5.
SQL

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

SQL

- SQL = *Structured Query Languages*
- *Une des raisons du succès des BD relationnelles.*
- SQL est basé sur l'algèbre et le calcul relationnels.
- 3 versions :
  - SQL86 ou SQL 1
  - SQL92 ou SQL2
  - SQL99 ou SQL3

Les langages

- LDD ou Langage pour la Définition des Données (*Data Definition Language - DDL*)
  - définition de l'ensemble du schéma de la base de données. (create, alter, rename, drop, etc.)
- LMD ou Langage pour la Manipulation des Données (*Data Manipulation Language - DML*)
  - consultation, modification, suppression et ajout des données. (select, insert, update, delete)
- LCD ou Langage pour le Contrôle des Données (*Data Control Language - DCL)*
  - définition des permissions d'accès, administration (grant , revoke, commit, rollback, etc..).

Schéma

- Les termes changent :
  - table = relation; row = tuple; column = attribute
- A partir de SQL2, on trouve la notion de schema.
- La création d'un schema retourne du privilège du DBA.

```
create schema test2 authorization ocure;
```

- Un schema regroupe les tables, contraintes, vues, etc..

Catalog

- Un catalogue est une collection nommée de schémas dans un SGBDR.
- Un catalogue regroupe les schémas de la base de données.

OlivierCuré *[ocure@univ-mlv.fr]*

5.1 SQL LDD

OlivierCuré *[ocure@univ-mlv.fr]*

LDD - Création de table

- Une table permet de spécifier une relation.
- En général, la déclaration d'un table regroupe:
  - le nom de la table
  - des attributs
  - des contraintes
  - la définition d'une clé primaire

```
CREATE TABLE table_name ({column_name
  data_type[DEFAULT default_exp]
  [column_constraint [,..] ] |
  table_constraint } [,..] );
```

LDD - Création de table(2)

- Exemple

```
create table personne (idPers int,
nomPers varchar(20) not null,
primary key(idPers));
```

- Les types de PostgreSQL :

- Entier : integer
- Numeric : decimal(p,s)
- Réel : real
- Booléen : bool
- Chaines de caractères : char(n) et varchar(n)
- Incrémentation automatique : serial
- Date et heure : timestamp
- Date : date

Il est possible de créer des types.

Contraintes d'intégrité

- Il s’agit d’une méthode déclarative pour assurer le respect des règles de gestion dans la base de données
- Les règles de gestion sont définies avec la table correspondante grâce aux commandes CREATE TABLE ou ALTER TABLE
- Ces règles sont validées quand les contraintes sont définies, et à chaque instruction INSERT, UPDATE ou DELETE

Contraintes d'intégrité

- Cinq types disponibles:
  - NOT NULL
  - Clé primaire
  - Clé unique
  - Clé étrangère
  - CHECK

Contraintes NOT NULL

- Elles garantissent qu'une colonne contient une valeur pour chaque ligne
- Elles sont en général définies dans la commande CREATE TABLE

Contraintes de clé primaire

- Elles garantissent qu'il n'y a pas deux lignes de la table qui contiennent la même valeur (duplicates) dans la ou les colonnes qui constitue(nt) la clé
- Toutes les colonnes de la clé primaire doivent être obligatoires (NOT NULL)
- Une seule clé primaire est autorisée par table
- Syntaxe :

```
ALTER TABLE nom_table ADD
CONSTRAINT nom_contrainte
PRIMARY KEY (nom_colonne,…,nom_colonne);
```

Contraintes de clé primaire (suite)

- Une colonne de clé primaire idéale est une colonne dont les valeurs changent rarement, car la mise à jour d'une clé primaire a généralement un impact sur d'autres tables
- Par convention, on ajoute souvent le suffixe _PK (primary key) au nom d'une clé primaire, elle est ainsi plus facile à détecter dans le dictionnaire des données
- Mieux vaut choisir des colonnes contenant des valeurs numériques courtes : meilleures performances et moins d'allocation mémoire.

Contraintes de clé primaire (exemple)

```
ALTER TABLE etudiant
ADDCONSTRAINT etudiant_pk
PRIMARY KEY (id_etudiant);
```

Contraintes de clé unique

- Mêmes règles que pour les clés primaires, mais autorisant des colonnes facultatives (pas not null)
- Il peut y avoir un nombre illimité de clés uniques dans une seule table
- Une norme courante consiste à suffixer leurs noms par « _uk »

Contraintes de clé unique (syntaxe)

- Syntaxe :

```
ALTER TABLE nom_table ADD
CONSTRAINT nom_contrainte
UNIQUE (nom_colonne,…,nom_colonne);
```

- Exemple : nom, prénom et n° de tel. doivent composer une valeur unique

```
ALTER TABLE etudiant ADD
CONSTRAINT etudiant_uk
UNIQUE (nom_etudiant, prenom_etudiant,
  no_tel_etu);
```

