## SQL Query Processing Order

SQL processes queries in a specific order, not from top to bottom as written. The order is:

1. FROM
2. JOIN
3. WHERE
4. GROUP BY
5. HAVING
6. SELECT
7. DISTINCT
8. ORDER BY
9. LIMIT/OFFSET

### Important Notes:

* GROUP BY prevents duplicates. However, when you specify multiple fields, it prevents duplicates only for the second field and not the first.
* Since GROUP BY executes before SELECT, an alias defined in SELECT cannot be used in GROUP BY.
* Fields in SELECT without aggregate functions (like MAX, MIN, AVG) must be listed in GROUP BY.

-- Higher Than 75 Marks

SELECT name

FROM students

WHERE marks > 75

ORDER BY SUBSTR(name, -3), id ASC;

-- Employee Names

SELECT name

FROM employee

ORDER BY name ASC;

-- Employee Salaries

SELECT name

FROM employee

WHERE salary > 2000 AND months < 10

ORDER BY employee\_id ASC;

-- Type of Triangle

SELECT A,

B,

C,

CASE

WHEN A = B AND B = C THEN 'Equilateral'

WHEN (A = B OR A = C OR B = C) AND (A + B > C AND A + C > B AND B + C > A) THEN 'Isosceles'

WHEN A <> B AND A <> C AND B <> C AND (A + B > C AND A + C > B AND B + C > A) THEN 'Scalene'

ELSE 'Not A Triangle'

END AS TriangleType

FROM TRIANGLES;

<> = Not equal to

-- The PADS

SELECT NAME || '(' || SUBSTR(OCCUPATION, 1, 1) || ')'

FROM OCCUPATIONS

ORDER BY NAME ASC;

SELECT 'There are a total of ' || COUNT(\*) || ' ' || LOWER(OCCUPATION) || 's.' AS OccupationCount

FROM OCCUPATIONS

GROUP BY OCCUPATION

ORDER BY COUNT(\*) ASC, OCCUPATION ASC;

-- Binary Tree Nodes\*

1)

SELECT b.N,

(CASE WHEN b.P IS NULL

THEN 'Root'

WHEN (SELECT COUNT(\*) FROM BST b2 WHERE b2.P = b.N) > 0

THEN 'Inner'

ELSE 'Leaf'

END)

FROM bst b

ORDER BY N;

2)

,

0

-- New Companies

1)

SELECT c.company\_code,

c.founder,

COUNT(DISTINCT lm.lead\_manager\_code) AS distinct\_lead\_managers,

COUNT(DISTINCT sm.senior\_manager\_code) AS distinct\_senior\_managers,

COUNT(DISTINCT m.manager\_code) AS distinct\_managers,

COUNT(DISTINCT e.employee\_code) AS distinct\_employees

FROM company c

INNER JOIN lead\_manager lm ON c.company\_code = lm.company\_code

INNER JOIN senior\_manager sm ON lm.lead\_manager\_code = sm.lead\_manager\_code

INNER JOIN manager m ON sm.senior\_manager\_code = m.senior\_manager\_code

INNER JOIN employee e ON e.manager\_code = m.manager\_code

GROUP BY c.company\_code, c.founder;

2)

SELECT COMPANY\_CODE, FOUNDER,

(SELECT COUNT(DISTINCT LEAD\_MANAGER\_CODE) FROM LEAD\_MANAGER WHERE COMPANY\_CODE = C.COMPANY\_CODE),

(SELECT COUNT(DISTINCT SENIOR\_MANAGER\_CODE) FROM SENIOR\_MANAGER WHERE COMPANY\_CODE = C.COMPANY\_CODE),

(SELECT COUNT(DISTINCT MANAGER\_CODE) FROM MANAGER WHERE COMPANY\_CODE = C.COMPANY\_CODE),

(SELECT COUNT(DISTINCT EMPLOYEE\_CODE) FROM EMPLOYEE WHERE COMPANY\_CODE = C.COMPANY\_CODE)

FROM COMPANY C

ORDER BY COMPANY\_CODE;

-- Revising Aggregations - The Count Function

SELECT COUNT(name) FROM city WHERE population > 100000;

-- Revising Aggregations - The Sum Function

SELECT SUM(population) FROM city WHERE district = 'California';

-- Revising Aggregations – Averages

SELECT AVG(population) FROM city WHERE district = 'California';

