

M2106 : Programmation et administration des bases de données

Cours 3/6 – LID

Guillaume Cabanac

guillaume.cabanac@univ-tlse3.fr



version du 18 février 2018

LID : Langage d'interrogation des données

- 1 Projection π
- 2 Sélection σ
- 3 Opérateurs ensemblistes \cup , \cap , \setminus et \exists
- 4 Groupement
- 5 Produit \times
- 6 Jointures \bowtie , \bowtie_{θ} , \bowtie_{θ} , \bowtie_{θ} et \bowtie_{θ}

Syntaxe simplifiée des requêtes SQL

```
-- Un résultat est une table, un select, une vue
-- ou l'application d'un opérateur ensembliste à deux résultats
Res := {Tab | Sel | Vue | Res1 {union [all] | intersect | minus} Res2}

-- Définition d'un résultat de type select (notion d'algèbre fermée)
Sel := select col1 [[as] aliasCol1], ..., fonc1 [[as] aliasColN], ...
      from Res1 [[as] aliasRes1], ... , ResN [[as] aliasResN]
      [where Cond1 {and | or} ... {and | or} CondN]
      [
        group by colX, ... , colY
        [having cond1 {and | or} ... {and | or} condN]
      ]
      [order by colZ [{asc} | desc] [nulls {first | last}]], ...

-- Définition d'une condition booléenne
Cond := {
  Oper1 [(+)] {= | < | > | {^ | ! | < | >}=} Oper2 [(+)]
  | Oper {= | < | >} {all | any | some} Res
  | Oper [not] null
  | Oper between borneMin and borneMax
  | Oper [not] like 'motif'
  | Oper [not] in Res           -- Opérateur '∈' en théorie des ensembles
  | [not] exists Res           -- Opérateur '∃' en logique
  | [not] Cond                 -- Opérateur '¬' en logique
  | ( Cond )                   -- Parenthésage
}
```

Légende : [X] l'élément X est optionnel
 {A | B | C} un des éléments A, B ou C est obligatoire

Base de données exemple

```
select * from adherent order by idA ;
```

IDA	NOM	PRENOM	VILLE	MAIL
1	Eastwood	Clint	Troyes	clint@eastwood.com
45	Vladuboudin	Tintin	Bruxelles	
69	Lovelace	Ada	Marseille	ada@hotmail.fr
999	Satan	Michel	Las Vegas	

```
select * from chien order by tatouage ;
```

TATOUAGE	NOM	DDN	SEXE	IDA	IDR
3	Lassie	04-MAY-10	F	999	CO
42	Rantanplan	01-MAY-09	M	45	
51	Milou	15-FEB-99	M	45	CA
666	Cerbère	08-MAR-11	M	999	CA
1664	Iench	12-DEC-12	F	69	

Projection d'une relation : $\pi_{att_1, \dots, att_N} R$ π Projection

La **projection** d'une relation R permet de n'afficher qu'un sous-ensemble des **attributs** de la relation.

```
-- Les colonnes projetées sont 'prenom' et 'nom'
select prenom, nom
from adherent ;
```

PRENOM	NOM
Clint	Eastwood
Tintin	Vladuboudin
Ada	Lovelace
Michel	Satan

Sélection dans une relation : $\sigma_{cond_1, \dots, cond_N} R$ σ Sélection

La **sélection** d'une relation R permet de n'afficher que les **tuples** de la relation correspondant aux **conditions booléennes** spécifiées.

```
-- Les lignes sélectionnées contiennent toutes les colonnes
-- de la table adherent ('*' = toutes les colonnes).
select *
from adherent
where idA = 69
      or idA = 999 ;
```

IDA	NOM	PRENOM	VILLE	MAIL
69	Lovelace	Ada	Marseille	ada@hotmail.fr
999	Satan	Michel	Las Vegas	

Sélection dans une relation : $\sigma_{cond_1, \dots, cond_N} R$

Chaînes, nombres et dates

Les opérateurs de sélection (σ) :

- $a = b$ Égalité entre a et b.
- $a <> b$ Différence entre a et b.
- $a \neq b$ ""
- $a \wedge b$ ""
- $a > b$ Supériorité de a par rapport à b.
- $a \geq b$ Supériorité ou égalité de a par rapport à b.
- $a < b$ Infériorité de a par rapport à b.
- $a \leq b$ Infériorité ou égalité de a par rapport à b.

Sélection dans une relation : $\sigma_{cond_1, \dots, cond_N} R$

Ensemble de valeurs

Les opérateurs de sélection (σ) sur opérande ensembliste :

- $a \text{ in } (X)$ Présence de la valeur a dans l'ensemble X.
- $a = \text{any } (X)$ ""
- $a = \text{some } (X)$ ""
- $a \text{ not in } (X)$ Absence de la valeur a dans l'ensemble X.
- $a <> \text{all } (X)$ ""
- $a > \text{all } (X)$ La valeur a est supérieure aux valeurs de X.
- $a \geq \text{all } (X)$ La valeur a est supérieure ou égale aux valeurs de X.
- De nombreuses autres combinaisons sont possibles...

NB : avec $X = x_1, \dots, x_N$

Sélection dans une relation : $\sigma_{cond_1, \dots, cond_N} R$

Cas particulier des valeurs null

Les opérateurs de sélection (σ) :

a is null Nullité de a (autrement dit : a n'est pas renseigné).

a is not null Non-nullité de a (autrement dit : a est renseigné).

