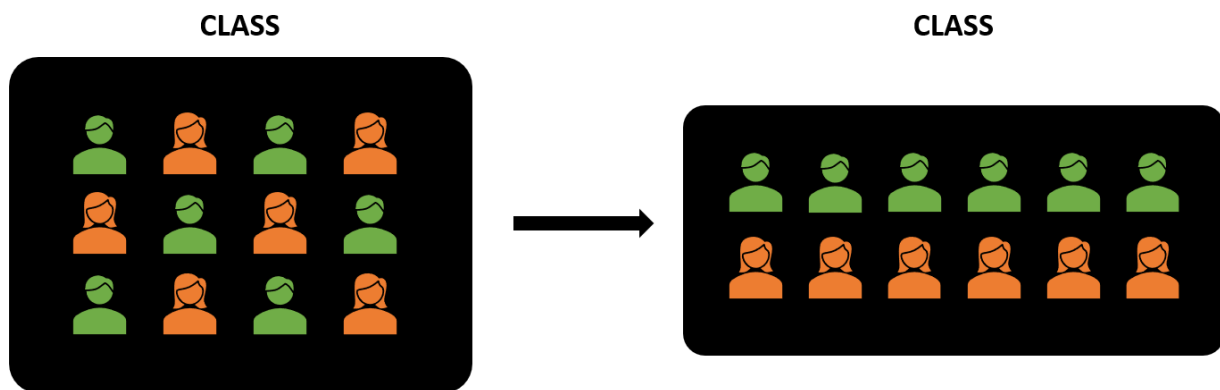




Partitioning data

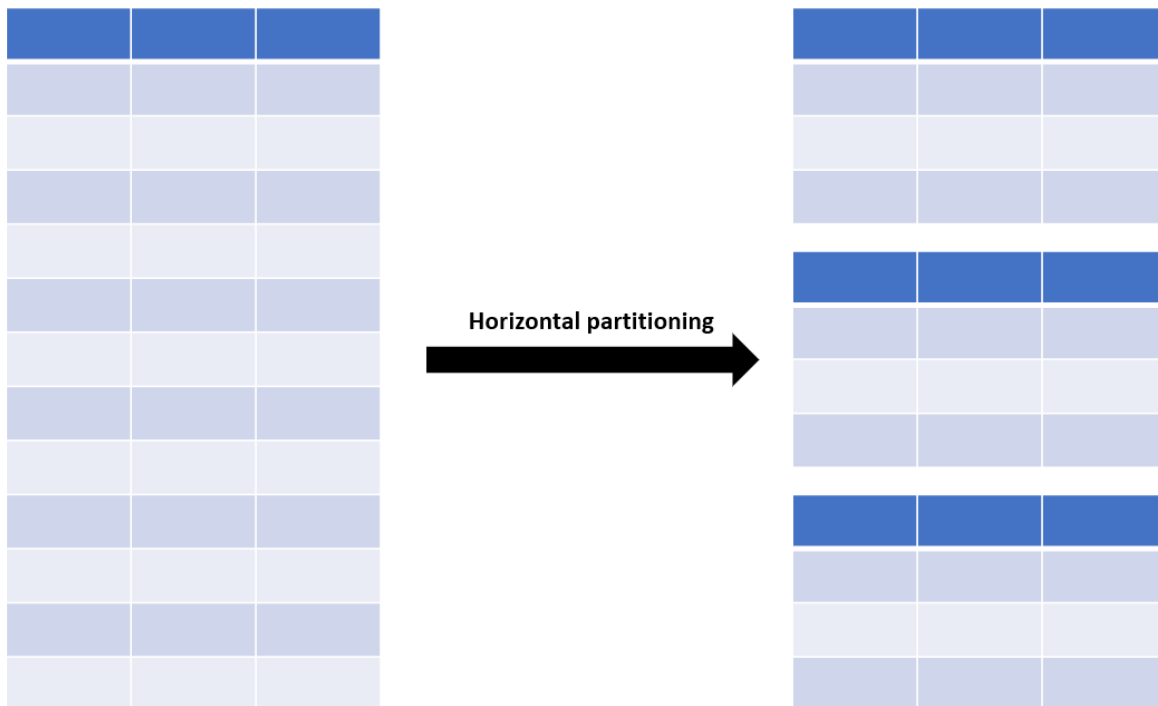
1. Horizontal partitioning

Vậy có cách nào làm cho query hiệu quả hơn ngoài việc index? Ý tưởng khá đơn giản dựa trên nguyên tắc chia để trị. Hãy tưởng tượng lớp học có 40 học sinh trong đó 12 nam và 8 nữ. Nếu cần tìm một bạn nữ tên Jane, thay vì scan từng học sinh, ta sẽ chia sẵn lớp học thành 2 dãy nam và nữ sao đó thực hiện scan trên dãy học sinh nữ.



