HMIN112M : Standard SQL

I.Mougenot

UM Faculté des Sciences Département Informatique

2020

Le langage du relationnel

SQL : Structured Query Language

- Standard : partagé par l'ensemble des SGBDR (Oracle mais aussi MySQL, ou PostgreSQL, ou SQL Server, ou . . .)
- Evolution du langage : SQL1 (SQL 89), SQL2 (SQL 92), SQL3 (SQL 99), dernière version en 2011
- 3 grands rôles
  1. manipulation des données (LMD)
  2. définition des données et de leurs structures (LDD)
  3. contrôle des données (LCD)

UNIVERSITÉ DE MONTPELLIER

SQL : langage de manipulation des données

Consultation (requêtage) de données provenant de une ou de plusieurs relations (tables)

```
SELECT [ALL | DISTINCT] { liste <colonne/
    expression> | * }
FROM liste [<proprietaire>.] { <table> | <vue> }
     [<alias >]
[WHERE <condition>]
[GROUP BY {liste expression }
                  [HAVING condition]] ;
```

Listing 1: Forme générale et partielle d'une requête (BNF)

Schéma exemple

Sur lequel seront construits les exemples

Un exemple réduit

Etudiant(<u>numINE</u>, nom, prenom, genre, age)
Enseignant(<u>codeE</u>, nom, prenom, genre, disc)
Module(<u>codeM</u>, libelle, nbreECTS)
Inscrit_dans(<u>numINE,codeM,annee</u>)
Enseigne_dans(<u>codeE,codeM,annee</u>)

En extension

Etudiant en extension

numINE	nom	prenom	genre	age
'201610'	'Dusol'	'Marie'	'f'	22
'201508'	'Dusol'	'Paul'	'm'	25
'201609'	'Bony'	'Paul'	'm'	22
'201201'	'Balard'	'Zoé'	'f'	29

Enseignant en extension

codeE	nom	prenom	genre	disc
'1_A'	'Dubois'	'Alice'	'f'	'info'
'2_B'	'Drapier'	'Paul'	'm'	'bio'
'3_B'	'Balard'	'Zoe'	'f'	'info'

Module en extension

codeM	libelle	nbreECTS
'HMIN112M'	'SI BD'	5
'HMIN111M'	'Programmation'	5

Inscrit_Dans en extension

numIne	codeM	annee
'201610'	'HMIN112M'	2020
'201610'	'HMIN111M'	2020
'201201'	'HMIN111M'	2020
'202002'	'HMIN111M'	2020

Consultation du contenu d'une table

par exemple Etudiant

```
SELECT *
FROM Etudiant ;

-- identique a :

SELECT numINE, nom, prenom, genre, age
FROM Etudiant ;
```

Listing 2: Consultation générale

Exemple d'une projection

Les nom et prénom des étudiants : $\Pi_{nom,prenom}(\text{Etudiant})$

```
SELECT  nom, prenom
FROM Etudiant ;
```

Listing 3: Projection sur nom et prénom

Autre exemple d'une projection avec un "distinct"

Les âges des étudiants : Π_{age}(Etudiant)

```
SELECT  distinct age
FROM Etudiant ;

-- distinct : pas de doublons
```

Listing 4: Projection sans doublons

Exemple d'une sélection

Etudiants de plus de 24 ans : $\sigma_{age>24}$(Etudiant)

```
SELECT  *
FROM Etudiant
WHERE age > 24 ;
```

Listing 5: Filtre sur l'âge

Ne pas confondre ordre SELECT et opération de sélection

Conjonction de conditions de sélection

Etudiants masculins de plus de 24 ans : $\sigma_{age>24\ and\ genre='m'}(\text{Etudiant})$

```
SELECT  *
FROM Etudiant
WHERE age > 24 AND genre = 'm' ;
```

Listing 6: Filtres sur l'âge et le genre

Disjonction de conditions de sélection

Etudiants masculins ou Etudiants de plus de 24 ans : $\sigma_{age>24\ or\ genre='m'}(\text{Etudiant})$

```
SELECT  *
FROM Etudiant
WHERE age > 24 OR genre = 'm' ;
```