-- Average Population

SELECT ROUND(AVG(population)) FROM city;

-- Japan Population

SELECT SUM(population) FROM city WHERE COUNTRYCODE = 'JPN';

-- Population Density Difference

SELECT MAX(population) - MIN(population) FROM city;

--The Blunder

SELECT CEIL(AVG(Salary)-AVG(REPLACE(Salary,'0','')))

FROM EMPLOYEES;

--Top Earners \*

SELECT MAX(SALARY\*MONTHS), COUNT(\*)

FROM EMPLOYEE

WHERE (SALARY\*MONTHS) = (SELECT MAX(SALARY\*MONTHS)

FROM EMPLOYEE);

**Açıklama:**

1. **Alt Sorgu (Subquery)**:

sql

Copy code

SELECT MAX(SALARY\*MONTHS)

FROM EMPLOYEE;

* + Bu alt sorgu, EMPLOYEE tablosunda SALARY ile MONTHS çarpımının en yüksek değerini bulur. Yani, tüm çalışanların maaş ve ay sayısını çarptığınızda elde edilen en büyük sonucu döner.

1. **Ana Sorgu**:

SELECT MAX(SALARY\*MONTHS), COUNT(\*)

FROM EMPLOYEE

WHERE (SALARY\*MONTHS) = (alt sorgudan dönen değer);

* + WHERE koşulu, alt sorgudan dönen en yüksek SALARY\*MONTHS değerini kullanarak, bu değeri eşleşen satırları filtreler.
  + SELECT MAX(SALARY\*MONTHS), COUNT(\*) kısmı, filtrelenen satırlar arasında en yüksek SALARY\*MONTHS değerini (bu durumda eşleşen değer) ve bu değere sahip olan satırların sayısını döner.

Whereden dönen koşul önce yapılır ve select ona göre filtrelenir.

--Weather Observation Station 2

SELECT

ROUND(SUM(LAT\_N), 2),

ROUND(SUM(LONG\_W), 2)

FROM

STATION;

Round 2 haneli kullanılabiliyor.

OUTPUT: 42850.04 47381.48

--Weather Observation Station 13

select round(sum(lat\_n),4) from station where lat\_n>38.7880 and lat\_n<137.2345;

--Weather Observation Station 14

select round(max(lat\_n),4) from station where lat\_n < 137.2345;

--Weather Observation Station 15

SELECT ROUND(long\_w, 4)

FROM station

WHERE lat\_n = (

SELECT MAX(lat\_n)

FROM station

WHERE lat\_n < 137.2345

);

--Weather Observation Station 16

select round(min(lat\_n),4)

from station

where lat\_n>38.7780;

--Weather Observation Station 17

select round (long\_w,4)

from STATION

where lat\_n = (select min(lat\_n) from station where lat\_n>38.7780);

--Weather Observation Station 18

SELECT

ROUND(ABS(MAX(LAT\_N) - MIN(LAT\_N))

+ ABS(MAX(LONG\_W) - MIN(LONG\_W)), 4)

FROM

STATION;

--Weather Observation Station 19

SELECT

ROUND(

SQRT(

POWER((MAX(lat\_n) - MIN(lat\_n)), 2)

+

POWER((MAX(long\_w) - MIN(long\_w)), 2)

), 4

) AS result

FROM station;

Weather Observation Station 20

SELECT CAST(LAT\_N AS DECIMAL (7,4))

FROM

(SELECT LAT\_N, ROW\_NUMBER() OVER (ORDER BY LAT\_N) as ROWNU

FROM STATION

)

WHERE ROWNU = ( SELECT ROUND((COUNT(LAT\_N)+1)/2,0)

FROM STATION

);

Açıklama:

Bu SQL sorgusu, bir `STATION` tablosundaki `LAT\_N` (enlem) değerlerinin medyanını hesaplar. Bu işlem adım adım şu şekilde gerçekleşir:

### Adım Adım Sorgu İşleyişi

#### 1. İç İçe Sorgu (Subquery) Çalışması

```sql

SELECT LAT\_N, ROW\_NUMBER() OVER (ORDER BY LAT\_N) as ROWNU

FROM STATION

```

Bu alt sorgu önce çalışır ve şu adımları gerçekleştirir:

- `STATION` tablosundan `LAT\_N` sütunundaki tüm değerleri seçer.

