DataRockie x KBTG SQL Day 2

สวัสดีนักเรียนทุกคนนนน สำหรับวันที่สอง เราจะเรียน (ไม่ basics) แล้วนะ 555+

เนื้อหาวันนี้จะเป็น intermediate + advanced SQL เทคนิค เช่น การเขียน `join` `window functions` และการเขียน `subqueries` (หรือการเขียน select ซ้อน select)

Review Day One

รีวิวเนื้อหาวันแรก เราเรียน SQL สำหรับ single table manipulation โดย clauses หลักๆที่เราเขียนเป็นประจำคือ `select` `where` `group by` `having` `order by` `limit`

- ลำดับการเขียน clauses สำคัญมาก `where` ต้องมาก่อน `group by` และ `having` ต้องมาหลัง `group by`
- คอลัมน์ที่อยู่ใน `group by` ต้องอยู่ใน `select` clause เสมอ
- เราสามารถใช้ `select` เพื่อสร้างคอลัมน์ใหม่ได้ด้วย และใช้ `as` เพื่อตั้งชื่อคอลัมน์ใหม่

```
SELECT
  country,
  COUNT(*) AS n
FROM customers
WHERE country <> 'Brazil'
GROUP BY country
HAVING n >= 5
ORDER BY n DESC
LIMIT 2;
```

Day 2 (Intermediate SQL)

วันนี้เราจะเรียนกันทั้งหมด 5 หัวข้อหลักๆคือ

- `case when`
- `create table`
- `join` ดึงข้อมูลจากหลายตารางพร้อมกัน อธิบายความแตกต่างของ `left join` vs. `inner join`
- subqueries
- window functions ตัวพื้นฐาน เช่น `row_number()`

1. CASE

เราใช้ `case` เพื่อเขียนเงื่อนไข (เหมือนการเขียน `if` ในโปรแกรม Excel)

โดย syntax ของ `case` จะเขียนแบบนี้

- ขึ้นต้นด้วย `case` ปิดท้ายด้วย `end`
- `when` ตามด้วยเงื่อนไข `then` คือผลลัพธ์หรือ value ถ้าเงื่อนไขนั้นเป็นจริง
- `else` คือเงื่อนไขอื่นๆ
- เราตั้งชื่อคอลัมน์ `case` ด้วย `as` เหมือนเดิม

```
-- we see NULL in company column
SELECT company FROM customers;

-- create new column called "segment"
-- NULL will be "End Customers"
-- other values will be "Corporate"
SELECT
  company,
  CASE WHEN company IS NULL THEN 'End Customers'
       ELSE 'Corporate'
  END AS segment
FROM customers;
```

2. CREATE TABLE

SQL มีวิธีการสร้างตารางหลักๆสองแบบคือ

1. `create table` แบบ manual

2. `create table` จาก `select` query ที่เราเขียนขึ้นมา

สร้างตารางใหม่ด้วย `create table` และใส่ข้อมูลด้วย `insert into`

```
-- drop existing table
DROP TABLE student;

-- create new table
CREATE TABLE student (
    id INT PRIMARY KEY,
    name TEXT NOT NULL,
    major TEXT NOT NULL,
    gpa REAL
);

INSERT INTO student VALUES
  (1, 'Toy', 'Economics', 3.25),
  (2, 'Joe', 'Marketing', 3.48),
  (3, 'Max', 'Marketing', 3.89);

-- select data from new table
SELECT * FROM student;
```

ถ้าต้องการลบตารางทิ้งให้พิมพ์ `drop` ตามด้วยชื่อ table ได้เลย

อันนี้ code สำหรับ tutorial นะครับ นักเรียนสามารถ copy แปะทำตามวีดีโอได้เลย เราจะลองสร้างอีกสองตาราง new_student และ address ตามลำดับ

```
CREATE TABLE new_student (
    id INT PRIMARY KEY,
    name TEXT NOT NULL,
    major TEXT NOT NULL,
    gpa REAL
);

INSERT INTO new_student VALUES
  (4, 'Jay', 'Data Science', NULL),
  (5, 'Mary', 'Data Science', NULL),
  (6, 'Anna', 'Data Science', NULL);

SELECT * FROM new_student;
```