⚠ L'expression « X = null » est **toujours** évaluée à faux !

```
select * from adherent where mail = null ;           ... where mail is null
-- no rows selected (K0)                          2 rows selected (OK : TV & MS)
```

⚠ L'expression « X <> null » est **toujours** évaluée à faux !

```
select * from adherent where mail <> null ;         ... where mail is not null
-- no rows selected (K0)                          2 rows selected (OK : CE & CM)
```

À retenir : toujours utiliser **is null** et **is not null**.

Projection et sélection dans une relation

Figure 7-1 Projection and Selection

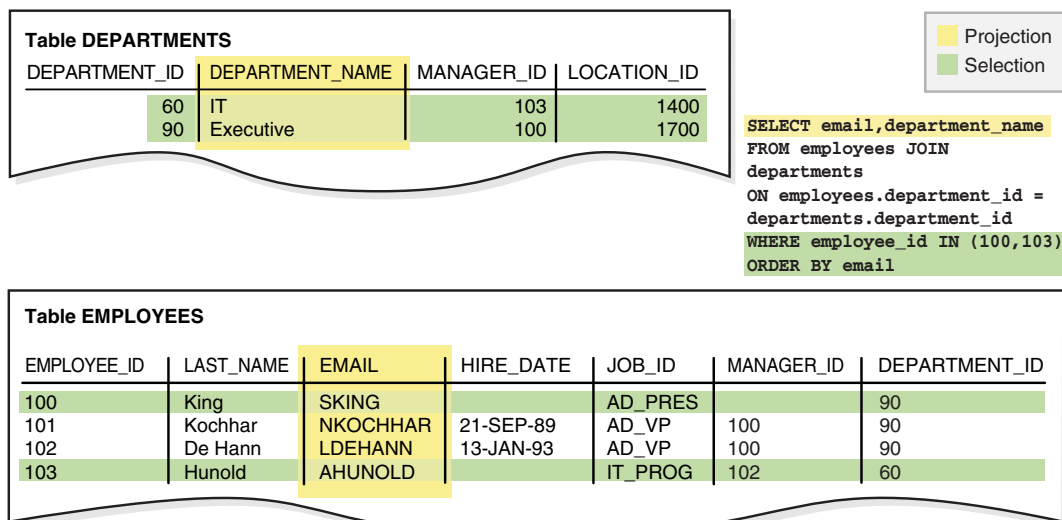


Illustration de la projection (π) et de la sélection (σ) en SQL

Tri des résultats

Clause order by

- Les résultats peuvent être **ordonnés** selon plusieurs clés de tri : nom ou position des colonnes.
- Deux **sens de tri** sont offerts : croissant (par défaut) ou décroissant.
- La position des **valeurs nulles** est également spécifiable.

```
-- Affichage des caractéristiques des chiens, ces derniers étant ordonnés par :
-- 1. idR croissant en affichant les valeurs nulles en haut de la liste,
-- 2. les lignes ex æquo sont départagées sur le champ nom (croissant),
-- 3. les lignes ex æquo sont départagées sur le champ ddn (décroissant).
select idR race, sexe, nom, tatouage, ddn "date de naissance"
from chien
order by idR nulls first, nom, ddn desc ;
```

```
-- Requête équivalent avec une notation positionnée (numéro de colonne)
select idR race, sexe, nom, tatouage, ddn "date de naissance"
from chien
order by 1 nulls first, 3, 5 desc ;
```

Élimination des doublons

Opérateur distinct

L'opérateur distinct « dédouble » le résultat :
il **élimine les lignes en double** dans le résultat.

<pre>select idR from chien order by idR nulls first ;</pre>	<pre>select distinct idR from chien order by idR nulls first ;</pre>
<pre>IDR ---</pre> <pre>CA CA CO ---</pre>	<pre>IDR ---</pre> <pre>CA CO ---</pre>

Composition de requêtes : $\pi_{att_1, \dots, att_N} (\sigma_{cond_1, \dots, cond_N} R)$ $\pi(\sigma)$ Sélection et projection

Le résultat de toute requête peut être à l'entrée d'une autre requête. On parle d'**algèbre fermée** : toute opération sur un résultat produit un nouveau résultat, qui est manipulable à son tour.

Parallèle avec les mathématiques : $\cos\left(\frac{3\pi}{4}\right) + \left(\sqrt[15]{42} \times 51!\right)$

```
-- Cette requête SQL spécifie une projection et une sélection
exécution
```

```
4.  select prenom, nom
1.  from adherent
2.  where idA = 69
3.  or idA = 999 ;
```

NOM	PRENOM
-----	-----
Lovelace	Ada
Satan	Michel

Composition de requêtes : les sous-requêtes

Généralisation de la composition de requêtes

Le résultat d'une **sous-requête** peut être à l'entrée d'un opérateur dans la **requête englobante**.

```
-- Speed dating pour chiens :
-- Adhérents disposant d'un mail, possédant une chienne et habitant
-- dans la même ville que les adhérents possédant un chien.
select nom, prenom
from adherent
where idA in (select idA --> possesseurs de chiennes
              from chien
              where sexe = 'F')
  and ville in (select ville --> ville des possesseurs de chiens
               from adherent
               where idA in (select idA --> possesseurs de chiens
                             from chien
                             where sexe = 'M'))
  and mail is not null --> possesseurs de chiennes avec email
order by nom, prenom ;
```

