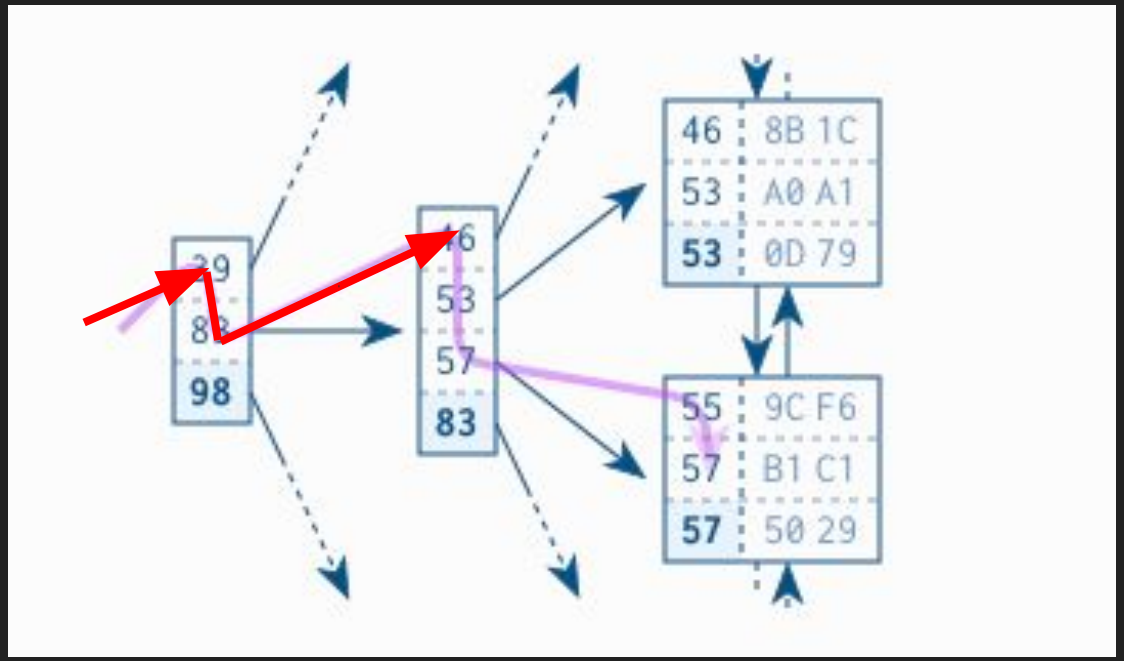


Table
(not sorted)

column 1	column 2	column 3	column 4
A	34	1	2
A	27	5	9
A	39	2	5
X	21	7	2
A	11	1	6
A	35	8	3
X	27	3	2
A	18	3	6
A	13	7	4

