Les sous requêtes

1. Les sous requêtes

Avec SQL il est possible d'imbriquer des requêtes un peu à la manière de poupées gigognes.

Mais pour toucher du doigt ce que l'on peut faire avec des sous requêtes, posons nous la question : **Où placer une sous requête ?**

Observons les types de résultats qu'une requête produit pour déterminer les emplacements qu'elle peut prendre au sein d'un ordre SELECT :

Résultat	Typologie des résultats
TIT_CODE CLI_NOM CLI_PRENOM -------- ---------- ---------- Mme. BOYER Martine Mme. GALLACIER Noëlle Mme. HESS Lucette Mme. LETERRIER Monique Mme. MARTINET Carmen Mme. DAVID Jacqueline Mme. MOURGUES Jacqueline Mme. ZAMPIERO Annick Mme. ROURE Marie-Louise Mme. DE CONINCK Patricia ...	Une table composée de colonnes et de lignes...
CLI_PRENOM ---------- Alain Marc Alain Paul Jean Alain Marcel Arsène André Daniel ...	Une table d'une seule colonne, c'est à dire une liste...
TIT_CODE CLI_NOM CLI_PRENOM -------- ---------- M. RAY Yannick Mme. ROURE Marie-Louise M. RECHUL Jacques M. ROUSSILLON Alain	Une ligne d'une table
MOYENNE -------- 406,74 F	Une table d'une seule ligne et d'une seule colonne, c'est à dire une valeur unique...
TIT_CODE CLI_NOM CLI_PRENOM -------- ---------- ----------	Pas de réponse (une table vide) et par opposition, une table NON vide...

Voyons maintenant comment imbriquer cela dans une autre requête...

Imbrication requête (résultat de requête)	Typologie de résultat et emplacement de la sous requête	Représentation
SELECT * FROM (TIT_CODE CLI_NOM CLI_PRENOM -------- ---------- ---------- Mme. BOYER Martine Mme. GALLACIER Noëlle Mme. HESS Lucette Mme. LETERRIER Monique Mme. MARTINET Carmen Mme. DAVID Jacqueline Mme. MOURGUES Jacqueline Mme. ZAMPIERO Annick Mme. ROURE Marie-Louise Mme. DE CONINCK Patricia)	Une requête renvoyant une table peut être imbriqué dans la clause FROM d'une autre requête...	SELECT * FROM (SELECT TIT_CODE, CLI_NOM, CLI_PRENOM FROM MaTable) ici, TableReponse est le renommage en table du résultat de la requête, car la clause FROM doit porter sur des tables nommées.
SELECT * FROM Matable WHERE uneColonne IN (CLI_PRENOM ---------- Alain Marc Alain Paul Jean Alain Marcel Arsène André Daniel)	Une requête renvoyant une seule colonne avec plusieurs valeurs peut être imbriqué dans le prédicat IN... mais aussi dans les prédicats ALL, SOME(ANY).	SELECT * FROM Matable WHERE uneColonne IN (SELECT CLI_NOM FROM MaTable)
SELECT * FROM maTable WHERE (valeur1, valeur2, valeur3) MATCH (TIT_CODE CLI_NOM CLI_PRENOM -------- ---------- M. RAY Yannick Mme. ROURE Marie-Louise M. RECHUL Jacques M. ROUSSILLON Alain)	Une requête renvoyant une seule ligne peut être imbriquée dans un prédicat MATCH et comparé à une ligne valuée.	SELECT * FROM maTable WHERE (valeur1, valeur2, valeur3) MATCH (SELECT TIT_CODE, CLI_NOM, CLI_PRENOM FROM T_CLIENT WHERE CLI_NOM LIKE'R%')
SELECT *, (MOYENNE -------- 406,74F) AS MOYENNE_DES_COMMANDES FROM MaTable ou SELECT * FROM MaTable WHERE MaColonne = (MOYENNE -------- 406,74F)	Une requête renvoyant une valeur unique peut être imbriquée partout ou une constante peut figurer...	SELECT *, (SELECT AVG(colonne) FROM UneTable) AS MOYENNE_DES_COMMANDES FROM MaTable ou SELECT * FROM MaTable WHERE MaColonne = (SELECT AVG(colonne) FROM UneTable) ou

