

BASES DE DONNÉES AVANCÉES

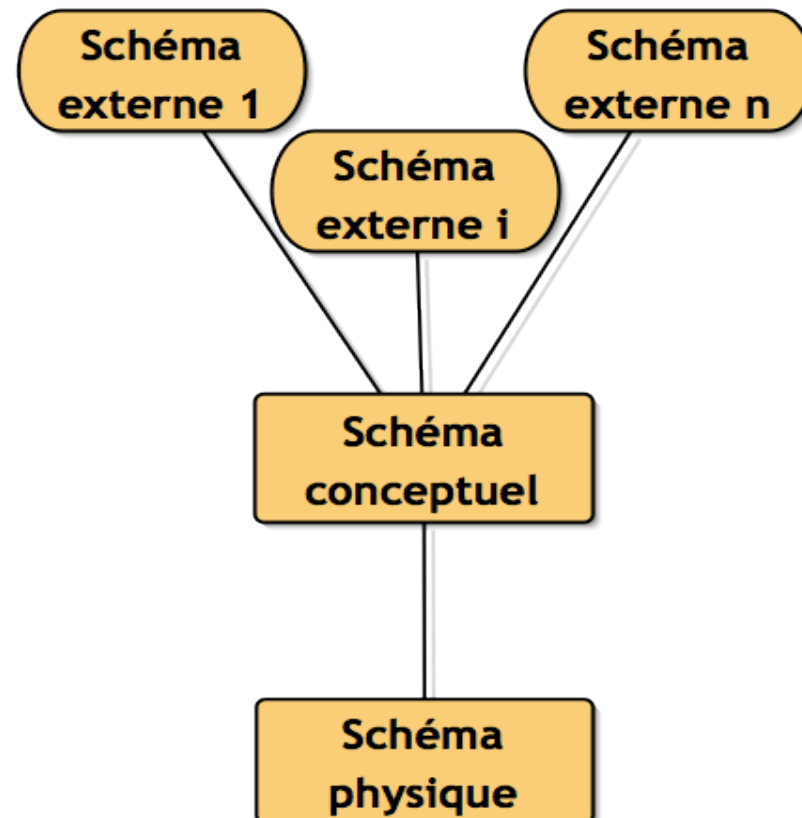
Nathalie Camelin

Université du Maine – Licence SPI Sem 6

Réaliser sa BdD relationnelle

2

- Concevoir un MEA
- Passer du MEA au MR
 - ▣ Ensemble de schémas de relation
- Choix du SGBD
 - ▣ Implémentation avec le langage SQL



* American National Standards Institute

**Standards Planning And Requirements Committee

Le langage SQL



- SQL : Structured Query Language
 - ▣ Le langage de communication avec un SGBD-R
- De nombreuses fonctionnalités
 - ▣ Gestion, insertion, suppression, modification de données
 - ▣ Opérations arithmétiques et de comparaison
 - ▣ Affichage des données
 - ▣ ...

Bref historique



- À la suite des travaux de Codd ...
 - ▣ 1974 : Lancement du projet System/R chez IBM
 - Basé sur les travaux de Chamberlin et Boyce
 - 1^{er} prototype nommé *SEQUEL*
 - *Structured English Query Language*
 - ▣ 1976 : Nouvelle implémentation SEQUEL/2
 - Fonctionnalités multi-utilisateurs et multi-tables
 - renommée SQL
 - ▣ 1978 : Tests concluants chez des clients pour IBM
 - Utilité et faisabilité

Produits commerciaux



- IBM : Développement de produits commerciaux basé sur le prototype System/R
 - ▣ 1981 : SQL/DS
 - ▣ 1983 : BD2
- 1979 : Oracle, première version commerciale par *Software Development Laboratories*

La norme SQL

- 1986 : Approbation ANSI de l'implémentation IBM
- 1987 : Première Norme ISO
- Puis :
 - ▣ 1989, 1992
 - ▣ 1999 (SQL3)
 - Intégration de parties plus avancées (SGBDRO, interface de programmation, gestion d'intégrité des données...)
 - ▣ 2003 (SQL:2003) , 2008 (SQL:2008)
 - Ajouts de manipulations XML
- 2011 (SQL:2011)
 - ▣ What's new? <http://www.sigmod.org/publications/sigmod-record/1203/pdfs/10.industry.zemke.pdf>

PostgreSQL



- Des atouts majeurs
 - ▣ Projet de SGBD-R non commercial le plus avancé
 - ▣ Projet Open Source toujours en développement
 - ▣ Existence de distributions commerciales (support tech.)