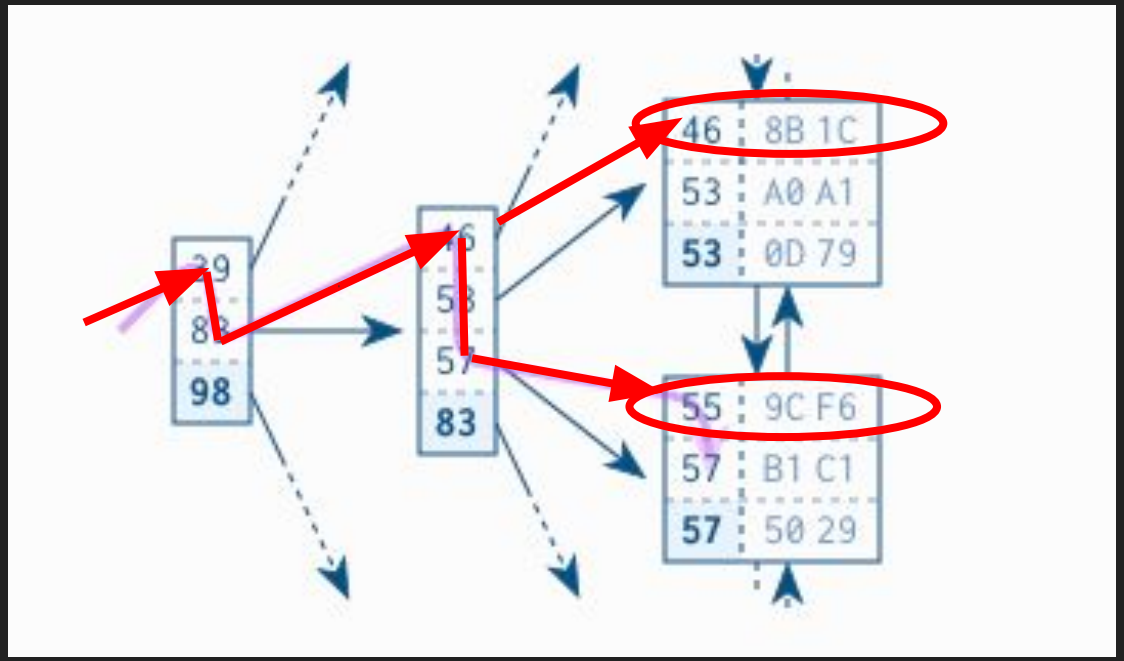


Table
(not sorted)

column 1	column 2	column 3	column 4
A	34	1	2
A	27	5	9
A	39	2	5
X	21	7	2
A	11	1	6
A	35	8	3
X	27	3	2
A	18	3	6
A	13	7	4

Range Scan

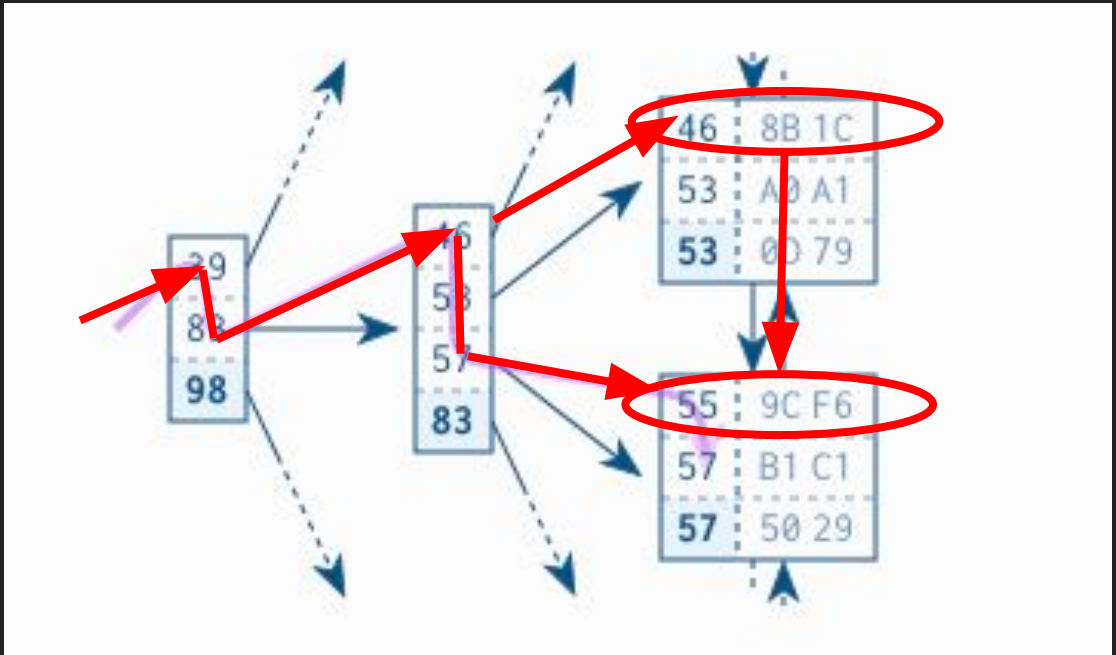
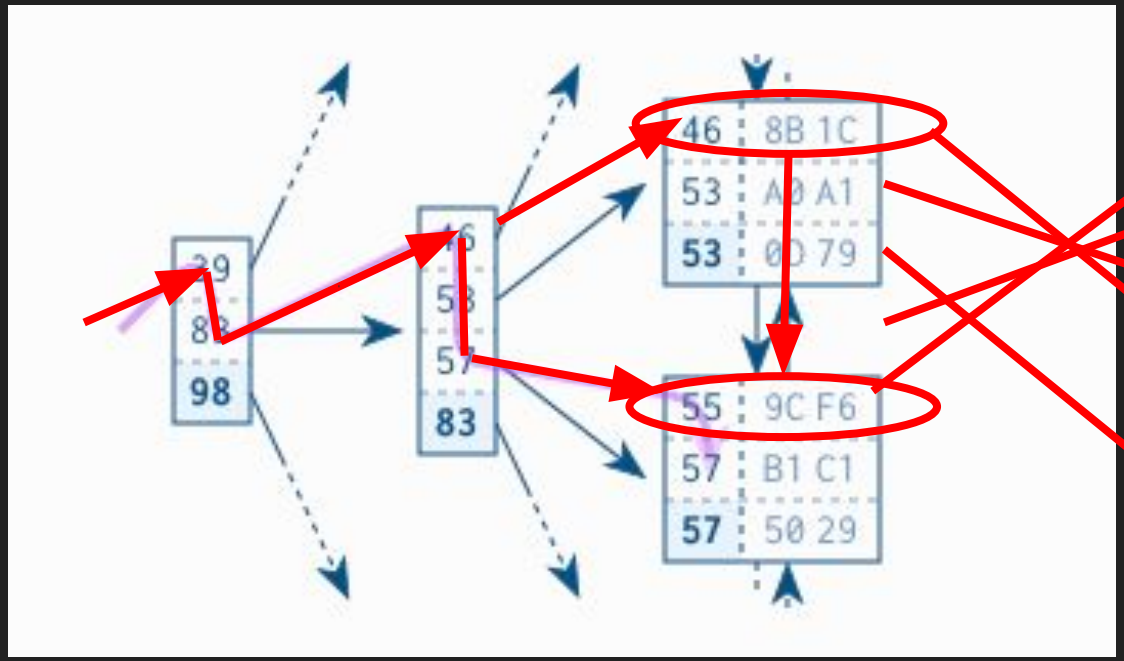


