INFORMATIQUE Développement d'applications
BLOC 1
UE05 Bases de données

Chapitre 3.3 : SQL - DML (avancé)

Vincent Reip

Janvier 2023

Objectif

- Au terme de ce chapitre, l'étudiant sera capable de :
  - Manipuler les données au sein d'un schéma relationnel en utilisant des fonctions avancées

Fonctions de calcul et groupements

- Le langage SQL offre des fonctions prédéfinies qui peuvent retourner une valeur calculée et/ou agrégée pour les tuples faisant partie de la sélection :
  - COUNT(*) : compte le nombre de tuples trouvés
    - Combien de clients sont enregistrés dans ma base de données ?
  - COUNT(nom-attr) : compte le nombre de valeurs pour un attribut
    - Combien de clients ont une adresse email connue dans ma base de données ?
  - AVG(nom-attr) : calcule la moyenne des valeurs pour un attribut
    - Quelle est le montant moyen d'une commande ?
  - SUM(nom-attr) : calcule la somme des valeurs pour un attribut
    - Quel est mon chiffre d'affaire total ?
  - MIN(nom-attr) : détermine la valeur minimale pour un attribut
    - Quel est la cote minimale obtenue par une étudiant ?
  - MAX(nom-attr) : détermine la valeur maximale pour un attribut
    - Quel est le plus gros montant facturé ?

Fonctions de calcul et groupements

- Ces fonctions peuvent aussi être utilisées conjointement avec une clause de groupement
  - Elles produiront alors une valeur pour chacun des groupes.
  - Les fonctions de calcul sont alors aussi appelées fonction de groupe ou d'agrégation.

    - « Regroupe-moi les étudiants par cursus et donne-moi le nombre moyen de crédits réussis en B1 pour chacun des cursus »

CURSUS	NB_MOYEN_CREDITS_REUSSIS
IINFORMATIQUE	23
MARKETING	26
ROBOTIQUE	24,5

    - « Regroupe-moi les clients par code postal et donne-moi le chiffre d'affaire engrangé pour chacune des communes »

CODE_POSTAL	CHIFFRE_AFFAIRE
4000	23.500 €
4800	17.650 €
5000	35.450 €
4020	9.865 €

COUNT(*)

```
SELECT COUNT(*)
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
```

- Requête retournant le nombre de tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - S'il n'y a pas de clause WHERE, la fonction retourne le nombre de tuples présents dans la relation
- Retourne une et une seule valeur entière positive
  - 0 si aucun tuple n'est sélectionné ou si la relation est vide

COUNT(*)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT COUNT(*)
FROM vin
```

COUNT(*)
5

```
SELECT COUNT(*)
FROM vin
WHERE cru = "VOLNAY"
```

COUNT(*)
2

```
SELECT COUNT(*)
FROM vin
WHERE cru = "MERLOT"
```

COUNT(*)
0

COUNT(nom-attribut)

```
SELECT COUNT([ALL | DISTINCT] nom-attribut)
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
```

- Requête retournant le nombre de valeurs d'une colonne pour les tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - On ne compte pas l'absence de valeur (null)
- Retourne une et une seule valeur entière positive
  - 0 si aucun tuple n'est sélectionné, si l'attribut n'a pas de valeur pour les tuples sélectionnés ou si la relation est vide
- L'utilisation du mot-clé DISTINCT permet de ne compter que les valeurs distinctes (valeur par défaut = ALL).

COUNT(nom-attribut)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	NULL	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT COUNT(*)
FROM vin
```

COUNT(*)
5

```
SELECT COUNT(millesime)
FROM vin
```

COUNT(millesime)
5

```
SELECT COUNT(qualite)
FROM vin
```

COUNT(qualite)
4

```
SELECT COUNT(DISTINCT millesime)
FROM vin
```

COUNT(DISTINCT millesime)
4

```
SELECT COUNT(qualite)
FROM vin
WHERE cru LIKE "V%"
```

COUNT(qualite)
2

AVG(nom-attribut)

```
SELECT AVG(nom_attr)
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
```

- Requête retournant la moyenne arithmétique des valeurs d'une colonne pour les tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - L'attribut doit être numérique
  - On ne tient pas compte de l'absence de valeur (null)
- Retourne une et une seule valeur ou null
  - null si aucun tuple n'est sélectionné, si l'attribut n'a pas de valeur pour les tuples sélectionnés ou si la relation est vide

AVG(nom-attribut)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	NULL	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT AVG(degre)
FROM vin
```

AVG(degre)
0.127