Listing 7: Filtres sur l'âge ou le genre

Sélection avec recherche approchée

Code module qui commence par HMIN :
$\sigma_{codeM\ LIKE\ 'HMIN\%'}(\text{Module})$

```
SELECT  codeM, libelle
FROM Module
WHERE codeM LIKE 'HMIN%' ;
```

Listing 8: Opérateur LIKE

% correspond à toute chaîne de caractères y compris la chaîne vide

Exemple d'une sélection suivie d'une projection

nom et prénom des étudiants de plus de 24 ans : $\Pi_{nom,prenom}(\sigma_{age>24}(\text{Etudiant}))$

```
SELECT  nom, prenom
FROM Etudiant
WHERE age > 24 ;
```

Listing 9: Sélection et projection

Exemple d'une jointure

Jointure natuelle sur numINE : Etudiant ⋈ Inscrit_Dans

```
SELECT  *
FROM Etudiant, Inscrit_Dans
WHERE Etudiant.numINE = Inscrit_Dans.numINE ;

SELECT  *
FROM Etudiant E, Inscrit_Dans I
WHERE E.numINE = I.numINE ;

-- en SQL 92
SELECT  *
```

Produit cartésien

Appel des deux tables sans conditions de jointure pour le rapprochement

```
SELECT  *
FROM Etudiant, Inscrit_Dans ;
```

Listing 11: Produit Cartésien

Quand on oublie la condition de jointure, on obtient le produit cartésien ...

Jointure naturelle et projection et sélection

Jointure : $\Pi_{E.numINE,nom,prenom,codeM}(\sigma_{genre='f'}(\text{Etudiant E}) \bowtie \text{Inscrit\_Dans I})$

```
SELECT  Etudiant.numINE, nom, prenom, codeM
FROM Etudiant, Inscrit_Dans
WHERE Etudiant.numINE = Inscrit_Dans.numINE and
    genre ='f' ;

SELECT E.numINE, nom, prenom, codeM
FROM Etudiant E, Inscrit_Dans I
WHERE E.numINE = I.numINE and genre ='f' ;

SELECT  E.numINE, nom, prenom, codeM
```

Semi-jointure et SELECT imbriqué

Jointure : $\Pi_{E.numINE,nom,prenom}(\sigma_{genre='f'}(\text{Etudiant E}) \ltimes \text{Inscrit\_Dans I})$

```
SELECT distinct Etudiant.numINE, nom, prenom
FROM Etudiant, Inscrit_Dans
WHERE Etudiant.numINE = Inscrit_Dans.numINE and
    genre ='f' ;

SELECT  numINE, nom, prenom
FROM Etudiant WHERE  genre ='f' AND numINE IN (
    SELECT numINE FROM Inscrit_Dans);
```

Listing 13: Semi-jointure

Auto-jointure

Auto-Jointure sur l'âge : $\Pi_{E2.*}$(Etudiant E1) $\bowtie$ Etudiant E2)

```
SELECT distinct E2.*
FROM Etudiant E1, Etudiant E2
WHERE E1.numINE = '201610' AND E1.numINE <> E2.
    numINE AND E1.age=E2.age ;

SELECT distinct E2.*
FROM Etudiant E1 JOIN Etudiant E2 ON E1.age=E2.
    age
WHERE E1.numINE = '201610' AND E1.numINE <> E2.
    numINE ;
```

Listing 14: Auto-jointure

Union

Union sur (nom,prenom) Etudiants et Enseignants :
$\Pi_{nom,prenom}(\text{Etudiant}) \cup \Pi_{nom,prenom}(\text{Enseignant})$

```
SELECT nom, prenom
FROM Etudiant
UNION
SELECT nom, prenom
FROM Enseignant;
```

Listing 15: Union

Relations de même schéma et le distinct s'opère avec l'UNION

Différence

(nom,prenom) Etudiants qui ne sont pas Enseignants : $\Pi_{nom,prenom}$(Etudiant) - $\Pi_{nom,prenom}$(Enseignant)

```
SELECT nom, prenom
FROM Etudiant
MINUS
SELECT nom, prenom
FROM Enseignant;
```