- `ROW\_NUMBER() OVER (ORDER BY LAT\_N)` ifadesi, `LAT\_N` değerlerine göre her satıra bir sıra numarası atar. Bu sıra numarası `ROWNU` adında yeni bir sütunda yer alır. `LAT\_N` değerleri küçükten büyüğe sıralanır.

#### 2. Medyan Pozisyonunu Hesaplama

```sql

SELECT ROUND((COUNT(LAT\_N)+1)/2,0)

FROM STATION

```

Bu alt sorgu çalışır ve şu adımları gerçekleştirir:

- `COUNT(LAT\_N)` ifadesi, `STATION` tablosundaki `LAT\_N` sütunundaki toplam satır sayısını döndürür.

- `(COUNT(LAT\_N)+1)/2` ifadesi, satır sayısının yarısına (ve bir ekleyerek) eşit olan pozisyonu hesaplar.

- `ROUND(...,0)` ifadesi, bu pozisyonu tam sayıya yuvarlar. Bu pozisyon medyan değerin bulunduğu satırın sıra numarasıdır.

#### 3. Ana Sorgunun Çalışması

```sql

SELECT CAST(LAT\_N AS DECIMAL (7,4))

FROM

(SELECT LAT\_N, ROW\_NUMBER() OVER (ORDER BY LAT\_N) as ROWNU

FROM STATION

) AS subquery

WHERE ROWNU = ( SELECT ROUND((COUNT(LAT\_N)+1)/2,0)

FROM STATION

);

```

Bu sorgu şu adımları gerçekleştirir:

1. \*\*İç İçe Sorgunun Sonucu (Subquery)\*\*:

- Önceden çalıştırılmış alt sorgunun sonucu, `subquery` adında geçici bir tablo olarak kullanılır. Bu tablo `LAT\_N` ve `ROWNU` sütunlarını içerir.

2. \*\*Medyan Pozisyonunun Hesaplanması\*\*:

- `WHERE ROWNU = ...` ifadesi, medyan pozisyonunu belirleyen alt sorgunun sonucunu kullanarak `ROWNU` değeri bu pozisyona eşit olan satırı seçer.

3. \*\*Sonuç Sütunu\*\*:

- Seçilen satırdaki `LAT\_N` değeri, `CAST(LAT\_N AS DECIMAL (7,4))` ifadesiyle belirli bir ondalık hassasiyette döndürülür.

### Özet

1. İç içe sorgu `STATION` tablosundan `LAT\_N` değerlerini alır ve her birine sıra numarası atar (`ROW\_NUMBER()`).

2. Medyan pozisyonu, satır sayısının yarısına denk gelen sıra numarası olarak hesaplanır.

3. Ana sorgu, sıra numarası medyan pozisyonuna eşit olan satırın `LAT\_N` değerini seçer ve belirli bir ondalık hassasiyette (`DECIMAL(7,4)`) döndürür.

--The Report

SELECT

CASE

WHEN g.grade < 8 THEN NULL

ELSE s.name

END,

g.grade,

s.marks

FROM students s

INNER JOIN grades g ON s.marks BETWEEN g.min\_mark AND g.max\_mark

ORDER BY g.grade DESC, s.name ASC;

-- Contest Leaderboard

SELECT h.hacker\_id, h.name, t1.total\_score

FROM (

SELECT hacker\_id, SUM(max\_score) AS total\_score

FROM (

SELECT hacker\_id, MAX(score) AS max\_score

FROM Submissions

GROUP BY hacker\_id, challenge\_id

) t

GROUP BY hacker\_id

) t1

JOIN Hackers h

ON h.hacker\_id = t1.hacker\_id

WHERE t1.total\_score <> 0

ORDER BY total\_score DESC, hacker\_id;

Açıklama:

Tabii, örneklerle açıklamak daha anlaşılır olabilir. Diyelim ki Submissions tablonuz aşağıdaki gibi veriler içeriyor:

### Submissions Tablosu

| **hacker\_id** | **challenge\_id** | **score** |
| --- | --- | --- |
| 1 | 101 | 80 |
| 1 | 102 | 90 |
| 1 | 103 | 85 |
| 2 | 101 | 75 |
| 2 | 102 | 95 |
| 2 | 103 | 70 |
| 3 | 101 | 60 |
| 3 | 102 | 70 |
| 3 | 103 | 80 |