```
SELECT AVG(degre)
FROM vin
WHERE qualite = 'A'
```

AVG(degre)
0.13

```
SELECT AVG(degre)
FROM vin
WHERE qualite = 'G'
```

AVG(degre)
null

AVG(nom-attribut)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.10
2	VOLNAY	1979	Bourgogne	B	0.13
3	CHENAS	1983	Beaujolais	A	NULL
4	VILLAGE	1998	Beaujolais Sud	B	0.13
5	JULIENAS	1986	Beaujolais	C	NULL

```
SELECT AVG(degre)
FROM vin
```

L'absence de valeur (NULL) n'est pas comptabilisée comme 0 dans le calcul de la moyenne arithmétique.

AVG(degre)
0.12

SUM(nom-attribut)

```
SELECT SUM(nom_attr)
FROM nom_relation
[JOIN …]
[WHERE expression_conditionnelle]
```

- Requête retournant la somme des valeurs d'une colonne pour les tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - L'attribut doit être numérique
  - On ne tient pas compte de l'absence de valeur (null)
- Retourne une et une seule valeur ou null
  - null si aucun tuple n'est sélectionné, si l'attribut n'a pas de valeur pour les tuples sélectionnés ou si la relation est vide

SUM(nom-attribut)

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	NULL	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT SUM(degre)
FROM vin
```

SUM(degre)
0.635

```
SELECT SUM(degre)
FROM vin
WHERE qualite = 'A'
```

SUM(degre)
0.26

```
SELECT SUM(degre)
FROM vin
WHERE qualite = 'G'
```

SUM(degre)
null

MIN(nom-attribut)

```
SELECT MIN(nom_attr)
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
```

- Requête retournant la valeur minimale d'une colonne pour les tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - L'attribut ne doit pas nécessairement être numérique
  - On ne tient pas compte de l'absence de valeur (null)
- Retourne une et une seule valeur ou null
  - null si aucun tuple n'est sélectionné, si l'attribut n'a pas de valeur pour les tuples sélectionnés ou si la relation est vide

MIN(nom-attribut)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	NULL	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT MIN(degre)
FROM vin
```

MIN(degre)
0.115

```
SELECT MIN(qualite)
FROM vin
```

MIN(qualite)
A

Que signifie être le minimum pour une chaîne de caractères ?

On se base sur l'ordre lexicographique (équivalent à l'ordre des mots dans un dictionnaire) mais en comparant les caractères selon leur valeur liée à leur table d'encodage (ASCII, Unicode…)

Qu'en est-il pour les dates ?

MAX(nom-attribut)

```
SELECT MAX(nom_attr)
FROM nom relation
[JOIN … ]
[WHERE expression_conditionnelle]
```

- Requête retournant la valeur maximale d'une colonne pour les tuples correspondants aux critères de sélection exprimés dans la clause WHERE
  - L'attribut ne doit pas nécessairement être numérique
  - On ne tient pas compte de l'absence de valeur (null)
- Retourne une et une seule valeur ou null
  - null si aucun tuple n'est sélectionné, si l'attribut n'a pas de valeur pour les tuples sélectionnés ou si la relation est vide

MAX(nom-attribut)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	NULL	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT MAX(degre)
FROM vin
```

MAX(degre)
0.14

```
SELECT MAX(qualite)
FROM vin
```

MAX(qualite)
C

Opérateur de groupement

```
SELECT attr_groupe , fonction_groupe
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
GROUP BY attr groupe
[ORDER BY … ]
```

- La clause GROUP BY permet d'indiquer sur quels attributs doivent s'effectuer les groupements
  - les tuples ayant des valeurs identiques pour les attributs mentionnés appartiendront au même groupe
- Les fonctions de groupe seront appliquées à chacun des groupes

Opérateur de groupement

- On ne peut trouver dans la clause SELECT que des éléments déterminant une seule valeur par groupe
  - Attributs faisant partie du groupement, fonctions de groupe ou constantes

```
SELECT code_postal, nom, AVG(salaire)
FROM EMPLOYE
GROUP BY codePostal
```

➔ ORA-00979:
not a GROUP BY expression

- Le groupement est effectué après la sélection
  - On ne peut donc utiliser une fonction de groupe dans une clause WHERE

```
SELECT code_postal
FROM EMPLOYE
WHERE MAX(salaire) > 25000
GROUP BY code_postal
```

➔ ORA-00934:
group function is not allowed here

GROUP BY

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115
6	GIGONDAS	1986	Côte du Rhône	D	0.120
7	JULIENAS	1990	Beaujolais	A	0,115