Composition de requêtes : les sous-requêtes

Exemples de sous-requêtes non-synchronisées

```
-- Chien le plus âgé ?
select * from chien where ddn >= all(select ddn from chien) ;
```

```
--   TATOUAGE NOM          DDN          SEXE          IDA IDR
--   -----
--           51 Milou      15-FEB-99 M           45 CA
```

```
-- Prime la plus faible ?
select prime
from participation
where prime <= all(select prime from participation) ;
```

```
-- PRIME
-- -----
--      1.00
```

```
-- Chiens de même sexe que Milou (tatouage 51)
select *
from chien
where sexe = (select sexe
              from chien
              where tatouage = 51)
and tatouage ^= 51 ; -- ^= est équivalent à <> et !=
```

```
--   TATOUAGE NOM          DDN          SEXE          IDA IDR
--   -----
--           666 Cerbère   08-MAR-11 M           999 CA
--           42 Rantanplan 01-MAY-09 M            45
```

Composition de requêtes : les sous-requêtes

Exemples de sous-requêtes synchronisées

Synchronisation de requêtes

Pour chaque ligne d'une requête, une sous-requête est déclenchée.

La **synchronisation** entre les deux requêtes est réalisée par le **passage de valeurs** de la requête englobante à la sous-requête.

```
-- Pour chaque ligne de 'concours c', la valeur 'c.ville' est passée en entrée
-- de la sous-requête qui est exécutée.
-- Le résultat obtenu 'max(dateC)' est ensuite comparé au 'dateC' courant pour
-- conserver la ligne courante (cas de l'égalité entre les valeurs) ou pas.
select ville, dateC dernierConcours
from concours c
where dateC = (select max(dateC)
              from concours
              where ville = c.ville) -- c.-à-d. where concours.ville = c.ville
order by dateC desc, ville ;
```

```
-- VILLE          DERNIERCONCOURS
-- -----
-- Rodez          28-SEP-12
-- Toulouse       05-MAY-13
```


Composition de requêtes : les vues

Vue

L'expression (\triangle pas le résultat !) d'une requête peut être **nommée** afin de :

- 1 décomposer des requêtes complexes,
- 2 réduire la visibilité de certaines données.

```
-- Définition de la vue
create view chienToulousain as
  select * from chien
  where idA in (select idA from adherent where ville = 'Toulouse') ;

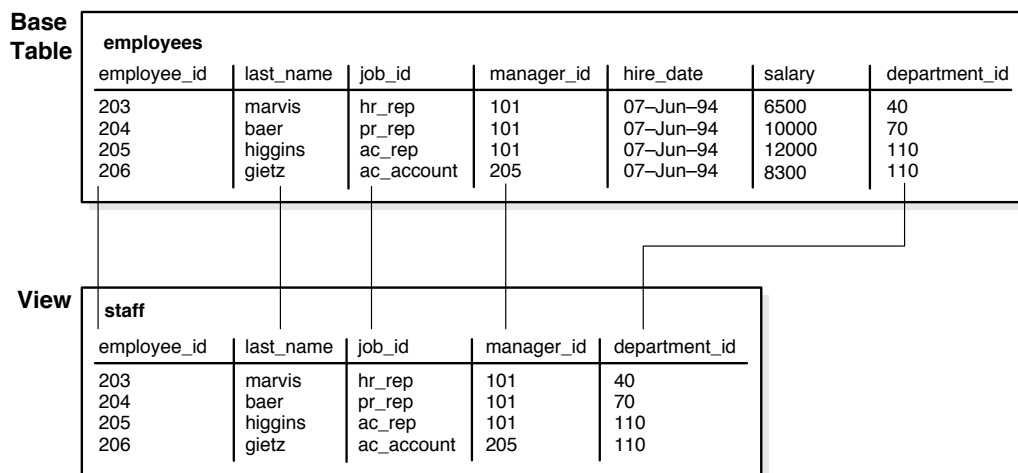
-- Utilisation de la vue : prime et classement des chiens Toulousains
select tatouage, prime, classement
from participation
where tatouage in (select tatouage
                  from chienToulousain)
order by prime desc ;

-- Cette requête est **réécrite** avant d'être exécutée
select tatouage, prime, classement
from participation
where tatouage in (select tatouage -- vue chienToulousain réécrite ci-dessous
                  from (select * from chien
                        where idA in (select idA from adherent where ville = '
                                     Toulouse'))))
order by prime desc ;
```

Composition de requêtes : les vues

Illustration issue de la documentation Oracle

Figure 5-5 An Example of a View



Réduction de la visibilité des données par masquage de hire_date et salary.