ou SELECT COUNT(*), MaColonne FROM MaTable GROUP BY MaColonne HAVING COUNT(*) = (MOYENNE -------- 406,74 F) ou SELECT * FROM MaTable T1 JOIN AutreTable T2 ON T1.colonne1 = T2.colonne2 - (MOYENNE -------- 406,74 F) Etc...		SELECT COUNT(*), MaColonne FROM MaTable GROUP BY MaColonne HAVING COUNT(*) = (SELECT AVG(colonne) FROM UneTable) ou SELECT * FROM MaTable T1 JOIN AutreTable T2 ON T1.colonne1 = T2.colonne2 - (SELECT AVG(colonne) FROM UneTable)
SELECT * FROM MaTable WHERE EXISTS (TIT_CODE CLI_NOM CLI_PRENOM -------- ------- ----------)	Une requête renvoyant des valeurs ou pas peut être imbriquée dans un prédicat EXISTS, UNIQUE et MATCH.	SELECT * FROM MaTable WHERE EXISTS (SELECT TIT_CODE CLI_NOM CLI_PRENOM FROM UneTable)

Nous verrons qu'il existe des opérateurs spécialisés, comme EXISTS pour traiter particulièrement de cas d'imbrication. En particulier, les opérateurs ALL, SOME(ANY), MATCH et UNIQUE.

REMARQUE IMPORTANTE : chaque fois que vous voudrez mettre une sous requête dans un ordre SQL, assurez vous que la sous requête est comprise dans une paire de parenthèses.

1.1. Sous requêtes renvoyant une seule valeur

Nous allons d'abord étudier ce que nous pouvons faire en utilisant des sous requêtes ne renvoyant qu'une valeur unique. La plupart du temps, nous avons l'assurance de ne renvoyer qu'une valeur unique si nous utilisons une requête dont l'unique colonne est le résultat d'un calcul statistique (agrégation) comme les MAX, MIN, AVG, COUNT et SUM. C'est pourquoi on trouvera souvent ce genre d'expression dans les requêtes imbriquées des filtres WHERE et HAVING, mais aussi parfois dans la clause SELECT.

1.1.1. Dans la clause SELECT

On peut placer dans la clause SELECT, à la place de colonnes, des sous requêtes, voire même combiner par une opération, une colonne et une sous requête.

Notre hôtelier voudrait connaître l'évolution du prix moyen de ses chambres par rapport à son tarif de référence au premier janvier 2000.
Le prix moyen des chambres, pour n'importe quelle date d'application peut être obtenu par :

SELECT TRF_DATE_DEBUT, AVG(TRF_CHB_PRIX) AS MOYENNE FROM TJ_TRF_CHB GROUP BY TRF_DATE_DEBUT;	TRF_DATE_DEBUT MOYENNE ---------- -------- 1999-01-01 255.2500 1999-09-01 280.6500 2000-01-01 306.0500 2000-09-01 382.2500 2001-01-01 407.6500

Le tarif de référence qui nous intéresse est visible sur la 3eme ligne de la réponse. Nous pouvons l'obtenir en précisant la requête :

SELECT AVG(TRF_CHB_PRIX) AS MOYENNE FROM TJ_TRF_CHB WHERE TRF_DATE_DEBUT = '2000-01-01';	TRF_DATE_DEBUT MOYENNE ---------- -------- 1999-01-01 -50.8000 1999-09-01 -25.4000 2000-01-01 .0000 2000-09-01 76.2000 2001-01-01 101.6000

Pour calculer l'écart, il suffit de reporter ce nombre en le soustrayant du prix moyen de la requête de l'exemple 1 :

SELECT TRF_DATE_DEBUT, AVG(TRF_CHB_PRIX) - 306.05 AS MOYENNE FROM TJ_TRF_CHB GROUP BY TRF_DATE_DEBUT;	TRF_DATE_DEBUT MOYENNE ---------- -------- 1999-01-01 -50.8000 1999-09-01 -25.4000 2000-01-01 .0000 2000-09-01 76.2000 2001-01-01 101.6000

Il ne suffit plus que de remplacer la valeur 306.50 par la requête qui l'a générée :

SELECT TRF_DATE_DEBUT, AVG(TRF_CHB_PRIX) - (SELECT AVG(TRF_CHB_PRIX) FROM TJ_TRF_CHB WHERE TRF_DATE_DEBUT = '2000-01-01') AS MOYENNE FROM TJ_TRF_CHB GROUP BY TRF_DATE_DEBUT;	TRF_DATE_DEBUT MOYENNE ---------- -------- 1999-01-01 -50.8000 1999-09-01 -25.4000 2000-01-01 .0000 2000-09-01 76.2000 2001-01-01 101.6000

NOTA : remarquez que nous n'avons plus besoin de nommer les colonnes de la sous requête. Observez aussi que la sous requête a été placée dans une paire de parenthèses.