Contraintes de clé étrangère

- Garantit que chaque valeur de la clé étrangère dans la table enfant existe dans une clé primaire ou unique précédemment définie dans la table parent
- Assure également que les mises à jour et suppressions dans la clé unique ou primaire de la table parent ne sont permises que s'il n'y a pas d'enregistrements enfants
- Une exception à cette règle est l'option DELETE CASCADE, qui génère une suppression automatique de tous les enregistrements enfants à la suppression de l'enregistrement parent

Contraintes de clé étrangère (syntaxe)

```
ALTER TABLE nom_table_enfant ADD
   CONSTRAINT nom_contrainte
   FOREIGN KEY (nom_colonne,…,nom_colonne)
   REFERENCES nom_table_parent
   [(nom_colonne,…,nom_colonne)]
   [ON DELETE action][ON UPDATE action;
```

- Une clé étrangère ne peut référencer que des clés primaires ou uniques existant dans la table parent ; la clé primaire est référencée par défaut si aucune colonne parent n'est spécifiée dans la liste de la clé étrangère
- La convention courante consiste à terminer les noms des clés étrangères par le suffixe _fk

On DELETE/UPDATE

- `[ON DELETE action] [ON UPDATE action]}`
- Valeurs de 'action' :
  - No action
  - Cascade
  - Set null
  - Set default

Contraintes de clé étrangère (exemple)

Toute nomination d'un professeur à un cours entraîne le fait que le professeur ne peut être supprimé de la table des professeurs.

```
ALTER TABLE nomination ADD
CONSTRAINT nomination_fk1
FOREIGN KEY (id_prof)
REFERENCES prof (id_prof);
```

- Nota : la création de la FK est impossible si la PK sur id_prof n'est pas réalisée auparavant

Contraintes CHECK

- Elles assurent le respect des règles de gestion sur la base des constantes et des valeurs de colonne de la ligne insérée ou mise à jour
- Syntaxe :

```
ALTER TABLE nom_table ADD
CONSTRAINT nom_contrainte
CHECK (condition);
```

- Par convention, on termine les contraintes CHECK par le suffixe « _ck »

Contraintes CHECK

- Le sexe doit être « F » ou « M » dans la table etudiant

```
ALTER TABLE etudiant ADD
CONSTRAINT sexe_ck
CHECK (sexe_etudiant IN ('F', 'M'));
```

- Les contraintes Check sont limitées à de simples validations par rapport à des constantes ou des valeurs des colonnes de la rangée courante, mais il n'est pas possible de valider les données par rapport au contenu d'autres rangées ou d'autres tables (il faut utiliser des triggers)

Supprimer des contraintes

- Syntaxe :

```
ALTER TABLE nom_table DROP CONSTRAINT
  nom_contrainte;
```

- Exemple :

```
ALTER TABLE etudiant DROP CONSTRAINT
  sexe_ck;
```

- Une façon plus simple de supprimer des contraintes est d'utiliser la syntaxe suivante, où l'option CASCADE supprime toutes les contraintes de clé étrangère

```
ALTER TABLE nom_table DROP PRIMARY KEY
  CASCADE;
```

Activation/Désactivation de contraintes

- Une contrainte peut être temporairement désactivée puis réactivée ultérieurement en utilisant la syntaxe

```
ALTER TABLE nom_table DISABLE
  CONSTRAINT nom_contrainte

ALTER TABLE nom_table ENABLE
  CONSTRAINT nom_contrainte
```

Exemple

```
CREATE TABLE realisateur(idreal INTEGER, nomreal
  VARCHAR(30),prenomreal VARCHAR(30), annaisreal
  INTEGER, PRIMARY KEY(idreal));

CREATE TABLE film (idfilm INTEGER PRIMARY KEY,
  titre VARCHAR(30),annee INTEGER, idreal INTEGER,
  FOREIGN KEY (idreal) REFERENCES realisateur
  (idreal));
```

LDD - Supprimer un schéma, une table

- Suppression d'un schéma :

```
DROP SCHEMA nomschema;
```

- Suppression d'une table du schéma à l'aide de l'instruction :

```
DROP TABLE nomtable;
```

LDD - Modifications

- Ajouter un attribut :

```
ALTER TABLE table_name ADD COLUMN attribute_name data_type;
```

- Supprimer un attribut :

```
ALTER TABLE table_name DROP COLUMN attribute_name;
```

- Ajouter une contrainte :

```
ALTER TABLE table_name ADD CHECK (attribute_name condition);

ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE
   (attribute_name);

ALTER TABLE table_name ADD FOREIGN KEY (attribute_name)
   REFERENCES table_name ( attribute_name);
```

LDD – Modifications (2)

- Supprimer une contrainte :

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

- Modifier la valeur par défaut :

```
ALTER TABLE table_name ALTER COLUMN attribute_name SET
   DEFAULT value;
```

- Renommer un attribut :

```
ALTER TABLE table_name RENAME COLUMN old_attribute_name TO
   new_attribute_name;
```