PostgreSQL



□ Historique

- ▣ 1977 : Projet Ingres débuté par l'Université de Berkeley
 - Relation Technologies/Ingres Corporation
- ▣ 1986 : Nouvelle équipe de l'Université de Berkeley
 - Ingres → Postgres
- ▣ 1996 : Ajout de nouvelles fonctionnalités par la communauté du logiciel libre
 - Postgres → PostgreSQL

PostgreSQL et la norme SQL ?

- Version installée à l'institut Claude Chappe
 - ▣ Côté serveur : 8.3.7
 - Documentation : <http://docs.postgresql.fr/8.3/>
 - ▣ Côté clients : psql (PostgreSQL) 8.4.X et 9.0.X
- Conformité avec la norme SQL
 - ▣ <http://docs.postgresql.fr/8.3/features.html>
 - ▣ Un extrait :

PostgreSQL supporte la plupart des fonctionnalités majeures de SQL:2003. Sur les 164 fonctionnalités requises pour une conformité « centrale » complète (*full Core conformance*), PostgreSQL se conforme à plus de 150. De plus, il existe une longue liste de fonctionnalités optionnelles supportées.

SQL : un langage fortement typé

- Une donnée \leftrightarrow Un type
- Un type \rightarrow Un ensemble d'opérations applicables
- Type : contrainte définit lors de la création de la table
 - ▣ !! important, y penser dès la modélisation !!
- Plus d'une 30aine de types définis dans PostgreSQL
 - ▣ Certains basés sur une norme ISO
 - ▣ D'autres non standards
 - !! compatibilité avec d'autres SGBDR ...

Types de données



- Booléens et binaires
- Caractères
- Numériques
 - ▣ Entiers
 - ▣ Réels
- Date et heure
- ...
- Types spécifiques
 - ▣ Géométrie, adresse réseau...

Le type NULL

- **NULL** correspond à :
 - ▣ méta-valeur représentant une absence de valeur
- Affectation :
 - ▣ Tout champ indépendamment de son type
 - !! sauf contrainte **NOT NULL**
- Référencement : le mot clé **NULL**
- **NULL** n'est pas
 - ▣ la valeur booléenne *false*
 - ▣ La chaîne de caractères vide ""
 - ▣ attention : 'NULL' n'est pas **NULL**
- Valeur par défaut d'un champ (si autorisé!)

5 catégories de commandes



- DDL - Data Definition Language
 - ▣ définition des éléments de la base de données : tables, champs, clés,...
- DML - Data Manipulation Language
 - ▣ manipulation des données : insertion, suppression, modification, extraction, ...
- DQL - Data Query Language
 - ▣ gestion des droits d'accès aux données
- DCL - Data Control Language
 - ▣ gestion des transactions
- SQL intégré

Commandes DML



- Langage de manipulation des données, 3 commandes de base:
 - ▣ **INSERT** : insertion de nouvelles données dans une table
 - ▣ **UPDATE** : mise à jour des données
 - ▣ **DELETE** : suppression d'enregistrements

- La commande **SELECT**
 - ▣ consultation de la base de données
 - ▣ associé à des mots clés et des clauses pour trouver et visualiser quasiment toutes les informations possibles
 - ▣ instruction la plus puissante et la plus complexe!

Toutes les clauses de la commande SELECT

- 9 clauses dont 7 optionnelles

```
SELECT [ ALL | DISTINCT ] { * | expression [ AS nom_affiché ] } [, ...]  
FROM nom_table [ [ AS ] alias ] [, ...] (version simplifiée)  
[ WHERE prédicat ]  
[ GROUP BY expression [, ...] ]  
[ HAVING condition [, ...] ]  
[ { UNION | INTERSECT | EXCEPT [ALL] } requête ]  
[ ORDER BY expression [ ASC | DESC ] [, ...] ]  
[ LIMIT { ALL | nombre } ]  
[ OFFSET début ]
```

Les opérateurs de bases

□ Projection : clause **SELECT**

- Rappel : En algèbre relationnelle, la projection élimine des attributs d'une relation
- Syntaxe :

```
SELECT att1, att2, ... attN FROM nom_table ;
```