Composition de requêtes : les vues « internes »

Vue « interne »

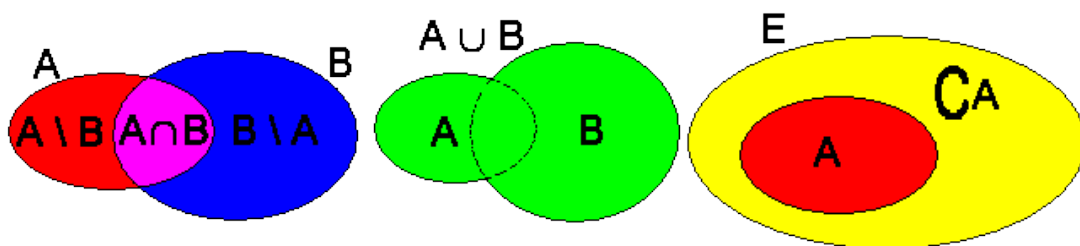
Une vue « interne » crée une vue temporaire le temps d'exécuter la requête. Cette syntaxe permet d'éviter la création de vues dont l'utilité se réduit à une seule requête.

```
-- Prime et classement des chiens Toulousains ?

-- utilisation d'une vue interne qui n'a pas d'existence propre
with chienToulousain as
(
  select *
  from chien
  where idA in (select idA
               from adherent
               where ville = 'Toulouse')
)
select tatouage, prime, classement
from participation
where tatouage in (select tatouage
                  from chienToulousain)
order by prime desc ;
```

Théorie des ensembles de Cantor (XIX^e siècle)

Quelques rappels sur les concepts et les notations. . .



http://www.maths-informatique-jeux.com/math/cours/images/operations_ensembles.png

- A se lit « l'ensemble A »
- $A \cup B$ se lit « A union B »
- $A \cap B$ se lit « A inter B »
- $A \setminus B$ se lit « A moins B »
- $\complement_E A$ se lit « le complément de A dans E »
- $x \in A$ se lit « l'élément x appartient à l'ensemble A »

Opérateurs ensemblistes

Union de deux résultats : $R_1 \cup R_2$

\cup Union

Les résultats des deux opérandes sont **concaténés verticalement**.
Les deux relations doivent être compatibles : même nombre d'attributs de mêmes types.

Commutativité : $A \cup B = B \cup A$

Parallèle avec les maths : $\{1, 6, 4\} \cup \{6, 4, 7\} = \{1, 4, 6, 7\}$ $\{5, 1\} \cup \emptyset = \{5, 1\}$

```
-- Avec l'opérateur 'union' : élimination autom. des doublons dans le résultat
select * from adherent where ville = 'Las Vegas'
union
select * from adherent where ville = 'Troyes' ;

-- Requête équivalente avec l'opérateur 'or'
select distinct *
from adherent
where ville = 'Las Vegas'
       or ville = 'Troyes' ;

-- Requête équivalente avec l'opérateur 'in' (∈ en logique)
select distinct *
from adherent
where ville in ('Las Vegas', 'Troyes') ;
```

Opérateurs ensemblistes

Union de deux résultats **avec conservation des doublons** : $R_1 \uplus R_2$

\uplus Union all

Les résultats des deux opérandes sont **concaténés verticalement** en conservant lignes identiques (doublons).
Les deux relations doivent être compatibles : même nombre d'attributs de mêmes types.

Commutativité : $A \uplus B = B \uplus A$

Parallèle avec les maths : $\{1, 6, 4\} \uplus \{6, 4, 7\} = \{1, 6, 4, 6, 4, 7\}$ $\{5, 1\} \uplus \emptyset = \{5, 1\}$

```
-- Avec l'opérateur 'union all' : doublons conservés dans le résultat
select idR from chien where idA = 999
union all
select idR from chien where idA = 45 ;
-- IDR
-- ---
-- CA
-- CO
-- CA
--

-- Avec l'opérateur 'union' : doublons éliminés dans le résultat :
-- {CA, CO, null}
```

Opérateurs ensemblistes

Intersection de deux résultats : $R_1 \cap R_2$

\cap Intersection

Le résultat contient les **tuples en commun** dans les deux opérandes.
Les deux relations doivent être compatibles : même nombre d'attributs de mêmes types.

Commutativité : $A \cap B = B \cap A$

Parallèle avec les maths : $\{1, 6, 4\} \cap \{6, 4, 7\} = \{4, 6\} = \{6, 4\}$ $\{5, 1\} \cap \emptyset = \emptyset$

```
-- Concours pour lesquels les chiens 1664 et 666 ont pu se croiser ?

-- Avec l'opérateur 'intersect'
select idC from participation where tatouage = 1664
intersect
select idC from participation where tatouage = 666 ;

-- Requête équivalente avec l'opérateur 'group by'
-- (plus de détail dans quelques diapos...)
select idC
from participation
where tatouage in (1664, 666)
group by idC
having count(*) = 2 ;
```

Opérateurs ensemblistes

Différence entre deux résultats : $R_1 \setminus R_2$

\setminus Différence

Le résultat contient les tuples d'une opérande R_1 **qui ne sont pas** dans une autre opérande R_2 . Les deux relations doivent être compatibles : même nombre d'attributs de mêmes types.