Listing 16: Différence

Relations de même schéma

Intersection

(nom,prenom) Etudiants qui sont aussi Enseignants :
$\Pi_{nom,prenom}(\text{Etudiant}) \cap \Pi_{nom,prenom}(\text{Enseignant})$

```
SELECT nom, prenom
FROM Etudiant
INTERSECT
SELECT nom, prenom
FROM Enseignant;
```

Listing 17: Intersection

Relations de même schéma

Calculs sur les tuples et attributs des tuples

Recours à des fonctions arithmétiques

```
SELECT count(*) FROM Etudiant;

SELECT min(age), max(age), avg(age), count(
    numINE) FROM Etudiant;

SELECT min(age), max(age), avg(age), count(
    numINE) FROM Etudiant WHERE genre ='f';
```

Listing 18: Exemples d'appels

count(), min(), max(), avg(), ... ici un seul tuple est retourné

Calculs avec plusieurs tuples retournés : partitionnement

Agréger sur certains attributs et faire un calcul sur d'autres : GROUP BY

```
SELECT genre, count(*) as nombre FROM Etudiant
    GROUP BY genre;
SELECT genre, avg(age) as nombre FROM Etudiant
    WHERE age < 30 GROUP BY genre;
SELECT E.numINE, nom, count(codeM) FROM Etudiant
     E JOIN Inscrit_Dans I ON E.numINE = I.numINE
GROUP BY E.numINE, nom;
```

Listing 19: Partitionnement

Partitionnement avec condition sur le partitionnement

GROUP BY ..attributs... HAVING condition

```
SELECT E.numINE, nom, count(codeM) FROM Etudiant
    E JOIN Inscrit_Dans I ON E.numINE = I.numINE
GROUP BY E.numINE, nom having count(codeM)>=2;
SELECT E.numINE, nom, count(codeM) FROM Etudiant
    E, Inscrit_Dans I WHERE E.numINE = I.numINE
GROUP BY E.numINE, nom having count(codeM)>=2;
```

Listing 20: Partitionnement et condition

Affiner la réponse : ici seuls les étudiants qui sont inscrits dans au moins 2 modules sont retournés

Partitionnement avec select imbriqué

GROUP BY ..attributs... HAVING expression sous-select

```
SELECT numINE, count(codeM) FROM Inscrit_Dans
GROUP BY numINE having count(codeM) >= ALL (
    select count(codeM) FROM inscrit_dans group
    by numINE);

-- avec certains SGBD dont Oracle
SELECT numINE, count(codeM) FROM Inscrit_Dans
GROUP BY numINE having count(codeM)=(select max(
    count(codeM)) FROM inscrit_dans group by
    numINE);
```

Listing 21: Partitionnement et sous-select

Opérateurs de comparaison et quantificateurs

Utilisation concertée d'un opérateur arithmétique et d'un quantificateur (universel ($\forall$) et existentiel ($\exists$))

- *les opérateurs de comparaison* $=$, $<$, $>$, $<=$,$>=$, $<>$.
- Ces opérateurs notés (Θ) peuvent être quantifiés.
  Soit S une expression dénotant un ensemble :
  A Θ ANY(S) signifie $(\exists\ x)(S(x) \wedge A\ \Theta\ x)$
  A Θ ALL(S) signifie $(\forall\ x)(S(x) \wedge A\ \Theta\ x)$

Connecteurs logiques

ou opérateurs booléens

- AND (en logique $\wedge$) : exprime une conjonction
- OU (en logique $\vee$) : exprime une disjonction
- NOT (en logique $\neg$) : exprime une négation

Predicat BETWEEN

Teste l'appartenance ou la non appartenance à un intervalle de valeurs

```
SELECT numINE FROM Etudiant
WHERE age BETWEEN 20 AND 30 ;
-- identique a
SELECT numINE FROM Etudiant
WHERE age >= 20 AND age <= 30 ;

SELECT numINE FROM Etudiant
WHERE age NOT BETWEEN 20 AND 24 ;
```