- Renommer une table :

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

5.2 SQL LMD

OlivierCuré *[ocure@univ-mlv.fr]*

SELECT

- La requête SELECT se compose de clauses SELECT et FROM qui sont toujours obligatoires
- Une requête SELECT de base se compose de 4 clauses, qui doivent apparaître dans l'ordre suivant :
  - `SELECT`
  - `FROM`
  - `WHERE`
  - `ORDER BY`
- Les clauses WHERE et ORDER BY sont facultatives et peuvent être ajoutées à SELECT FROM, d'autres clauses seront présentées plus tard dans le cours

SELECT (syntaxe)

- Syntaxe simplifiée :

```
SELECT nom_colonne,…,nom colonne | *
FROM nom_table,…,nom_table;
```

- L'instruction peut comprendre soit une liste de colonnes séparées par des virgules, soit par un seul astérisque, mais pas les deux
- Les colonnes sont affichées dans lequel elles sont listées
- L'astérisque indique que toutes les colonnes d'une table spécifiée seront extraites
- Les noms de tables sont séparés par des virgules

SELECT (exemple)

idfilm	titre	annee	idreal
1	Alien	1979	1
2	Reservoir Dogs	1992	2
3	The thing	1982	3
4	Volte-face	1997	4
5	Pulp fiction	1995	2
6	Terminator	1984	5
7	Ghost of Mars	2001	3
8	Mad Max	1979	6
9	Mad Max2	1981	6

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945

```
SELECT nomreal, titre

FROM realisateur r, film f

WHERE prenomreal LIKE
   'John' AND
   r.idreal=f.idreal;
```

Résultat

titre	nomreal
The thing	Carpenter
Volte-face	Woo
Ghost of Mars	Carpenter

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

Préfixer une colonne

- Utilisé pour spécifier la table à laquelle une colonne appartient
- Syntaxe : table.colonne
- Il est nécessaire lorsque deux tables possèdent un nom de colonne identique et que l'une des colonnes est référencée dans une instruction SQL ; sans requête, la requête serait ambiguë
- Exemple : id_etudiant existe dans au moins deux tables; le préfixe est donc obligatoire

SELECT etudiant.id_etudiant

FROM etudiant, compte_etudiant ;

(Nota : les jointures seront abordées ultérieurement)

La notion d'alias

- On les utilise pour renommer un objet à l’intérieur d’une instruction SQL
- Ils sont de deux types :

- Alias de colonne, utilisés pour remplacer des noms de colonnes longs ou énigmatiques

- Alias de table, utilisés pour remplacer des noms de table longs

La notion d'alias (syntaxe)

- Syntaxe :

```
SELECT nom_colonne [AS] alias_colonne,…,nom_colonne
  [AS] alias_colonne
FROM nom_table alias_table,…,nom_table alias_table
```

- Toute colonne ou table peut être suivie d'un alias
- Les alias qui contiennent plus d'un mot doivent être entre doubles quotes
- Les alias de colonne ne peuvent être référencés que dans la clause ORDER BY (non dans WHERE)

La notion d'alias (exemple)

- Les préfixes de colonnes améliorent la performance, en raison de la réduction du temps d'analyse de la requête

```
/* Exécute une requête sur la table étudiant en
  plaçant les colonnes id_etudiant et nom_etudiant
  respectivement par N0 et Nom de famille*/

   SELECT id_etudiant N0, nom_etudiant '' Nom de
     famille'' FROM etudiant;

/* Utilise des alias de table pour l'exemple
  précédent comportant deux tables*/

   SELECT e.id_etudiant, c.id_etudiant FROM
     etudiant e, compte_etudiant c;
```

DISTINCT

- On utilise DISTINCT pour s'assurer qu'une seule occurrence de chaque ensemble de lignes dupliquées est renvoyée
- Le SELECT DISTINCT s'applique à toutes les colonnes spécifiées dans la requête
- DISTINCT force le tri des résultats
- Syntaxe :

```
SELECT DISTINCT nom_colonne,
  …,nom_colonne | * FROM table
```

DISTINCT (exemple)

- Toutes les villes qui comptent un ou plusieurs professeurs

```
SELECT p.ville_prof FROM prof p;
```

- Liste unique des villes où habitent les professeurs

```
SELECT DISTINCT p.ville_prof FROM
  prof p;
```

Clause WHERE

- La clause WHERE sert à filtrer une ou des tables
- Syntaxe :

```
SELECT nom_colonne,…,nom_colonne
FROM table
WHERE condition;
```

- Exemple :

```
SELECT prenom_etudiant, sexe_etudiant
FROM etudiant
WHERE sexe_etudiant='F' ;
```

Nota : F est entre simples quotes

Université de Marne-la-vallée

Opérateurs de comparaison

Opérateur	Signification
IN	Egal à n'importe quel membre d'un ensemble
BETWEEN x AND y	Supérieur ou égal à x et inférieur ou égal à y
LIKE	Concordance de modèles (pattern matching)
IS NULL	Teste si une colonne est NULL

