**FAQ reeks 3: subqueries**

**Algemene opmerkingen**

* Tijdens reeks 3 (subqueries) en reeks 4 (aggregatie, grouping- & having-keywords) moeten er typisch iets geavanceerdere concepten gebruikt worden om de oefeningen correct op te kunnen lossen. Het is daarom zeer belangrijk om goed na te denken over alle queries die je indient en om steeds volledig te proberen begrijpen waarom een bepaalde query (niet) correct is.
* De meeste oefeningen kunnen opgelost worden door gebruik te maken van een subquery in de WHERE-clausule, en door gebruik te maken van de operatoren (ALL, ANY, IN...) en concepten (scalaire subquery, gecorreleerde subquery...) die opgelijst worden in de introductieslides van reeks 3. Subqueries in de FROM-clausule, waarmee je een tijdelijke tabel definieert die aan de door de subquery opgelegde vereisten voldoet, kunnen eveneens van pas komen. Subqueries in de SELECT-clausule dienen minder frequent gebruikt te worden in deze reeks, maar zijn typisch nuttig in combinatie met aggregatie-operatoren (zie reeks 4). We merken ook dat het vaak eenvoudiger is om te redeneren over en een oefeningen op te lossen met de (NOT) IN operator of een bepaald type join in plaats van over/met de (NOT) EXISTS operator.
* Soms is het mogelijk om het gebruik van subqueries te vermijden of om queries te vereenvoudigen door (1) de resulterende rijen te sorteren (door middel van het ORDER BY-keyword) en (2) de eerste x rijen te behouden (door middel van het LIMIT-keyword). Dit is vooral relevant wanneer je de top x van een bepaald iets (bv. tracks die het meest afgespeeld zijn...) wil opvragen. Het gebruik van deze keywords zal er echter voor zorgen dat *ex aequos* niet altijd in het eindresultaat worden opgenomen. Veronderstel, bijvoorbeeld, dat er 2 tracks bestaan die een gelijk aantal keer afgespeeld zijn en dit aantal ook overeenkomt met het maximale aantal keer dat een track afgespeeld is. In dit geval zal er door middel van LIMIT 1 een resultatentabel teruggeven worden waarin er slechts 1 van beide tracks opgenomen is, wat dus niet volledig correct is. Bovendien zal er tijdens de evaluatieoefeningen steeds expliciet gevraagd worden om alle ex aequos op te nemen in het eindresultaat.
* In tegenstelling tot de eerste twee reeksen leggen we voor de meeste oefeningen uit deze reeks een aantal verschillende mogelijke oplossingsmethoden uit. Herinner je dat we reeds in het FAQ-bestand van reeks 1 vermeld hebben dat het nuttig kan zijn om te proberen om de oefeningen op verschillende manieren op te lossen en zeker voor reeks 2, 3 en 4 kan dit je SQL-inzichten en SQL-skills sterk verbeteren. Je zal ook merken dat het soms mogelijk is om bepaalde oefeningen uit deze reeks zonder subqueries op te lossen, maar we raden toch aan om voor elke oefening minstens 1 oplossing in te dienen waarin je subqueries gebruikt.

**Radio 3.1**

Mogelijke oplossingsmethoden:

1. LEFT/RIGHT JOIN.
2. NOT IN operator:
   1. Inner query: Selecteer alle tracks die gebruikt worden in een playlist.
   2. Outer query: Hou enkel de tracks over die niet voorkomen in de resultatentabel van de inner query.
3. NOT EXISTS operator (gecorreleerde subquery):
   1. Inner query: Controleer voor elke track, door middel van een gecorreleerde subquery, of het gebruikt wordt in een playlist.
   2. Outer query: Hou enkel de tracks over waarvoor de conditie van de inner query ‘vals’ is.

**Radio 3.2**

Let bij deze oefening goed op hoe je zoekt naar de *jongste* presentator (en niet naar de oudste presentator).

Mogelijke oplossingsmethoden:

1. Scalaire subquery:
   1. Inner query: Selecteer de geboortedatum van de jongste presentator. Hiervoor heb je echter een aggregatiefunctie nodig (zie reeks 4).
   2. Outer query: Selecteer alle data met betrekking tot presentatoren wiens geboortedatum gelijk is aan het resultaat van de inner query.
2. ALL operator:
   1. Inner query: Selecteer de geboortedatums van alle presentatoren.
   2. Outer query: Selecteer alle data met betrekking tot presentatoren wiens geboortedatum later valt of gelijk is aan alle geboortedatums die voorkomen in het resultaat van de inner query.

**Radio 3.3**

De methodes om deze oefening op te lossen zijn gelijkaardig aan de methodes die gebruikt kunnen worden om Radio 3.2 op te lossen. Vergeet echter niet om, zowel in de inner als in de outer query, enkel vrouwelijke presentatrices te beschouwen (presenter.sex = ‘f’).

**Radio 3.4**

Mogelijke oplossingsmethoden:

1. Subqueries in FROM-clausule en INNER JOIN:
   1. Query 1: Selecteer alle tracks van Leonard Cohen (subq1).
   2. Query 2: Selecteer alle tracks die niet van Leonard Cohen zijn (subq2).
   3. subq1 INNER JOIN subq2 ON (subq1.title = subq2.title).
2. IN operator:
   1. Inner query: Selecteer alle titels van tracks van Leonard Cohen.
   2. Outer query: Selecteer alle titels van tracks die niet van Leonard Cohen zijn en die voorkomen in de resultatentabel van de inner query.
3. EXISTS operator (gecorreleerde subquery):
   1. Inner query: Controleer voor elke titel van een track van Leonard Cohen, door middel van een gecorreleerde subquery, of het ook voorkomt als titel van een track van een artiest die niet Leonard Cohen is.
   2. Outer query: Hou enkel de titels van de tracks van Leonard Cohen over waarvoor de conditie van de inner query ‘waar’ is.

