BASES DE DONNÉES AVANCÉES

Nathalie Camelin

Université du Maine – Licence SPI Sem 6

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

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 ]
```

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 chainer 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) ;
```

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	

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

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 chainer 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 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	mgr	hiredate	sal	comm	deptno	avg(sal)
7369	SMITH	CLERK	7902	1980-12-17	800		20	1037,50
7499	ALLEN	SALESMAN	7698	1981-02-20	1600	300	30	1400,00
7521	WARD	SALESMAN	7698	1981-02-22	1250	500	30	1400,00
7566	JONES	MANAGER	7839	1981-04-02	2975		20	2758,33
7654	MARTIN	SALESMAN	7698	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		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

Exemple de GROUP BY


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

job	avg(sal)
SALESMAN	1400,00
MANAGER	2758,33
CLERK	1037,50
PRESIDENT	5000,00
ANALYST	3000,00

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	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

Exemple de HAVING


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

job	avg(sal)
SALESMAN	1400,00
MANAGER	2758,33
PRESIDENT	5000,00
ANALYST	3000,00

- 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%';
```

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

Récapitulatif de la commande SELECT

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

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

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

Quelques objets du SGBD (suite)

- ROLE
  - Notion d' « utilisateur » ou « groupe d'utilisateurs »
  - Avant la 8.1 : USER et GROUP
- INDEX
  - Construction physique d'une structure de données sur une ou plusieurs colonnes d'une table
- SEQUENCE
  - Suite d'entiers qui s'incrémente à chaque appel
- FUNCTION
  - Opération stockée

ROLE

- Mot clé ROLE (!! Non standard SQL)
- Entité qui peut posséder des objets et avoir des droits
  - Rôle de la base ≠ utilisateur du système d'exploitation
  - Global aux bases de données
  - Affecte les droits d'accès sur ses objets pour les autres rôles
- Catalogue des rôles : pg_roles

```
SELECT rolname FROM pg_roles;
```

```
\du
```

ROLE - Utilisateur

- Un « user » est un rôle qui a le droit de connexion
  - Détermination des droits par rapport au rôle spécifié lors de la connexion

```
psql -U nom_user
```

  - session_user : nom_user
- L'utilisateur initial : *postgres*
  - Rôle par défaut d'un SGBD postgreSQL nouvellement installé
  - Droits d'un super-utilisateur :
    - Aucune vérification de droits avant l'exécution d'une action

ROLE - Groupement

- Un rôle peut être à la fois une personne ou un groupe de personnes
  - Intérêt : partage commun de droits
- Possibilité de changer de rôle

```
SET ROLE to group_user ;
```

  - Si nom_user fait partie du groupe group_user
  - current_user prend alors la valeur group_user
- Quel rôle ?

```
SELECT current_user, session_user;
```

```
RESET ROLE;
```

Exemple de ROLE

- Création d'un rôle

```
CREATE ROLE etudiant LOGIN ;
CREATE USER etudiant ;                -- identique à la ligne précédente

CREATE ROLE prof SUPERUSER ;          -- création d'un super-utilisateur

CREATE ROLE etu_db CREATEDB ;         -- possibilité de créer des bases de données

CREATE ROLE etu_role CREATEROLE ; -- possibilité de créer des rôles
```

- Modification d'un rôle

```
ALTER ROLE etudiant WITH PASSWORD 'toto13' ;
```

- Suppression d'un rôle

```
DROP ROLE etudiant;
```

INDEX

- Notion non standard
  - Méthode courante augmentant les performances du SGBD
  - Méthodes d'indexation : arbres, tables de hachage
  - Algorithme de recherche spécifique au type d'index
  - Même intérêt que les index de livre

- Pourquoi utiliser un index ?
  - ++ : Retrouver une ligne spécifique plus rapidement
  - – : Ajout une surcharge au SGBD (maj à chaque modif)
  - → À utiliser à bon escient

INDEX

- Un index créé automatiquement sur chaque clé primaire
- Création d'index

```
CREATE INDEX nom_index
    ON nom_table [USING methode] ( nom_colonne [,...] )
    [ WHERE expression ]
```

  - WHERE : création d'un index partiel
- Suppression d'index

```
DROP INDEX nom_index [,...] [RESTRICT/CASCADE];
```

- Modification avec ALTER INDEX

SEQUENCE

- Table permettant la génération d'entier

```
CREATE SEQUENCE nom_seq [INCREMENT BY increment ]
   [ MINVALUE valeurmin | NO MINVALUE ]
   [ MAXVALUE valeurmax | NO MAXVALUE ]
   [ START [ WITH ] début ]
   [ CACHE cache ]
   [ [ NO ] CYCLE ]
   [ OWNED BY { table.colonne | NONE } ]
```

- Utilisation des valeurs avec
  - nextval() : valeur suivante
  - curval() : valeur courante
  - setval() : initialisation de la valeur courante

Exemple de SEQUENCE