เสร็จแล้วจะรวม 2 ตารางนี้เข้าด้วยกัน ใน SQL เราใช้ `union` เพื่อทำการ append tables

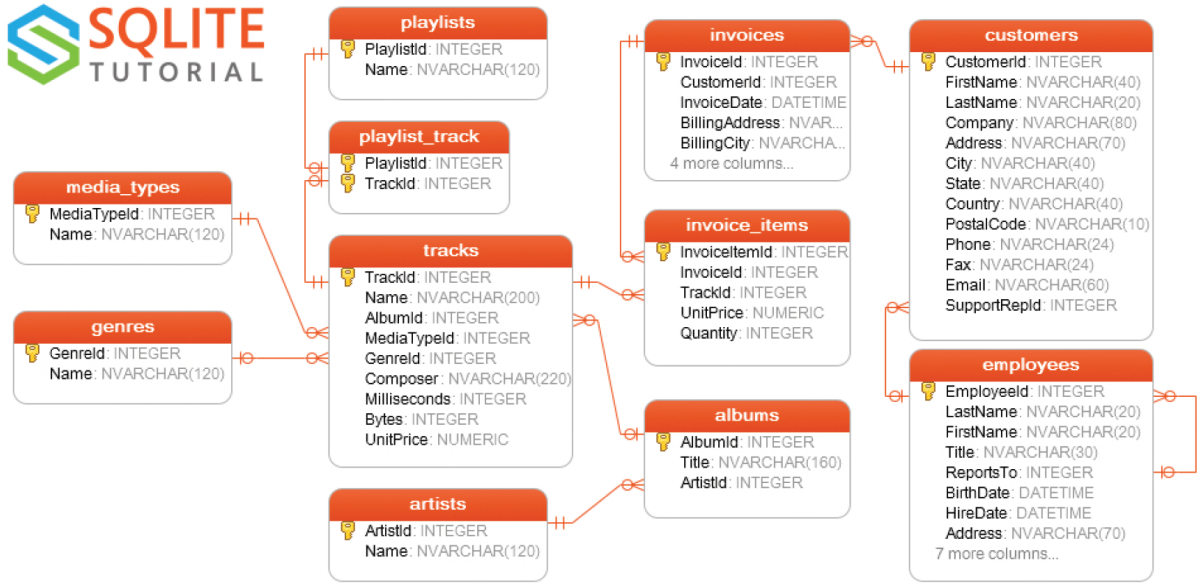
```
-- append two tables together
CREATE TABLE all_student AS
  SELECT * FROM student
  UNION
  SELECT * FROM new_student;
```

อัพเดท value ในตารางด้วย `update` และ `set` value ตามเงื่อนไข (record) ที่เราต้องการ

```
-- update Jay's gpa score
UPDATE all_student
SET gpa = 3.55
WHERE name = 'Jay';

SELECT * FROM all_student;
```

3. JOINs



รูปแบบการ `join` ที่ใช้เยอะที่สุดใน SQL คือ `inner` และ `left join` โดย syntax การเขียน `join` ก็ไม่ยากเลย เราใช้ primary key และ foreign key ในการ merge tables เข้าด้วยกัน

Tip - ER Diagram สำคัญมาก ถ้ามี ให้ปรินท์ออกมาวางข้างๆกันได้เลย ERD เป็นเหมือนแผนที่ช่วยให้เราเข้าใจ database ได้ง่ายขึ้น ความท้าทายในชีวิตจริงคือไม่ค่อยมีบริษัททำ ERD แจกพนักงานเท่าไหร่ 555+ ต้องเรียนรู้เอง on the job training ไปเรื่อยๆ

```
-- basic syntax
SELECT *
FROM tableA
JOIN tableB ON tableA.key = tableB.key;
```

- inner join จะ return เฉพาะ rows ที่ key ตรงกันเท่านั้น (matched)