Table
(not sorted)

column 1	column 2	column 3	column 4
A	34	1	2
A	27	5	9
A	39	2	5
X	21	7	2
A	11	1	6
A	35	8	3
X	27	3	2
A	18	3	6
A	13	7	4

Range Scan


98
83
46 8B 1C
53 A0 A1
53 0D 79
55 9C F6
57 B1 C1
57 50 29

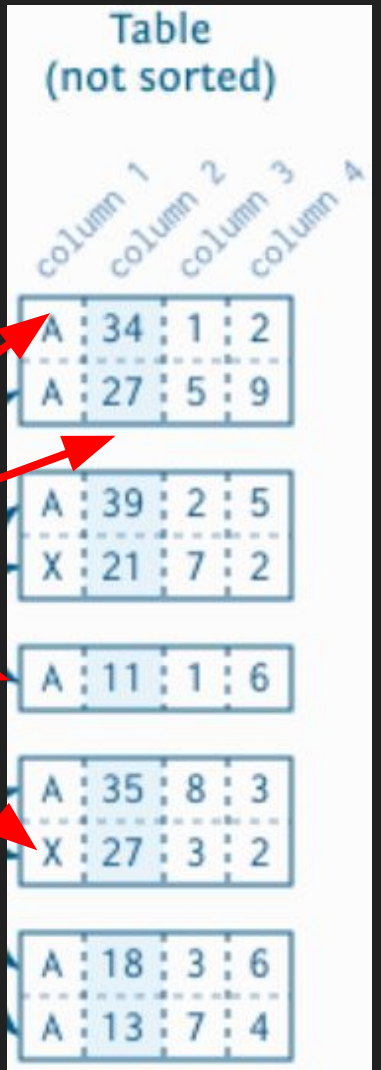


Table
(not sorted)
column 1
column 2
column 3
column 4
A 34 1 2
A 27 5 9
A 39 2 5
X 21 7 2
A 11 1 6
A 35 8 3
X 27 3 2
A 18 3 6
A 13 7 4

```
#+begin_src sql :results code
SELECT first_name, last_name, subsidiary_id, phone_number
  FROM employees
 WHERE last_name  = 'WINAND'
   AND subsidiary_id = 30
#+end_src
#+begin_src
```

```
-------------------------------------------------------------------------
|Id |Operation                    | Name          | Rows | Cost |
-------------------------------------------------------------------------
| 0 |SELECT STATEMENT             |               |    1 |   30 |
|*1 | TABLE ACCESS BY INDEX ROWID| EMPLOYEES     |    1 |   30 |
|*2 |  INDEX RANGE SCAN           | EMPLOYEES_PK |   40 |    2 |
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
  1 - filter("LAST_NAME"='WINAND')
  2 - access("SUBSIDIARY_ID"=30)
#+end_src
```