### Adım Adım Açıklama

1. **İç Sorgu - Her Challenge İçin Maksimum Puanları Hesaplama:**

İç sorgu her hacker\_id ve challenge\_id kombinasyonu için maksimum puanı hesaplar:

sql

Copy code

SELECT hacker\_id, challenge\_id, MAX(score) AS max\_score

FROM Submissions

GROUP BY hacker\_id, challenge\_id

Bu sorgunun sonucu şöyle olur:

| **hacker\_id** | **challenge\_id** | **max\_score** |
| --- | --- | --- |
| 1 | 101 | 80 |
| 1 | 102 | 90 |
| 1 | 103 | 85 |
| 2 | 101 | 75 |
| 2 | 102 | 95 |
| 2 | 103 | 70 |
| 3 | 101 | 60 |
| 3 | 102 | 70 |
| 3 | 103 | 80 |

Burada, her hacker için her challenge'daki en yüksek puanlar gösterilmektedir.

1. **Orta Sorgu - Toplam Maksimum Puanları Hesaplama:**

Orta sorgu, iç sorgudan gelen maksimum puanları toplar ve her hacker için toplam maksimum puanı hesaplar:

sql

Copy code

SELECT hacker\_id, SUM(max\_score) AS total\_score

FROM (

-- İç sorgu burada

) t

GROUP BY hacker\_id

İç sorgunun sonucunu kullanarak toplama işlemi yapılır:

| **hacker\_id** | **total\_score** |
| --- | --- |
| 1 | 80 + 90 + 85 = 255 |
| 2 | 75 + 95 + 70 = 240 |
| 3 | 60 + 70 + 80 = 210 |

Bu orta sorgunun sonucu her hacker için toplam maksimum puanı hesaplar.

1. **Dış Sorgu - Hacker Bilgilerini Getirme ve Sonuçları Filtreleme:**

Son olarak, dış sorgu hacker bilgilerini getirir ve sıfırdan büyük toplam puanları sıralar:

sql

Copy code

SELECT h.hacker\_id, h.name, t1.total\_score

FROM (

-- Orta sorgu burada

) t1

JOIN Hackers h

ON h.hacker\_id = t1.hacker\_id

WHERE t1.total\_score <> 0

ORDER BY total\_score DESC, hacker\_id

Bu sorgunun sonucu, örneğin Hackers tablosundaki isimlerle birlikte hacker'ların toplam puanlarını ve sıralamasını gösterir. Diyelim ki Hackers tablosu şu şekilde:

| **hacker\_id** | **name** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

Sonuç şu şekilde olur:

| **hacker\_id** | **name** | **total\_score** |
| --- | --- | --- |
| 1 | Alice | 255 |
| 2 | Bob | 240 |
| 3 | Charlie | 210 |

### Özet

1. İç sorgu: Her hacker ve challenge kombinasyonu için maksimum puanı bulur.
2. Orta sorgu: Bu maksimum puanları toplar ve her hacker için toplam puanı hesaplar.
3. Dış sorgu: Hacker bilgilerini getirir ve toplam puana göre sıralar.

Bu şekilde, her hacker'ın en yüksek puanlarının toplamını alabilir ve sıralayabilirsiniz.

İç sorgu da mesela 1 hacker\_id si için 3 tane 101 challange\_id olsaydı max\_score kısmında en yüksek olan mı gelecekti

ChatGPT

Evet, doğru. İç sorguda MAX(score) fonksiyonu kullanıldığında, her hacker\_id ve challenge\_id kombinasyonu için en yüksek score değeri seçilir.

### İç Sorgunun Davranışı

Diyelim ki Submissions tablonuzda bir hacker (hacker\_id = 1) ve bir challenge (challenge\_id = 101) için birden fazla kayıt var. İç sorgu şu şekilde çalışır:

sql

Copy code

SELECT hacker\_id, challenge\_id, MAX(score) AS max\_score

FROM Submissions

GROUP BY hacker\_id, challenge\_id

### Örnek Veri

Varsayalım ki Submissions tablonuz şu şekilde:

| **hacker\_id** | **challenge\_id** | **score** |
| --- | --- | --- |
| 1 | 101 | 70 |
| 1 | 101 | 80 |
| 1 | 101 | 75 |
| 1 | 102 | 90 |

### İç Sorgunun Çalışması

İç sorgunun çıktısı şu olacaktır:

| **hacker\_id** | **challenge\_id** | **max\_score** |
| --- | --- | --- |
| 1 | 101 | 80 |
| 1 | 102 | 90 |

* **MAX(score)**: hacker\_id = 1 ve challenge\_id = 101 için en yüksek score değeri 80 olarak hesaplanır. Bu, 70, 80 ve 75 puanları arasında en yüksek olanıdır.
* **GROUP BY hacker\_id, challenge\_id**: Her hacker ve challenge kombinasyonu için gruplama yapılır, ve her kombinasyon için en yüksek puan elde edilir.