Nếu có thêm học sinh vào lớp, chỉ cần sắp xếp bạn đó vào đúng nhóm là ok. Trên thực tế, việc thay đổi giới tính gần như không xảy ra, tuy nhiên nếu xảy ra thì việc cần làm là sắp xếp bạn đó về đúng nhóm của mình là xong.

Partitioning table chính là ví dụ phía trên. Chúng ta chia table lớn thành nhiều table nhỏ hơn, các table nhỏ hơn gọi là **partition table**, kế thừa toàn bộ cấu trúc của parent table, từ column cho đến kiểu dữ liệu. Việc chia nhỏ này được gọi là **horizontal partitioning**.



Horizontal partitioning có những ưu điểm sau:

- Giới hạn vùng dữ liệu phải scan trên table trong một vài trường hợp. Nếu ta cần tìm một học sinh tên John Doe không phân biệt giới tính thì việc partition như ví dụ trên không đem lại hiệu quả.
- **Partition table** cũng giống như một table thường nên ta có thể thực hiện index cho nó. Dẫn đến việc tốn ít cost hơn để maintain index table, do số lượng record ít hơn.
- Ngoài ra, việc xóa các dữ liệu trên **partition table** sẽ nhanh hơn và không ảnh hưởng đến các **partition** khác.

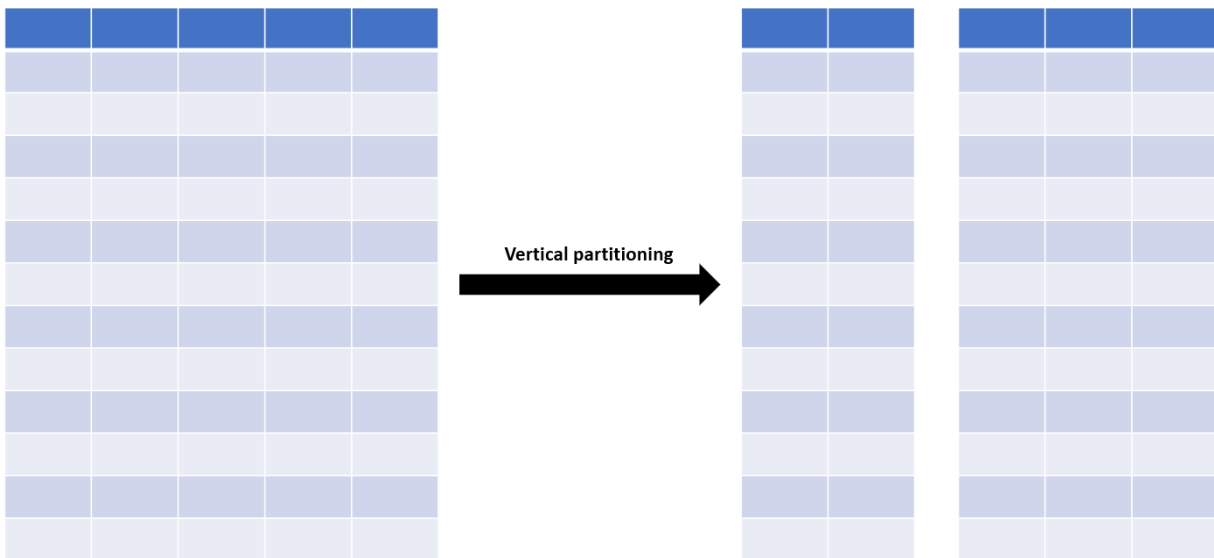
Với những ưu điểm trên, ta cần lưu ý khi thực hiện **horizontal partitioning** để đạt hiệu quả tối đa:

- Áp dụng với các table rất lớn. Thường là quá size của memory.
- Việc partition trên điều kiện nào phải dựa vào tính chất và tần suất của các query.

2. Vertical partitioning

Vertical partitioning cũng là việc chia một table ra thành nhiều **partition table** nhưng theo chiều.. dọc. Ví dụ một table 100 columns được partition thành 4 tables mỗi table 25 columns. Về cơ bản sẽ không có một tiêu chuẩn hay công thức cụ thể nào cho việc **vertical partitioning**. Ta chỉ cần chú ý đến việc nhóm các columns có tần suất query cùng nhau thành một partition.