Listing 22: BETWEEN

Predicat IN

Teste l'appartenance ou la non appartenance à un ensemble de valeurs

```
SELECT numINE FROM Etudiant
WHERE age IN (20,22,24) ;
-- identique a
SELECT numINE FROM Etudiant
WHERE age=20 OR age=22 OR age=24 ;
SELECT numINE FROM Etudiant
WHERE age NOT IN (20,22,24) ;
-- identique a
SELECT numINE FROM Etudiant
WHERE age <>20 AND age <>22 AND age <>24 ;
```

Predicat EXISTS

Teste la non vacuité d'un ensemble (test logique sur vide ou non vide)

```
SELECT numINE FROM Etudiant E
WHERE EXISTS (SELECT * FROM INSCRIT_DANS I WHERE
     E.numINE = I.numINE);

SELECT codeM FROM Module M
WHERE NOT EXISTS (SELECT * FROM INSCRIT_DANS I
    WHERE M.codeM = I.codeM);
```

Listing 24: EXISTS

Retourne une valeur booléenne : "true" ou "false"

Predicat NULL

Notion de valeur manquante : NULL est un valeur non renseignée (différente de zéro). Le test sur la valeur manquante se fait avec IS puisque ce n'est pas une valeur en tant que telle

```
SELECT numINE FROM Etudiant E
WHERE age IS NULL;
SELECT numINE FROM Etudiant E
WHERE prenom IS NOT NULL;
```

Listing 25: NULL

BDR : Hypothèse du monde clos : seules les données qui sont énoncées dans la base sont vraies

Notion de tri

Les tuples retournés dans la réponse peuvent être triés sur la valeur croissante ou décroissante d'un, ou de plusieurs attributs

```
SELECT * FROM ETUDIANT ORDER BY prenom;

SELECT * FROM ETUDIANT ORDER BY prenom DESC;

SELECT nom, prenom, age FROM ETUDIANT ORDER BY
    prenom,age DESC;

SELECT nom, prenom FROM ETUDIANT ORDER BY prenom
    ,age DESC;
```

Listing 26: ORDER BY

Traduire la division en SQL

Absence d'opérateur physique pour la division en SQL

nécessite de l'exprimer autrement :

1. double différence
2. double "not exists"
3. partitionnement

Retour sur l'exemple vu en algèbre relationnelle

$\Pi_{numEtudiant,codeModule}(\text{Inscrit\_Dans}) \div \Pi_{codeModule}(\text{Module})$

équivaut à :

$\Pi_{numEtudiant}(\text{Inscrit\_Dans}) - \Pi_{numEtudiant}((\Pi_{numEtudiant}(\text{Inscrit\_Dans}) \times \Pi_{codeModule}(\text{Module})) - \Pi_{numEtudiant,codeModule}(\text{Inscrit\_Dans}))$

Double Différence : basée sur la théorie

Renvoyer les étudiants qui sont inscrits dans au moins un module, et qui ne sont pas parmi les étudiants qui ne sont pas inscrits à au moins un module

```
CREATE OR REPLACE VIEW NonInscritsATout AS
SELECT numINE, m.codeM FROM Inscrit_Dans, Module
    m
MINUS
SELECT numINE, codeM FROM Inscrit_Dans ;

SELECT numINE FROM Inscrit_Dans
MINUS
SELECT numINE FROM NonInscritsATout;
```

Double Test de vacuité : NOT EXISTS

Une double négation équivaut à une affirmation : donner les étudiants tels qu'il n'y ait pas de modules auquels ils ne soient pas inscrits

```
SELECT NumINE FROM Etudiant E
WHERE NOT EXISTS (SELECT * FROM Module M
WHERE NOT EXISTS (SELECT * FROM Inscrit_Dans I
WHERE E.numINE = I.numINE AND M.codeM = I.codeM)
    );
```

Listing 28: Façon 2

Partitionnement

Retourner les étudiants qui sont inscrits à un nombre de modules égal au nombre total de modules

```
SELECT numINE FROM Inscrit_Dans GROUP BY numINE
    HAVING COUNT(CodeM)
= (SELECT COUNT(*) FROM MODULE);
```