GROUP BY qualite :
4 valeurs différentes pour l'attribut « qualite »
➔ il y aura donc 4 groupes
Pour chacun des groupes de tuples, on applique la fonction de groupe MIN

```
SELECT MIN(degre) AS Minimum
FROM vin
GROUP BY qualite
```

Minimum
0.115
0.135
0.120
0.115

GROUP BY (1/4)

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1978	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115
6	CHENAS	1995	Beaujolais	B	0.12
7	JULIENAS	1994	Beaujolais	A	0.125
8	CHENAS	1996	Beaujolais	A	0.13

```
SELECT cru, MIN(degre) AS Minimum
FROM vin
WHERE millesime > 1980
GROUP BY cru
```

GROUP BY (2/4)

1. On applique la clause WHERE sur l'ensemble des tuples

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
~~2~~	~~VOLNAY~~	~~1979~~	~~Bourgogne~~	~~B~~	~~0.135~~
3	CHENAS	1983	Beaujolais	A	0.14
~~4~~	~~VILLAGE~~	~~1978~~	~~Beaujolais Sud~~	~~D~~	~~0.125~~
5	JULIENAS	1986	Beaujolais	C	0.115
6	CHENAS	1995	Beaujolais	B	0.12
7	JULIENAS	1994	Beaujolais	A	0.125
8	CHENAS	1996	Beaujolais	A	0.13

```
SELECT cru, MIN(degre) AS Minimum
FROM vin
WHERE millesime > 1980
GROUP BY cru
```

GROUP BY (3/4)

2. On effectue les groupements

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12

idVin	cru	millesime	region	qualite	degre
3	CHENAS	1983	Beaujolais	A	0.14
6	CHENAS	1995	Beaujolais	B	0.12
8	CHENAS	1996	Beaujolais	A	0.13

idVin	cru	millesime	region	qualite	degre
5	JULIENAS	1986	Beaujolais	C	0.115
7	JULIENAS	1994	Beaujolais	A	0.125

```
SELECT cru, MIN(degre) AS Minimum
FROM vin
WHERE millesime > 1980
GROUP BY cru
```

GROUP BY (4/4)

3. Pour chaque groupement, on applique la clause SELECT

a) Trouver dans chaque groupement les tuples ayant la valeur minimum pour l'attribut 'degre'

b) Appliquer la projection sur ces tuples

cru	Minimum
VOLNAY	0.12
CHENAS	0.12
JULIENAS	0.115

```
SELECT cru, MIN(degre) AS Minimum
FROM vin
WHERE millesime > 1980
GROUP BY cru
```

GROUP BY + ORDER BY

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT qualite, MIN(degre) AS Minimum
FROM vin
GROUP BY qualite
ORDER BY qualite
```

qualite	Minimum
A	0.12
B	0.135
C	0.115
D	0.125

EXERCICES

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

1. Afficher le degré moyen des vins dont le nom de cru commence par la lettre ‘V’
2. Afficher, pour chaque région et nom de cru, le millésime le plus ancien.

EXERCICES

```
SELECT AVG(degre)
FROM vin
WHERE cru LIKE 'V%
```

AVG(degre)
0.1266667

```
SELECT cru, region,
       MIN(millesime) AS annee_min
FROM vin
GROUP BY cru, region
```

cru	region	annee_min
CHENAS	Beaujolais	1983
JULIENAS	Beaujolais	1986
VILLAGE	Beaujolais Sud	1998
VOLNAY	Bourgogne	1979

EXERCICES

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

1. Afficher le total des ventes pour chaque magasin
2. Afficher le nombre de ventes pour chaque montant

EXERCICES

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

```
SELECT magasin, SUM(montant) AS total
FROM vente
GROUP BY magasin
```

magasin	total
Los Angeles	1800
San Diego	550
Boston	1500

```
SELECT montant, COUNT(*) AS nombre
FROM vente
GROUP BY montant
```

montant	nombre
1500	1
250	1
800	1
300	2
700	1

Sélection de groupes