□ Sélection : clause **WHERE**

- Rappel : En algèbre relationnelle, la sélection sur la condition C permet de garder les n-uplets qui satisfont C.
- Syntaxe :

```
SELECT * FROM nom_table WHERE condition ;
```


Exemple : la table *emp*

```
SELECT * FROM emp;
```

empno	name	job	mgr	hiredate	sal	comm
7369	SMITH	CLERK	7902	1980-12-17	800	
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300
7521	WARD	SALESMAN	7698	1981-02-22	1250	500
7566	JONES	MANAGER	7839	1981-04-02	2975	
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400
7698	BLAKE	MANAGER	7839	1981-05-01	2850	
7782	CLARK	MANAGER	7839	1981-06-09	2451	
7788	SCOTT	ANALYST	7566	1982-12-09	3000	
7839	KING	PRESIDENT		1981-11-17	5000	
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0
7876	ADAMS	CLERK	7788	1983-01-12	1100	
7900	JAMES	CLERK	7698	1981-12-03	950	
7902	FORD	ANALYST	7566	1981-12-03	3000	
7934	MILLER	CLERK	7782	1982-01-23	13000	

Exemple de Projection et Sélection

```
SELECT name, hiredate, sal FROM emp WHERE sal >= 1500 ;
```

empno	name	hiredate	sal	hiredate	sal	comm
7369	ALLEN	1981-02-20	1600	1980-12-17	800	
7499	JONES	1981-04-02	2975	1981-02-20	1600	300
7521	BLAKE	1981-05-01	2850	1981-02-22	1250	500
7566	CLARK	1981-06-09	2450	1981-04-02	2975	
7654	SCOTT	1982-12-09	3000	1981-09-28	1250	1400
7698	KING	1981-11-17	5000	1981-05-01	2850	
7782	TURNER	1981-09-08	1500	1981-06-09	2451	
7788	FORD	1981-12-03	3000	1982-12-09	3000	
7839	KING	PRESIDENT		1981-11-17	5000	
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0
7876	ADAMS	CLERK	7788	1983-01-12	1100	
7900	JAMES	CLERK	7698	1981-12-03	950	
7902	FORD	ANALYST	7566	1981-12-03	3000	
7934	MILLER	CLERK	7782	1982-01-23	13000	

Clause SELECT et ses opérateurs



- * : tous les champs de la table
- ALL : retourne toutes les lignes (par défaut)
- DISTINCT : suppression des doublons
- +, -, *, / : opérations mathématiques de base
- || : concaténation de champs de type caractères
- AS : nommer une colonne calculée

Exemples d'op. de la clause SELECT

```
SELECT empno || ' ' || name AS id_nom  
FROM emp where job = 'ANALYST';
```

id_nom
7788_SCOTT
7902_FORD

```
SELECT DISTINCT job FROM emp ;
```

job
ANALYST
CLERC
MANAGER
PRESIDENT
SALESMAN

```
SELECT sal+comm FROM emp ;
```

? column ?
1900
1750
2650
1500

Clause WHERE

- poser une condition sur les lignes
- syntaxe :

```
WHERE expression1 OPERATEUR expression2
```

- Les opérateurs logiques
 - ▣ comparaison : 6 opérateurs : `=`, `<>`, `<`, `>`, `<=`, `>=`
 - ▣ étendue : `BETWEEN` *valeur1* `AND` *valeur2*
 - ▣ appartenance : `IN` (*ensemble_valeurs*)
 - ▣ correspondance à un modèle : `LIKE` *modele*
 - ▣ `IS NULL`

Exemple de clause WHERE

- Qui a été embauché de août 1981 à août 1982?

```
SELECT * FROM emp WHERE hiredate BETWEEN '1981-08-01' AND '1982-08-01' ;
```

- Qui est vendeur ou employé de bureau?

```
SELECT * FROM emp WHERE job IN ('salesman','clerk') ;
```

- Quel employé a son nom commençant par la lettre A ?

```
SELECT * FROM emp WHERE name LIKE 'A%' ;
```

Correspondance à un modèle

- % correspond à un ensemble de caractères
 - ▣ 'A%' match 'ALLEN'
 - ▣ '%O%' match 'FORD', 'SCOTT' et 'JONES'
 - ▣ '%K' match 'CLARK' mais pas 'BLAKE'
- _ correspond à un seul caractère
 - ▣ 'K_NG' match 'KING' et 'KONG'
 - ▣ '_LA%' match 'BLAKE' et 'CLARK'