```
CREATE SEQUENCE mon_num START 3 INCREMENT BY 2;

SELECT nextval('mon_num');
      -- retourne 3

SELECT nextval('mon_num');
      -- retourne 5

INSERT INTO ma_table VALUES (nextval('mon_num')) ;

ALTER SEQUENCE mon_num RESTART WITH 3 ;
      -- réinitialisation à 3 ;

SELECT setval('mon_num',3,false);
      -- idem
```

FUNCTION

- Identificateur demandant l'exécution au sein d'une requête SQL d'une opération programmée
  - Le résultat en retour s'insère en lieu et place de l'appel de la fonction
- Objet lié à une base de données (tout comme les autres objets : tables, vues, index ...)
- Intérêts :
  - Étendre les capacités de SQL par la création de nouvelles fonctions
  - Simplifier l'élaboration des requêtes SQL

FUNCTION

- Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )
RETURNS type_retour
AS 'definition'
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

- CREATE OR REPLACE
  - !! le prototype reste inchangé
- Arguments :
  - Types d'arguments
  - Non obligatoires mais ()

FUNCTION

- Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )
RETURNS type_retour
AS 'definition'
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

- *type_retour*
  - Type de données renvoyé par la fonction
- AS '*definition*':
  - Code de la fonction pour le SQL ou les langages procéduraux de type PL/pgSQL
  - Chemin en absolu du fichier contenant le code objet pour le C

FUNCTION

- Syntaxe

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ argtype [, ...] ] )
RETURNS type_retour
AS 'definition'
LANGUAGE 'nom_langage' [ WITH ( attribut [, ...] ) ]
```

- *LANGUAGE*
  - Précise le nom du langage du corps de la fonction
- *attribut :*
  - *isstrict* : si un des paramètres est NULL alors la fonction renvoie NULL et n'est pas exécutée
  - *immutable* ...

FUNCTION

- Exemple de déclaration d'une fonction

```
CREATE FUNCTION nom_leader (varchar(20))
RETURNS varchar(30) AS
'SELECT nom
FROM individu join leader on id_ind=id_musicien join cd using (id_cd)
WHERE titre_cd=$1'
LANGUAGE 'SQL';
```

- Appel de la fonction

```
select nom_leader('evidence');
```

```
 nom_leader
------------
 coltrane
(1 row)
```

!!!! Le langage doit avoir été créé au préalable ...

Les fonctions prédéfinies

- 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

Les fonctions prédéfinies

- 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

5 catégories de commande

- 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

Gestion des privilèges

- Liste de contrôle d'accès (ACL)
  - Quels rôles possèdent quels droits sur quels objets ?
  - Consultée avant toute action
  - Pour chaque objet :
    - Ensemble de privilèges
    - Ensemble de restriction
  - Possibilité de modifier les droits sur un objet :
    - Par le super-utilisateur ou le propriétaire
    - Octroi de privilèges : GRANT
    - Restriction de privilèges : REVOKE

Attribution de privilèges

- GRANT

GRANT *privilege* [,...] ON *objet* [,...] TO
{ *username* | GROUP *groupname* | PUBLIC } [, ...]

- Exemples de valeurs pour *privilege* :
  - SELECT (lecture)
  - INSERT (insertion de lignes)
  - UPDATE (modification de colonnes)
  - DELETE (suppression de lignes)
  - ALL

!!! Attention les droits dépendent des objets ...

Attribution de privilèges

- GRANT

```
GRANT privilege [,...] ON objet [,...] TO
{ username | GROUP groupname | PUBLIC } [, ...]
```

- Exemples de valeurs pour *objet* :
  - bases de données : ON DATABASE nom_bd
  - table : ON [TABLE] nom_table
  - schéma : ON SCHEMA nom_schema
  - langage : ON LANGUAGE nom_langage
  - fonctions : ON FUNCTION nom_fonction

Attribution de privilèges

- GRANT

```
GRANT privilege [,...] ON objet [,...] TO
{ username | GROUP groupname | PUBLIC } [, ...]
```

- *TO* :
  - Un utilisateur particulier : *username*
  - Un groupe d'utilisateur : GROUP groupname
  - Tous les utilisateurs : PUBLIC

Restriction des privilèges

- REVOKE : Restriction de privilèges

```
REVOKE privilege [,...] ON objet [,...] FROM
{ username | GROUP groupname | PUBLIC } [, ...]
```

- Remarque :
  - Le fait de supprimer des droits à public n'affecte pas ceux à qui l'on a octroyé spécifiquement les droits.

Accès aux différents schémas

- Donner les droits d'accès à son schéma

```
GRANT usage ON schema mon_schema TO nom_utilisateur ;
```

- Révoquer les droits

```
REVOKE create ON schema public FROM public ;
```

Interdiction de créer des objets dans le schéma public

- !! L'octroi de droits sur le schema n'octroie pas de droits sur les objets de ce schema ...

