Bases de données

SQL

Xavier Tannier
xavier.tannier@sorbonne-universite.fr

Bases de données

- Un langage de gestion d'une base de données permettant l'ensemble des opérations :
  - Création et mise à jour de la structure
  - Interrogation et mise à jour des données
  - Gestion des accès concurrentiels, des pannes
  - Gestion de l'intégrité et de la sécurité des données
- Plusieurs versions (1986, 1989, 1992, 1999, 2003, 2008)
- Tous les SGBD ne fournissent pas toutes les fonctionnalités, et certains en fournissent plus.

Types de domaines

Types de domaines en SQL

`char(n)`	Chaîne de caractères de longueur fixe n
`varchar(n)`	Chaîne de caractères de longueur variable inférieure ou égale à *n*
`int`	Valeur entière (taille dépendant de la machine)
`smallint`	Petite valeur entière (taille dépendant de la machine)
`numeric(p,d)` ou `decimal(p,d)`	Valeur numérique, précision *p* (nombre de chiffres significatifs), nombre de chiffres après la virgule *d* (valeurs de *p* et *d* par défaut dépendant du système)
`real` ou `float` ou `double precision`	Réel à virgule flottante, de plus en plus précis

Tous les domaines contiennent également la valeur NULL

Types de domaines en SQL

`date`	Date du calendrier grégorien
`time`	Heure
`timestamp`	Date + Heure
`interval`	Intervalle de dates

Format des informations calendaires :
- `YYYY-MM-DD` pour les dates
- `HH:MM:SS.mmm` pour les heures

Exemple :
- 2017-09-01 16:06:00.000

Types non normalisés

`text`	Texte
`image`	Image
`OLE`	Objet OLE (Windows)
`boolean`	Booléen
`money`	= numeric(max, 2)
`autoinc`	Entier à incrémentation automatique

Structure

Création de table

```
CREATE TABLE Episode (
  ID int(11),
  Serie varchar(60),
  Saison smallint(6),
  Episode smallint(6),
  Diffusion date,
  Titre varchar(60)
)
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Contraintes

- Non nullité

```
CREATE TABLE Episode (
  ID int(11) NOT NULL,
  Serie varchar(60) NOT NULL,
  Saison smallint(6) NOT NULL,
  Episode smallint(6) NOT NULL,
  Diffusion date,
  Titre varchar(60) NOT NULL
)
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Contraintes

- Unicité

```
CREATE TABLE Episode (
  ID int(11) NOT NULL,
  Serie varchar(60) NOT NULL,
  Saison smallint(6) NOT NULL,
  Episode smallint(6) NOT NULL,
  Diffusion date,
  Titre varchar(60) NOT NULL,
  UNIQUE (Serie, Saison, Episode)
)
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Contraintes

- Clé primaire

```
CREATE TABLE Episode (
  ID int(11) AUTO_INCREMENT,
  Serie varchar(60),
  Saison smallint(6),
  Episode smallint(6),
  Diffusion date,
  Titre varchar(60),
  PRIMARY KEY (ID),
  UNIQUE (Serie, Saison, Episode)
)
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Clés

- Clé étrangère

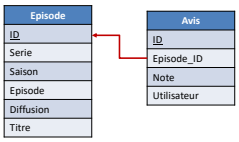
```
CREATE TABLE Avis (
  ID int(11) AUTO_INCREMENT,
  Episode_ID int(11) NOT NULL,
  Note smallint(6) NOT NULL,
  Utilisateur varchar(30) NOT NULL,
  PRIMARY KEY (ID),
  FOREIGN KEY (Episode_ID)
      REFERENCES Episode(ID)
)
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Avis
ID
Episode_ID
Note
Utilisateur

Clés

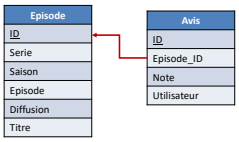
- Clé étrangère et intégrité référentielle



  - Quand on supprime un épisode alors qu'un avis sur cet épisode existe :
    - **ON DELETE RESTRICT** : la suppression est rejetée (par défaut)
    - **ON DELETE CASCADE** : la suppression est validée et les avis sur l'épisode sont supprimés
    - **ON DELETE SET NULL** : la suppression est validée et le champ Episode_ID passe à **NULL** (si autorisé)
    - **ON DELETE SET DEFAULT** : la suppression est validée et le champ Episode_ID passe à la valeur par défaut (si spécifiée)

Clés

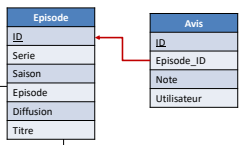
- Clé étrangère et intégrité référentielle



  - Quand on modifie la clé primaire d'un épisode alors qu'un avis sur cet épisode existe :
    - **ON UPDATE RESTRICT** : la modification est rejetée (par défaut)
    - **ON UPDATE CASCADE** : la modification est validée et les avis sur l'épisode sont modifiés
    - **ON UPDATE SET NULL** : la modification est validée et le champ Episode_ID passe à **NULL** (si autorisé)
    - **ON UPDATE SET DEFAULT** : la modification est validée et le champ Episode_ID passe à la valeur par défaut (si spécifiée)

Clés

- Clé étrangère et intégrité référentielle, exemple :