Opérateurs de négation et de conjonction

- négation d'une condition : **NOT**
 - ▣ exclure des enregistrements d'un ensemble de résultats
 - ▣ **NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL**

- conditions multiples : **OR** et **AND**
 - ▣ expression1 **AND** expression2
 - vrai si expression1 ET expression2 à TRUE
 - ▣ expression1 **OR** expression2
 - vrai si expression1 OU expression2 à TRUE
 - ▣ Possibilité de chaîner plusieurs opérateurs : importance de l'ordre d'écriture!

Opérateurs arithmétiques

- 4 opérateurs arithmétiques
 - ▣ addition(+), soustraction(-), multiplication(*) et division (/)
 - ▣ attention à la valeur NULL !

```
SELECT * FROM emp WHERE sal + comm >500;
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30

Sous-requêtes et filtrage

- Utiliser le résultat d'une requête comme élément de comparaison dans la condition de la clause WHERE
- 4 opérateurs spécifiques aux sous-requête :
 - ▣ IN, EXISTS, ALL, ANY
 - ▣ possibilité de négation : NOT
- Remarque :
 - ▣ IN équivalent à =ANY
 - ▣ NOT IN équivalent à <>ALL

Exemple de sous-requête

- Qui sont les personnes dont les manager sont rattachés au département 20?

```
SELECT * FROM emp WHERE mgr = ANY  
      (SELECT empno FROM emp WHERE deptno=20) ;
```

empno	name	job	mgr	hiredate	sal	comm
7369	SMITH	CLERK	7902	1980-12-17	800	
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300
7521	WARD	SALESMAN	7698	1981-02-22	1250	500
7566	JONES	MANAGER	7839	1981-04-02	2975	
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400
7698	BLAKE	MANAGER	7839	1981-05-01	2850	
7782	CLARK	MANAGER	7839	1981-06-09	2451	
7788	SCOTT	ANALYST	7566	1982-12-09	3000	
7839	KING	PRESIDENT		1981-11-17	5000	
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0
7876	ADAMS	CLERK	7788	1983-01-12	1100	
7900	JAMES	CLERK	7698	1981-12-03	950	
7902	FORD	ANALYST	7566	1981-12-03	3000	
7934	MILLER	CLERK	7782	1982-01-23	13000	

Exemple de sous-requête

- Qui est la personne qui a le plus gros salaire ?

```
SELECT * FROM emp WHERE sal >= ALL (select sal from emp) ;
```

Ordre d'évaluation dans le WHERE

Ordre d'évaluation	Type d'opérateur
1	signe positif (+), signe négatif (-)
2	multiplication (*), division(/)
3	addition(+), soustraction(-)
4	BETWEEN, IN, LIKE, IS NULL, =, <> ,< ,> ,<= ,>=
5	NOT
6	AND
7	OR

Les fonctions



- Un grand nombre de fonctions Postgre
 - ▣ Beaucoup hors standard SQL
 - !! compatibilité avec d'autres SGBD
 - ▣ Ex : fonction « première valeur non nulle »
 - Postgre → COALESCE()
 - Oracle → NVL()
- Fonctions typées
 - ▣ Type d'argument d'entrée
 - ▣ Type d'argument de retour

Quelques fonctions

- Fonctions mathématiques

- ▣ valeur absolue **ABS**, arrondi **ROUND**, racine carré **SQRT**, puissance **POWER**, **SIN**, **EXP**, **LOG**

- Fonctions de caractères

- Fonctions de formatage des types de données, fonctions de dates, fonctions système, ...

- Fonctions d'agrégation

- Doc : fonctions et opérateurs

- ▣ <http://docs.postgresql.fr/8.3/functions.html>

Exemples de fonctions de caractères



- remplacement de caractères
 - ▣ `TRANSLATE(chaine, val1, val2),`
`REPLACE(chaine, char1, char2)`
- modification de la casse
 - ▣ `UPPER(chaine), LOWER(chaine)`
- extraction de sous-chaine
 - ▣ `SUBSTR(chaine, départ, longueur)`
- longueur de chaine
 - ▣ `LENGTH(chaine)`

Exemples de fonctions de caractères