1.1.2. Dans les filtres WHERE et HAVING

C'est l'endroit le plus classique pour placer une sous-requête.

Premier exemple :

Intéressons de savoir quelles sont les chambres au 01/01/2000 qui ont un prix voisin à + ou - 10 F de la moyenne des prix au 1/1/2000...
Nous savons déjà que la moyenne des prix au 1/1/200 de toutes les chambres a déjà été calculée par la requête de l'exemple 2 et sa valeur est 306.50.
Nous pouvons donc formuler ainsi la requête :

SELECT CHB_ID, TRF_CHB_PRIX FROM TJ_TRF_CHB WHERE TRF_CHB_PRIX BETWEEN 296.5 AND 316.50 AND TRF_DATE_DEBUT = '2000-01-01';	CHB_ID TRF_CHB_PRIX ---------- ---------- 2 300.0000 6 300.0000 9 300.0000 16 300.0000 19 300.0000

Il semble qu'il faudrait envisager de placer deux fois la sous requête... Mais une petite astuce due aux propriétés des équations va nous permettre de résoudre ce problème. En effet , si nous retirons du prix de la chambre la valeur 306.50, la requête devient :

SELECT CHB_ID, TRF_CHB_PRIX FROM TJ_TRF_CHB WHERE TRF_CHB_PRIX - 306.50 BETWEEN -10 AND +10 AND TRF_DATE_DEBUT = '2000-01-01';	CHB_ID TRF_CHB_PRIX ---------- ---------- 2 300.0000 6 300.0000 9 300.0000 16 300.0000 19 300.0000

Ce qui d'ailleurs est le strict énoncé du départ. Dès lors le remplacement de cette somme par la requête de l'exemple 2 est un jeu d'enfant :

```
SELECT CHB_ID, TRF_CHB_PRIX
FROM   TJ_TRF_CHB
WHERE  TRF_CHB_PRIX - (SELECT AVG(TRF_CHB_PRIX)
                       FROM TJ_TRF_CHB
                       WHERE TRF_DATE_DEBUT = '2000-01-01') BETWEEN -10 AND +10
AND TRF_DATE_DEBUT = '2000-01-01';
```

```
CHB_ID     TRF_CHB_PRIX
---------- ------------
2          300.0000
6          300.0000
9          300.0000
16         300.0000
19         300.0000
```

Deuxième exemple :

Notre hôtelier désire savoir quels sont les mois pour lesquels le taux d'occupation de son hôtel à dépassé les 2/3.
Calculer le taux d'occupation (c'est à dire le nombre de chambre occupé chaque mois) est assez simple. Il suffit de compter le nombre d'occurences de la table ou sont stockées les informations des réservations (TJ_CHB_PLN_CLI).

```
SELECT EXTRACT(YEAR FROM PLN_JOUR) AS ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) AS MOIS,
     COUNT(*) AS NOMBRE
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR)
ORDER  BY ANNEE, MOIS ;
```

```
ANNEE      MOIS       NOMBRE
---------- ---------- ------
1999       1          404
1999       2          354
1999       3          405
1999       4          382
1999       5          436
1999       6          392
1999       7          394
1999       8          424
1999       9          399
1999       10         419
...
```

De même calculer un taux d'occupation de 66.67% consiste à faire le décompte des chambres et le multiplier par ce facteur, ce qui donne un taux d'occupation par nuit, que l'on peut ensuite ramener au mois par une côte mal taillée de 30 jours (référence comptable) :

```
SELECT COUNT(*) * 30 * 0.6667 AS
TAUX_OCCUPATION_MOYEN
FROM T_CHAMBRE ;
```

```
TAUX_OCCUPATION_MOYEN
---------------------
400.0200
```

Dès lors nous avons les éléments pour imbriquer nos requêtes...
Le filtrage d'un agrégat (calcul statistique) ne peut se faire que par le filtre HAVING :

```
SELECT EXTRACT(YEAR FROM PLN_JOUR) AS ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) AS MOIS,
     COUNT(*) AS NOMBRE
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR)
HAVING COUNT(*) > 400.02
ORDER  BY ANNEE, MOIS;
```

```
ANNEE      MOIS       NOMBRE
---------- ---------- ------
1999       1          404
1999       3          405
1999       5          436
1999       8          424
1999       10         419
1999       12         440
2000       1          418
2000       2          402
2000       3          422
2000       4          401
...
```