```
CREATE TABLE Avis (
  ID int(11) AUTO_INCREMENT,
  Episode_ID int(11) NOT NULL,
  Note smallint(6) NOT NULL,
  Utilisateur varchar(30) NOT NULL,
  PRIMARY KEY (ID),
  FOREIGN KEY (Episode_ID)
      REFERENCES Episode(ID)
      ON DELETE RESTRICT
      ON UPDATE CASCADE
)
```

Contraintes

- Vérification

```
CREATE TABLE Avis (
  ID int(11) AUTO_INCREMENT,
  Episode_ID int(11) NOT NULL,
  Note smallint(6) NOT NULL,
  Utilisateur varchar(30) NOT NULL,
  PRIMARY KEY (ID),
  CHECK (Note > 0),
  CHECK (Note <= 5)
)
```

Avis
ID
Episode_ID
Note
Utilisateur

Contraintes

- Vérification

```
CREATE TABLE Date (
  Annee int(11) NOT NULL,
  Mois smallint(6) NOT NULL,
  Jour smallint(6) NOT NULL,
  Type_calendrier VARCHAR(10),
  Jour_de_la_semaine VARCHAR(10),
  CHECK (Type_calendrier IN ("grégorien",
            "julien", "hégirien")),
  CHECK (Jour_de_la_semaine IN ("lundi",
       "mardi", "mercredi", "jeudi",
       "vendredi", "samedi", "dimanche"))
)
```

Date
Annee
Mois
Jour
Type_calendrier
Jour_de_la_semaine

Contraintes

- Nommage des contraintes
  - Clarifie les messages d'erreur
  - Permet une modification plus simple

```
CONSTRAINT note_positive CHECK (Note > 0),
CONSTRAINT fk_episode FOREIGN KEY (Episode_ID)
      REFERENCES Episode(ID)
```

Contraintes

- Ajout/suppression de contraintes

```
ALTER TABLE Avis
ADD CONSTRAINT fk_episode FOREIGN KEY (Episode_ID)
      REFERENCES Episode(ID)
```

```
ALTER TABLE Avis
DROP CONSTRAINT fk_episode
```


Sélection de contenu

Requête SQL

- Forme typique d'une requête SQL

```
SELECT A1, A2, … An
FROM T1, T2, … Tn
WHERE P
```

A_i est un attribut
T_i est une relation
P est un prédicat

Le résultat d'une requête SQL est une relation

La clause SELECT

- Liste les attributs souhaités en retour de la requête
- Opération de **projection**

```
SELECT Serie, Saison, Episode, Titre
FROM Episode
```

Episode

ID	Serie	Saison	Episode	Diffusion	Titre
342	Game of Thrones	1	1	17/04/2011	Winter is coming
343	Game of Thrones	1	2	24/04/2011	The Kingsroad
456	Game of Thrones	2	4	22/04/2012	Garden of Bones
673	Homeland	2	3	14/10/2012	State of Independence
843	Homeland	3	2	06/10/2013	Uh... Oh... Ah...
...	...	...		...	

La clause SELECT

```
SELECT *
FROM Episode
```

Episode

ID	Serie	Saison	Episode	Diffusion	Titre
342	Game of Thrones	1	1	17/04/2011	Winter is coming
343	Game of Thrones	1	2	24/04/2011	The Kingsroad
456	Game of Thrones	2	4	22/04/2012	Garden of Bones
673	Homeland	2	3	14/10/2012	State of Independence
843	Homeland	3	2	06/10/2013	Uh... Oh... Ah...
...	...	...		...	

La clause SELECT

(par défaut)

```
SELECT [ALL] Serie, Saison
FROM Episode
```

Serie	Saison
Game of Thrones	1
Game of Thrones	1
Game of Thrones	1
Game of Thrones	2
Game of Thrones	2
...	...