L'opérateur IN

- On utilise IN pour comparer une colonne à une liste de valeurs
- Retourne vrai si la valeur de colonne est égale à une valeur de la liste
- NOT IN retourne vrai si la valeur de colonne n'est égale à aucune valeur de la liste
- Exemple :

```
SELECT p.id_prof

FROM prof p

WHERE p.ville_prof IN ('NEW YORK', 'SAN
FRANCISCO');
```

L'opérateur BETWEEN

- On utilise BETWEEN pour comparer une colonne à un intervalle inclusif
- Retourne vrai si la valeur de colonne est comprise dans l’intervalle, bornes comprises
- NOT BETWEEN retourne vrai si la valeur est hors des bornes

L'opérateur BETWEEN (exemple)

- Exemples :

```
SELECT prix_cursus
FROM cursus
WHERE prix_cursus BETWEEN 2000 AND 4000;
-------------------------------------
SELECT prix_cursus
FROM cursus
WHERE prix_cursus NOT BETWEEN 3000 AND 4000;
```

L'opérateur LIKE

- On utilise LIKE pour comparer des colonnes de type caractère à une chaîne en utilisant la concordance de modèles
- Retourne vrai si la valeur de la colonne correspond à la chaîne
- NOT LIKE retourne vrai si la valeur ne correspond pas à la chaîne
- La chaîne peut contenir des caractères génériques
- Le souligné (_) correspond exactement à un caractère quelconque
- Le signe pourcentage (%) correspond à zéro caractères ou plusieurs

L'opérateur LIKE (exemple)

- Extraction des universités dont le pays commence par « U »

```
SELECT id_universite, nom_université FROM universite

WHERE pays_universite LIKE 'U%';
```

- Extraction des universités dont la ville ne termine pas par «RD»

```
SELECT u.id_universite, u.ville_universite FROM
  université u WHERE u.ville_universite NOT LIKE '%RD';
```

- Extraction des professeurs dont le nom commence par J, suivi exactement de 3 caractères et se terminant par E

```
SELECT nom_prof FROM prof WHERE nom_prof LIKE 'J___E';
```

La valeur NULL

- Les valeurs sont parfois :
  - Inconnues.
  - Non applicables.
- SQL permet l'utilisation de la valeur "null". Celle-ci impose quelques nouvelles notions :
  - Comparaison avec `is [not] null`.

La valeur NULL (2)

- Apparition de nouveaux opérateurs (jointures externes).
- Une logique avec 3 valeurs (true, false et null).

AND	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

OR	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

L'opérateur IS NULL

- Une expression arithmétique qui contient un NULL renvoie toujours NULL, par exemple ajouter 10 à NULL renvoie toujours NULL
- L’opérateur sert à tester si une colonne contient un valeur (quelle qu’elle soit)

L'opérateur IS NULL (exemple)

```
SELECT p.nom_prof,
  p.adr1_prof,p.adr2_prof
FROM prof p
WHERE p.adr2_prof IS NULL;
-------
SELECT p.id_prof, p.nom_prof
FROM prof p
WHERE p.adr1_prof IS NOT NULL;
```

Clause WHERE à conditions multiples

- On combine les conditions multiples grâce aux opérateurs logiques AND et OR
- Une clause WHERE peut contenir un nombre illimité de conditions combinées

L'opérateur AND

- AND combine deux conditions et ne renvoie un résultat que si les deux conditions sont vérifiées
- Exemple :

```
SELECT p.nom_prof
FROM prof p
WHERE p.sexe_prof = 'F'
AND p.pays_prof = 'Canada';
```

L'opérateur OR

- OR combine deux conditions et renvoie un résultat si l'une des deux conditions est vérifiée
- Exemple:

```
SELECT nom_etudiant
FROM etudiant
WHERE datenais_etu > '01/01/1970'
OR datenais_etu < '31/12/1960';
```

  - Dans cette requête il y a un piège si l'on utilise OR ou bien AND. Quel est-il ?

L'opérateur NOT

- NOT est utilisé pour inverser une condition
- Exemple : extraire les comptes dont la facture ou le versement n'est pas supérieur à 2.000

```
SELECT id_etudiant FROM compte

WHERE NOT (montant_facture>2000 OR
  montant_versement>2000);
```

Nota : NOT(condition_A AND condition_B) est équivalent à NOT condition_A OR NOT condition_B

Règles de précédence des opérateurs logiques

- La précédence est l'ordre dans lequel les expressions sont évaluées : celle-ci est, par défaut et par ordre décroissant NOT,puis AND,puis OR
- Les parenthèses servent à outrepasser la précédence par défaut

Règles de précédence des opérateurs logiques (exemple)