La touche finale consistant à remplacer la valeur numérique 400.02 par la requête de l'exemple 9 en n'oubliant pas les parenthèses :

```
SELECT EXTRACT(YEAR FROM PLN_JOUR) AS ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) AS MOIS,
     COUNT(*) AS NOMBRE
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR)
HAVING COUNT(*) > (SELECT COUNT(*) * 30 * 0.6667
               FROM   T_CHAMBRE)
ORDER  BY ANNEE, MOIS ;
```

```
ANNEE      MOIS
NOMBRE
----------- ----------- ------
1999       1          404
1999       3          405
1999       5          436
1999       8          424
1999       10          419
1999       12          440
2000       1          418
2000       2          402
2000       3          422
2000       4          401
...
```

Ce n'est pas plus compliqué que cela ! En fait, il faut comprendre que les mécanismes de base pour l'implémentation des sous requêtes sont toujours les mêmes : décomposition de la requête en éléments simples, création des requêtes élémentaires puis réassemblage du tout.

NOTA : on peut aussi placer une sous requête dans le filtre de jointure ON...

1.2. Sous requêtes renvoyant une liste (Opérateurs IN, ALL et ANY(SOME))

Une liste de valeurs, c'est à dire une colonne, ne peut être utilisée comme critère de comparaison que par des opérateurs spécialisés.
C'est le cas de l'opérateur IN, mais aussi des opérateurs ALL et ANY (ou SOME) que nous allons étudier dans un second temps..

1.2.1. Dans le prédicat IN

L'opérateur IN est utilisable dans tous les prédicats, c'est pourquoi on le retrouve dans les filtres WHERE et HAVING. Pour alimenter une liste de valeur pour le prédicat IN, il suffit de placer une requête ne renvoyant qu'une seule colonne.

Premier exemple :

Monsieur BOUVIER vient réserver une chambre, et comme il s'y prend à l'avance, il aimerait prendre une chambre dans laquelle il n'a jamais dormi au cours de l'année 2001...
Dans ce genre de cas, la négation se fait par différence : toutes les chambres de l'hôtel MOINS les chambres dans lesquelles monsieur BOUVIER a déjà couché au cours de l'année 2001.

Trouver les chambres occupées par monsieur BOUVIER au cours de l'année 2001, n'est pas bien difficile :

```
SELECT DISTINCT C.CHB_ID
FROM   TJ_CHB_PLN_CLI J J
     JOIN T_CLIENT C
          ON J.CLI_ID = C.CLI_ID
WHERE  C.CLI_NOM ='BOUVIER'
  AND  EXTRACT(YEAR FROM J.PLN_JOUR) =
2001;
```

```
CHB_ID
-----------
1
2
3
4
6
7
9
10
11
12
...
```

De même, trouver toutes les chambres de l'hôtel, relève de la plus élémentaire des requêtes :

SELECT CHB_ID FROM T_CHAMBRE;	CHB_ID ---------- 1 2 3 4 5 6 7 8 9 10 ...

Dès lors l'utilisation du IN, et plus particulièrement ici du NOT IN, va permettre de faire le lien entre les deux requêtes :

SELECT CHB_ID FROM T_CHAMBRE WHERE CHB_ID NOT IN (SELECT DISTINCT C.CHB_ID FROM TJ_CHB_PLN_CLI JOIN T_CLIENT C ON J.CLI_ID = C.CLI_ID WHERE C.CLI_NOM ='BOUVIER' AND EXTRACT(YEAR FROM J.PLN_JOUR) = 2001);	CHB_ID ------ 5 8

Notre client pourra coucher dans l'une des chambres 5 ou 8...

NOTA : beaucoup de requêtes utilisant le IN (comme le NOT IN) peuvent être simplifiées en utilisant des jointures. Le IN par des jointures internes, le NOT IN par des jointures externes associées à une clause HAVING COUNT(...) = 0. En général les performances seront meilleures en utilisant une jointure que dans le cas d'une sous requête avec [NOT] IN.

Ainsi notre exemple précédent peut se réécrire :

SELECT DISTINCT H.CHB_ID FROM T_CHAMBRE H LEFT OUTER JOIN TJ_CHB_PLN_CLI J ON H.CHB_ID = J.CHB_ID LEFT OUTER JOIN T_CLIENT C ON J.CLI_ID = C.CLI_ID AND CLI_NOM ='BOUVIER' AND EXTRACT(YEAR FROM J.PLN_JOUR) = 2001 GROUP BY H.CHB_ID, J.CHB_ID HAVING COUNT(C.CLI_ID) = 0 ORDER BY H.CHB_ID;	CHB_ID ------ 5 8

Second exemple :

Le gérant de l'hôtel voudrais savoir quels sont les mois (et les années) qui ont eu un nombre de nuitées égal aux nuitées enregistrées au cours de n'importe quel mois de janvier ?