```
SELECT attr_groupe , fonction_groupe
FROM nom_relation
[JOIN … ]
[WHERE expression_conditionnelle]
GROUP BY attr_groupe
HAVING expression_conditionnelle_sur_groupe
[ORDER BY … ]
```

- La clause HAVING permet de sélectionner les groupes qui feront partie du résultat
  - la condition de sélection peut porter sur les éléments de la clause SELECT mais aussi sur d'autres fonctions de groupe calculable sur chacun des groupes

Pas de HAVING sans GROUP BY
Pas de GROUP BY sans fonction de groupe

HAVING

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT qualite, MIN(degre) AS Minimum
FROM vin
GROUP BY qualite
HAVING MIN(degre) > 0.12
```

qualite	Minimum
A	0.12
B	0.135
C	0.115
D	0.125

qualite	Minimum
B	0.135
D	0.125

HAVING

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT qualite, MIN(degre) AS Minimum
FROM vin
GROUP BY qualite
HAVING MAX(degre) >= 0.14
```

qualite	Minimum
A	0.12
B	0.135
C	0.115
D	0.125

qualite	Minimum
A	0.12

HAVING

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT qualite, MIN(degre) AS Minimum
FROM vin
GROUP BY qualite
HAVING qualite IN ('A', 'B', 'C')
```

qualite	Minimum
A	0.12
B	0.135
C	0.115
D	0.125

qualite	Minimum
A	0.12
B	0.135
C	0.125

Une condition ne faisant pas intervenir de fonction de groupe doit être placée dans la clause WHERE (même si les SGBD sont tolérants)

EXERCICES

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20
7	Boston	300	1999-07-22

1. Afficher le total des ventes par magasin, pour autant que ce total atteigne 1500$
2. Pour chaque magasin, afficher le montant minimum d'une vente pour autant que le nombre de ventes dans ce magasin soit supérieur à 2

EXERCICES

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20
7	Boston	300	1999-07-22

```
SELECT magasin, SUM(montant) AS total
FROM vente
GROUP BY magasin
HAVING SUM(montant) >= 1500
```

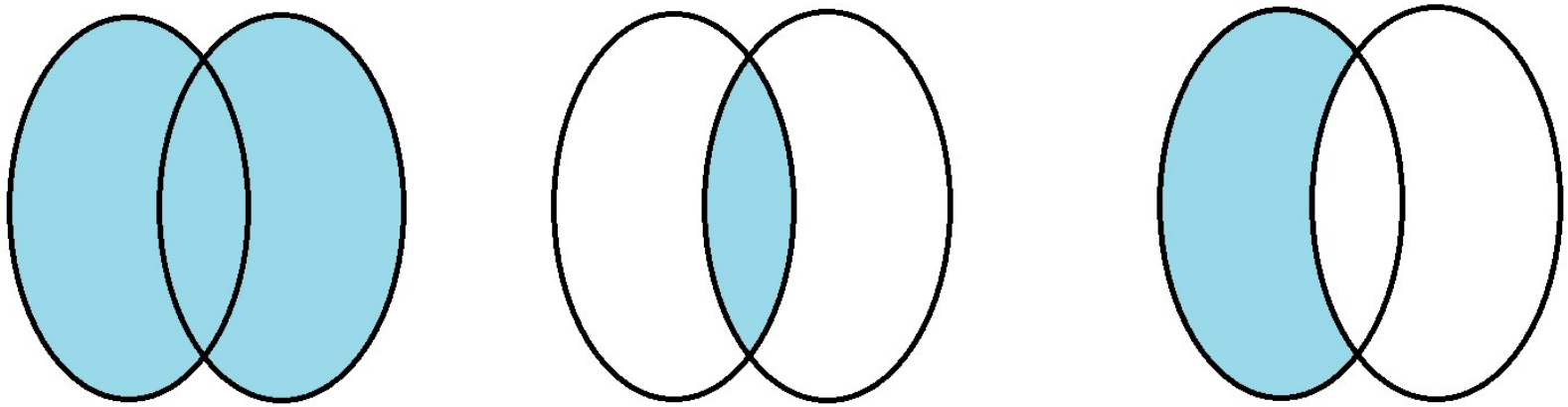
magasin	total
Los Angeles	1800
Boston	1800

```
SELECT magasin, MIN(montant) AS minimum
FROM vente
GROUP BY magasin
HAVING COUNT(*) > 2
```

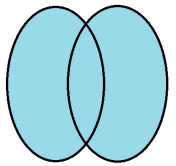
magasin	minimum
Boston	300

Opérateurs ensemblistes

- Les opérateurs ensemblistes permettent de combiner (« verticalement ») les résultats de plusieurs requêtes de type ‘SELECT’
  - Les requêtes peuvent porter sur des tables différentes
  - Les résultats des requêtes doivent être de schéma compatible