- Exemple : extraction des étudiants nés avant le 1er janvier 1960 ou après le 1er janvier 1980 et dont le numéro de téléphone commence par 4

```
SELECT nom_etudiant
FROM etudiant
WHERE no_tel_etu LIKE  ’4%’
AND (datenais_etu > ’01/01/60’
OR datenais_etu < ’01/01/80’);
```

Les jointures

- Il est souvent nécessaire d'extraire des informations figurant dans plusieurs tables et de les regrouper dans un seul listing ; ceci est réalisé par des jointures entre tables spécifiées dans la clause FROM
- Attention ! Même si la base de données comprend des relations « en dur », elles ne sont pas prises en compte dans les requêtes

Les jointures (2)

- Les tables spécifiées dans la clause FROM sont automatiquement combinées en produit cartésien
- Le produit cartésien est la combinaison de chaque ligne de la première table avec chaque ligne de la seconde
- La jointure utilise une clause WHERE pour restreindre le produit cartésien

Jointure conditionnelle (exemple)

- La jointure doit se baser sur une condition qui met en correspondance les deux colonnes correspondantes dans les deux tables

```
select i.no_inscription,
  e.nom_etudiant
from inscription i, etudiant e
where i.id_etudiant=e.id_etudiant;
```

Clause ORDER BY

- ORDER BY trie les lignes à extraire avant qu'elles ne soient affichées
- Syntaxe simplifiée :

ORDER BY expression [ASC | DESC]

- Le tri est croissant par défaut
- L'ordre des expressions détermine la priorité du tri

Clause ORDER BY (exemple)

- Tri des etudiants dans l'ordre décroissant des villes, puis classer les etudiants de la même ville par date de naissance :

```
SELECT nom_etudiant, ville_etudiant,
  datenais_etu from etudiant

order by ville_etudiant DESC,
  datenais_etu;
```

Nota : les valeurs NULL apparaissent en bas d'un tri en ordre décroissant (les plus élevées)

Tri par expression

- Trier les comptes des étudiants par leur reste à payer

```
SELECT no_compte_etu,
  montant_facture -
  montant_versement SOLDE

from compte

order by montant_facture -
  montant_versement ;
```

Tri par alias de colonne

- Trier les instructeurs par rémunération totale

```
SELECT no_compte_etu,
  montant_facture -
  montant_versement SOLDE
from compte order by solde;
```

Les fonctions d'aggrégats

- Elles retournent un résultat par groupe de lignes au lieu d’un résultat par ligne
  - Pour des raisons de simplicité, nous supposerons qu’un seul groupe est extrait
  - Une requête SELECT peut extraire des groupes multiples en utilisant une clause GROUP BY, qui sera abordée plus loin
- Fonctions agrégats courantes :
  - MAX, MIN, SUM, AVG, COUNT

Les fonctions MIN et MAX

- Syntaxe simplifiée et règles
  - MAX(expr) | MIN(expr)
  - Retourne la valeur minimum ou maximum de toutes les valeurs de expr
- Exemples

```
select min(ville_etudiant) from
  etudiant;

select max(c.prix_cursus)
    from cursus c;
```

Les fonctions SUM et AVG

- Syntaxe simplifiée et règles
  - SUM(expr) | AVG(expr)
  - Retourne la somme ou la moyenne de toutes les valeurs de n
  - Les valeurs NULL ne sont pas comprises dans le calcul
- Exemples

```
SELECT SUM(salaire) FROM prof;

SELECT AVG(salaire) FROM prof;
```

La fonction COUNT

- Syntaxe et règles
  - COUNT([DISTINCT] expr) |COUNT(*)
  - Retourne le nombre de lignes où expr n'est pas NULL
  - DISTINCT retourne le nombre total de lignes uniques où expr n'est pas NULL
  - Astérisque retourne le nombre total de lignes, y compris les doublons et les NULLs

Regroupement de données

- Un besoin courant consiste à totaliser l'information à un niveau de groupe, c'est l'utilité de la fonction GROUP BY
- Syntaxe simplifiée
  - `SELECT liste-select`
  - `FROM liste-table`
  - `[WHERE conditions]`
  - `[GROUP BY nom_colonne,…,nom_colonne]`
  - `[ORDER BY liste-tri];`

Regroupement de données

- Règles
  - GROUP BY doit apparaître après la clause WHERE et avant la clause ORDER BY
  - Toutes les colonnes de la clause SELECT doivent être dans la clause GROUP BY ou faire partie d'une fonction d'agrégat
  - Si des valeurs `null` se trouvent dans l'attribut de regroupement, un groupe est créé pour celles-ci.