Ngoài ra, với **vertical partitioning**, best practice là sử dụng chung một PK cho toàn bộ các partition table.



Vậy lợi ích của **vertical partitioning** là gì?

Trước tiên cần hiểu về cách data được lưu trữ xuống disk (HDD/SSD). Về cơ bản, các records được lưu thành một khối dữ liệu có độ lớn gần tương tự nhau được gọi là block. Do đó, nếu một table chứa số lượng column ít đồng nghĩa với việc tăng đường số lượng records lưu trữ trên một block. Như vậy nếu query các column trong cùng block, các xử lý tính toán I/O sẽ giảm đi phần nào dẫn tới việc tăng performance. Mình có giải thích kĩ hơn về block và page trong [bài này](#).

Trong thực tế, **vertical partitioning** nên được chú ý ngay từ khi design database vì việc này ảnh hưởng trực tiếp đến cách query và vận hành hệ thống do phải chia thành các table thực. **Horizontal partitioning** đơn giản hơn một chút tuy nhiên vẫn cần dừng hệ thống trong 1 khoảng thời gian để làm các thao tác sau:

- Back-up data cũ.
- Tạo partition table.

- Insert data vào table mới.

3. Horizontal partitioning by range

Bản chất của **partitioning** là phân chia ra các **partition table** dựa trên điều kiện phân vùng **partition key**. Với **partition by range**, điều kiện partition có thể là:

- Phân chia theo thời gian.
- Phân chia theo numeric từ 1 đến 5, 6 đến 10...
- Phân chia theo bảng chữ cái A, B, C; D, E, F...

Khi thực hiện **range partition** ta quan tâm đến giá trị min và max của mỗi **partition** để thực hiện việc phân chia. Ví dụ table **ENGINEER** có thông tin về ngày bắt đầu làm việc (start_date), ta có thể dựa trên column này để phân chia **partition** theo từng quý. Như đã nói ở trên, ta cần back-up data và tạo lại table mới nhé:

```
DROP TABLE ENGINEER;
CREATE TABLE ENGINEER
(
    id bigserial NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL
) PARTITION BY RANGE(started_date);

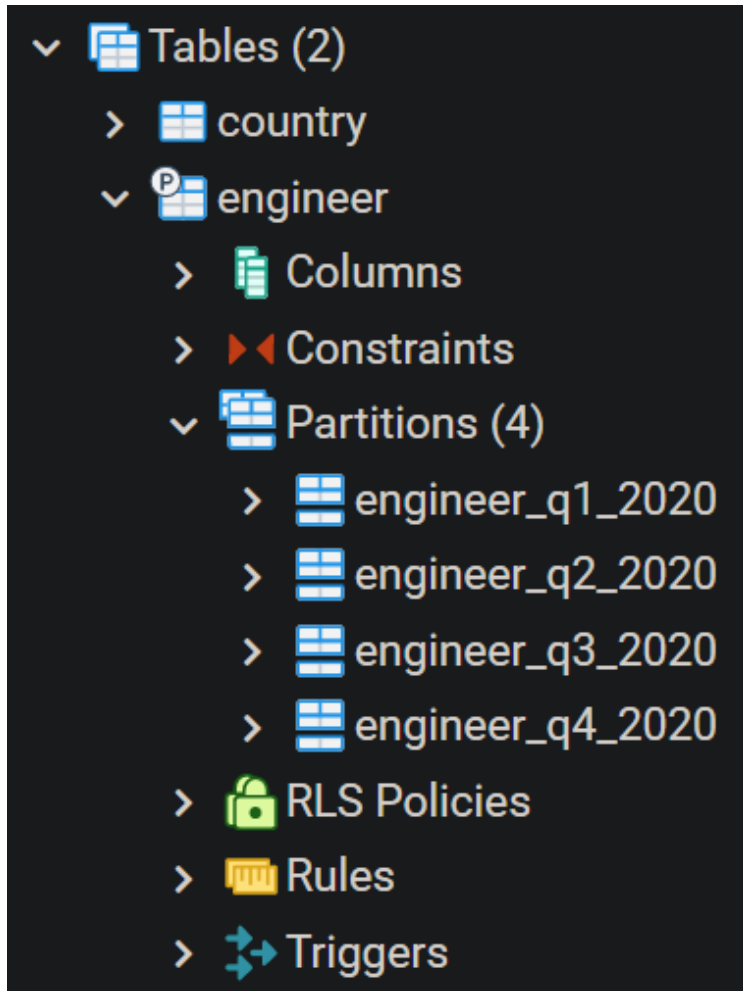
CREATE TABLE ENGINEER_Q1_2020 PARTITION OF ENGINEER FOR VALUES
    FROM ('2020-01-01') TO ('2020-04-01');

CREATE TABLE ENGINEER_Q2_2020 PARTITION OF ENGINEER FOR VALUES
    FROM ('2020-04-01') TO ('2020-07-01');

CREATE TABLE ENGINEER_Q3_2020 PARTITION OF ENGINEER FOR VALUES
    FROM ('2020-07-01') TO ('2020-10-01');



CREATE TABLE ENGINEER_Q4_2020 PARTITION OF ENGINEER FOR VALUES
    FROM ('2020-10-01') TO ('2020-12-31');
```