```
SELECT ename, SUBSTR(job,1,3)  
FROM emp WHERE sal>2000;
```

ename	job
JONES	MAN
BLAKE	MAN
CLARK	MAN
SCOTT	ANA
KING	PRE
FORD	ANA

```
SELECT DISTINCT lower(translate(job,'A','K'))  
FROM emp ;
```

job
knklyst
clerc
mknkger
president
sklesmkn

```
SELECT replace(job,'s','z') FROM emp ;
```

!! ne remplace rien du tout !!

Fonctions d'agrégation

- Calcul d'UNE SEULE VALEUR à partir d'un ensemble de valeurs en entrée
- Syntaxes :
 - ▣ nom_agrégat (expression)
 - ▣ nom_agrégat (ALL expression)
 - ▣ nom_agrégat (DISTINCT expression)
 - ▣ nom_agrégat (*)
- Utilisation dans :
 - ▣ Clause **SELECT**
 - ▣ Clause **HAVING**

Fonctions d'agrégation



- Les plus fréquemment utilisées :
 - ▣ **COUNT** : compte du nombre de valeurs non NULL
 - ▣ **SUM** : somme de valeurs numériques ou intervalle
 - ▣ **AVG** : moyenne de valeurs numériques ou intervalle
 - ▣ **MIN** et **MAX** : valeur minimale/maximale de valeurs numériques, chaîne de caractères ou date
 - ▣ **STDDEV** : écart-type de valeurs numériques
 - ▣ **VARIANCE** : variance de valeurs numériques

Exemple de fonctions d'agrégation

□ Exemple avec *count*

▣ Compter un nombre de lignes

```
SELECT count(mgr) AS manager,  
       count(ALL mgr) AS tous_manager,  
       count(DISTINCT mgr) AS nombre_manager,  
       count(*) AS toutes_lignes  
from emp ;
```

manager	tous_manager	nombre_manager	toutes_lignes
13	13	6	14

Exemples de fonctions d'agrégation

```
SELECT COUNT (DISTINCT job) FROM emp ;
```

count
5

```
SELECT AVG (comm) FROM emp ;
```

avg
550

```
SELECT ename, MAX (sal) FROM emp ;
```

Réponse du SGBD :

ERREUR: la colonne « emp.ename » doit apparaître dans la clause GROUP BY ou doit être utilisé dans une fonction d'agrégat

Toutes les clauses de la commande SELECT

- 9 clauses dont 7 optionnelles

```
SELECT [ ALL | DISTINCT ] { * | expression [ AS nom_affiché ] } [, ...]  
FROM nom_table [ [ AS ] alias ] [, ...] (version simplifiée)  
[ WHERE prédicat ]  
[ GROUP BY expression [, ...] ]  
[ HAVING condition [, ...] ]  
[ { UNION | INTERSECT | EXCEPT [ALL] } requête ]  
[ ORDER BY expression [ ASC | DESC ] [, ...] ]  
[ LIMIT { ALL | nombre } ]  
[ OFFSET début ]
```

Opérateurs ensemblistes

- combiner le résultat de 2 requêtes ou plus
 - ▣ **UNION**: mettre en communs tous les n-uplets
 - ▣ **INTERSECT** : identifier les n-uplets similaires
 - ▣ **EXCEPT** : identifier les n-uplets appartenant à un ensemble mais pas à l'autre
- syntaxe :

`requête_1 { UNION | INTERSECT | EXCEPT } [ALL] requête_2 [...]`

 - ▣ même schéma pour requête_1 et requête_2
- !! **DISTINCT** par défaut → **ALL**
- possibilité de chaîner plusieurs opérateurs : évaluer de gauche à droite

Exemple de l'op. EXCEPT

```
SELECT * FROM emp WHERE job = 'SALESMAN'
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30

```
SELECT * FROM emp WHERE sal < 1300
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980-12-17	800		20
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7876	ADAMS	CLERK	7788	1983-01-12	1100		20
7900	JAMES	CLERK	7698	1981-12-03	950		30

```
SELECT * FROM emp WHERE job = 'SALESMAN'  
EXCEPT  
SELECT * FROM emp WHERE sal < 1300
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30

La clause ORDER BY

- SANS : ordre des n-uplets aléatoire et non garanti
- Trier les n-uplets résultats de la requête