Création de groupes d'utilisateurs

- Créations de rôles

```
CREATE ROLE group_test;
CREATE ROLE toto;
CREATE ROLE tata;
CREATE ROLE titi;
```

- Ajout d'utilisateurs au sein d'un groupe

```
GRANT group_test TO toto,tata,titi;
```

- Suppression d'utilisateurs

```
REVOKE group_test FROM toto;
ALTER GROUP group_test DROP USER toto;   -- idem
```

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

Les transactions

- Groupe d'opérations formant une unité indivisible
  - Toutes effectuées
  - Ou aucune
- Participe à l'intégrité des données
- Par défaut chaque instruction est elle-même une transaction
- Déclaration explicite par un bloc transactionnel

```
BEGIN ;
 -- listes d'instructions
COMMIT ;
```

Exemple de transactions

- Un virement financier :

```
BEGIN ;
UPDATE comptes SET balance = balance - 100.00
    WHERE nom = 'Alice';
UPDATE comptes SET balance = balance + 100.00
    WHERE nom = 'Bob';
COMMIT ;
```

- Les deux opérations sont nécessaires ou aucune ne doit être exécutée.

Gestion des transactions

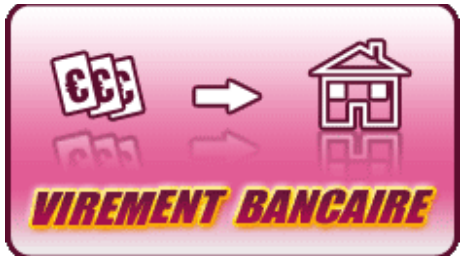
- Possibilité de placer des points de retournement
  - SAVEPOINT
- Possibilité d'annuler des opérations :
  - ROLLBACK TO

```
BEGIN;
UPDATE comptes SET balance = balance - 100.00
    WHERE nom = 'Alice';
SAVEPOINT mon_pointdesauvegarde;
UPDATE comptes SET balance = balance + 100.00
    WHERE nom = 'Bob';
-- oups ... oublions ça et créditons le compte de Wally
ROLLBACK TO mon_pointdesauvegarde;
UPDATE comptes SET balance = balance + 100.00
    WHERE nom = 'Wally';
COMMIT;
```

Le modèle transactionnel des SGBDR

- Exemple d'une transaction

Luke



Leia

- Transaction bancaire de 100€:
  1. Compte bancaire de Luke débité de 100€
  2. Compte bancaire de Leia crédité de 100€
  3. Mise à jour autres informations

Contraintes ACID

- Atomicité
  - L'ensemble des opérations liées à une transaction est indivisible
    - Tout : L'ensemble des opérations est réalisé sans aucune erreur

    OU
    - Rien : L'ensemble des opérations est annulé car une opération a levé une erreur
  - Exemple :
    - Le compte de Luke ne peut pas être débité si celui de Leia n'est pas crédité

Contraintes ACID

- Cohérence
  - Les modifications apportées doivent être valides en fin de transaction

    →Conformes à la base et à ses contraintes d'intégrité
    - Cohérence à l'instant T et T+1
    - Si une ou plusieurs opérations met(tent) en cause l'intégrité alors :
      - L'ensemble des opérations est annulé

      OU
      - Le système modifie les données dépendantes
  - Exemple :
    - Le compte de Luke ne peut pas être débité si celui-ci devient négatif et que cela n'est pas autorisé

Contraintes ACID

- Isolation
  - Toute modification de la base au cours d'une transaction doit être invisible aux autres transactions
    - Seuls les états terminaux sont visibles

    → Invisibilité des états intermédiaires de la base
  - Exemple :
    - Le compte de Luke ne peut pas être affiché avec un solde(T)-100€ si celui-ci de Leia n'est pas affiché avec un solde(T)+100€

Contraintes ACID

- Durabilité
  - Toutes modifications apportées lors d'une transaction réussie sont définitives.
    - Ne peuvent disparaître suite à un problème technique.
  - Exemple :
    - Le compte de Leia est crédité ne peut apparaître avec un solde ne tenant pas compte du crédit si la transaction a été validée. Et ce, même en cas de dysfonctionnement et restauration du système.

Quelques mécanismes de garantie ACID

- Notion de bloc transactionnel en SQL Standard
  - Start transaction … commit;
  - savepoint et rollback

    →Atomicité, Cohérence
- Définition de différents niveaux de verrouillage

  → Isolation
  - Plus haut niveau : Serializable

  Plus d'infos : http://docs.postgresql.fr/9.2/transaction-iso.html
- Journalisation des transactions
  - Toute transaction validée est notifiée dans un journal

    →Durabilité

  Plus d'infos : http://docs.postgresql.fr/9.1/wal.html