Thay vì back-up, mình sẽ insert data mới luôn.



Explain query một engineer với id = 10:



```
EXPLAIN SELECT * FROM ENGINEER WHERE id = 10;
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN  text 		
1	Append (cost=0.00..2380.00 rows=4 width=56)		
2	-> Seq Scan on engineer_q1_2020 (cost=0.00..595.90 rows=1 width=56)		
3	Filter: (id = 1)		
4	-> Seq Scan on engineer_q2_2020 (cost=0.00..591.69 rows=1 width=56)		
5	Filter: (id = 1)		
6	-> Seq Scan on engineer_q3_2020 (cost=0.00..600.41 rows=1 width=56)		
7	Filter: (id = 1)		
8	-> Seq Scan on engineer_q4_2020 (cost=0.00..592.00 rows=1 width=56)		
9	Filter: (id = 1)		

Sau khi thực hiện **partition**, một table **ENGINEER** sẽ bao gồm 4 **table partition**. Lúc này, query plan phải sequence scan trên toàn bộ partition để tìm ra record thỏa mãn điều kiện `id = 10`.

Chưa thấy có gì đặc biệt, thử tìm kiếm các engineer bắt đầu làm việc từ Apr 01, 2020:

```
EXPLAIN SELECT * FROM ENGINEER WHERE started_date = '2020-04-01';
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN  text 		
1	Append (cost=0.00..591.69 rows=241 width=56)		
2	-> Seq Scan on engineer_q2_2020 (cost=0.00..591.69 rows=241 width=56)		
3	Filter: (started_date = '2020-04-01'::date)		

Lúc này, query plan chỉ scan trên partition **engineer_q2_2020**. Đã thấy sự lợi hại của partition. Mặc dù ta không specific chỉ định partition **engineer_q2_2020** nhưng

PostgreSQL biết điều đó và tối ưu luôn cho chúng ta. Câu query trên tương tự với:

```
EXPLAIN SELECT * FROM ENGINEER_Q2_2020 WHERE started_date = '2020-04-01';
```

Mỗi **partition** được coi là một table riêng biệt và kế thừa các đặc tính của table. Ta hoàn toàn có thể thêm index cho từng partition để tăng performance cho query, được gọi là **local index**. Hoặc thêm index cho parent table, được gọi là **global index**.

Quick question, bạn có nhận ra điều gì đặc biệt khi tạo **partition table** không? Đó là nó không có PK, hay nói cách khác, không thể tạo constraint PK hay unique trên các column (riêng lẻ, không combine với partition column) của table. Thử nhé:

```
CREATE TABLE TEST_PK
(
    id bigserial NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL,
    PRIMARY KEY (id)
) PARTITION BY RANGE(started_date);

CREATE TABLE TEST_UNQ
(
    id bigserial UNIQUE NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL
) PARTITION BY RANGE(started_date);
```

Bạn có biết vì sao không? Nếu chưa thì chờ câu trả lời ở cuối bài.

Với ví dụ trên, ta có tổng cộng 4 partitions cho 4 quý của năm 2020. Nếu insert hoặc update một record không thuộc năm 2020 thì sao nhỉ:

```
UPDATE ENGINEER SET started_date = '2021-01-01';
```

```
INSERT INTO ENGINEER(first_name, last_name, gender, country_id, title, started_date, created)
VALUES('Hermina', 'Kuhlman', 3, 229, 'Backend Engineer', '2021-09-23', current_timestamp);
```

Cả 2 query đều báo lỗi vì.. không biết nhét chúng vào partition table nào. Do đó, nếu không chắc chắn về tập data của mình, ta cần tạo một table default partition để chứa các record không biết phân loại vào đâu.

```
CREATE TABLE ENGINEER_DEFAULT_PARTITION PARTITION OF ENGINEER DEFAULT;
```

PostgreSQL version 11 trở xuống không support default partition, các vesion từ 11 trở lên mới thực hiện được nhé. Bây giờ chạy lại query insert trên sẽ ngon ngay.