UNION



```
Requête_SELECT
UNION
Requête_SELECT
```

- L'union permet de réunir les tuples issus de deux requêtes portant éventuellement sur des tables différentes
  - Les éventuels doublons seront supprimés
    - UNION ALL permet de garder les doublons
  - Les colonnes retournées par les requêtes doivent être en nombre identique et de même type

UNION

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

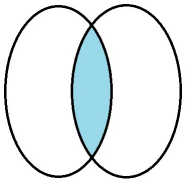
vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT montant, date
FROM vente

UNION

SELECT total, date
FROM vente_par_internet
```

montant	date
1500	1999-06-01
250	1999-06-05
800	1999-10-02
300	1999-12-10
300	1999-04-02
700	1999-06-20
250	1999-06-01
535	1999-01-10
320	1999-04-02
750	1999-01-12

INTERSECTION



```
Requête_SELECT
INTERSECT
Requête_SELECT
```

- L'intersection permet de retourner les tuples identiques issus de deux requêtes portant éventuellement sur des tables différentes
  - Les éventuels doublons seront supprimés
  - Les colonnes retournées par les requêtes doivent être en nombre identique et de même type
- L'intersection n'est pas une opération primitive : elle peut être obtenue grâce à une jointure
  - Elle n'est d'ailleurs pas proposée par tous les SGBD

```
SELECT DISTINCT e.nom, e.prenom
FROM ETUDIANT e
JOIN PROFESSEUR p ON e.nom = p.nom AND e.prenom = p.prenom
```

INTERSECTION

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

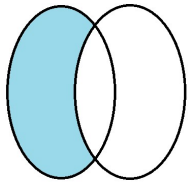
idVPI	date	total
1	1999-06-01	1500
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT montant, date
FROM vente

INTERSECT

SELECT total, date
FROM vente_par_internet
```

Dates auxquelles il y a eu une vente par internet et une vente en magasin de même montant

montant	date
1500	1999-06-01

DIFFERENCE



```
Requête_SELECT
MINUS (ou EXCEPT)
Requête_SELECT
```

- La différence permet de retourner les tuples issus de la première requête qui ne sont pas présents dans la deuxième requête. Les requêtes peuvent éventuellement porter sur des tables différentes
  - Les éventuels doublons seront supprimés
  - Les colonnes retournées par les requêtes doivent être en nombre identique et de même type
- L'intersection n'est pas une opération primitive : elle peut être obtenue grâce au prédicat NOT IN associé à une sous-requête
  - Elle n'est d'ailleurs pas proposée par tous les SGBD
- ORACLE, MySQL, : MINUS – PostgreSQL, SQLServer : EXCEPT

DIFFERENCE

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT date
FROM vente_par_internet
MINUS
SELECT date
FROM vente
```

Dates auxquelles seulement une vente par internet a eu lieu

date
1999-01-10
1999-01-12

Opérateur ensembliste + ORDER BY

- Il est possible de trier le résultat en appliquant une clause ORDER BY
  - En fin de requête
  - En spécifiant des attributs de la première requête

```
SELECT nom, prenom
FROM ETUDIANT

INTERSECT

SELECT name, firstname
FROM TEACHER

ORDER BY nom
```

```
SELECT nom, prenom
FROM ETUDIANT
ORDER BY nom ASC

INTERSECT

SELECT name, firstname
FROM TEACHER
```

ORA-00933: SQL command not properly ended

```
SELECT nom, prenom
FROM ETUDIANT

UNION

SELECT name, firstname
FROM TEACHER

ORDER BY name ASC
```

ORA-00904: "Name": invalid identifier

Requêtes de même schéma

- Pour pouvoir appliquer un opérateur ensembliste, il faut que les ensembles de valeurs (résultat des requêtes) soient « compatibles »
  - Même nombre de colonnes
  - Types « compatibles » - des conversions implicites ou explicites peuvent être faites

```
SELECT nom, prenom
FROM ETUDIANT

UNION

SELECT name, firstname
FROM TEACHER
```

```
SELECT nom, prenom, gsm
FROM ETUDIANT

UNION

SELECT name, firstname
FROM TEACHER
```

ORA-01789: query block has incorrect number of result columns

```
SELECT nom, date_de_naissance
FROM ETUDIANT

UNION

SELECT name, firstname
FROM TEACHER
```

ORA-01790: expression must have same datatype as corresponding expression

Requêtes de même schéma

```
SELECT nom, prenom
FROM ETUDIANT

UNION

SELECT name, 'Jules'
FROM TEACHER
```

```
SELECT nom, prenom
FROM ETUDIANT

UNION

SELECT name, id_teacher
FROM TEACHER
```

ORA-01790: expression must have same datatype as corresponding expression