Top

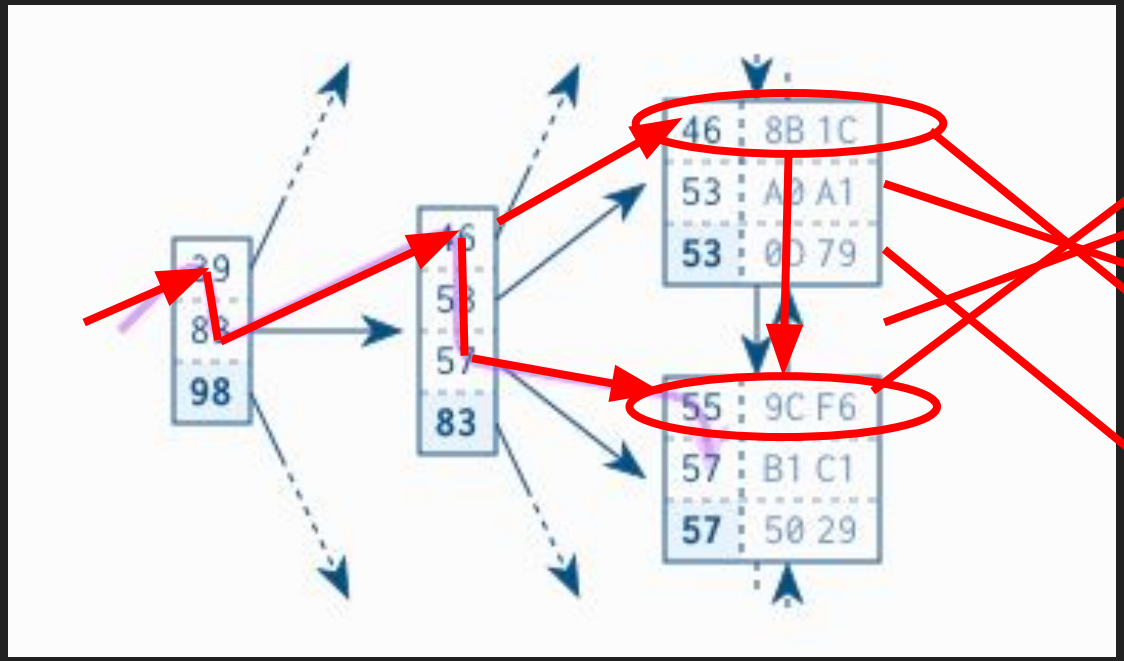
Se você tem um banco chave-valor, isso já é o suficiente.

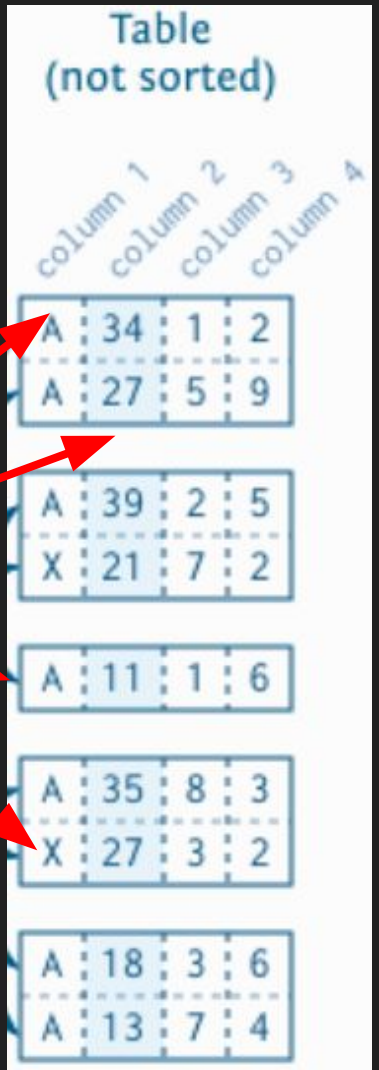
Então é só jogar índice em tudo e já era?

Não é bem assim...

Caso degenerado: tabela esparça

Caso degenerado: tabela esparça





Caso degenerado: tabela esparça

```
#+begin_src
Total rows: 100000

-----------------------------------------------------------
| Id | Operation          | Name      | Rows   | Cost |
-----------------------------------------------------------
|  0 | SELECT STATEMENT   |           | 100000 |  477 |
|* 1 |  TABLE ACCESS FULL| EMPLOYEES |    100 |  477 |
-----------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("LAST_NAME"='WINAND' AND "SUBSIDIARY_ID"=30)
#+end_src
```