Vài câu hỏi khác được đặt ra:

- Có thể tạo một partition mới trong quá trình runtime không? Có thể có nhiều hơn 1 partition chứa cùng một khoảng thời gian không? Ví dụ partition: 2020-01-01:2020-04-01 và partition: 2020-02-01:2020-05-01.
- Có thể drop một partition trong quá trình runtime mà giữ nguyên data của partition đó không?

Với câu đầu tiên, ta hoàn toàn có thể tạo một partition mới, ví dụ là default partition ở trên. Với ý thứ hai, bản chất của partition là phân chia tập data ra thành các partition độc lập với nhau dựa trên các điều kiện cho trước. Vậy nên để không vi phạm quy tắc đó, ta không thể tạo 2 partition với **overlapping key**. Nôm na là một record không thể thuộc nhiều hơn một partition, các **partition key** không được trùng nhau hoặc chứa một phần trùng nhau.

Câu thứ hai, có hai trường hợp:

- Không có **default partition**: tất nhiên rồi, muốn giữ lại các record ta cần một partition phù hợp để chứa record đó. Trong trường hợp không có **default partition**, không có cách nào có thể giữ lại các record sau khi drop partition table.
- Có **default partition**: thực ra đây là trick question, nếu không hiểu bản chất rất dễ bị.. ăn cú lừa thế kỉ . Không cần quan tâm đến partition hay không partition, có **default partition table** hay không có **default partition table**, **DROP** table vẫn là **DROP** table, xóa toàn bộ tất cả các data, index, trigger, constraint.. và định nghĩa của table.

4. Partition by list

Với **list partitioning**, việc phân chia ra các partition dựa trên key được định nghĩa dưới dạng list of value. Ví dụ với table **ENGINEER**, các engineer có title **Backend Engineer**, **Frontend Engineer**, **Fullstack Engineer** nhóm vào thành một partition; **BA** và **QA** một partition; còn lại là **default partition**. Triển khai:

```
DROP TABLE ENGINEER CASCADE;

CREATE TABLE ENGINEER
(
    id bigserial NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL
) PARTITION BY LIST(title);

CREATE TABLE ENGINEER_ENGINEER PARTITION OF ENGINEER FOR VALUES
    IN ('Backend Engineer', 'Frontend Engineer', 'Fullstack Engineer');

CREATE TABLE ENGINEER_BA_QA PARTITION OF ENGINEER FOR VALUES
    IN ('BA', 'QA');

CREATE TABLE ENGINEER_DEFAULT PARTITION OF ENGINEER DEFAULT;
```

Về mục đích cũng không khác gì **partition by range** ngoài việc chia nhỏ ra thành nhiều partition. Tuy nhiên, trường hợp áp dụng sẽ có khác nhau đôi chút:

List partition phân chia table dựa trên danh sách các giá trị cho trước, không theo khoảng giá trị như range partition. Do đó, nó phù hợp phân chia dữ liệu theo những giá trị cụ thể, giống bài toán phân chia nam, nữ ở phần trước.

5. Partition by hash

Lại là **hash**, nhưng lần này là **hash partition**. Idea không có gì phức tạp, các bước thực hiện như sau:

- Thực hiện hash partition key ra hash value.
- Modulus để tìm partition cho record. Ví dụ record có partition key hash value = 5, tổng số lượng partition là 3 (0, 1, 2), lấy $5 \% 3 = 2$. Vậy record đó nằm ở partition thứ ba.

```

DROP TABLE ENGINEER CASCADE;

CREATE TABLE ENGINEER
(
    id bigserial NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL
) PARTITION BY HASH(country_id);

CREATE TABLE ENGINEER_P1 PARTITION OF ENGINEER
    FOR VALUES WITH (MODULUS 3, REMAINDER 0);

CREATE TABLE ENGINEER_P2 PARTITION OF ENGINEER
    FOR VALUES WITH (MODULUS 3, REMAINDER 1);

CREATE TABLE ENGINEER_P3 PARTITION OF ENGINEER
    FOR VALUES WITH (MODULUS 3, REMAINDER 2);