```
SELECT nom, prenom
FROM ETUDIANT

UNION

SELECT name, TO_CHAR(id_teacher)
FROM TEACHER
```

Sous-requêtes

- Principe : exploiter le résultat d'une sous-requête dans la clause WHERE de la requête principale.
- La sous-requête peut faire intervenir d'autres tables, des jointures, d'autres sous-requêtes, etc…
- On distingue 3 types de relations utilisées avec des sous-requêtes :
    - relation de comparaison : =, <, >, <=, >= (ANY et ALL)
    - relation d'appartenance : IN, NOT IN
    - relation d'existence : EXISTS, NOT EXISTS

Sous-requêtes : relation de comparaison unique

```
SELECT nom_attr1, nom_attr2, …
FROM nom_relation
[JOIN … ]
WHERE nom_attr (=|<|>|<=|>=) sous_requête
[…..]
```

• On compare la valeur d'un attribut de la requête principale avec le résultat de la sous-requête

- Le résultat de la sous-requête doit être composés d'une seule valeur (1 ligne, 1 colonne) dont le type est « comparable » avec l'attribut comparé

Sous-requêtes : relation de comparaison unique

etudiant

idEtudiant	Nom	Prenom	Localite
1	Dupont	Luc	Verviers
2	Durand	Noémie	Liège
3	Frion	Stéphane	Verviers
4	Marcoli	Diana	Spa
5	Préant	Marc	Arlon
6	El Madri	Dris	Liège

enseignant

idEnseignant	Nom	Localite
1	Altenberg	Verviers
2	Pirard	Herve
3	Pirard	Spa
4	Paarisel	Namur

La sous-requête renvoie une seule valeur (Verviers) avec certitude puisque la clause WHERE exploite la clé primaire idEnseignant qui par définition est unique.

```
SELECT e.nom, e.prenom
FROM etudiant e
WHERE e.localite = (SELECT localite
                    FROM enseignant
                    WHERE idEnseignant = 1)
```

Nom	Prenom
Dupont	Luc
Frion	Stéphane

Sous-requêtes : relation de comparaison unique

etudiant

idEtudiant	Nom	Prenom	Localite
1	Dupont	Luc	Verviers
2	Durand	Noémie	Liège
3	Frion	Stéphane	Verviers
4	Marcoli	Diana	Spa
5	Préant	Marc	Arlon
6	El Madri	Dris	Liège

enseignant

idEnseignant	Nom	Localite
1	Altenberg	Verviers
2	Pirard	Herve
3	Pirard	Spa
4	Paarisel	Namur

```
SELECT e.nom, e.prenom
FROM etudiant e
WHERE e.localite = (SELECT localite
                    FROM enseignant
                    WHERE nom = 'Pirard')
```

ORA-01427: single-row subquery returns more than one row

Sous-requêtes : relation de comparaison unique

etudiant

idEtudiant	Nom	Prenom	Localite
1	Dupont	Luc	Verviers
2	Durand	Noémie	Liège
3	Frion	Stéphane	Verviers
4	Marcoli	Diana	Spa
5	Préant	Marc	Arlon
6	El Madri	Dris	Liège

enseignant

idEnseignant	Nom	Localite
1	Altenberg	Verviers
2	Pirard	Herve
3	Pirard	Spa
4	Paarisel	Namur

```
SELECT e.nom, e.prenom
FROM etudiant e
WHERE e.localite = (SELECT localite, nom
                    FROM enseignant
                    WHERE idEnseignant = 1)
```

ORA-00913: too many values

Sous-requêtes : relation de comparaison multiple

```
SELECT nom_attr1, nom_attr2, …
FROM nom_relation
[JOIN … ]
WHERE nom_attr (=|<|>|<=|>=) (ANY|ALL) sous_requête
[…..]
```

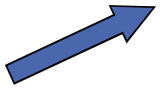
- On compare la valeur d'un attribut de la requête principale avec tous les tuples issus du résultat de la sous-requête
  - ANY : tuple sélectionné si la condition de comparaison est vérifiée pour au moins un tuple de la sous-requête
  - ALL : tuple sélectionné si la condition de comparaison est vérifiée pour tous les tuples de la sous-requête
- Les tuples issus de la sous-requête doivent être composés d'un seul attribut dont le type est « comparable » avec l'attribut comparé

Sous-requêtes : relation de comparaison multiple

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT *
FROM vin
WHERE qualite = ANY (SELECT qualite
                     FROM vin
                     WHERE region = 'Beaujolais')
```



qualite
A
C

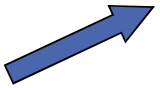
idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
3	CHENAS	1983	Beaujolais	A	0.14
5	JULIENAS	1986	Beaujolais	C	0.115