▣ syntaxe :

```
ORDER BY expression [ ASC | DESC ] [, ...]
```

- expression : champ, ordinal ou opération mathématique de base
 - ASC : ordre ascendant (par défaut)
 - DESC : ordre descendant
- Tri possible selon plusieurs champs dans l'ordre précisé

Exemple du ORDER BY

```
SELECT * FROM emp ORDER BY name ASC;
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7876	ADAMS	CLERK	7788	1983-01-12	1100		20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30
7782	CLARK	MANAGER	7839	1981-06-09	2450		10
7902	FORD	ANALYST	7566	1981-12-03	3000		20
7900	JAMES	CLERK	7698	1981-12-03	950		30
7566	JONES	MANAGER	7839	1981-04-02	2975		20
7839	KING	PRESIDENT		1981-11-17	5000		10
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7934	MILLER	CLERK	7782	1982-01-23	1300		10
7788	SCOTT	ANALYST	7566	1982-12-09	3000		20
7369	SMITH	CLERK	7902	1980-12-17	800		20
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30

Exemple du ORDER BY

```
SELECT sal, comm FROM emp ORDER BY 2, 1;
```

sal	comm
1500	0
1600	300
1250	500
1250	1400
800	
950	
1100	
1300	
2450	
2850	
2975	
3000	
3000	
5000	

Remarque : La valeur NULL est considérée supérieure à toute autre valeur.

Les clauses LIMIT et OFFSET

- Restreint le nombre de n-uplets renvoyés
- syntaxe :

```
LIMIT { ALL | nombre }  
OFFSET début
```

 - ▣ **ALL** : par défaut
 - ▣ *nombre* : nombre total de n-uplets à afficher
 - ▣ *début* : indice à partir duquel on affiche les n-uplets résultats
- Attention
 - ▣ À n'utiliser qu'en complément de ORDER BY
 - ▣ À ne pas utiliser à la place d'un **MAX**
 - Risques d'incohérence

Exemple de ORDER BY/LIMIT/OFFSET

```
SELECT * FROM emp ORDER BY ename ASC LIMIT 4 OFFSET 1;
```

empno	name	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980-12-17	800		20
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30
7566	JONES	MANAGER	7839	1981-04-02	2975		20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30
7782	CLARK	MANAGER	7839	1981-06-09	2450		10
7788	SCOTT	ANALYST	7566	1982-12-09	3000		20
7839	KING	PRESIDENT		1981-11-17	5000		10
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30
7876	ADAMS	CLERK	7788	1983-01-12	1100		20
7900	JAMES	CLERK	7698	1981-12-03	950		30
7902	FORD	ANALYST	7566	1981-12-03	3000		20
7934	MILLER	CLERK	7782	1982-01-23	1300		10

La clause GROUP BY

- Classement des données par groupe
 - ▣ sous-ensemble de lignes ayant même valeur pour les attributs précisés

- Syntaxe

GROUP BY *expression* [, ...]

- ▣ ***expression*** : nom de colonne en entrée, nom ou numéro de colonne en sortie ou expression sur les champs en entrée
 - ▣ Après la clause WHERE et avant la clause ORDER BY
- Remarque :
 - ▣ Chaque champ de la clause SELECT → dans la clause GROUP BY (sauf agrégat)
 - ▣ !! Une seule ligne produite par groupe !!

Exemple de GROUP BY

```
SELECT job, avg(sal) FROM emp GROUP BY job;
```

empno	name	job	avg(sal)	date	sal	comm	deptno	avg(sal)
7369	SMITH	SALESMAN	1400,00	1980-12-17	800		20	1037,50
7499	ALLEN	MANAGER	2758,33	1981-02-20	1600	300	30	1400,00
7521	WARD	CLERK	1037,50	1981-02-22	1250	500	30	1400,00
7566	JONES	PRESIDENT	5000,00	1981-04-02	2975		20	2758,33
7654	MARTIN	ANALYST	3000,00	1981-09-28	1250	1400	30	1400,00
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30	2758,33
7782	CLARK	MANAGER	7839	1981-06-09	2450		10	2758,33
7788	SCOTT	ANALYST	7566	1982-12-09	3000		20	3000,00
7839	KING	PRESIDENT	5000	1981-11-17	5000		10	5000,00
7844	TURNER	SALESMAN	7698	1981-09-08	1500	0	30	1400,00
7876	ADAMS	CLERK	7788	1983-01-12	1100		20	1037,50
7900	JAMES	CLERK	7698	1981-12-03	950		30	1037,50
7902	FORD	ANALYST	7566	1981-12-03	3000		20	3000,00
7934	MILLER	CLERK	7782	1982-01-23	1300		10	1037,50

une seule ligne par groupe :
la fonction moyenne est appliquée au sein
de chaque groupe

Exemples de GROUP BY

- Attention : Une seule ligne retournée par groupe