GROUP BY (exemple)

- Pour calculer le nombre d'étudiants par pays...
  - Il faut grouper les étudiants par pays
  - Puis calculer le compte

```
select pays_etudiant,
  count(id_etudiant)
    from etudiant
    group by pays_etudiant;
```

GROUP BY (exemple)

- Comment n'extraire que les groupes (pays_etudiant) qui comptent plus de 10 étudiants ? En fait c'est impossible avec les seules clauses déjà connues car :
  - Impossible d'utiliser la clause WHERE parce que la condition doit être basée sur le compte de toutes les lignes du groupe
  - Impossible d'utiliser des fonctions agrégats dans la clause WHERE
  - Les conditions de la clause WHERE s'appliquent à chaque ligne avant qu'elles ne soient groupées
  - D'où…

La clause HAVING

- On utilise la clause HAVING pour restreindre les groupes de lignes qui sont retournés en spécifiant des conditions sur les résultats de ces groupes
- Syntaxe simplifiée :

```
SELECT liste-select
FROM liste-table
[WHERE conditions]
[GROUP BY nom_colonne,…,nom_colonne]
[HAVING conditions]
[ORDER BY liste-tri];
```

Nota : on ne peut pas utiliser de fonctions agrégats dans la clause WHERE

La clause HAVING (exemple)

- Pour compter le nombre d'étudiants par pays en ne prenant en considération que les groupes dont le nombre est supérieur à 10 (même exemple que précédemment, mais avec une condition sur le groupe)

```
select pays_etudiant, count(id_etudiant)
    from etudiant
    group by pays_etudiant
    having count(id_etudiant)>10;
```

GROUP BY avec jointures

- On peut utiliser des jointures en conjonction avec un `group by`.
- L'exécution du `group` by et autres fonctions est réalisée après la jointure des relations.
- Exemple :

```
SELECT annee FROM film as f, realisateur as r
  WHERE f.idDir = r.idDir GROUP BY annee HAVING
  annee=1979 // 1979 2
```

Union dans SQL

- Contrainte : les ensembles des n-uplets doivent être compatibles dans l'union.
- Syntaxe : les films de 'Carpenter' ou 'Miller'.

```
SELECT nomFilm FROM film f, realisateur r WHERE
  r.nom LIKE 'Miller' and r.idreal=f.idreal

UNION

SELECT nomFilm FROM film f, realisateur r WHERE
  r.nom LIKE 'Carpenter' and r.idreal=f.idreal
```

Union, Intersect et Différence dans SQL

- On pouvait aussi écrire :

```
SELECT nomFilm FROM film f, realisateur r WHERE
  (r.nom LIKE 'Carpenter' or r.nom LIKE 'Miller'
  and r.idreal=f.idreal);
```

- Même contrainte pour l'intersection. Instruction : `INTERSECT`.
- Et la différence avec la syntaxe `EXCEPT` (postgresql) ou `MINUS` (Oracle) suivant le SGBDR.

Union, Intersect et except dans SQL (2)

- Par défaut, suppression des duplicats, comme dans l'algèbre relationnel.
- `UNION ALL`, `INTERSECT ALL` et `EXCEPT ALL` permettent de garder les duplicats.

LMD – Insertion

Nom de la table où l'on ajoute les n-uplets.

Liste des attributs à renseigner.

```
INSERT INTO table (att1,
  att2, .., attn) VALUES (val1,
  val2, ..., valn);

INSERT INTO table  VALUES (val1,
  val2, ..., valm);
```

Les valeurs du nouveau n-uplet.
Il faut respecter l'ordre de la définition
ou de la description de la table.

LMD – Mise-à-jour

Nom de la table modifiée

Modification de(s) la valeur(s).
Attributs séparés par des ",".

```
UPDATE table

SET attribute=value

WHERE condition;
```

Condition permettant d'identifier le(s) n-uplet(s).
Le plus souvent, opération sur la clé primaire.

Exemple -Mise-à-jour

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945

```
UPDATE realisateur SET
   annaisreal = '1951',
   prenomreal = 'Jon' WHERE
   idreal=3;
```

Résultat

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	Jon	1951
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

Exemple - Insertion

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945

```
INSERT INTO
   realisateur(idreal,nom
   real,prenomreal)

VALUES (7,'Raimi','Sam');

OU

INSERT INTO realisateur

VALUES
   (7,'Raimi','Sam',null)
   ;
```

Résultat

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945
7	Raimi	Sam	

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

LMD – Suppression

Nom de la table où l'on supprime le(s) n-uplet(s).

```
DELETE FROM table

WHERE condition;
```

Identification du n-uplet à supprimer.
Souvent une opération sur la clé primaire.