Sous-requêtes : relation de comparaison multiple

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT *
FROM vin
WHERE millesime < ANY (SELECT millesime
                       FROM vin
                       WHERE region LIKE 'Beaujolais%')
```



millesime
1983
1998
1986

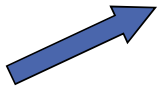
idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
5	JULIENAS	1986	Beaujolais	C	0.115

Sous-requêtes : relation de comparaison multiple

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT *
FROM vin
WHERE degre > ALL (SELECT degre
                   FROM vin
                   WHERE cru = 'VOLNAY')
```



degre
0.12
0.135

idVin	cru	millesime	region	qualite	degre
3	CHENAS	1983	Beaujolais	A	0.14

Sous-requêtes : relation de comparaison multiple

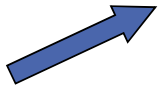
vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT *
FROM vente
WHERE montant > ALL (SELECT total
                     FROM vente_par_internet)
```



total
250
535
320
750

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
3	Boston	800	1999-10-02

Sous-requêtes : relation d'appartenance

```
SELECT nom_attr1, nom_attr2, …
FROM nom_relation
[JOIN … ]
WHERE nom_attr [NOT] IN sous_requête
[…..]
```

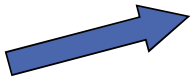
- On vérifie que la valeur d'un attribut de la requête principale est présente (ou non) dans le résultat de la sous-requête
  - Similaire à la forme = ANY (<> ALL)
- Les tuples issus de la sous-requête doivent être composés d'un seul attribut dont le type est « comparable » avec l'attribut comparé

Sous-requêtes : relation d'appartenance

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT *
FROM vin
WHERE region IN (SELECT region
                 FROM vin
                 WHERE cru LIKE 'V%')
```



region
Bourgogne
Bourgogne
Beaujolais Sud

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
4	VILLAGE	1998	Beaujolais Sud	D	0.125

Sous-requêtes : relation d'appartenance

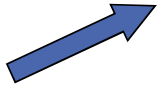
vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT *
FROM vente
WHERE montant IN (SELECT total
                  FROM vente_par_internet)
```



total
250
535
320
750

idVente	magasin	montant	date
2	San Diego	250	1999-06-05

Sous-requêtes : relation d'existence

```
SELECT nom_attr1, nom_attr2, …
FROM nom_relation
[JOIN … ]
WHERE [NOT] EXISTS sous_requête
[…..]
```

- Un tuple de la requête de base sera sélectionné si la sous-requête retourne au moins un tuple [EXISTS] ou si la sous-requête ne retourne aucun tuple [NOT EXISTS]
  - La valeur et le schéma des tuples retournés n'a pas d'importance
- Le résultat de la sous-requête doit idéalement varier pour chaque tuple de la requête de base
  - la sous-requête devrait donc utiliser dans sa clause WHERE un ou plusieurs attributs de la requête de base

Sous-requêtes : relation d'existence

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT v.magasin, v.montant, v.date
FROM vente v
WHERE EXISTS (SELECT *
              FROM vente_par_internet vi
              WHERE v.date = vi.date)
```

La sous-requête sera exécutée pour chaque tuple de la requête principale (donc pour chacun des tuples de la table vente). Le résultat de cette sous-requête sera potentiellement différent à chaque itération vu la présence de v.date dans la clause WHERE.

magasin	montant	date
Los Angeles	1500	1999-06-01
Los Angeles	300	1999-04-02

Magasins, montants et dates de vente pour lesquelles il y a eu, le même jour, une vente par internet.

Sous-requêtes : relation d'existence

etudiant

idEtudiant	Nom	Prenom	Localite
1	Dupont	Luc	Verviers
2	Durand	Noémie	Liège
3	Frion	Stéphane	Verviers
4	Marcoli	Diana	Spa
5	Préant	Marc	Arlon
6	El Madri	Dris	Liège

enseignant

idEnseignant	Nom	Localite
1	Altenberg	Verviers
2	Pirard	Herve
3	Pirard	Spa
4	Paarisel	Namur

```
SELECT et.nom, et.prenom
FROM etudiant et
WHERE EXISTS (SELECT *
              FROM enseignant en
               WHERE en.localite = 'Verviers')
```

Quel résultat ?