```
SELECT job, avg(sal) FROM emp WHERE name like '%A%' GROUP BY job;
```

job	avg(sal)
SALESMAN	1366,67
MANAGER	2650,00
CLERK	1025,00

```
SELECT name, job, avg(sal) FROM emp WHERE name like '%A%'  
GROUP BY job;
```

!! ne fonctionne pas !! :


il existe plusieurs valeurs de name par job ...

Exemple de GROUP BY


- possibilités d'avoir des sous-groupes
- l'ordre des colonnes n'a pas d'importance

```
SELECT mgr, job, avg(sal) FROM emp WHERE name like '%A%'  
group by job, mgr;
```

```
SELECT mgr, job, avg(sal) FROM emp WHERE name like '%A%'  
group by mgr, job;
```



job	mgr	sal
MANAGER	7839	2650
CLERK	7788	1100
SALESMAN	7698	1367
CLERK	7698	950



job	mgr	sal
CLERK	7698	950
MANAGER	7839	2650
CLERK	7788	1100
SALESMAN	7698	1367

La clause HAVING

- Restriction appliquée sur le groupe
- **HAVING** est au **GROUP BY** ce que le **WHERE** est au **SELECT**
 - ▣ **HAVING** → restriction sur les groupes
 - ▣ **WHERE** → restriction sur les enregistrements
- même syntaxe que **WHERE** mais :
 - ▣ fonction d'agrégation
 - ▣ expression figurant dans la clause **GROUP BY**

Exemple de HAVING

```
SELECT job, avg(sal) FROM emp GROUP BY job HAVING avg(sal)>1300;
```

empno	name	job	avg(sal)	e	sal	comm	deptno
7369	SMITH	SALESMAN	1400,00	17	800		20
7499	ALLEN	SALESMAN	2758,33	20	1600	300	30
7521	WARD	SALESMAN	5000,00	22	1250	500	30
7566	JONES	MANAGER	3000,00	02	2975		20
7654	MARTIN	SALESMAN	7698	1981-09-28	1250	1400	30
7698	BLAKE	MANAGER	7839	1981-05-01	2850		30
7782	CLARK	MANAGER	7839	1981-06-09	2450		10



Cherchez l'intrus ...

```
SELECT avg(sal) FROM emp HAVING ename LIKE '%A%';
```

```
SELECT ename FROM emp GROUP BY job HAVING ename LIKE '%A%';
```

```
SELECT job, avg(sal) FROM emp GROUP BY job HAVING job LIKE '%A%';
```

7902	FORD	ANALYST	7500	1981-12-03	3000		20
7934	MILLER	CLERK	7782	1982-01-23	1300		10

Toutes les clauses de la commande SELECT

- 9 clauses dont 7 optionnelles

```
SELECT [ ALL | DISTINCT ] { * | expression [ AS nom_affiché ] } [, ...]  
FROM nom_table [ [ AS ] alias ] [, ...] (version simplifiée)  
[ WHERE prédicat ]  
[ GROUP BY expression [, ...] ]  
[ HAVING condition [, ...] ]  
[ { UNION | INTERSECT | EXCEPT [ALL] } requête ]  
[ ORDER BY expression [ ASC | DESC ] [, ...] ]  
[ LIMIT { ALL | nombre } ]  
[ OFFSET début ]
```

La clause FROM



- Clause indispensable pour sélectionner l'ensemble des données sur lesquelles travailler
- Réellement obligatoire?
 - ▣ Oui : travailler sur des données
 - ▣ Non : appeler une fonction
- Possibilité de jointure
 - ▣ Travailler sur des données issues de plusieurs tables
 - ▣ Résultat : une relation

Les jointures



- Sélection des données de travail sur plusieurs tables
 - ▣ Possibilité de jointure réflexive
- Mot clé : JOIN
- Dans la clause FROM
- 3 types de jointures :
 - ▣ Jointure croisée : CROSS JOIN
 - ▣ Jointure interne : INNER JOIN / NATURAL JOIN
 - ▣ Jointure externe : [RIGHT/LEFT/FULL] OUTER JOIN

Jointure croisée : CROSS JOIN

- Produit cartésien entre 2 tables
 - ▣ $R1 \times R2 \rightarrow R3$ regroupant exclusivement toutes les possibilités de combinaison des occurrences de R1 et R2
- Syntaxe :