```

Như vậy với **hash partition**, không dễ để đoán được một record nằm ở partition nào. Cho dù bạn có biết thuật toán hash, cũng cần thêm một vài step để biết được đáp án. Vậy nên, **hash partition** phù hợp với:


- Các dữ liệu không nhất thiết phải thuộc cùng một group để nhóm vào một partition.
- Không cần thực hiện các lệnh đặc biệt với một nhóm data nào đó, ví dụ như drop.
- Với **hash partition**, chỉ cần đạt được mục đích cố gắng chia thành nhiều partition cân bằng nhau là được. Như vậy đủ để tăng performance cho query.

Insert lại data và query thử nhé:

```

EXPLAIN SELECT * FROM ENGINEER WHERE COUNTRY_ID = 1

```

Data Output	Explain	Messages	Notifications
	QUERY PLAN text 		
1	Seq Scan on engineer_p3 engineer (cost=0.00..859.61 rows=423 width=56)		
2	Filter: (country_id = 1)		

Query plan sẽ tự biết record nằm ở **partition table** nào và thực hiện sequence scan trên table đó luôn. Nếu **query plan** và **PostgreSQL** thông minh đến vậy thì tại sao không partition toàn bộ các table để tăng query performance? Có bí ẩn gì đằng sau mà ta chưa biết không?

Trước khi đi tìm câu trả lời, tổng kết lại những lợi ích mà **partition** đem lại nhé:

- Biến một logical table thành nhiều physical table, giảm không gian tìm kiếm, tăng performance cho query trong trường hợp tận dụng được điều kiện WHERE với **partition key**.
- Xóa bỏ các data cũ một cách dễ dàng, nhanh chóng so với cách truyền thống. Ví dụ cần xóa tất cả các record có giới tính nam, nếu chia partition theo giới tính từ đầu. Chỉ cần TRUNCATE/DROP **partition** đó là ok. Không cần seq scan để DELETE record với WHERE condition.


Ngoài ra, nó cũng có một vài hạn chế cơ bản cần quan tâm:

- Unique constraint, PK constraint không thể thực hiện trên table chính mà phải thực hiện ở **partition table**.
- Các **partition table** không thể có column nào khác mà không khai báo ở parent table. Nói cách khác, nó kế thừa toàn bộ column và data type của parent table.
- Một vài vấn đề liên quan đến trigger. Ví dụ BEFORE ROW trigger ON INSERT. Nó thực hiện trigger function/procedure... trước khi row được insert vào table.


6. Partition pruning

Quay lại phần trước, sử dụng data đã tạo với **hash partition**, explain 2 query sau đều ra **computation cost** như nhau:

```
EXPLAIN SELECT * FROM ENGINEER WHERE country_id = 1;
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN text 		
1	Seq Scan on engineer_p3 engineer (cost=0.00..859.61 rows=424 width=56)		
2	Filter: (country_id = 1)		



```
EXPLAIN SELECT * FROM ENGINEER_P3 WHERE country_id = 1;
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN text 		
1	Seq Scan on engineer_p3 (cost=0.00..859.61 rows=424 width=56)		
2	Filter: (country_id = 1)		