Bottomline

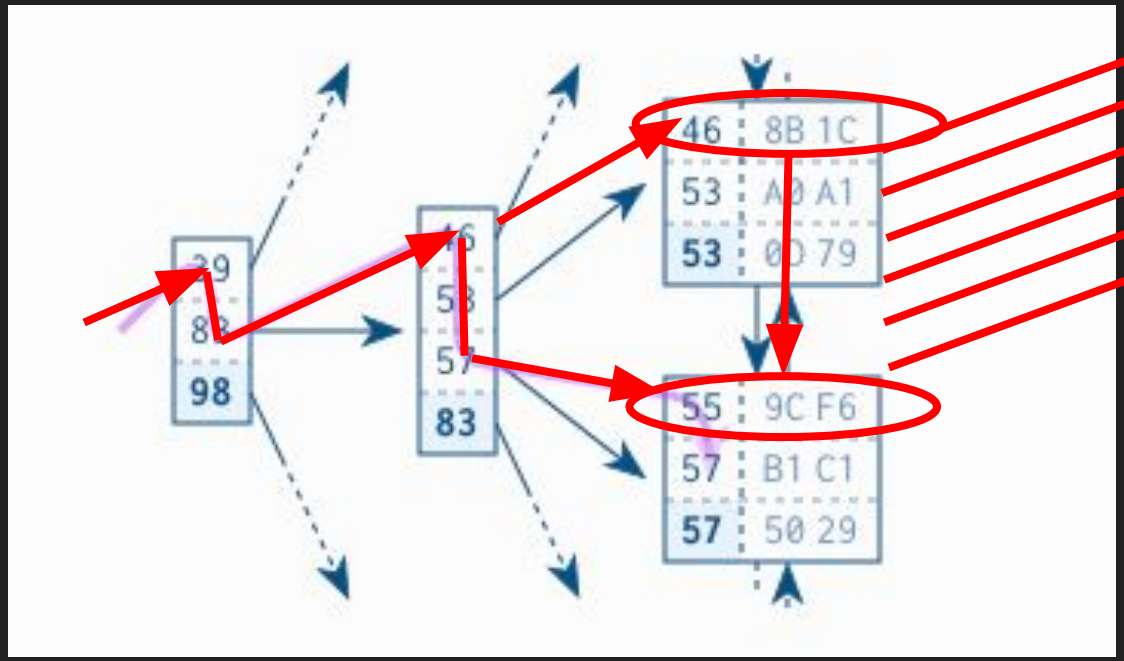
O índice não faz milagre

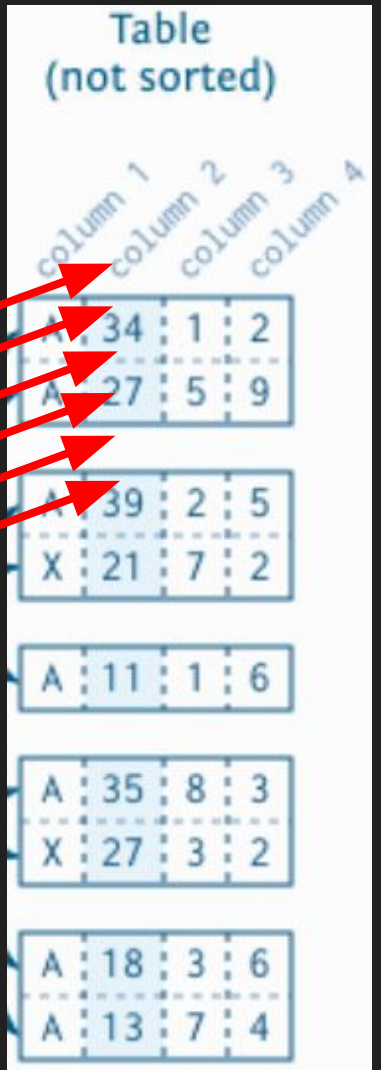
Se a engine de consulta do banco não tiver certeza que o uso do índice vai reduzir o número de leituras, ele não será usado. (ESTATISTICAS!)

Como resolve?