```
SELECT DISTINCT Serie, Saison
FROM Episode
```

Serie	Saison
Game of Thrones	1
Game of Thrones	2
...	...

La clause WHERE

- Conditions sur le résultat
- Opération de **sélection**

```
SELECT *
FROM Episode WHERE Serie = "Game of Thrones"
```

Episode

ID	Serie	Saison	Episode	Diffusion	Titre
342	Game of Thrones	1	1	17/04/2011	Winter is coming
343	Game of Thrones	1	2	24/04/2011	The Kingsroad
456	Game of Thrones	2	4	22/04/2012	Garden of Bones
673	Homeland	2	3	14/10/2012	State of Independence
843	Homeland	3	2	06/10/2013	Uh... Oh... Ah...
...	...	...		...	

La clause WHERE

```
SELECT *
FROM Episode WHERE Serie = "Game of Thrones"
                 AND Saison = 1
```

AND, OR, NOT *=, <, >, <>, etc.*

Episode

ID	Serie	Saison	Episode	Diffusion	Titre
342	Game of Thrones	1	1	17/04/2011	Winter is coming
343	Game of Thrones	1	2	24/04/2011	The Kingsroad
456	Game of Thrones	2	4	22/04/2012	Garden of Bones
673	Homeland	2	3	14/10/2012	State of Independence
843	Homeland	3	2	06/10/2013	Uh... Oh... Ah...
...	...	...		...	

La clause WHERE

```
SELECT *
FROM Episode WHERE Serie = "Homeland"
                 AND (Saison, Episode) = (2, 3)
```

```
SELECT *
FROM Episode WHERE Serie = "Homeland" AND Saison = 3
       AND
       Diffusion BETWEEN '2015/10/01' AND '2015/12/31'
```

```
SELECT *
FROM Episode WHERE Diffusion IS NULL
```

La clause FROM

- Liste des relations impliquées dans la requête
- Opération de **produit cartésien**

```
SELECT *
FROM Episode, Avis
```

- En général, aucun intérêt en l'absence de contraintes exprimées dans la clause WHERE ou de jointures

La clause FROM

Episode

ID	Serie	Saison	Episode
342	Game of Thrones	1	1
343	Game of Thrones	1	2
456	Game of Thrones	2	4
673	Homeland	2	3
843	Homeland	3	2

Avis

ID	Episode_ID	Note
123	342	2
562	342	5
134	342	4
890	673	2
342	673	4

SELECT * FROM Episode, Avis

Episode.ID	Serie	Saison	Episode	Avis.ID	Episode_ID	Note
342	Game of Thrones	1	1	123	342	2
342	Game of Thrones	1	1	562	342	5
342	Game of Thrones	1	1	134	342	4
342	Game of Thrones	1	1	890	673	2
342	Game of Thrones	1	1	342	673	4
343	Game of Thrones	1	2	123	342	2
343	Game of Thrones	1	2	562	342	5
343	Game of Thrones	1	2	134	342	4
343	Game of Thrones	1	2	890	673	2
343	Game of Thrones	1	2	342	673	4
...	...	...	...	...	...	...

La clause FROM

Episode

ID	Série	Saison	Épisode
342	Game of Thrones	1	1
343	Game of Thrones	1	2
456	Game of Thrones	2	4
673	Homeland	2	3
843	Homeland	3	2

Avis

ID	Episode_ID	Note
123	342	2
562	342	5
134	342	4
890	673	2
342	673	4

(ou avec « JOIN », voir plus tard)

SELECT * FROM Episode E, Avis A WHERE E.ID = A.Episode_ID

Episode.ID	Serie	Saison	Episode	Avis.ID	Episode_ID	Note
342	Game of Thrones	1	1	123	342	2
342	Game of Thrones	1	1	562	342	5
342	Game of Thrones	1	1	134	342	4
673	Homeland	2	3	890	673	2
673	Homeland	2	3	342	673	4

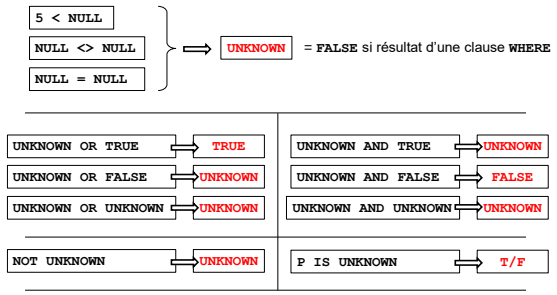
Renommage

- Opération de **renommage**
- Avec ou sans le mot-clé **AS**