PostgreSQL query plan tự động chuyển sang partition **ENGINEER_3** để thực hiện scan vì nó biết record có country_id = 1 nằm ở đó. Mấu chốt chính là **partition pruning**, nó là thứ bí ẩn đằng sau giúp tối ưu plan nhằm tăng performance cho các query có điều kiện dựa trên partition table cụ thể. Mặc định, **partition pruning** ở chế độ ON, bây giờ OFF nó đi và thực hiện lại query nhé:

```
SET enable_partition_pruning = off;

EXPLAIN SELECT * FROM ENGINEER WHERE country_id = 1;
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN  text 		
1	Append (cost=0.00..2382.13 rows=426 width=56)		
2	-> Seq Scan on engineer_p1 engineer_1 (cost=0.00..757.88 rows=1 width=56)		
3	Filter: (country_id = 1)		
4	-> Seq Scan on engineer_p2 engineer_2 (cost=0.00..762.51 rows=1 width=56)		
5	Filter: (country_id = 1)		
6	-> Seq Scan on engineer_p3 engineer_3 (cost=0.00..859.61 rows=424 width=56)		
7	Filter: (country_id = 1)		

Vỡ alo, giờ thì seq scan từng partition để tìm ra record phù hợp

Partition pruning chỉ phát huy sức mạnh khi điều kiện WHERE phù hợp với **partition key**.

7. Multi-level partition

Mình có nhận được câu hỏi khá hay ở bài trước:

Tại một thời điểm nào đó, toàn bộ 95% query chỉ rơi vào một partition table thì sao, chúng ta xử lý thế nào?

Trước khi trả lời câu hỏi cần phân tích lại về mục đích của **partitioning**.

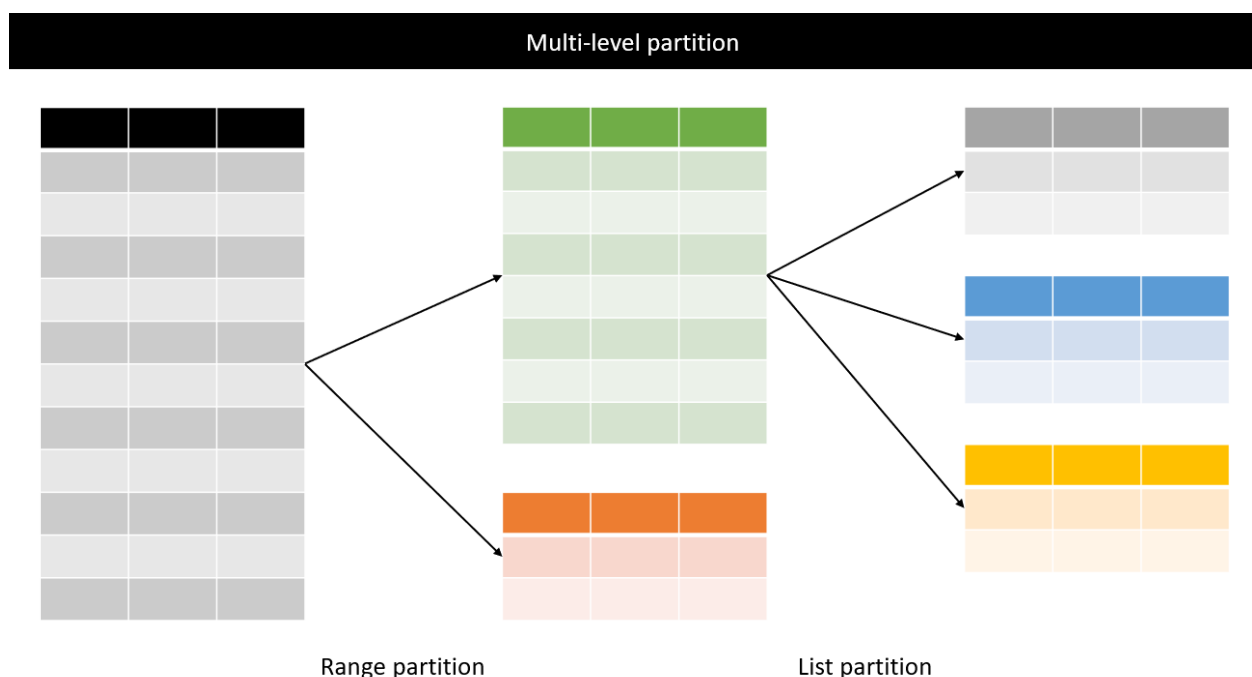
Phân chia table lớn thành nhiều partition table, từ đó giảm không gian tìm kiếm để tăng query performance. Time complexity giảm từ $O(n)$ xuống $O(k)$ với $k < n$. Đó cũng là lý do vì sao với các table nhỏ, partition không phát huy được nhiều sức mạnh.

Ví dụ trong 10s có 1000 queries đều rơi vào một partition, quá tốt rồi, nếu **query plan** xác định được partition ngay từ đầu và thực hiện **partition pruning**, chẳng phải mục đích ban đầu đã đạt được rồi hay sao. Nên việc 95% query vào một hay nhiều partition không phải

là một vấn đề ta cần quan tâm. Thậm chí nếu không partition ở đây, không gian tìm kiếm tăng lên và làm giảm query performance.

Vấn đề sẽ thực sự xảy ra khi **query plan/query execution** không thể **partition pruning** dẫn đến việc phải scan toàn bộ các partition. Việc scan có thể được thực hiện seq hoặc parallel giữa các partition tuy nhiên nó phụ thuộc vào điều kiện tìm kiếm. Tuy nhiên, **computation cost** lúc này sẽ gần bằng việc không partition thậm chí tốn hơn vì có rất nhiều behind the scenes mà ta không nắm được hết. Lúc này tội lỗi không nằm ở **partition** nữa mà là cách design của chúng ta có vấn đề. Đấy cũng là lý do vì sao không phải table nào cũng đi **partition** để hưởng chút lợi ích từ việc đó. Giống như việc mang dao mổ trâu đi giết gà vậy, trói gà không chặt, chém mạnh quá trượt vào tay chân thì còn chết nữa.

Ta thấy được lợi ích chính của **partition** là thu hẹp không gian tìm kiếm, vậy nếu có thể tiếp tục thu hẹp **partition** của 95% query kia chẳng phải tốt hơn sao? Quầy thôi, lúc này ta cần **multi-level partition**. Nghe có vẻ nguy hiểm nhưng lại chẳng có gì, nhìn vào hình minh họa sau cũng dễ dàng hiểu được.



Quanh đi quẩn lại vẫn dựa trên Tree data structure. Nếu phân tích được và bắt thóp 95% query đó dựa trên condition gì, ta hoàn toàn có thể partition tiếp **partition table** thành những partition nhỏ hơn để phục vụ bài toán. Tuy nhiên, nó cũng có những drawback mình đã nói ở trên. Hãy cân nhắc thật kĩ trước khi quyết định.