Non commutativité : $A \setminus B \neq B \setminus A$

Parallèle avec les maths : $\{1, 6, 4\} \setminus \{6, 4, 7\} = \{1\}$ $\{5, 1\} \setminus \emptyset = \{5, 1\}$

```
-- Participants au concours 12 qui n'ont pas participé au concours 13 ?

-- Avec l'opérateur 'minus' : élimination des doublons dans le résultat
select tatouage from participation where idC = 12
minus
select tatouage from participation where idC = 13 ;

-- Requête équivalente avec l'opérateur 'not in' ( $\notin$ ) et une sous-requête
select tatouage
from participation
where idC = 12
and tatouage not in (select tatouage
from participation
where idC = 13) ;
```

Opérateurs ensemblistes

Existence de tuples \exists

\exists Existence

L'opérateur exists renvoie vrai si la sous-requête contient des lignes, faux sinon.

```
-- Propriétaires de chiens : ces deux requêtes donnent le même résultat
select *
from adherent
where exists (select *
              from chien
              where idA = adherent.idA) ;

select *
from adherent
where idA in (select idA
              from chien
              where idA is not null) ;

-- IDA NOM          PRENOM      VILLE      MAIL
-----
-- 45 Vladuboudin  Tintin      Bruxelles
-- 69 Lovelace     Ada         Marseille  ada@hotmail.fr
-- 999 Satan       Michel      Las Vegas

-- NB : la version avec 'exists' est synchronisée
```

Fonctions d'agrégation

Appliquées à un résultat dans son intégralité

Définition

Une fonction d'agrégation **synthétise** les valeurs d'**une colonne** (en entrée) **en une seule valeur** (en sortie).

Parallèle avec les maths : $3 + 6 = 9$ $\min(3, 4, 5) = 3$

Fonction	Description
<code>count(*)</code>	Nombre total de lignes
<code>count(col)</code>	Nombre de lignes dont la valeur de col n'est pas vide
<code>count(distinct col)</code>	Nombre de valeurs non nulles distinctes dans col
<code>min(col)</code>	Valeur minimale dans col
<code>max(col)</code>	Valeur maximale dans col
<code>sum(col)</code>	Somme des valeurs de col
<code>avg(col)</code>	Moyenne des valeurs de col
<code>stddev(col)</code>	Écart-type des valeurs de col
<code>median(col)</code>	Valeur médiane de col

Fonctions d'agrégation

Appliquées à un résultat dans son intégralité. Quelques exemples...

```
select count(*) nbTot, count(idR) as nbRacés, count(*) - count(idP) nbBâtards,
       count(distinct idR) nbRacesDist
from chien ;
```

```
--      NBTOT      NBRACÉS  NBBÂTARDS  NBRACESDIST
-- ----
--      5          3          2          2
```

```
-- Nombre de chiens qui ont une race (requêtes équivalentes)
select count(idR)      select count(*)
from chien ;          from chien
                       where idR is not null ;
```

```
-- COUNT(IDR)          COUNT(*)
-- ----
--      3              3
```

```
-- Statistiques sur les gains du chien 1664
select count(prime) nbPrimes, sum(prime) argentGagné, avg(prime) primeMoyenne,
       min(classement) meilleurePlace
from participation
where tatouage = 1664 ;
```

```
--      NBPRIMES  ARGENTGAGNÉ  PRIMEMOYENNE  MEILLEUREPLACE
-- ----
--      4          1563.35      564.45        1
```

Fonctions d'agrégation

Appliquées à des regroupements

Opérateur group by ... having

L'opérateur group by définit **comment grouper** des lignes.

L'opérateur having définit **comment filtrer les groupes**.

Les fonctions d'agrégation projetées sont calculées **pour chaque groupe**.

```
select idR, count(*)
from chien
where idR is not null
group by idR
order by 2 desc, 1 ;
```

```
-- IDR      NBCHIENS  NBPROPRIOS
-- ----
-- CA          2          2
-- CO          1          1
```

```
select idR, count(*)
from chien
where idR is not null
group by idR
having count(*) > 1
order by 2 desc, 1 ;
```

```
-- IDR      NBCHIENS  NBPROPRIOS
-- ----
-- CA          2          2
```

Fonctions d'agrégation

Appliquées à des regroupements

Opérateur group by ... having

⚠ Règle absolue : $\{S_1, \dots, S_n\} \subseteq \{G_1, \dots, G_k\} \cup \{\text{count}, \text{min}, \text{max}, \dots\}$

```
select S1, ..., Sn
from R1, ..., Rm
...
group by G1, ..., Gk ;
```

```
-- Nombre de chiens par race et propriétaire ?
-- KO
-- {idR, idA, count}  $\not\subseteq$  {idR, count...}
select idR, idA, count(*) nbCh
from chien
group by idR ;

-- OK
-- {idR, idA, count}  $\subseteq$  {idR, idA, count, ...}
select idR, idA, count(*) nbCh
from chien
group by idR, idA ;

-- #fail SQL Error: ORA-00979:
-- not a GROUP BY expression
```

```
-- Mauvaise utilisation : utiliser un 'group by' au lieu d'un 'distinct'
-- KO
select idR
from chien
group by idR ;

-- OK
select distinct idR
from chien ;
```

Fonctions d'agrégation

Appliquées à des regroupements. Exemples...

```
-- Chiens qui ont participé aux concours sans jamais avoir de prime (losers)
select tatouage
from participation
group by tatouage
having count(prime) = 0 ;
```

```
-- Chiens qui, lorsqu'ils sont classés, ont toujours eu
-- le même classement (mais pas 2e)
select tatouage
from participation
where classement is not null
group by tatouage
having count(distinct classement) = 1
and max(classement) != 2 ;
```