```
SELECT E.Serie, E.Saison, E.Episode, A.Note,
       A.Note * 20 [AS] Note_sur_100
FROM Episode [AS] E, Avis [AS] A
WHERE E.ID = A.Episode_ID AND Note_sur_100 > 90
```

- Oracle : on doit omettre le **AS**

Comparaisons de valeurs NULL



Ordre

Ordre

- Rappel : dans la base, AUCUN ordre n'est acquis (pas même l'ordre d'insertion)
- L'ordre alphabétique est assuré par **`ORDER BY`**

```
SELECT * FROM Episode
ORDER BY Serie, Saison, Episode
```

```
SELECT * FROM Episode
ORDER BY Episode DESC
```


Opérations sur les ensembles

Union

Fusionner les clients et les fournisseurs :

```
(SELECT Id, Nom, Adresse, Ville, Pays FROM Client)
UNION
(SELECT Id, Nom_Fournisseur, Adresse_Fournisseur,
       Ville_Fournisseur, Pays_Fournisseur FROM Fournisseur)
```

- Les relations reliées doivent avoir le même nombre d'attributs
- Les attributs fusionnés doivent avoir le même type

Intersection

Les clients communs à deux magasins :

```
(SELECT Id, Nom, Prenom, Date_naissance FROM Client_Magasin1)
INTERSECT
(SELECT Id, Nom, Prenom, Date_naissance FROM Client_Magasin2)
```

- En MySQL, pas d'**INTERSECT** :

```
SELECT DISTINCT valeur FROM Client_Magasin1
WHERE valeur IN (
  (SELECT valeur FROM Client_Magasin2)
)
```


Différence

Les clients qui n'ont jamais rien commandé

```
(SELECT Id, Nom, Prenom FROM Clients)
EXCEPT
(SELECT Id_Client, Nom_Client, Prenom_Client FROM Commande)
```

- En Oracle, mot-clé **MINUS**
- En MySQL, pas de **EXCEPT** ou **MINUS** :

```
SELECT DISTINCT valeur FROM Clients
WHERE valeur NOT IN (
  (SELECT valeur FROM Commande)
)
```


Jointures

Jointures

- Une jointure est un **produit cartésien** entre deux relations, avec des contraintes sur l'association entre les tuples des deux relations
- La jointure et les contraintes sont exprimées dans la clause **`FROM`**

Jointures

- Cours

IdCours	TitreCours	ECTS
INFO-101	Algorithmique	5
INFO-311	Java	5
INFO-413	Fouille de données	2,5
OPTRO-321	Optique ondulatoire	2,5

- Prerequis

IdCours	PrereqId
INFO-311	INFO-101
INFO-514	INFO-101
OPTRO-321	OPTRO-301

- NB :
  - pas de pré-requis pour INFO-413
  - pas d'information sur le cours INFO-514

Jointure interne

```
SELECT * FROM Cours INNER JOIN Prerequis
                ON Cours.IdCours = Prerequis.IdCours
```

Cours

IdCours	PrereqId
INFO-311	INFO-101
INFO-514	INFO-101
OPTRO-321	OPTRO-301

Prerequis

IdCours	TitreCours	ECTS
INFO-101	Algorithmique	5
INFO-311	Java	5
INFO-413	Fouille de données	2,5
OPTRO-321	Optique ondulatoire	2,5

Cours.IdCours	PrereqId	Prerequis.IdCours	TitreCours	ECTS
INFO-311	INFO-101	INFO-311	Java	5
OPTRO-321	OPTRO-301	OPTRO-321	Optique ondulatoire	2,5

Jointure interne