Là encore, il convient de décortiquer la question pour en trouver les requêtes élémentaires...
Pour connaître le nombre de nuitées des mois de janvier, il suffit de totaliser la colonne CHB_PLN_CLI_NB_PERS de la table TJ_CHB_PLN_CLI, comme suit :

SELECT COUNT (CHB_PLN_CLI_NB_PERS) AS PERSONNE, EXTRACT(YEAR FROM PLN_JOUR) AS ANNEE FROM TJ_CHB_PLN_CLI WHERE EXTRACT(MONTH FROM PLN_JOUR) = 1 GROUP BY EXTRACT(YEAR FROM PLN_JOUR);	PERSONNE ANNEE ---------- ----- 404 1999 415 2001 418 2000

Étendre le comptage à tous les mois de toutes les années n'est pas plus difficile :

```
SELECT COUNT (CHB_PLN_CLI_NB_PERS) AS PERSONNE,
     EXTRACT(YEAR FROM PLN_JOUR) ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) MOIS
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR);
```

```
PERSONNE    ANNEE       MOIS
----------- ----------- ----
412         2000        12
389         2001        11
405         1999        3
402         2000        2
415         2001        1
382         1999        4
422         2000        3
379         2001        2
424         1999        8
413         2000        7
...
```

Maintenant, pour ne plus filtrer ces comptages que sur les valeurs retournées par la première requête (404, 415, 418), il ne suffit plus que d'utiliser la clause HAVING comme suit :

```
SELECT COUNT (CHB_PLN_CLI_NB_PERS) AS PERSONNE,
     EXTRACT(YEAR FROM PLN_JOUR) ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) MOIS
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR)
HAVING COUNT (CHB_PLN_CLI_NB_PERS) IN (404, 415, 418);
```

```
PERSONNE    ANNEE       MOIS
----------- ----------- ----
415         2001        1
404         1999        1
415         2001        7
404         2001        4
418         2000        1
```

C'est à dire en reprenant le résultat de la requête 16 dans une liste IN.

Il ne nous reste plus qu'à remplacer le contenu de la dernière parenthèse située après le IN par la première requête à laquelle on ne laisse subsister que la colonne de comptage dans la clause select :

```
SELECT COUNT (CHB_PLN_CLI_NB_PERS) AS PERSONNE,
     EXTRACT(YEAR FROM PLN_JOUR) ANNEE,
     EXTRACT(MONTH FROM PLN_JOUR) MOIS
FROM   TJ_CHB_PLN_CLI
GROUP  BY EXTRACT(YEAR FROM PLN_JOUR), EXTRACT(MONTH FROM
PLN_JOUR)
HAVING COUNT (CHB_PLN_CLI_NB_PERS) IN (SELECT COUNT
(CHB_PLN_CLI_NB_PERS)
                         FROM   TJ_CHB_PLN_CLI
                         WHERE  EXTRACT(MONTH FROM PLN_JOUR) = 1
                         GROUP  BY EXTRACT(YEAR FROM PLN_JOUR),
                                EXTRACT(MONTH FROM PLN_JOUR));
```

```
PERSONNE    ANNEE       MOIS
----------- ----------- ----
415         2001        1
404         1999        1
415         2001        7
404         2001        4
418         2000        1
```

C'est bien évidemment le résultat attendu !

1.4. Sous requêtes renvoyant une table

N'importe quelle requête est capable de renvoyer une table... Car un résultat de requête est bien une table. C'est l'essence même de la fonction d'une requête.
Or, où place t-on une table dans une requête ? Dans la clause FROM.
Autrement dit il est possible de placer une sous requête dans la clause FROM de n'importe quelle requête à la place d'un nom de table !
Nous verrons aussi qu'il est possible de placer une sous requête dans des prédicats très particuliers de SQL 3, à l'aide des opérateurs FOR ALL, FOR ANY et FOR SOME...

Continuons notre recherche entamée précédemment. Notre hôtelier voudrait bien éviter aux autres clients les rituelles batailles de polochons qui suivent les matches et pénalisent le sommeil du juste. Il cherche donc à savoir si au moins un étage de son hôtel permet de coucher les 24 personnes qui compose cette équipe (joueurs, remplaçants, entraîneurs...). Il voudrait donc savoir quel est le maximum de la somme des couchages des étages...