Fonctions d'agrégation

Équivalence de requêtes : admirez la concision de la construction `group by ... having!`

```
-- Requête avec 'group by' ... 'having'
select idR, count(*) nbCh
from chien
where idR is not null
group by idR
having count(*) > 1
order by 2 desc, 1 ;

-- Requête décomposée et réécrite sans 'group by' ... 'having'
-- Elle donne strictement le même résultat
select idR, nbCh
from
(
  -- calcul du nombre de de chiens d'une race donnée -- synchronisation sur t
  select idR, (select count(*) from chien where idR = t.idR) nbCh
  from
  (
    -- Calcul des idR distincts (clause du 'group by')
    select distinct idR
    from chien
    where idR is not null
  ) as t
)
-- traduction de la clause 'having'
where nbCh > 1
order by 2 desc, 1 ;
```

Produit cartésien : $R_1 \times R_2$

\times Produit cartésien

Le résultat d'un produit cartésien entre deux relations R_1 et R_2 contient $|R_1| \times |R_2|$ lignes : ce sont **tous les couples de lignes possibles**, issues de R_1 et R_2 (concaténation horizontale).

```
-- Taille du résultat : |adherent| * |chien| = 4 * 5 = 20 lignes
select * from adherent, chien ;
select * from adherent cross join chien ; -- Syntaxe ANSI pour info (non utilisée en TD/TP)
```

IDA	NOM	PRENOM	VILLE	MAIL	TATOUAGE	NOM	DDN	SEXE	IDA	IDR
45	Vladuboudin	Tintin	Bruxelles		1664	Iench	12-DEC-12	F	69	
45	Vladuboudin	Tintin	Bruxelles		666	Cerbère	08-MAR-11	M	999	CA
45	Vladuboudin	Tintin	Bruxelles		51	Milou	15-FEB-99	M	45	CA
45	Vladuboudin	Tintin	Bruxelles		42	Rantanplan	01-MAY-09	M	45	
45	Vladuboudin	Tintin	Bruxelles		3	Lassie	04-MAY-10	F	999	CO
69	Lovelace	Ada	Marseille	ada@hotmail.fr	1664	Iench	12-DEC-12	F	69	
69	Lovelace	Ada	Marseille	ada@hotmail.fr	666	Cerbère	08-MAR-11	M	999	CA
69	Lovelace	Ada	Marseille	ada@hotmail.fr	51	Milou	15-FEB-99	M	45	CA
69	Lovelace	Ada	Marseille	ada@hotmail.fr	42	Rantanplan	01-MAY-09	M	45	
69	Lovelace	Ada	Marseille	ada@hotmail.fr	3	Lassie	04-MAY-10	F	999	CO
999	Satan	Michel	Las Vegas		1664	Iench	12-DEC-12	F	69	
999	Satan	Michel	Las Vegas		666	Cerbère	08-MAR-11	M	999	CA
999	Satan	Michel	Las Vegas		51	Milou	15-FEB-99	M	45	CA
999	Satan	Michel	Las Vegas		42	Rantanplan	01-MAY-09	M	45	
999	Satan	Michel	Las Vegas		3	Lassie	04-MAY-10	F	999	CO
1	Eastwood	Clint	Troyes	clint@eastwood.com	1664	Iench	12-DEC-12	F	69	
1	Eastwood	Clint	Troyes	clint@eastwood.com	666	Cerbère	08-MAR-11	M	999	CA
1	Eastwood	Clint	Troyes	clint@eastwood.com	51	Milou	15-FEB-99	M	45	CA
1	Eastwood	Clint	Troyes	clint@eastwood.com	42	Rantanplan	01-MAY-09	M	45	
1	Eastwood	Clint	Troyes	clint@eastwood.com	3	Lassie	04-MAY-10	F	999	CO

Erreur de débutant. . .



Confusion fatale entre produit cartésien \times et jointure \bowtie

L'oubli de clauses de jointures conserve des **données invalides** !

```
-- Les propriétaires et leurs chiens ?
-- KO : oubli de la clause de jointure (adherent.idA = chien.idA)
select *
from adherent, chien ;
```

```
----- Il faudrait qu'ils soient égaux !!! -----
|
V
IDA NOM          PRENOM VILLE      MAIL          TATOUAGE NOM      DDN          SEXE  IDA  IDR
-----
[...]
69 Lovelace      Ada   Marseille  ada@hotmail.fr  666 Cerbère     08-MAR-11 M   999 CA
[...]
999 Satan        Michel Las Vegas          51 Milou        15-FEB-99 M   45 CA
[...]
1 Eastwood       Clint Troyes    clint@eastwood.com 666 Cerbère     08-MAR-11 M   999 CA
```

```
-- Ada   NE possède PAS Cerbère !
-- Satan NE possède PAS Milou !
-- Clint NE possède PAS Cerbère !
```

Jointure naturelle : $R_1 \bowtie R_2$

\bowtie Jointure naturelle

Une jointure naturelle est un **produit** cartésien suivi d'une **sélection** portant sur les **attributs en commun** dans les deux relations :

$$R_1 \bowtie R_2 = \sigma_{R_1.att=R_2.att}(R_1 \times R_2)$$

Propriété : $|R_1 \bowtie R_2| \leq |R_1 \times R_2|$

```
select *
from chien, race
where chien.idR = race.idR ;
```

```
-- Avec syntaxe ANSI (à titre informatif : non utilisée en TD/TP)
select *
from chien natural join race ;
```

```

=
|
V
TATOUAGE NOM      DDN          SEXE  IDA  IDR  IDA  IDR  LIBELLE      DESCRIPTION
-----
666 Cerbère     08-MAR-11 M   999 CA  CA  Caniche      Petit et frisé
51 Milou        15-FEB-99 M   45 CA  CA  Caniche      Petit et frisé
3 Lassie        04-MAY-10 F   999 CO  CO  Colley       Poilu et doré
```

```
-- NB : Seuls les chiens possédant une race sont listés
-- ==> Inutile de spécifier 'where race.idR is not null' dans cet exemple
```