Listing 29: Façon 3

Accès en écriture : insertion de tuples dans une table

Insertion d'un tuple pour tous les attributs de la table ou pour un sous-ensemble d'attributs

```
INSERT INTO Etudiant VALUES ('201610','Dusol','
    Marie','f',22);

INSERT INTO Etudiant (numINE, nom, prenom)
    VALUES ('201711','Dubois','Marc');
```

Listing 30: Insertion

Accès en écriture : suppression de tuples dans une table

Toujours le tuple dans son ensemble

```
DELETE FROM Etudiant WHERE numINE = '201711' ;

DELETE FROM Etudiant ;

DELETE FROM Etudiant WHERE numINE in (select
    numINE FROM
        INSCRIT_DANS WHERE codeM = 'HMIN112M');
```

Listing 31: Insertion

Accès en écriture : mise à jour de tuples dans une table

Toujours le tuple dans son ensemble

```
UPDATE Etudiant SET age = 27 WHERE numINE =
    '201711' ;

UPDATE Etudiant SET age = age + 1  ;

UPDATE Etudiant SET age = 27, genre='M' WHERE
    numINE = '201711' ;
```

Listing 32: Mise à jour

SQL : langage de définition de données

Création, mise à jour et suppression des objects du schéma (table, vue, contrainte, utilisateur, . . .)

```
CREATE
ALTER
DROP
```

Listing 33: Principaux ordres

SQL : CREATE

Création des objects du schéma

```
CREATE TABLE Etudiant
(numIne varchar(12) constraint Etudiant_PK
    PRIMARY KEY,
nom varchar(20), prenom varchar(20), age number
    (3), genre char(1)) ;

CREATE TABLE Modules
(codeM varchar(8) constraint Module_PK PRIMARY
    KEY,
libelle varchar(15), nbreECTS integer) ;
```

Listing 34: Exemple CREATE

SQL : CREATE

Création des objects du schéma

```
CREATE TABLE Inscrit_Dans
(numINE varchar(12),
codeM varchar(8), annee integer, PRIMARY KEY(
    numIne,codeM), CONSTRAINT ID_FK1
FOREIGN KEY (numINE) REFERENCES Etudiant(numINE)
     ON DELETE CASCADE, CONSTRAINT ID_FK2
FOREIGN KEY (codeM) REFERENCES Module(codeM) ON
    DELETE CASCADE) ;
```

Listing 35: Exemple CREATE

Enumération des types de contrainte

- Contraintes intra-table
  - PRIMARY KEY (UNIQUE + NOT NULL)
  - NULL / NOT NULL
  - CHECK
  - UNIQUE
- Contrainte inter-tables
  - FOREIGN KEY

Possibilité de nommer ou non la contrainte

SQL : ALTER

Evolution du schéma : reste limité pour ce qui concerne le relationnel

```
ALTER TABLE Etudiant ADD CONSTRAINT dom_genr
    CHECK (genre in ('f','m'));

ALTER TABLE Module ADD niveau varchar(4);
ALTER TABLE Module MODIFY niveau varchar(6);
ALTER TABLE Module DROP COLUMN niveau;
```

Listing 36: Exemple ALTER

SQL : ALTER

Activer/Désactiver ou encore supprimer une contrainte

```
ALTER TABLE Etudiant DISABLE CONSTRAINT dom_genr
    ;
ALTER TABLE Etudiant ENABLE CONSTRAINT dom_genr;
ALTER TABLE Etudiant DROP CONSTRAINT dom_genr;
```

Listing 37: Exemple ALTER

SQL : DROP

Supprimer un objet du schéma

```
DROP TABLE Inscrit_Dans;
DROP TABLE Etudiant;

DROP  TABLE Etudiant CASCADE CONSTRAINTS;
DROP TABLE Inscrit_Dans  CASCADE CONSTRAINTS;;
```