Nous avions vu que le calcul de la totalité des places de couchage par étage est assez aisé :

SELECT SUM(CHB_COUCHAGE) AS COUCHAGE FROM T_CHAMBRE GROUP BY CHB_ETAGE;	COUCHAGE ---------- 23 22 9

De ce résultat nous pourrions extraire le maximum. Il sufirait de reprendre le résultat de cette requête, et de faire :

SELECT MAX(COUCHAGE) AS MAX_COUCHAGE FROM (COUCHAGE ---------- 23 22 9)	MAX_COUCHAGE ---------- 23

En remplaçant le résultat par la requête de l'exemple 25, nous obtenons :

SELECT MAX(COUCHAGE) AS MAX_COUCHAGE FROM (SELECT SUM(CHB_COUCHAGE) AS COUCHAGE FROM T_CHAMBRE GROUP BY CHB_ETAGE) ;	MAX_COUCHAGE ---------- 23

C'est moins que la composition de l'équipe, mais je vous rassure, notre hôtelier, qui à du métier, a finalement eût l'idée de rajouter dans la plus grande chambre, un lit d'enfant...

ATTENTION : lorsque l'on place une sous requête en tant que table dans la clause FROM d'une requête, il faut pouvoir donner un nom à cette table ne serait-ce que parce qu'elle peut être jointes aux autres. Il convient donc de lui donner systématiquement un surnom.
Ici le surnom choisit a été la lettre T, comme Table !

1.5. Sous requêtes vide, non vide

Le principe est le suivant : si la sous requête renvoie un résultat quelconque, alors le prédicat vaut vrai. Si le sous requête ne renvoit aucune ligne, le prédicat vaut faux. SQL 2 à prévu deux prédicats spécillisés qui sont EXISTS et UNIQUE.

1.5.1. Dans le prédicat EXISTS

Le prédicat EXISTS permet de tester l'existence ou l'absence de données dans la sous requête.
Si la sous requête renvoie au moins une ligne, même remplie de marqueurs NULL, le prédicat est vrai. Dans le cas contraire le prédicat à valeur fausse, y compris si l'évaluation à la valeur UNKNOWN (dans la cas d'un comparaison avec un marqueur NULL).
Le prédicat EXISTS peut être combinés avec l'opérateur de négation NOT.

Nous voulons obtenir le total du couchage de l'hôtel, toutes chambres confondues, à condition qu'il y ait au moins une chambre dotée d'un couchage pour au moins 3 personnes :

SELECT SUM(CHB_COUCHAGE) AS TOTAL_COUCHAGE FROM T_CHAMBRE WHERE EXISTS (SELECT * FROM T_CHAMBRE WHERE CHB_COUCHAGE >= 3);	TOTAL_COUCHAGE ---------- 54

En fait l'utilisation d'un prédicat EXISTS n'a pas grand sens sans l'utilisation des sous requêtes corrélées.

IMPORTANT

- le prédicat EXISTS est en général plus rapide que le prédicat IN
- le comportement du prédicat EXISTS face au retour de marqueurs NULL renvoyés par la sous requête diffère sensiblement d'un SGBD à l'autre. En particulier DB2 et Oracle ne sont pas conforme à la norme SQL2
- le prédicat EXISTS n'a aucun intérêt sans une sous requête corrélée
- il convient de toujours utiliser l'étoile comme unique contenu de la clause SELECT de la sous requête car dans ce cas particulier, le moteur SQL choisit une constante la plus adaptée à un traitement performant

1.5.2. Dans le prédicat UNIQUE

UNIQUE est un raffinement du prédicat EXISTS. Cette extension du fonctionnement du prédicat EXISTS porte sur le doublonnage des lignes renvoyées. En effet, UNIQUE exige qu'il n'y ait aucun doublon dans les lignes renvoyées par la sous requête. En d'autres termes UNIQUE vaut faux si au moins deux lignes renvoyées par la sous requête comporte les mêmes données.

Nous voulons obtenir le total du couchage de l'hôtel, toutes chambres confondues, à condition qu'il n'y ait qu'une seule chambre dotée d'un couchage pour exactement 5 personnes :

SELECT SUM(CHB_COUCHAGE) AS TOTAL_COUCHAGE FROM T_CHAMBRE WHERE UNIQUE (SELECT CHB_COUCHAGE FROM T_CHAMBRE WHERE CHB_COUCHAGE = 3);	TOTAL_COUCHAGE ----------

Il existe deux chambres (6 et 17, d'ID 6 et 16) dotées de 5 places de couchage. La sous requête renvoie donc deux lignes possédant des données identiques. UNIQUE vaut alors faux, aucun résultat n'est donc renvoyé.
En fait l'utilisation d'un prédicat EXISTS n'a pas grand sens sans l'utilisation des sous requêtes corrélées.