Semi-jointures gauche $R_1 \bowtie R_2$ et droite $R_1 \bowtie R_2$ \bowtie Semi-jointure \bowtie

Une semi-jointure gauche \bowtie (resp. droite \bowtie) est une **jointure naturelle** dont on ne **projette** que les attributs provenant de la relation de gauche (resp. droite).

$$R_1 \bowtie R_2 = \pi_{R_1.*} (R_1 \bowtie R_2)$$

$$R_1 \bowtie R_2 = \pi_{R_2.*} (R_1 \bowtie R_2)$$

```
-- Semi-jointure gauche (pour la droite : select race.*)
select chien.*
from chien, race
where chien.idR = race.idR ;
```

```
-- Le résultat ne contient que les colonnes de chien
TATOUAGE NOM      DDN      SEXE      IDA IDR
-----
666 Cerbère      08-MAR-11 M      999 CA
51 Milou        15-FEB-99 M      45 CA
3 Lassie        04-MAY-10 F      999 CO
```

 θ -jointure : $R_1 \bowtie_{\theta} R_2$ θ -jointure

Une θ -jointure exprime une **condition booléenne** de sélection avec les opérateurs binaires $\leq, <, =, \neq, >, \geq$.

```
-- Couples mâle et femelles avec un mâle plus âgé que la femelle ?
select male.tatouage m, femelle.tatouage f
from chien male, chien femelle
where male.ddn > femelle.ddn
      and male.sexe = 'M'
      and femelle.sexe = 'F' ;

--           M           F
-- -----
--           666           3

-- NB : cette requête illustre le cas particulier de l'auto-jointure sur chien.
```

Équi-jointure : $R_1 \bowtie_{R_1.att_i=R_2.att_j} R_2$

$\bowtie_{R_1.att_i=R_2.att_j}$ équi-jointure

L'équi-jointure est une forme particulière de θ -jointure, où θ est l'opérateur d'égalité.

```
-- Propriétaires et leurs chiens ?
select *
from adherent, chien
where adherent.idA = chien.idA ;

-- NB1 : Ce n'est pas une jointure naturelle car les tables adherent et chien
--        ont 2 colonnes en commun: nom et idA.
--        Or, ici on ne joint que sur la colonne idA !
-- Avec syntaxe ANSI : attention, à manipuler avec précaution...
select * from adherent natural join chien ; -- naturelle : KO
select * from adherent join chien using (idA) ; -- theta : OK
```

```
-----(-)-----
|
V
IDA NOM          PRENOM VILLE      MAIL          TATOUAGE NOM          DDN          SEXE  IDA  IDR
-----
45 Vladuboudin  Tintin Bruxelles          51 Milou      15-FEB-99 M      45  CA
45 Vladuboudin  Tintin Bruxelles          42 Rantanplan  01-MAY-09 M      45
69 Lovelace     Ada   Marseille  ada@hotmail.fr  1664 Iench   12-DEC-12 F      69
999 Satan      Michel Las Vegas          3 Lassie     04-MAY-10 F     999  CO
999 Satan      Michel Las Vegas          666 Cerbère   08-MAR-11 M     999  CA

-- NB2 : Client Eastwood n'est pas présent dans le résultat car il ne possède aucun chien.
```

Expressivité de l'équi-jointure

Reprise de l'exemple de la diapositive 16 : « Speed dating pour chiens »

```
-- Speed dating pour chiens :
-- Adhérents disposant d'un mail, possédant une chienne et habitant
-- dans la même ville que les adhérents possédant un chien.
```

```
-- Version imbriquée
select nom, prenom
from adherent
where idA in (select idA --> possesseurs de chiennes
             from chien
             where sexe = 'F')
  and ville in (select ville --> ville des possesseurs de chiens
               from adherent
               where idA in (select idA --> possesseurs de chiens
                             from chien
                             where sexe = 'M'))
  and mail is not null --> possesseurs de chiennes avec email
order by nom, prenom ;
```

```
-- Version relationnelle (avec jointures)
select a1.nom, a1.prenom
from adherent a1, chien c1, adherent a2, chien c2
where a1.idA = c1.idA -- jointure 1/2 : a1 possède c1
  and a2.idA = c2.idA -- jointure 2/2 : a2 possède c2
  and c1.sexe = 'F' -- c1 est une chienne
  and c2.sexe = 'M' -- c2 est un chien
  and a1.ville = a2.ville -- a1 et a2 habitent dans la même ville
  and a1.mail is not null -- a1 possède une adresse mail
order by a1.nom, a1.prenom ;
```

Auto-jointure $R \bowtie_{R.att_i, \theta R.att_j} R$ $\bowtie_{R.att_i, \theta R.att_j}$ auto-jointure