**Radio 3.5**

Er is een verschil tussen het genre van een radioprogramma en het genre van tracks die gespeeld zijn tijdens de uitzending van een radioprogramma. Let dus goed op welk genre je hier dient te gebruiken.

Mogelijke oplossingsmethoden:

1. Subqueries in FROM-clausule en INNER JOIN:
   1. Query 1: Selecteer alle tracks die gespeeld zijn tijdens een uitzending van het radioprogramma met titel ‘De La Soul’ (subq1).
   2. Query 2: Selecteer de genres en artiesten van alle tracks (subq2).
   3. subq1 INNER JOIN subq2 ON (subq1.genre = subq2.genre).
2. IN operator:
   1. Inner query: Selecteer alle genres van tracks die gespeeld zijn tijdens een uitzending van het radioprogramma met titel ‘De La Soul’.
   2. Outer query: Selecteer alle artiesten die tracks gecomponeerd hebben waarvan het genre ook voorkomt in de resultatentabel van de inner query.
3. EXISTS operator (gecorreleerde subquery):
   1. Inner query: Controleer voor elke artiest, door middel van een gecorreleerde subquery, of er een track bestaat die gecomponeerd is door deze artiest, en waarvan het genre van deze track ook voorkomt als genre van tracks die gespeeld zijn tijdens een uitzending van het radioprogramma met titel ‘De La Soul’.
   2. Outer query: Hou enkel artiesten over waarvoor de conditie van de inner query ‘waar’ is.

**Radio 3.6**

Let op, het is niet mogelijk om de unnest-functie te gebruiken binnen een conditie in de JOIN- en WHERE-clausule.

Mogelijke oplossingsmethode met subqueries in de FROM-clausule en INNER JOIN:

1. Query 1 en 2: Maak twee (gelijke- resultatentabellen waarin elke rij bestaat uit 1 presenterid en 1 bijhorende hobby, door uitvoering van de unnest-functie op de hobbies kolom (subq1 en subq2).
2. subq1 INNER JOIN subq2 ON (subq1.hobby = subq2.hobby AND subq1.presenterid != subq2.presenterid).

Om ervoor te zorgen dat de achternaam van de eerste presentator alfabetisch voor de achternaam van de tweede presentator gesorteerd wordt, kan je opnieuw gebruik maken van de ‘<’ operator (zie ook Radio 2.8).

**Radio 3.7**

Mogelijke oplossingsmethoden:

1. LEFT/RIGHT JOIN.
2. NOT IN operator:
   1. Inner query: Selecteer alle albums waar er een track op staat die gespeeld is in een uitzending.
   2. Outer query: Hou enkel de albums over die niet voorkomen in de resultatentabel van de inner query.
3. NOT EXISTS operator (gecorreleerde subquery):
   1. Inner query: Controller voor elk album, door middel van een gecorreleerde subquery, of er een track opstaat die gespeeld is in een uitzending.
   2. Outer query: Hou enkel de albums over waarvoor de conditie van de inner query ‘vals’ is.

**Radio 3.8**

Mogelijke oplossingsmethoden:

1. INNER JOIN, NOT IN operator:
   1. Inner query: Selecteer alle presentatoren die samen met iemand anders een radioprogramma presenteren.
   2. Outer query: Hou enkel deze presentatoren over die voorkomen in de resultatentabel van de inner query.
2. EXISTS operator:
   1. Inner query: Controleer voor elke presentator, door middel van een gecorreleerde subquery, of er een andere presentator bestaat die een radioprogramma presenteert dat ook door de eerste presentator wordt gepresenteerd.
   2. Outer query: Hou enkel de presentatoren over waarvoor de conditie van de inner query ‘waar’ is.

**Radio 3.9**

Deze oefening kan op het eerste zicht wat lastig zijn om te vertalen naar een SQL query. Daarom raden we aan om de vraagstelling vanuit een ander standpunt te benaderen zodat deze oefening iets eenvoudiger wordt om op te lossen. Hint: eeuwige co-presentatoren zijn presentatoren die *nooit* een radioprogramma alleen presenteren. Als eerste stap kan je dus alle presentatoren selecteren die minstens 1 radioshow alleen presenteren. Hierna kan je dan alle presentatoren selecteren die niet voorkomen in deze verzameling.

Een andere manier om deze oefening op te lossen is door het aantal radioprogramma’s dat een presentator *in totaal* presenteert te vergelijken met het aantal radioprogramma’s dat een presentator *alleen* presenteert. Hoewel dit redelijk eenvoudig is, heb je hiervoor wel kennis in verband met aggregatiefuncties nodig (zie reeks 4).

**Radio 3.10**

De methodes om deze oefening op te lossen zijn gelijkaardig aan de methodes die gebruikt kunnen worden om Radio 3.4 op te lossen.

**Radio 3.11**

De methodes om deze oefening op te lossen zijn gelijkaardig aan de methodes die gebruikt kunnen worden om Radio 3.5 op te lossen. Let wel opnieuw goed op welk genre (radioprogramma/track) je hier dient te gebruiken.