Sous-requêtes : relation d'existence

etudiant

idEtudiant	Nom	Prenom	Localite
1	Dupont	Luc	Verviers
2	Durand	Noémie	Liège
3	Frion	Stéphane	Verviers
4	Marcoli	Diana	Spa
5	Préant	Marc	Arlon
6	El Madri	Dris	Liège

enseignant

idEnseignant	Nom	Localite
1	Altenberg	Verviers
2	Pirard	Herve
3	Pirard	Spa
4	Paarisel	Namur

```
SELECT et.nom, et.prenom
FROM etudiant et
WHERE EXISTS (SELECT *
              FROM enseignant en
               WHERE en.localite = et.localite )
```

Quel résultat ?

Sous-requêtes : relation d'existence

vente

idVente	magasin	montant	date
1	Los Angeles	1500	1999-06-01
2	San Diego	250	1999-06-05
3	Boston	800	1999-10-02
4	San Diego	300	1999-12-10
5	Los Angeles	300	1999-04-02
6	Boston	700	1999-06-20

vente_par_internet

idVPI	date	total
1	1999-06-01	250
2	1999-01-10	535
3	1999-04-02	320
4	1999-01-12	750

```
SELECT v.magasin, v.montant, v.date
FROM vente v
WHERE NOT EXISTS (SELECT *
                  FROM vente_par_internet vi
                    WHERE v.date = vi.date AND vi.total > v.montant )
```

magasin	montant	date
Los Angeles	1500	1999-06-01
San Diego	250	1999-06-05
Boston	800	1999-10-02
San Diego	300	1999-12-10
Boston	700	1999-06-20

Magasins, montants et dates pour lesquels il n'y a pas eu, le même jour, de vente par internet pour un montant supérieur.

CASE…WHEN…END

```
SELECT nom_attr1, nom_attr2, … ,
        CASE expression
            WHEN valeur THEN resultat
            WHEN valeur THEN resultat
             ….
            [ELSE resultatElse]
        END
FROM nom_relation
[JOIN … ]
[WHERE …]
[…..]
```

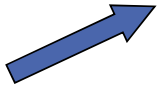
- Permet d'évaluer une expression et de fournir un résultat en fonction de la valeur de cette expression
- L'instruction CASE…WHEN…END se positionne 'comme un attribut' dans la requête
  - Séparé par des virgules
  - Produira sont résultat dans une colonne

CASE…WHEN…END

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT cru, millesime,
        CASE qualite
          WHEN 'A' THEN 'Très bon vin'
          WHEN 'B' THEN 'Bon vin'
          WHEN 'C' THEN 'Bof…'
          WHEN 'D' THEN 'A éviter'
          ELSE 'Qualité inconnue'
        END AS appreciation
FROM vin
```



cru	millesime	appreciation
VOLNAY	1983	Très bon vin
VOLNAY	1979	Bon vin
CHENAS	1983	Très bon vin
VILLAGE	1998	A éviter
JULIENAS	1986	Bof…

CASE…WHEN…END

```
SELECT nom_attr1, nom_attr2, … ,
            CASE
                  WHEN expression THEN resultat
                  WHEN expression THEN resultat
                   ….
                 [ELSE resultatElse]
             END
FROM nom_relation
[JOIN … ]
[WHERE …]
[…..]
```

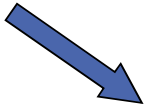
- L'expression a évaluer peut être placée au niveau de chaque clause WHEN
    - Offre plus de combinaisons différentes grâce aux opérateurs logiques et de comparaison (AND, OR, =, <,>, LIKE,…)

CASE…WHEN…END

vin

idVin	cru	millesime	region	qualite	degre
1	VOLNAY	1983	Bourgogne	A	0.12
2	VOLNAY	1979	Bourgogne	B	0.135
3	CHENAS	1983	Beaujolais	A	0.14
4	VILLAGE	1998	Beaujolais Sud	D	0.125
5	JULIENAS	1986	Beaujolais	C	0.115

```
SELECT cru, millesime,
        CASE
          WHEN qualite = 'A' OR qualite = 'B' THEN 'A acheter'
          WHEN qualite = 'C' AND degre < 0.12 THEN 'A goûter'
          ELSE 'Ne pas acheter'
        END AS conseil
FROM vin
```



cru	millesime	conseil
VOLNAY	1983	A acheter
VOLNAY	1979	A acheter
CHENAS	1983	A acheter
VILLAGE	1998	Ne pas acheter
JULIENAS	1986	A goûter