```
SELECT * FROM Cours INNER JOIN Prerequis
                   ON Cours.IdCours = Prerequis.IdCours
```

Sémantiquement équivalent à :

```
SELECT * FROM Cours, Prerequis
WHERE Cours.IdCours = Prerequis.IdCours
```

Différence ?

Jointure interne

```
SELECT * FROM Cours INNER JOIN Prerequis
```

fait le produit cartésien sans contrainte
(rarement ce que l'on souhaite)

```
SELECT * FROM Cours NATURAL JOIN Prerequis
```

fait la jointure naturelle, mais peu conseillé

Jointure externe

```
SELECT * FROM Cours LEFT OUTER JOIN Prerequis
                   ON Cours.IdCours = Prerequis.IdCours
```

Cours

IdCours	PrereqId
INFO-311	INFO-101
INFO-514	INFO-101
OPTRO-321	OPTRO-301

Prerequis

IdCours	TitreCours	ECTS
INFO-101	Algorithmique	5
INFO-311	Java	5
INFO-413	Fouille de données	2,5
OPTRO-321	Optique ondulatoire	2,5

Cours.IdCours	PrereqId	Prerequis.IdCours	TitreCours	ECTS
INFO-311	INFO-101	INFO-311	Java	5
OPTRO-321	OPTRO-301	OPTRO-321	Optique ondulatoire	2,5
INFO-514	INFO-101	*null*	*null*	*null*

Jointure externe

```
SELECT * FROM Cours RIGHT OUTER JOIN Prerequis
                    ON Cours.IdCours = Prerequis.IdCours
```

Cours

IdCours	PrereqId
INFO-311	INFO-101
INFO-514	INFO-101
OPTRO-321	OPTRO-301

Prerequis

IdCours	TitreCours	ECTS
INFO-101	Algorithmique	5
INFO-311	Java	5
INFO-413	Fouille de données	2,5
OPTRO-321	Optique ondulatoire	2,5

Cours.IdCours	PrereqId	Prerequis.IdCours	TitreCours	ECTS
null	*null*	INFO-101	Algorithmique	5
INFO-311	INFO-101	INFO-311	Java	5
null	*null*	INFO-413	Fouille de données	2,5
OPTRO-321	OPTRO-301	OPTRO-321	Optique ondulatoire	2,5

Jointure externe

```
SELECT * FROM Cours FULL OUTER JOIN Prerequis
                    ON Cours.IdCours = Prerequis.IdCours
```

Cours

IdCours	PrereqId
INFO-311	INFO-101
INFO-514	INFO-101
OPTRO-321	OPTRO-301

Prerequis

IdCours	TitreCours	ECTS
INFO-101	Algorithmique	5
INFO-311	Java	5
INFO-413	Fouille de données	2,5
OPTRO-321	Optique ondulatoire	2,5

Cours.IdCours	PrereqId	Prerequis.IdCours	TitreCours	ECTS
null	*null*	INFO-101	Algorithmique	5
INFO-311	INFO-101	INFO-311	Java	5
null	*null*	INFO-413	Fouille de données	2,5
OPTRO-321	OPTRO-301	OPTRO-321	Optique ondulatoire	2,5
INFO-514	INFO-101	*null*	*null*	*null*

Chaînes de caractères

Chaînes de caractère

- Égalité partielle avec LIKE

```
SELECT DISTINCT Serie FROM Episode
WHERE Serie LIKE "%Dead%"
```

Serie
The Walking Dead
Fear the Walking Dead
Dead Like Me
Ash vs Evil Dead
…

  - **'%'** = n'importe quelle chaîne de caractères
  - **'_'** = n'importe quel caractère
  - Les motifs sont sensibles à la casse ("%Dead%" ≠ "%dead%")

Chaînes de caractère

- Autres opérations possibles
  - Concaténation
  - Manipulation de la casse
  - Longueur
  - Sous-chaînes
  - Remplacements,
  - …
- Attention, implémentation souvent très dépendante du système
  (ex : CONCAT avec n paramètres, CONCAT avec 2 paramètres, '||', '+')

Agrégation

Fonctions d'agrégation

Moyenne