- left join เราใช้ตารางซ้ายเป็นตัวตั้ง และดึงคอลัมน์จากตารางขวามาถ้า key ตรงกัน (matched) คล้ายๆกับการเขียน `vlookup()` ในโปรแกรม Excel

Join Example #1

ดึงข้อมูลทุกคอลัมน์จากตาราง genres และ tracks

```
-- join table genres and tracks
SELECT * FROM genres
JOIN tracks ON genres.genreid = tracks.genreid;
```

Tip - เราใช้ `as` เพื่อตั้งชื่อย่อคอลัมน์ได้ หลังจาก `join` tables เสร็จแล้ว เราสามารถเขียนทุก clauses ที่เราเรียนมาในวันแรกได้หมดเลย ตั้งแต่ `where` `group by` `having` `order by` `limit`

```
-- join table genres and tracks
SELECT * FROM genres AS g
JOIN tracks AS t ON g.genreid = t.genreid
WHERE g.name = 'Rock';
```

Join Example #2

นับจำนวนเพลงทั้งหมดที่อยู่ในแต่ละ genre

```
-- join table genres and tracks
SELECT g.name, COUNT(*) AS n_tracks
FROM genres AS g
JOIN tracks AS t ON g.genreid = t.genreid
GROUP BY g.name
ORDER BY n_tracks DESC;
```

Join Example #3

ลูกค้า ID ที่ 35 ชอบฟังเพลงประเภทไหนบ้าง (genre)?

Tip - เราใช้ `distinct` เพื่อดึง unique values ออกมา

```
SELECT
  DISTINCT firstname, lastname, g.name
FROM customers c
JOIN invoices inv  ON c.customerid = inv.customerid
JOIN invoice_items invi ON inv.invoiceid = invi.invoiceid
JOIN tracks t ON invi.trackid = t.trackid
JOIN genres g ON t.genreid = g.genreid
WHERE c.customerid = 35;
```

Join Example #4

ลูกค้าในประเทศอเมริกา แต่ละคนมี total invoice เท่าไหร่กันบ้าง (i.e. sum total)

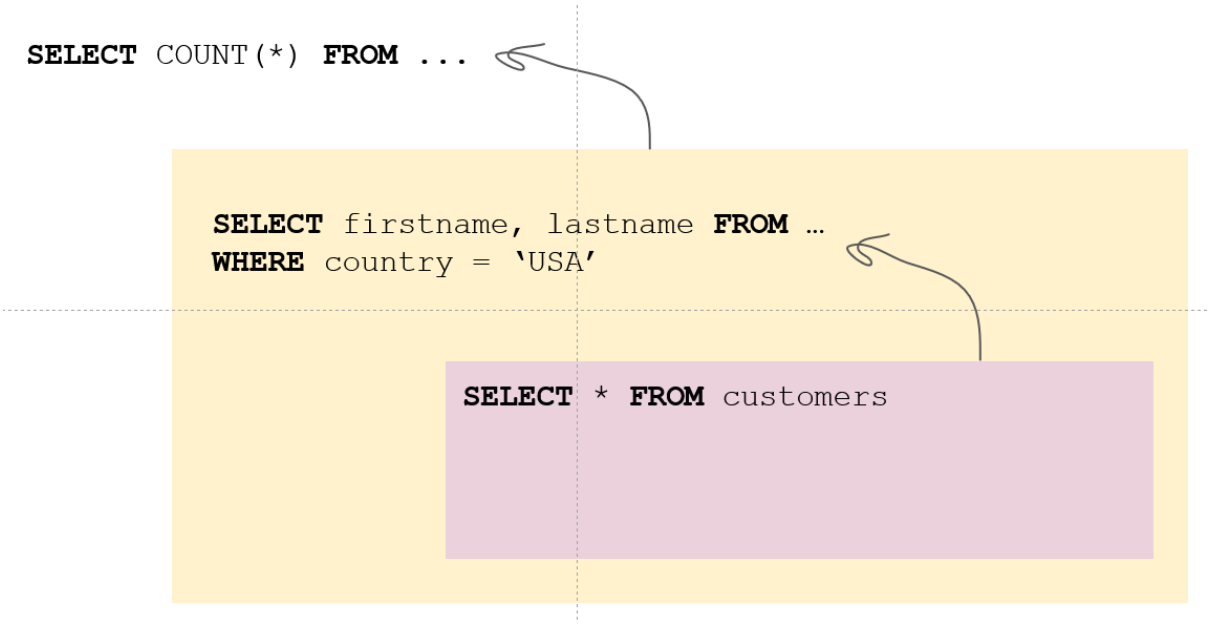
Tip - เราใช้ `column index` ใน `group by` และ `order by` แทนชื่อคอลัมน์จริงก็ได้

```
SELECT
   customers.customerid,
   firstname ||  ' ' || lastname AS fullname,
   SUM(total) AS total_inv
FROM customers
JOIN invoices ON customers.customerid = invoices.customerid
WHERE country = 'USA'
GROUP BY 1,2
ORDER BY 3 DESC;
```