-- Placements

SELECT

s.name AS student\_name

FROM students s

JOIN friends f ON s.id = f.id

JOIN packages p ON s.id = p.id

JOIN packages fp ON f.friend\_id = fp.id

WHERE fp.salary > p.salary

AND p.salary <> fp.salary

ORDER BY fp.salary;

Symmetric Pairs

SELECT X, Y FROM FUNCTIONS F1

WHERE EXISTS(SELECT \* FROM FUNCTIONS F2 WHERE F2.Y = F1.X

AND F2.X = F1.Y AND F2.X > F1.X) AND (X != Y)

UNION

SELECT X,Y FROM FUNCTIONS F1 WHERE X = Y AND

((SELECT COUNT(\*) FROM FUNCTIONS WHERE X = F1.X AND Y = F1.X) > 1)

ORDER BY X;

Alternative Queries

--Draw The Triangle 1

SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ')

FROM DUAL

CONNECT BY ROWNUM <= 20

ORDER BY 1 DESC;

### Sorgunun Genel Yapısı

sql

Copy code

SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ')

FROM DUAL

CONNECT BY ROWNUM <= 20

ORDER BY 1 DESC;

### Sorgunun Detaylı Açıklaması

1. **SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ')**:
   * SYS\_CONNECT\_BY\_PATH: Bu işlev, hiyerarşik bir sorguda belirli bir düğüme ulaşana kadar geçen yolun (path) bir dizgesini döndürür. İkinci parametre olarak verilen dizeyi (bu durumda '\* ') kullanarak yolu birleştirir.
   * NULL: Bu parametre, genellikle kök düğüm olarak adlandırılır ve burada bir başlangıç noktası olarak NULL verilmiştir.
   * '\* ': Yıldız işaretini ve bir boşluk karakterini temsil eder. Bu, her adımda bir yıldız ve boşluk eklemek için kullanılır.
2. **FROM DUAL**:
   * DUAL: Oracle veritabanında tek satırlık, tek sütunluk sahte bir tablodur. Genellikle sorgularda veri olmadan işlemler yapmak için kullanılır.
3. **CONNECT BY ROWNUM <= 20**:
   * CONNECT BY: Bu, Oracle'da hiyerarşik sorgular oluşturmak için kullanılır. Bu durumda, bir rekürsif (tekrarlayan) sorgu oluşturulmaktadır.
   * ROWNUM <= 20: Bu koşul, sorgunun 20 kez tekrarlanmasını sağlar. ROWNUM, sonuç kümesindeki her bir satıra atanan geçici bir sıra numarasıdır.
4. **ORDER BY 1 DESC**:
   * ORDER BY 1: Bu, ilk sütuna göre sıralama yapmayı belirtir. İlk sütun SYS\_CONNECT\_BY\_PATH(NULL, '\* ') ifadesidir.
   * DESC: Bu, sıralamanın azalan düzende yapılacağını belirtir.

-- Draw The Triangle 2

SELECT SYS\_CONNECT\_BY\_PATH(NULL, '\* ')

FROM DUAL

CONNECT BY ROWNUM <= 20

ORDER BY 1 ASC;

-- Print Prime Numbers

SET SERVEROUTPUT ON;

DECLARE

COUNTER NUMBER;

K NUMBER;

RESULT CLOB;

BEGIN

RESULT := '';

FOR N IN 2..1000 LOOP

COUNTER := 0;

K := FLOOR(N/2);

FOR i IN 2..K LOOP

IF(MOD(N,i)=0) THEN

COUNTER :=1;

END IF;

END LOOP;

IF (COUNTER = 0) THEN

RESULT := RESULT||N||'&';

END IF;

END LOOP;

DBMS\_OUTPUT.PUT\_LINE(SUBSTR(RESULT,0,length(RESULT)-1));

END;

/