```

DROP TABLE ENGINEER;
CREATE TABLE ENGINEER
(
    id bigserial NOT NULL,
    first_name character varying(255) NOT NULL,
    last_name character varying(255) NOT NULL,
    gender smallint NOT NULL,
    country_id bigint NOT NULL,
    title character varying(255) NOT NULL,
    started_date date,
    created timestamp without time zone NOT NULL
) PARTITION BY RANGE(started_date);

CREATE TABLE ENGINEER_Q1_2020 PARTITION OF ENGINEER FOR VALUES
    FROM ('2020-01-01') TO ('2020-04-01') PARTITION BY LIST(title);



CREATE TABLE ENGINEER_DF PARTITION OF ENGINEER DEFAULT;

CREATE TABLE ENGINEER_Q1_2020_SE PARTITION OF ENGINEER_Q1_2020
    FOR VALUES IN ('Backend Engineer', 'Frontend Engineer', 'Fullstack Engineer');
CREATE TABLE ENGINEER_Q1_2020_BA PARTITION OF ENGINEER_Q1_2020
    FOR VALUES IN ('BA', 'QA');
CREATE TABLE ENGINEER_Q1_2020_DF PARTITION OF ENGINEER_Q1_2020 DEFAULT;

```

Insert data và query xem thế nào:


```
EXPLAIN SELECT * FROM ENGINEER WHERE started_date = '2020-02-02';
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN  text 		
1	Append (cost=0.00..597.33 rows=286 width=56)		
2	-> Seq Scan on engineer_q1_2020_ba engineer_1 (cost=0.00..138.76 rows=80 width=46)		
3	Filter: (started_date = '2020-02-02'::date)		
4	-> Seq Scan on engineer_q1_2020_se engineer_2 (cost=0.00..228.75 rows=103 width=60)		
5	Filter: (started_date = '2020-02-02'::date)		
6	-> Seq Scan on engineer_q1_2020_df engineer_3 (cost=0.00..228.39 rows=103 width=60)		
7	Filter: (started_date = '2020-02-02'::date)		

Khá dễ hiểu, started_date nằm trong partition **ENGINEER_Q1** nên query plan thực hiện seq scan trên toàn bộ partition con của nó.

Thay đổi query, thêm điều kiện tìm kiếm **title**:

```
EXPLAIN SELECT * FROM ENGINEER WHERE started_date = '2020-02-02' AND title = 'BA';
```

Data Output	Explain	Messages	Notifications
	QUERY PLAN text		
1	Seq Scan on engineer_q1_2020_ba engineer (cost=0.00..154.31 rows=41 width=46)		
2	Filter: ((started_date = '2020-02-02'::date) AND ((title)::text = 'BA'::text))		

Ngon rồi, nhờ **partition pruning** nên query plan chỉ thực hiện scan trên partition **ENGINEER_Q1_BA**.

Bạn có thể thêm **ANALYZE** để xem thêm thông số về **query execution** nhé. Với các table ít record, độ chênh lệch về **execution time** không đáng kể. Tuy nhiên với table lớn cỡ vài hoặc vài chục GB, tác dụng là rõ ràng luôn (với query có where condition phù hợp).