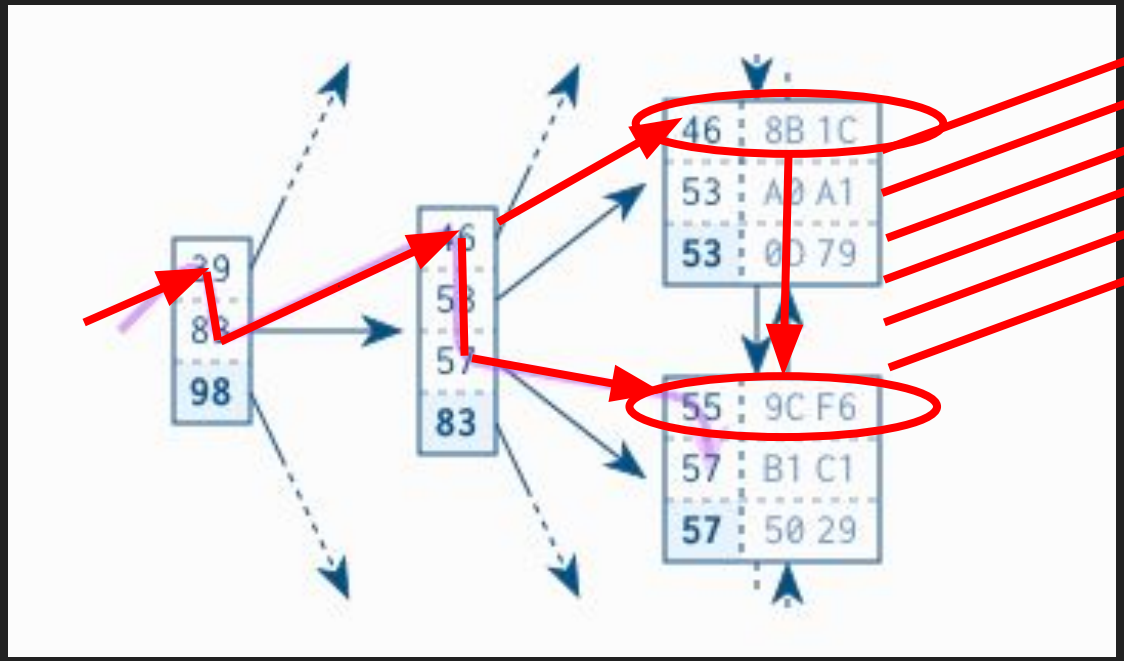
1. "Clusterizar" a tabela
2. Evitar tocar a tabela por completo (covering indx)
3. Diminua o result-set
4. Diminua o número de registros por página (desperdiça espaço)
5. Pede ajuda pro DBA