IMPORTANT

- Le prédicat UNIQUE est en général beaucoup plus rapide que certaines constructions telles que le prédicat NOT EXIST associé au SELECT DISTINCT oue encore une sous requête avec un filtre HAVING pour comptre les lignes dédoublées...
- Le comportement du prédicat UNIQUE face au retour de marqueurs NULL renvoyés par la sous requête peut diffèrer sensiblement d'un SGBD à l'autre.
- Le prédicat UNIQUE n'a aucun intérêt sans une sous requête corrélée.
- Contrairement à EXISTS, il convient de toujours spécifier les colonnes visées dans la clause SELECT.

NOTA : ce prédicat est hélas rarement implémenté dans les SGBD... Dommage !

2. Les sous requêtes corrélées

Nous allons maintenant rajouter une couche à la puissance des sous requêtes en étudiant comment on peut corréler une sous requête à la requête au sein de laquelle elle est imbriquée.

NOTA : On trouve parfois les mots sous requêtes "imbriquées", sous requêtes "liées" pour désigner la techniques des sous requêtes corrélées.

En fait une sous requête corrélée est une sous requête qui s'exécute pour chaque ligne de la requête principale et non une fois pour toute. Pour arriver à un tel résultat, il suffit de faire varier une condition (en général un prédicat) en rappelant dans la sous requête la valeur d'une colonne de la requête principale.

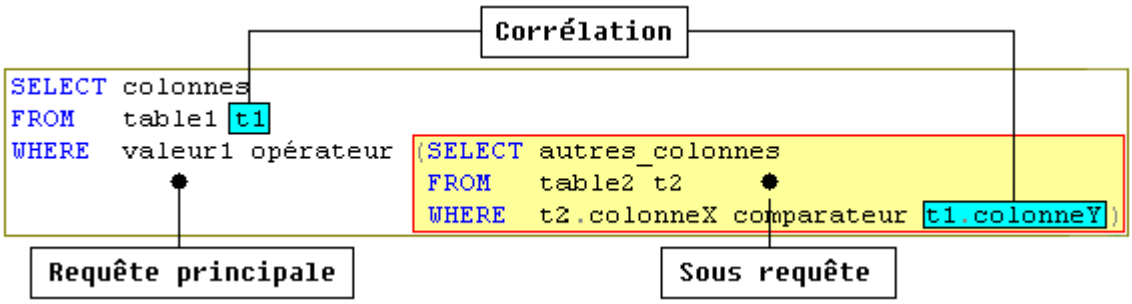
Il est plus facile de comprendre cette notion à l'aide d'une comparaison à un langage procédural, mais aussi en visualisant graphiquement une telle imbrication.

Le principe procédural est simple. Les sous requêtes corrélées correspondent en fait à des boucles imbriquées. Voici un exemple de procédure permettant de lire un fichier et de scruter chaque ligne du fichier à la recherche d'un mot précis :

```
Procedure RechercheMot (LeMot : string, leFichier : File)
Var
  NumLigne     integer;
  Ligne        string
  NumCar       integer;
  LongueurMot  integer;
endVar
Proc
  LeFichier.open()
  LongueurMot := LeMot.length
  FOR NumLigne from 1 to LeFichier.Length()
  |  Ligne := LeFichier.readLine()
  |  FOR NumCar from 1 to Ligne.length() - LongueurMot + 1
  | |   if leMot = Ligne.substring(NumCar, LongueurMot)
  | |   then
  | |      screen.prompt("Mot " + LeMot + " trouvé à la ligne : " + string(NumLigne))
  | |   endif
  |  ENDFOR
  ENDFOR
  LeFichier.close()
endProc
```

Nous voyons bien que les deux boucles sont imbriquées. C'est à dire qu'il faut recommencer à rechercher l'occurrence du mot dans chacune des lignes du fichier que l'on "balaye". C'est exactement comme cela que fonctionnent les requêtes imbriquées. La requête principale donne à la sous requête de nouvelles conditions d'exécution à chaque fois que la requête principale trouve une ligne correspondant à ses propres filtres.

De manière graphique, le principe d'une sous requête est le suivant :



Cherchons donc à trouver les clients qui ont un prénom en commun. Autrement dit pour qu'un client soit sélectionné, il faut qu'un autre client porte le même prénom. Nous apellerons cela l'homoprenymie !