Exemple - Insertion

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945
7	Raimi	Sam	

```
DELETE FROM realisateur
WHERE idreal = 7;
```

Résultat

idreal	nomreal	prenomreal	annaisreal
1	Scott	Ridley	1943
2	Tarantino	Quentin	1963
3	Carpenter	John	1948
4	Woo	John	1946
5	Cameron	James	1954
6	Miller	George	1945

Université de Marne-la-vallée

OlivierCuré *[ocure@univ-mlv.fr]*

Création de la BD Company

- create table department (dnum int primary key, lib varchar(30));
- create table project(pnum int primary key, pnom varchar(20), dno int, foreign key (dno) references department (dnum));
- create table employee (nss int primary key, nom varchar(20), dno int, foreign key (dno) references department (dnum));
- create table works_on (enss int , pno int, heures decimal(3,1), primary key(enss,pno), foreign key (enss) references employee (nss), foreign key (pno) references project (pnum));

Enrichissements de la BD Company

```
 /* Valeurs de departement */
insert into department values(5,'research');
insert into department values(4,'Administration');
insert into department values(1,'HeadQuarters');
/* Instances de project */
insert into project values(1,'ProductX',5);
insert into project values(2,'ProductY',5);
insert into project values(3,'ProductZ',5);
insert into project
  values(10,'Computerization',4);
 insert into project
  values(20,'Reorganization',1);
 insert into project values(30,'NewBenefits',4);
```

Enrichissements de la BD Company (2)

```
insert into employee values(1,'Smith',5);
insert into employee values(2,'Wong',5);
insert into employee values(3,'Zelaya',4);
insert into employee values(4,'Wallace',4);
insert into employee values(5,'Narayan',5);
insert into employee values(6,'Borg',1);
insert into works_on values(2,3,10);
insert into works_on values(2,10,10);
insert into works_on values(2,20,10);
insert into works_on values(3,10,30);
insert into works_on values(3,30,10);
```

Données de Company

department

dnum	lib
5	research
4	Administration
1	HeadQuarters

employee

nss	nom	dno
1	Smith	5
2	Wong	5
3	Zelaya	4
4	Wallace	4
5	Narayan	5
6	Borg	1

project

pnum	pnom	dno
1	ProductX	5
2	ProductY	5
3	ProductZ	5
10	Computerization	4
20	Reorganization	1
30	Newbenefits	4

works_on

enss	pno	heures
1	1	32.5
1	2	7.5
2	2	10.0
2	3	10.0
2	10	10.0
2	20	10.0
3	10	30.0
3	30	10.0
4	3	40.0

Requêtes imbriquées

- Une requête où la clause `WHERE` contient une sous-requête `SELECT`.
- Utilité : lorsqu'il faut chercher une valeur dans la bd et l'utiliser dans une condition.
- On utilise parfois l'opérateur `IN` qui compare une valeur avec un ensemble de valeurs du même type.

Requêtes imbriquées (2)

```
SELECT .. FROM .. WHERE attx IN (SELECT
  attx FROM ..);
```

- L'opérateur `IN` compare une valeur v à un ensemble de valeurs V. Evaluation à TRUE s'il y a une correspondance. (`NOT IN`).
- Exemple :

```
SELECT pnum FROM project WHERE pnum IN
  (SELECT pnum FROM works_on WHERE enss=1
  AND heures>20); // réponse : 1
```

Université de Marne-la-vallée

Requêtes imbriquées (3)

```
SELECT .. FROM .. WHERE attx = (SELECT
  attx FROM ..);
```

- L'opérateur = compare une valeur v à une autre valeur, et non un ensemble.
- Exemple :

```
SELECT pnum FROM project WHERE pnum = (SELECT pnum
  FROM works_on WHERE enss=1 AND heures>20); //
  réponse : 1

SELECT pnum FROM project WHERE pnum = (SELECT pnum
  FROM works_on WHERE enss=1 AND heures>3); // Pas
  de réponse
```

Université de Marne-la-vallée

Requêtes imbriquées (4)

```
SELECT .. FROM .. WHERE attx = (SELECT
  attx FROM ..);
```

- L'opérateur de comparaison `IN` est identique à `=ANY` et `=SOME`.
- Soit op $\in$ {>,>=,<,<=,<>}. On peut écrire : `WHERE att1 op ALL|ANY (SELECT ..)`.
- v > ALL V retourne TRUE si la valeur de v est supérieure à toutes les valeurs de l'ensemble V.

```
SELECT pnum FROM project WHERE pnum > ALL
  (SELECT pnum FROM works_on);
```

EXISTS, NOT EXISTS

- Un autre opérateur de comparaison sur les ensembles.
- Contrôle si le résultat d'une requête imbriquée est vide (ne contient pas de n-uplets) ou non vide.
- Exemple:

```
SELECT nom FROM employee e WHERE
  EXISTS (SELECT enss FROM works_on
  w WHERE e.nss=w.enss);
```

Jointure, format général

```
SELECT (column_list) FROM
  table_name [INNER | {LEFT | RIGHT
  | FULL } OUTER ] JOIN table_name
  ON qualification_list WHERE ...
```

- Par défaut, une jointure interne (inner join).