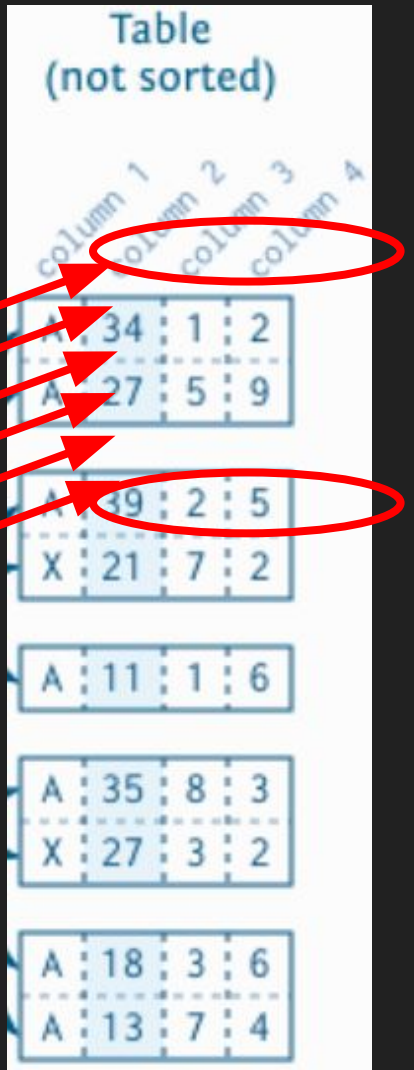
Clustered-table





Clustered-table

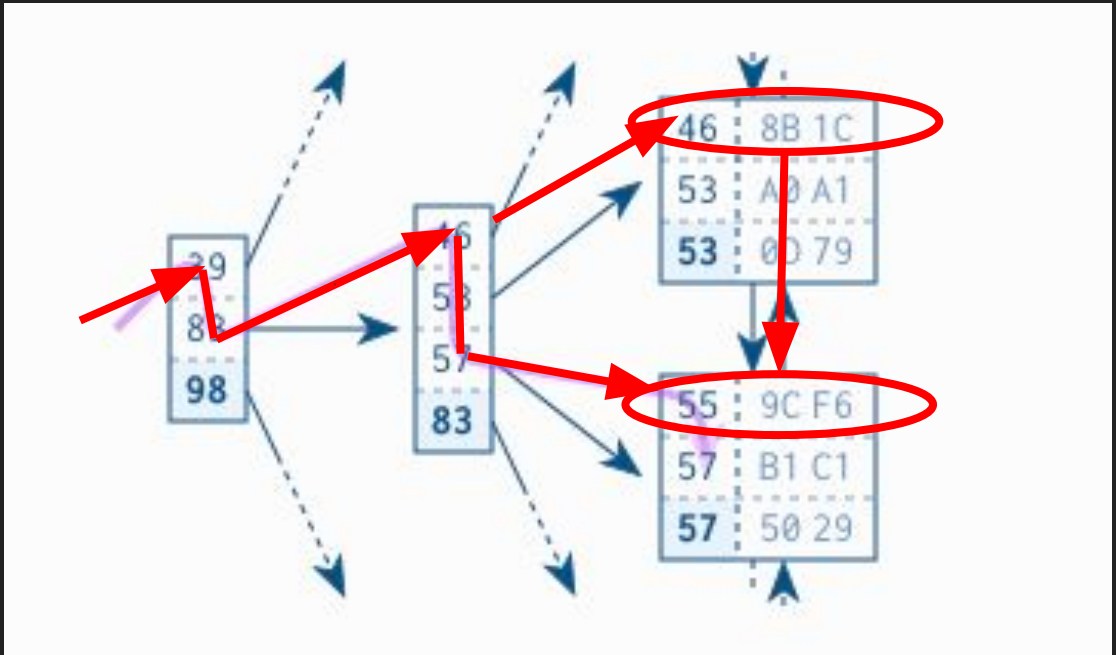


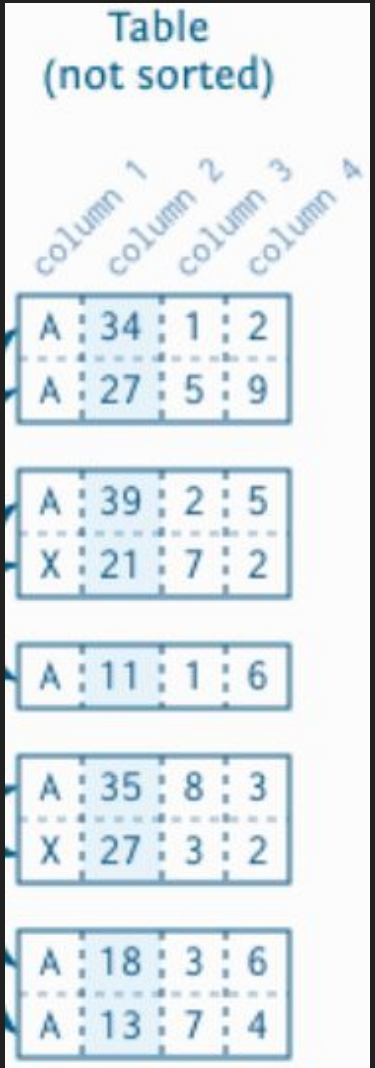


Index-only scan

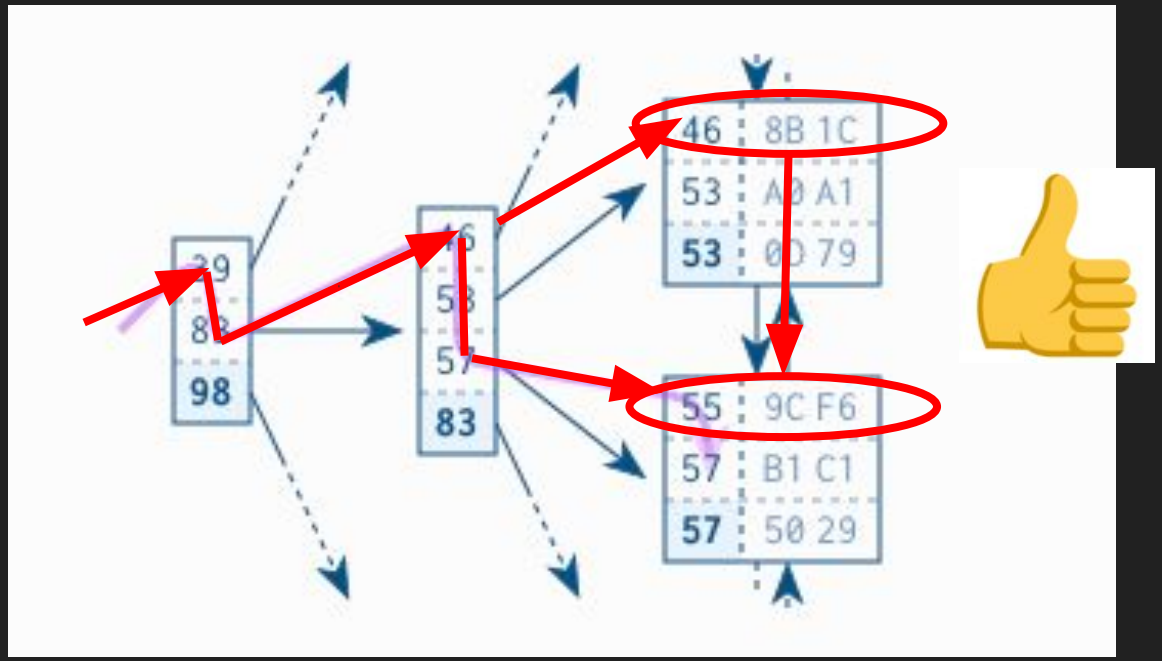
```
#+begin_src sql :results code
CREATE INDEX sales_sub_eur
    ON sales ( subsidiary_id, eur_value )
SELECT SUM(eur_value) FROM sales
 WHERE subsidiary_id = ?
#+end_src

#+begin_src
-----------------------------------------------------------------
| Id  | Operation          | Name          |  Rows | Cost |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT   |               |     1 |  104 |
|   1 |  SORT AGGREGATE    |               |     1 |      |
|*  2 |   INDEX RANGE SCAN| SALES_SUB_EUR | 40388 |  104 |
-----------------------------------------------------------------
#+end_src
```