```
SELECT AVG(Salaire) FROM Enseignant
WHERE Departement = "Informatique"
```

Minimum

```
SELECT MIN(Note), Nom_Etudiant FROM Note
WHERE Matiere = "BD"
```

Maximum

```
SELECT MAX(Altitude), Nom_Sommet FROM Sommet
WHERE Pays = "France"
```

Somme

```
SELECT SUM(Prix) FROM Depense
WHERE Date BETWEEN 2015/09/01 AND 2015/09/30
```

Compte

```
SELECT COUNT(*) FROM Etudiants
WHERE Stage_Etranger IS NULL
```

```
SELECT COUNT(DISTINCT Serie) FROM Episode
```


Agrégation : GROUP BY

Moyenne

```
SELECT Departement, AVG(Salaire) FROM Enseignant
GROUP BY Departement
```

Enseignant

Nom	Prénom	Departement	Salaire
Grolleau	Alexis	Informatique	12000
Camus	Thomas	Informatique	14000
Remond	Philippe	Informatique	15000
Schmit	Ludovic	Biologie	11000
Burel	Laurent	Biologie	13000
Legros	Antoine	Biologie	12500
Chabanne	Jean-Luc	Biologie	12000
Cohen	Loïc	Physique	12500
Boutin	Paul	Physique	13000

Departement	Salaire
Informatique	13666
Biologie	12125
Physique	12750

Agrégation : GROUP BY

Compte

```
SELECT Departement, COUNT(*) Nombre FROM Enseignant
GROUP BY Departement
```

Enseignant

Nom	Prénom	Departement	Salaire
Grolleau	Alexis	Informatique	12000
Camus	Thomas	Informatique	14000
Remond	Philippe	Informatique	15000
Schmit	Ludovic	Biologie	11000
Burel	Laurent	Biologie	13000
Legros	Antoine	Biologie	12500
Chabanne	Jean-Luc	Biologie	12000
Cohen	Loïc	Physique	12500
Boutin	Paul	Physique	13000

Departement	Nombre
Informatique	3
Biologie	4
Physique	2

Agrégation : GROUP BY

Compte

```
SELECT Nom, COUNT(*) Nombre FROM Enseignant
GROUP BY Departement
```

FAUX

- Les attributs de la clause SELECT doivent apparaître
  - Dans le GROUP BY
  - Dans une fonction d'agrégation
- Ici on agrège par département donc le nom d'une personne n'est plus une information accessible

Agrégation : HAVING

Compte

```
SELECT Departement, COUNT(*) Nombre FROM Enseignant
GROUP BY Departement HAVING Nombre > 2
```

Enseignant

Nom	Prénom	Departement	Salaire
Grolleau	Alexis	Informatique	12000
Camus	Thomas	Informatique	14000
Remond	Philippe	Informatique	15000
Schmit	Ludovic	Biologie	11000
Burel	Laurent	Biologie	13000
Legros	Antoine	Biologie	12500
Chabanne	Jean-Luc	Biologie	12000
Cohen	Loïc	Physique	12500
Boutin	Paul	Physique	13000

Departement	Nombre
Informatique	3
Biologie	4

Agrégation : HAVING

```
SELECT Departement, COUNT(*) Nombre FROM Enseignant
GROUP BY Departement HAVING Nombre > 2
```

- Le **WHERE** est appliqué AVANT la formation des groupes (on crée les groupes à partir des lignes qui vérifient les contraintes)
- Le **HAVING** est appliqué APRÈS la formation des groupes (on sélectionne certains groupes parmi ceux formés)

Agrégation et NULL

- Toutes les opérations d'agrégation sauf le **COUNT** ignorent les tuples contenant des valeurs **NULL** sur les attributs agrégés.
- Si la collection ne contient que des valeurs **NULL** :
  - **COUNT** retourne 0
  - Les autres retournent **NULL**

Sous-requêtes

Sous-requêtes

- Sous-requête dans le **WHERE**