4. Subqueries

subqueries คือการเขียน `select` ซ้อน `select` ช่วยเพิ่มลูกเล่นในการเขียน query ให้มีความซับซ้อน และสร้างสรรค์มากยิ่งขึ้น

วิธีการมองหา subqueries ง่ายๆ ให้มองหา `select` ที่อยู่ในวงเล็บ (inner query)



ตัวอย่างง่ายๆของ subqueries เช่น

```
SELECT * FROM tracks
WHERE bytes = (SELECT MAX(bytes) FROM tracks);
```

```
SELECT * FROM tracks
WHERE bytes < (SELECT AVG(bytes) FROM tracks);
```

ตำแหน่งของ inner query สามารถเกิดขึ้นได้หลายที่ ที่เราเห็นบ่อยๆตามลำดับ เช่น

- `where`
- `from`
- `join`
- `select`

เราสามารถเขียน subquery เพื่อลดขนาด table ก่อนไปทำ operation อื่นๆต่อก็ได้

```
SELECT * FROM (
    SELECT firstname, lastname, email
    FROM customers WHERE company IS NULL
)
WHERE firstname LIKE 'A%';
```

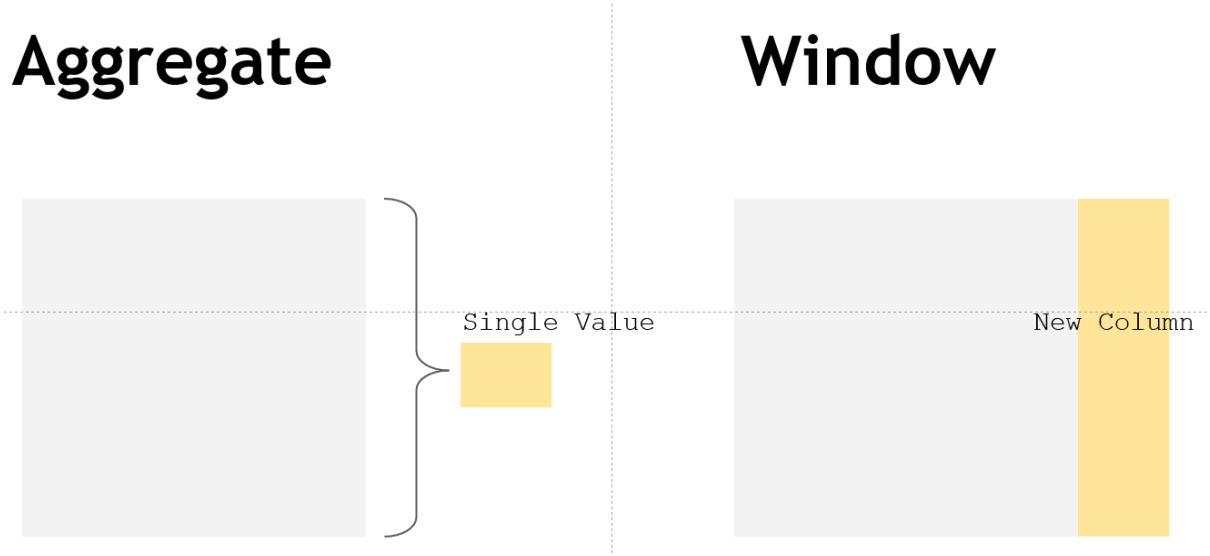
มาลองดูตัวอย่าง subqueries ของ `join` กันบ้าง

```
-- subquery
SELECT
  firstname || ' ' || lastname AS fullname,
  SUM(total) AS total_invoice,
  COUNT(total) AS n_invoice
FROM customers AS t1
JOIN (
      SELECT * FROM invoices
      WHERE STRFTIME('%Y', invoicedate) = '2010'
     ) AS t2
ON t1.customerid = t2.customerid
WHERE country = 'USA'
GROUP BY 1
ORDER BY 2 DESC;
```