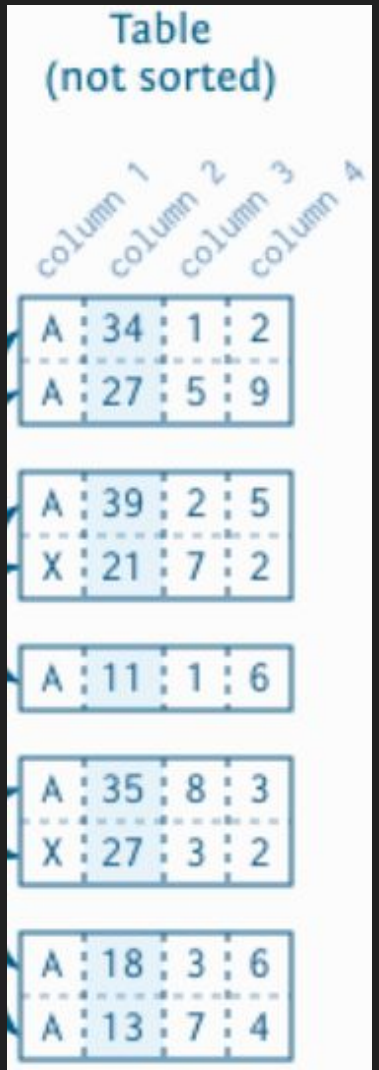
Clustered-table





Clustered-table





A performance depende não só da sua consulta, mas também da *disposição dos dados no disco*, i.e. a ordem em que foram escritos na tabela (!!)

(ou seja, tem coisas que código apenas não resolve)

E não é só isso...

Tudo o que falamos também é condicionado às "estatísticas" de uso, e é radicalmente influenciado por caches em vários nívels e afins.

A engine do banco é *muito* esperta.

Tudo o que hipotetizamos precisa ser *verificado*

Tudo o que falamos também é influenciado por:

- Qual banco vc ta usando (postgresql, mysql, oracle, sqlserver…)
- Configuração do banco
- Hardware
- Sistema de arquivos e separação de volumes
- Hardware
- Rede
- etc

Se vc quiser ficar manjeiro

https://www.amazon.com.br/PostgreSQL-High-Performance-Ibrar-Ahmed/dp/1784392979

Discussão

Tá. Falamos isso tudo que provavelmente a maior parte da plateia já sabia.

Mas por que isso é importante?????

Conclusão

Falar de performance é difícil.

Existem diversas dimensões que precisamos avaliar sobre o que *performance* significa. Latência? Throughput? Consumo de memória? CPU? Variância de cada um desses? Dependência desses com escalabilidade horizontal/vertial, etc, etc.

Até um sistema web simples (e.g. `backend` + `banco` + `workers` + `jquery`), daqueles que cabe no heroku já possuem várias peças que influenciam a experiência total do usuário.

Dizer que "O framework " é lento e partir pra uma reescrita sem entender essas coisas e sem métricas, além de não resolver o problema, é um jeito muito eficiente de queimar dinheiro e neurônio

Obrigado

(dsclp qualquer coisa)

Contatos

Milhouse (Renan Ranelli)

milhouseonsoftware.com

renanranelli @ Twitter

/rranelli @ Github