L'auto-jointure est une forme particulière de θ -jointure portant sur **une seule relation**. L'utilisation d'**alias** est nécessaire pour distinguer les relations.

```
-- Couples de chiens (mâle et femelle) de même race ?
select male.tatouage lui, femelle.tatouage elle
from chien male, chien femelle -- auto-jointure avec emploi des alias
where male.idR = femelle.idR -- mâle et femelle sont de même race
      and male.sexe = 'M' -- relation de gauche : mâles
      and femelle.sexe = 'F' ; -- relation de droite : femelles

-- La condition ci-dessous est inutile car partition disjointe sur sexe
-- and male.tatouage <> femelle.tatouage

-- Écriture de la même requête avec le mot clé optionnel 'as'
select male.tatouage as lui, femelle.tatouage as elle -- 'as' : alias de colonne
from chien as male, chien as femelle -- 'as' : alias de table
where male.idR = femelle.idR
      and male.sexe = 'M'
      and femelle.sexe = 'F' ;
```

Jointure externe gauche $R_1 \bowtie R_2$ Jointure externe gauche $R_1 \bowtie R_2$

Résultat de $R_1 \bowtie R_2$ auquel on **ajoute les lignes de R_1** qui n'ont pas été appariées avec R_2 .

```
-- Les propriétaires, avec leurs chiens éventuels ?
select *
from adherent, chien
where adherent.idA = chien.idA (+) ;

-- Avec syntaxe ANSI
select *
from adherent left outer join chien on adherent.idA = chien.idA ;
```

IDA	NOM	PRENOM	VILLE	MAIL	TATOUAGE	NOM	DDN	SEXE	IDA	IDR
1	Eastwood	Clint	Troyes	clint@eastwood.com						
45	Vladuboudin	Tintin	Bruxelles		51	Milou	15-FEB-99	M	45	CA
45	Vladuboudin	Tintin	Bruxelles		42	Rantanplan	01-MAY-09	M	45	
69	Lovelace	Ada	Marseille	ada@hotmail.fr	1664	Iench	12-DEC-12	F	69	
999	Satan	Michel	Las Vegas		3	Lassie	04-MAY-10	F	999	CO
999	Satan	Michel	Las Vegas		666	Cerbère	08-MAR-11	M	999	CA

```
-- Autre utilité : les propriétaires qui n'ont pas de chien ?
select *
from adherent, chien
where adherent.idA = chien.idA (+)
      and chien.idA is null ;
```

Jointure externe droite $R_1 \bowtie_{\infty} R_2$ Jointure externe droite $R_1 \bowtie_{\infty} R_2$

Résultat de $R_1 \bowtie R_2$ auquel on ajoute les lignes de R_2 qui n'ont pas été appariées avec R_1 .

```
-- Les propriétaires, avec leurs chiens éventuels ?
select *
from chien, adherent
where chien.idA (+) = adherent.idA ;

-- Avec syntaxe ANSI
select *
from chien right outer join adherent on chien.idA = adherent.idA ;
```

TATOUAGE	NOM	DDN	SEXE	IDA	IDR	IDA	NOM	PRENOM	VILLE	MAIL
	51 Milou	15-FEB-99	M	45	CA	45	Eastwood	Clint	Troyes	clint@eastwood.com
	42 Rantanplan	01-MAY-09	M	45		45	Vladuboudin	Tintin	Bruxelles	
1664	Iench	12-DEC-12	F	69		69	Lovelace	Ada	Marseille	ada@hotmail.fr
3	Lassie	04-MAY-10	F	999	CO	999	Satan	Michel	Las Vegas	
666	Cerbère	08-MAR-11	M	999	CA	999	Satan	Michel	Las Vegas	

```
-- Autre utilité : les propriétaires qui n'ont pas de chien ?
select *
from chien, adherent
where chien.idA (+) = adherent.idA
and chien.idA is null ;
```

Jointure externe complète $R_1 \bowtie_{\infty} R_2$ Jointure externe complète $R_1 \bowtie_{\infty} R_2$

$$R_1 \bowtie_{\infty} R_2 = (R_1 \bowtie R_2) \cup (R_1 \bowtie_{\infty} R_2)$$

```
select *
from chien, race
where chien.idR (+) = race.idR
union
select *
from chien, race
where chien.idR = race.idR (+)

-- Avec syntaxe ANSI
select *
from chien full outer join race on chien.idR = race.idR ;
```

TATOUAGE	NOM	DDN	SEXE	IDA	IDR	IDR	LIBELLE	DESCRIPTION
	3 Lassie	04-MAY-10	F	999	CO	CO	Colley	Poilu et doré
	42 Rantanplan	01-MAY-09	M			45		
	51 Milou	15-FEB-99	M	45	CA	CA	Caniche	Petit et frisé
	666 Cerbère	08-MAR-11	M	999	CA	CA	Caniche	Petit et frisé
	1664 Iench	12-DEC-12	F	69				
						BA	Berger Allemand	cf. Rintintin

Sources

Les illustrations sont reproduites à partir des documents suivants :

- Oracle Database Documentation Library 11.1
(<http://www.oracle.com/pls/db111/homepage>)
 - Oracle[®] Database : Concepts 11g Release 1 (11.1) B28318-06
- Oracle Database Documentation Library 11.2
(<http://www.oracle.com/pls/db112/homepage>)
 - Oracle[®] Database : Concepts 11g Release 2 (11.2) E40540-01