Jointures

- SELECT * FROM employee e, works_on w WHERE e.nss = w.enss;
- SELECT * FROM employee e JOIN works_on w ON e.nss=w.enns;

```
nss | nom | dno | enss | pno | heures
-----+---------+-----+------+-----+--------
1 | Smith | 5 | 1 | 1 | 32.5
1 | Smith | 5 | 1 | 2 | 7.5
2 | Wong | 5 | 2 | 2 | 10.0
2 | Wong | 5 | 2 | 3 | 10.0
2 | Wong | 5 | 2 | 10 | 10.0
2 | Wong | 5 | 2 | 20 | 10.0
3 | Zelaya | 4 | 3 | 10 | 30.0
3 | Zelaya | 4 | 3 | 30 | 10.0
4 | Wallace | 4 | 4 | 3 | 40.0
```

Jointures externes

```
SELECT nss, nom FROM employee e
  JOIN works_on w ON e.nss=w.enns;
```

nss	nom
1	Smith
2	Wong
3	Zelaya
4	Wallace

```
SELECT DISTINCT nss, nom FROM
  employee e LEFT OUTER JOIN
  works_on ON e.nss=w.enss;
```

nss	nom
1	Smith
2	Wong
3	Zelaya
4	Wallace
5	Narayan
6	Borg

Contraintes d'intégrités

- DROP TABLE works_on;
- DROP TABLE employee;
- DROP TABLE project;
- DROP TABLE department;
- CREATE TABLE department (dnum INT PRIMARY KEY, lib VARCHAR(30));
- CREATE TABLE project (pnum INT PRIMARY KEY, pnom VARCHAR(20), dno INT, FOREIGN KEY (dno) REFERENCES department (dnum) ON DELETE CASCADE);
- CREATE TABLE employee (nss INT PRIMARY KEY, nom VARCHAR(20), dno INT, FOREIGN KEY (dno) REFERENCES department (dnum) ON DELETE CASCADE);
- CREATE TABLE works_on (enss INT, pno INT, heures DECIMAL(3,1), PRIMARY KEY(enss,pno) FOREIGN KEY (enss) REFERENCES employee (nss) ON DELETE CASCADE, FOREIGN KEY (pno) REFERENCES project (pnum) ON DELETE CASCADE);

Contraintes d'intégrités (2)

- DELETE FROM department WHERE lib ILIKE 'res%';

employee

nss	nom	dno
3	Zelaya	4
4	Wallace	4
6	Borg	1

(3 rows)

project

pnum	pnom	dno
10	Computerization	4
20	Reorganization	1
30	Newbenefits	4

(3 rows)

works_on

enss	pno	heures
3	10	30.0
3	30	10.0

Contraintes d'intégrités (3)

- DROP TABLE works_on;
- DROP TABLE employee;
- DROP TABLE project;
- DROP TABLE department;
- CREATE TABLE department (dnum INT PRIMARY KEY, lib VARCHAR(30));
- CREATE TABLE project (pnum INT PRIMARY KEY, pnom VARCHAR(20), dno INT, FOREIGN KEY (dno) REFERENCES department (dnum) ON DELETE SET NULL on UPDATE SET NULL);
- CREATE TABLE employee (nss INT PRIMARY KEY, nom VARCHAR(20), dno INT, FOREIGN KEY (dno) REFERENCES department (dnum) ON DELETE SET NULL on UPDATE SET NULL);
- CREATE TABLE works_on (enss INT, pno INT, heures DECIMAL(3,1), PRIMARY KEY(enss,pno) FOREIGN KEY (enss) REFERENCES employee (nss) ON DELETE CASCADE, FOREIGN KEY (pno) REFERENCES project (pnum) ON DELETE CASCADE);

Contraintes d'intégrités (4)

- DELETE FROM department WHERE lib ILIKE 'res%';

works_on

enss	pno	heures
1	1	32.5
1	2	7.5
2	2	10.0
2	3	10.0
2	10	10.0
2	20	10.0
3	10	30.0
3	30	10.0
4	3	40.0

(9 rows)

employee

nss	nom	dno
3	Zelaya	4
4	Wallace	4
6	Borg	1
1	Smith	
2	Wong	
5	Narayan	

(6 rows)

project

pnum	pnom	dno
10	Computerization	4
20	Reorganization	1
30	Newbenefits	4
1	ProductX	
2	ProductY	
3	ProductZ	

Contraintes d'intégrités (6)

- UPDATE department SET dnum=6 WHERE dnum=5;

employee

nss	nom	dno
3	Zelaya	4
4	Wallace	4
6	Borg	1
1	Smith	
2	Wong	
5	Narayan	

(6 rows)

project

pnum	pnom	dno
10	Computerization	4
20	Reorganization	1
30	Newbenefits	4
1	ProductX	
2	ProductY	
3	ProductZ	

(6 rows)

works_on

enss	pno	heures
1	1	32.5
1	2	7.5
2	2	10.0
2	3	10.0
2	10	10.0
2	20	10.0
3	10	30.0
3	30	10.0
4	3	40.0

Quelques requêtes

- `UPDATE employee SET dno = (SELECT dnum FROM department WHERE lib LIKE 'res%');`
- `INSERT INTO employee select 7,'toto', dnum from department where lib like 'res%';`
- `DELETE FROM project WHERE pnum = (SELECT pno FROM works_on WHERE enss=1);`