Une première idée qui vient immédiatement à l'esprit consiste à faire :

```
SELECT CLI_ID, CLI_NOM, CLI_PRENOM
FROM   T_CLIENT
WHERE  CLI_PRENOM IN (SELECT
CLI_PRENOM
                FROM   T_CLIENT) ;
```

```
CLI_ID  CLI_NOM
CLI_PRENOM
------- ----------- ----------
1       DUPONT      Alain
2       MARTIN      Marc
3       BOUVIER     Alain
4       DUBOIS      Paul
5       DREYFUS     Jean
6       FAURE       Alain
7       LACOMBE     Paul
8       DUHAMEL     Evelyne
9       BOYER       Martine
10      MARTIN      Martin
...
```

La réponse est le contenu complet de la table des clients... Tous les clients auraient-ils un homoprényme ? Non, bien entendu, mais la faute que nous avons commise est simple : un client porte bien évidemment le même prénom que... lui même !

C'est pourquoi nous devons retirer dans la sous requête la référence au client que l'on scrute dans la requête principale.
La requête propre est donc celle-ci :

```
SELECT CLI_ID, CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C1
WHERE  CLI_PRENOM IN (SELECT CLI_PRENOM
                FROM   T_CLIENT C2
                WHERE  C1.CLI_ID <> C2.CLI_ID);
```

```
CLI_ID  CLI_NOM     CLI_PRENOM
------- ----------- ----------
3       BOUVIER     Alain
1       DUPONT      Alain
6       FAURE       Alain
25      LE GUILLARD Alain
69      LEI         Alain
65      NOCENTINI   Alain
77      ROUSSILLON  Alain
71      BOURA       André
13      PHILIPPE    André
38      ALBERT      Christian
32      LEBAILLIF   Christian
...
```

Soit 53 occurences.

Ici, la corrélation entre la requête principale et la sous requête se fait dans la clause WHERE et porte sur le fait que l'identifiant des clients scrutés dans la sous requête ne doit pas être le même que dans la requête principale : *C1.CLI_ID <> C2.CLI_ID*.
Notons que pour obtenir cette corrélation, il faut donner des surnoms à nos tables.

Plus élégament, nous pouvons écrire cette requête à l'aide d'une clause EXISTS qui sera notablement plus performante :

```
SELECT CLI_ID, CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C1
WHERE  EXISTS (SELECT *
               FROM   T_CLIENT C2
               WHERE  C1.CLI_ID <> C2.CLI_ID
                 AND C1.CLI_PRENOM = C2.CLI_PRENOM) ;
```

```
CLI_ID  CLI_NOM     CLI_PRENOM
------- ----------- ----------
3       BOUVIER     Alain
1       DUPONT      Alain
6       FAURE       Alain
25      LE GUILLARD Alain
69      LEI         Alain
65      NOCENTINI   Alain
77      ROUSSILLON  Alain
71      BOURA       André
13      PHILIPPE    André
38      ALBERT      Christian
32      LEBAILLIF   Christian
...
```

Ce qui, bien évidemment donne le même résultat.

Si maintenant nous exigeons qu'il n'y ait qu'un seul homoprényme de nos clients, alors le prédicat UNIQUE vient à notre secours :

```
SELECT CLI_ID, CLI_NOM, CLI_PRENOM
FROM   T_CLIENT C1
WHERE  UNIQUE (SELECT CLI_PRENOM
               FROM   T_CLIENT C2
               WHERE  C1.CLI_ID <> C2.CLI_ID
                 AND C1.CLI_PRENOM = C2.CLI_PRENOM) ;
```

```
CLI_ID  CLI_NOM     CLI_PRENOM
------- ----------- ----------
13      PHILIPPE    André
71      BOURA       André
32      LEBAILLIF   Christian
38      ALBERT      Christian
90      JOLY        Christophe
87      BERTRAND    Christophe
24      CHTCHEPINE  Dominique
35      PICOT       Dominique
51      DAVID       Jacqueline
56      MOURGUES    Jacqueline
43      MONTEIL     Jean
5       DREYFUS     Jean
53      BERGER      Jean-Pierre
75      MARTIN      Jean-Pierre
66      LAYANI      Lionel
27      LECUYER     Lionel
18      FAYOLLE     Olivier
45      ORELL       Olivier
72      CARDONA     Philippe
41      PLATONOFF   Philippe
```

Notons qu'ils ne sont plus que 20 occurences, et non plus 53, soit 10 paires de clients homoprénymes. En fait un nombre pair était bien évidemment attendu !