```
SELECT DISTINCT valeur FROM Clients
WHERE valeur NOT IN (
  (SELECT valeur FROM Commande)
)
```

- Sous-requête dans le **FROM**
  - Les départements qui paient plus que la moyenne

```
SELECT Departement, Salaire_Moyen
FROM (SELECT Departement, AVG(Salaire) AS Salaire_Moyen
      FROM Enseignant
      GROUP BY Departement)
WHERE Salaire_Moyen > 12000
```

Sous-requêtes

- Sous-requête scalaire
  - Les enseignants qui gagnent plus du dixième du budget personnel de leur département

```
SELECT Nom FROM Enseignant E1 WHERE
       Salaire * 10 > (SELECT SUM(Salaire)
                       FROM Enseignant E2
                       WHERE E1.Departement = E2.Departement)
```

  - Une erreur est levée à l'exécution si la requête renvoie plus d'un élément

Comparaison d'ensembles

Comparaison d'ensembles

  - Les enseignants qui gagnent plus que le plus haut salaire du département d'informatique

```
SELECT Nom FROM Enseignant
WHERE Salaire > ALL (SELECT Salaire FROM Enseignant
                     WHERE Departement="informatique")
```

ou SOME ou ANY

  - Un cours donné en 2019 qui était déjà donné en 2018

```
SELECT Nom_Cours FROM Cours C1
WHERE C1.annee=2019 AND EXISTS (
                     SELECT * FROM Cours C2
                     WHERE C2.Nom_Cours=C1.Nom_Cours
                           AND C2.annee=2018)
```

ou NOT EXISTS ou UNIQUE

La clause WITH

La clause WITH

- Création d'une vue temporaire pour simplifier la requête
  - Tous les enseignants qui ont le salaire le plus haut

```
WITH Salaire_Max.Valeur AS (SELECT MAX(Salaire) FROM
                            Enseignant)
SELECT Nom FROM Enseignant E, Salaire_Max S
WHERE E.Salaire = S.Valeur
```

  - (pas le plus simple pour cette requête,
    mais très utile pour des requêtes plus complexes)

SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

POLYTECH SORBONNE

SQL

Modification de contenu

Bases de données

Xavier Tannier
xavier.tannier@sorbonne-universite.fr

Modification de contenu

- Ajouter des éléments
- Supprimer des éléments
- Modifier des éléments

Ajout d'éléments

– Ajouter un tuple

AUTOINCREMENT

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

```
INSERT INTO Episode VALUES
(NULL, 'Big Bang Theory', 9, 1, '2015/09/21',
                          'The Matrimony Momentum')
```

```
INSERT INTO Episode
('Serie', 'Titre', 'Saison', 'Episode') VALUES
('Big Bang Theory', 'The Separation Oscillation', 9, 2)
```

– Ajouter plusieurs tuples

```
INSERT INTO Episode VALUES
(NULL, 'Big Bang Theory', 9, 1, '2015/09/21', 'The Matrimony Momentum')
(NULL, 'Big Bang Theory', 9, 2, '2015/09/28', 'The Separation Oscillation')
(NULL, 'Big Bang Theory', 9, 3, '2015/10/05', 'The Bachelor Party Corrosion')
(NULL, 'Big Bang Theory', 9, 4, '2015/10/12', 'The 2003 Approximation')
```


Ajout d'éléments

– Ajouter un avis aléatoire entre 0 et 5 dans la table Avis pour chaque épisode de la table Episode

```
INSERT INTO Avis
SELECT NULL, ID, FLOOR(RAND() *6), 'AUTO' FROM Episode
```

Episode
ID
Serie
Saison
Episode
Diffusion
Titre

Avis
ID
Episode_ID
Note
Utilisateur

Suppression d'éléments

– Supprimer toutes les séries de la table Episode

```
DELETE FROM Episode
```

– Supprimer la saison 1 de la série « Plus belle la vie »

```
DELETE FROM Episode
WHERE Serie="Plus belle la vie" AND Saison=1
```

– Supprimer les notes inférieures à la moyenne des notes dans les avis

```
DELETE FROM Avis
WHERE Note < (SELECT AVG(Note) FROM Avis)
```

FAUX

Pourquoi ?

Mise à jour d'éléments

– On ne note plus les séries sur 5, mais sur 10… Mise à jour des avis existants