5. Window Functions

Window functions คือ analytical functions ของ SQL โดยเราใช้ `window functions` เพื่อสร้างคอลัมน์ใหม่ สามารถจับกลุ่มหรือเรียงข้อมูลตามที่เราต้องการได้

ด้านล่างคือ diagram ความแตกต่างของ aggregate vs. window functions



Tip - window functions จะตามหลังด้วย `over()` เสมอ ภายใน `over()` เราสามารถเขียน `partition by` และ `order by` ได้ด้วย

Window Example #1 (Naive!)

สร้างคอลัมน์ตัวเลข 1:n ในตาราง result set ที่เราดึงขึ้นมา

```
SELECT
    firstname,
    lastname,
    ROW_NUMBER() OVER() AS rowNum
FROM customers;
```

สร้างคอลัมน์ตัวเลขจับกลุ่มตามประเทศ

```
SELECT
    firstname,
    lastname,
    country,
    ROW_NUMBER() OVER( PARTITION BY country ) AS rowNum
FROM customers;
```

สร้างคอลัมน์ตัวเลขจับกลุ่มตามประเทศ เรียงตามนามสกุล

```
SELECT
    firstname,
    lastname,
    country,
    ROW_NUMBER() OVER( PARTITION BY country ORDER BY lastname ) AS rowNum
FROM customers;
```

Window Example #2

คำนวณคอลัมน์ running total (หรือ cumulative sum ง่ายๆ)

วิธีการเขียน window functions ง่ายที่สุดคือใช้ `aggregate functions` ตามด้วย `over()`

```
-- create new table totalRevenue
CREATE TABLE totalRevenue AS
    SELECT
      STRFTIME('%Y', invoicedate) AS year,
      STRFTIME('%m', invoicedate) AS month,
      SUM(total) AS total_revenue
    FROM invoices
    GROUP BY year,month;

-- calculate running total
SELECT
    year,
  month,
  total_revenue,
  SUM(total_revenue) OVER(PARTITION BY year ORDER BY month) AS runningTotal
FROM totalRevenue;
```

Window Example #3

ฟังก์ชัน `rank()` เพื่อทำ ranking ง่ายๆ

```
SELECT * FROM (
SELECT
  g.name,
  t.name,
  ROUND(t.bytes/ (1024*1024.0), 2) AS MB,
  RANK() OVER(PARTITION BY g.name ORDER BY t.bytes DESC) AS ranking
FROM genres g
JOIN tracks t ON g.genreid = t.genreid
)
WHERE ranking = 1
ORDER BY MB DESC;
```

Window Example #4

ฟังก์ชัน `ntile()` เพื่อแบ่ง segment ข้อมูลเป็นกลุ่มเท่าๆกัน

```
SELECT name, segment, COUNT(*) AS n FROM (
SELECT
  genres.name,
  tracks.name,
  tracks.bytes,
```

```
  NTILE(4) OVER(PARTITION BY genres.name ORDER BY tracks.bytes) AS segment
FROM genres
JOIN tracks ON genres.GenreId = tracks.GenreId )
GROUP BY 1 , 2;
```