Listing 38: Exemple DROP

SQL : VIEW

Aller vers les besoins de restitution et protection du schéma

```
CREATE OR REPLACE VIEW LesFillesDeHMIN112M AS
SELECT Etudiant.*, 'HMIN112M' as code, annee
    FROM Etudiant JOIN Inscrit_Dans
ON Etudiant.numINE = Inscrit_Dans.numINE
WHERE genre = 'f' and codeM = 'HMIN112M';

SELECT * FROM LesFillesDeHMIN112M ;
```

Listing 39: Exemple VIEW

SQL : VIEW et OUTER JOIN

Notion de jointure externe

```
DROP VIEW LesInscritsOuPas ;
CREATE VIEW LesInscritsOuPas AS
SELECT Etudiant.*, codeM, annee FROM Etudiant
    LEFT OUTER JOIN Inscrit_Dans
ON Etudiant.numINE = Inscrit_Dans.numINE ;

SELECT * FROM LesInscritsOuPas ;
```

Listing 40: Exemple VIEW

SQL : Autoriser des accès aux objets de son schéma

GRANT / REVOKE

```
GRANT SELECT, UPDATE ON Etudiant TO PUBLIC;

GRANT ALL ON Etudiant TO P00000009432;

GRANT ALL ON Etudiant TO P00000009432 WITH GRANT
    OPTION;
```

Listing 41: Exemples GRANT

SQL : Autoriser des accès aux objets de son schéma

GRANT : Exemples d'usage

```
GRANT SELECT, UPDATE ON Etudiant TO PUBLIC;

-- autre usager
SELECT * FROM E20200009999.Etudiant ;

UPDATE E20200009999.Etudiant SET age = age + 1 ;
```

Listing 42: Schéma E20200009999

SQL : Enlever l'autorisation d'accès aux objets de son schéma

GRANT / REVOKE

```
REVOKE SELECT, UPDATE ON Etudiant FROM PUBLIC;

REVOKE ALL ON Etudiant FROM P00000009432;
```

Listing 43: Exemples REVOKE

SQL : Enlever des accès aux objets de son schéma

GRANT : Exemples d'usage

```
REVOKE SELECT, UPDATE ON Etudiant FROM PUBLIC;

-- autre usager
SELECT * FROM E20200009999.Etudiant ;

la table E20200009999.Etudiant n'est plus
    visible
```

Listing 44: Schéma E20200009999

SQL : Le GRANT OPTION

GRANT / REVOKE

```
-- Pierre
GRANT ALL ON Etudiant TO Pauline WITH GRANT
    OPTION;
-- Pauline
GRANT ALL ON Pierre.Etudiant TO Yasmine ;
-- Pierre
REVOKE ALL ON Etudiant FROM Pauline ;
-- par effet rebond : REVOKE ALL ON Etudiant
    FROM Yasmine ;
```