```
SELECT * FROM table_1 CROSS JOIN table_2
```

 - ▣ identique à :

```
SELECT * FROM table_1, table_2
```
- Quand l'utiliser?
 - ▣ Besoin de toutes les possibilités de combinaison entre des valeurs de différentes tables

Jointure interne : [INNER] JOIN

- Retour uniquement des lignes satisfaisant la condition de jointure
- Condition de jointure :
 - ▣ Condition comparant des champs compatibles
 - même type de données
 - !! même signification !!
 - ▣ 2 syntaxes exprimant la condition :
 - **USING** nom_attr

```
SELECT * FROM table_1 JOIN table_2 USING nom_attr_12
```

- **ON** table_1.nom_attr_a op_comp table_2.nom_attr_b
- ▣ Possibilité de condition avec conjonction/négation

Jointure naturelle : NATURAL JOIN

- Cas particulier d'une jointure interne : Condition de jointure implicite

```
SELECT * FROM table_1 NATURAL JOIN table_2;
```

- ▣ La valeur de TOUS les attributs ayant même nom dans les 2 tables doivent être égales
 - Relie les tables en faisant correspondre toutes les attribut portant le même nom
- À n'utiliser qu'avec prudence...

Remarques sur la jointure interne

□ Quand l'utiliser?

- ▣ Lorsque l'on veut faire une sélection a priori sur la « relation de travail »
- ▣ Équivalent à un l'utilisation d'un CROSS JOIN et d'un WHERE mais moins efficace

□ Que se passe-t-il?

- ▣ Si je fais une jointure interne où aucune des combinaison de tuples ne satisfait la condition?
- ▣ Si je fais une jointure naturelle sur deux tables n'ayant aucun nom d'attribut commun?
- ▣ Si je fais une jointure naturelle réflexive?

Jointure externe : OUTER JOIN

□ Retourne :

- ▣ Les lignes de chaque table qui satisfont la condition de jointure (idem INNER JOIN)
- ▣ PLUS : Les lignes de la table [droite/gauche] qui ne la satisfont pas
 - [RIGHT/LEFT/FULL] OUTER JOIN

□ Condition de jointure identique à celle du INNER JOIN

□ Quand utiliser une jointure externe?

- ▣ Lorsqu'en plus de la sélection sur la relation de travail, on veut garder toutes les données d'une table en particulier (ou les deux)

Les jointures

- Possibilité de plusieurs jointures dans un seul **FROM**
 - ▣ Lecture de gauche à droite
- Possibilité de faire une jointure réflexive
 - ▣ Obligation de renommer (au minimum) une table avec **AS**
 - Attributs disponibles par notation préfixée :
alias_table.nom_attribut
- Possibilité de faire une jointure avec un ensemble de tuples issu d'une requête **SELECT**
 - ▣ Obligation de nommer l'ensemble de tuples avec **AS**
 - À n'utiliser que si nécessaire. Par exemple, pour récupérer un attribut calculé.

Récapitulatif de la commande SELECT

SELECT	<i>noms des colonnes à afficher</i>
FROM	<i>nom de la table contenant les n-uplets</i>
WHERE	<i>condition(s) à remplir par les lignes</i>
GROUP BY	<i>condition(s) de regroupement des lignes</i>
HAVING	<i>condition(s) à remplir par le groupe</i>
UNION/INTERSECT/EXCEPT	<i>opérateurs ensemblistes</i>
ORDER BY	<i>ordre d'affichage</i>
LIMIT	<i>nombre de n-uplets à afficher</i>
OFFSET	<i>numéro du premier n-uplet affiché</i>

Ordre d'exécution du SELECT



1. FROM
2. WHERE
3. Fonctions de groupe / GROUP BY
4. HAVING
5. SELECT
6. UNION/INTERSECTION/EXCEPT
7. ORDER BY
8. DISTINCT
9. OFFSET
10. LIMIT