Listing 45: Exemple GRANT OPTION

Un niveau de structuration supplémentaire

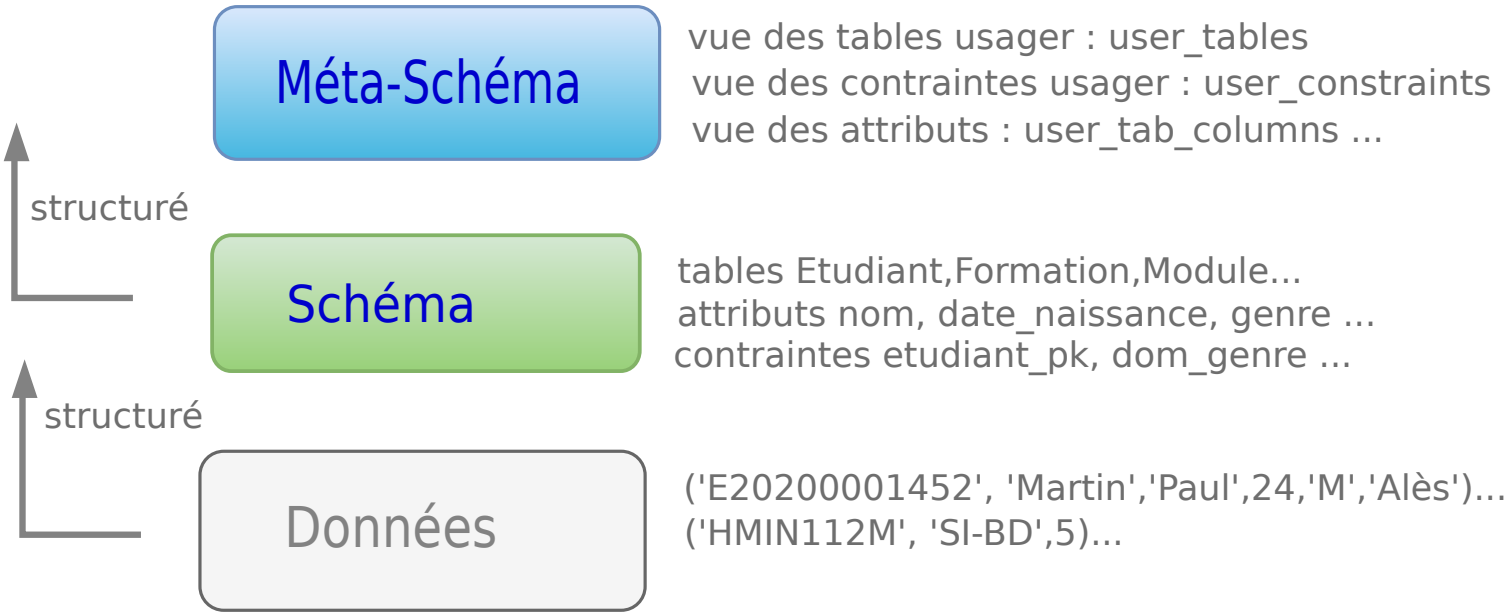
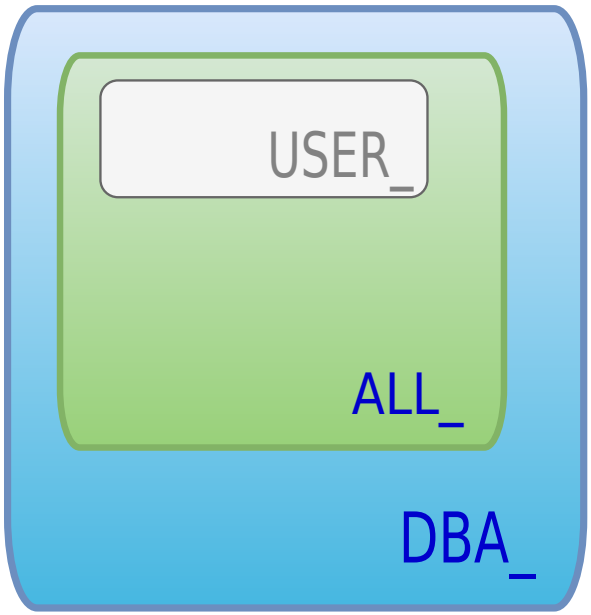


Figure: Méta-schéma ou dictionnaire de données

Vues statiques



vues statiques

préfixe user_ : vues du schéma utilisateur

préfixe all_ : vues des objets sur lesquels l'usager a desaccès

préfixe dba_ : vues sur tous les objets de la base

Figure: Trois catégories imbriquées

SQL : Exemple de consultation des vues du méta-schéma

USER_TABLES, DBA_USERS

```
desc USER_TABLES
-- collecter les statistiques
analyze table emp compute statistics;
SELECT table_name, num_rows FROM USER_TABLES;

SELECT username, created, last_login FROM
    DBA_USERS;
```

Listing 46: Exemples Vues Statiques

SQL : Exemple de consultation des vues du méta-schéma

USER_TAB_PRIVS

```
desc USER_TAB_PRIVS
-- retailler les colonnes
col privilege for a20
col grantor for a20
set linesize 200

SELECT grantee, owner, table_name, grantor,
    privilege FROM user_tab_privs;
```

Listing 47: Exemples Vues Statiques