```
UPDATE Avis
SET Note = Note * 2
```

– On s'est trompé, on a décalé les épisodes de la saison 2 de Suits

```
UPDATE Episode
SET Episode = Episode - 1
WHERE Serie="Suits" AND Saison=2
```


Mise à jour d'éléments

– On ne note plus les séries en continu entre 0 et 10, mais 0, 5 ou 10

```
UPDATE Avis
SET Note = CASE
           WHEN Note < 3 THEN 0
           WHEN Note < 7 THEN 5
           ELSE 10
           END
```

Les vues

Les vues

- Une vue est une « relation virtuelle », une vision partielle et/ou agrégée d'informations contenue dans une base
- Les données d'une vue sont totalement dépendantes des autres données de la table (une vue ne contient aucune information qui ne soit pas ailleurs dans la base)
- Le résultat d'une requête de sélection, stockée pour un usage régulier

- Intérêt : évite ou empêche l'accès à l'ensemble du modèle logique
  - Informations sur le personnel, mais sans les salaires
  - Informations sur les épisodes d'une série avec les commentaires des téléspectateurs

Vue : exemple

```
CREATE VIEW Vue_Enseignant AS
SELECT Nom, Prenom, Departement FROM Enseignant
```

Nom	Prénom	Departement	Salaire
Grolleau	Alexis	Informatique	12000
Camus	Thomas	Informatique	14000
Remond	Philippe	Informatique	15000
Schmit	Ludovic	Biologie	11000
Burel	Laurent	Biologie	13000
Legros	Antoine	Biologie	12500
Chabanne	Jean-Luc	Biologie	12000
Cohen	Loïc	Physique	12500
Boutin	Paul	Physique	13000

Nom	Prénom	Departement
Grolleau	Alexis	Informatique
Camus	Thomas	Informatique
Remond	Philippe	Informatique
Schmit	Ludovic	Biologie
Burel	Laurent	Biologie
Legros	Antoine	Biologie
Chabanne	Jean-Luc	Biologie
Cohen	Loïc	Physique
Boutin	Paul	Physique

Les vues

- On peut créer une vue à partir de n'importe quelle requête de sélection
- On peut ensuite faire des requêtes sur une vue comme sur une relation classique

- Une vue est mise à jour lorsque les données sous-jacentes sont modifiées
- On peut sous certaines conditions modifier les données par l'intermédiaire des vues (**`UPDATE`** ou **`INSERT`** dans la vue), mais c'est à manipuler avec précautions…

Les droits d'accès

Types de droits

- Droits sur la structure
  - Création de nouvelles relations
  - Modification de relations
  - Suppression de relations
  - Création/suppression d'index

- Droits sur le contenu
  - Lecture (**`SELECT`**)
  - Insertion (**`INSERT`**)
  - Modification (**`UPDATE`**)
  - Suppression (**`DELETE`**)

Spécification d'un droit

```
GRANT <actions>
ON <relation ou vue>
TO <utilisateurs>
```

- *Actions* :
  - **SELECT**
  - **INSERT** ([liste de colonnes])
  - **UPDATE** ([liste de colonnes])
  - **DELETE**
  - **ALL PRIVILEGES**

  (séparées par des virgules)
- *Utilisateurs* :
  - **PUBLIC** (tous les utilisateurs)
  - Un identifiant d'utilisateur
  - Un rôle (voir plus loin)

Révocation d'un droit

```
REVOKE <actions>
ON <relation ou vue>
TO <utilisateurs>
```

- Si on révoque **PUBLIC**, tout le monde perd ses privilèges, sauf ceux qui ont obtenu le droit explicitement par ailleurs

Rôle

- Un rôle = un groupe d'utilisateurs

```
CREATE ROLE enseignant;

GRANT enseignant TO xtannier;
GRANT enseignant TO rboulin;
GRANT enseignant TO ptalaud;

CREATE ROLE droit_modif;

GRANT droit_modif TO lechef;
GRANT droit_modif TO enseignant;

GRANT SELECT, UPDATE ON Table_Enseignant TO droit_modif;
```

Autres types de droits

- Droits de référencement pour créer des clés étrangères

```
GRANT REFERENCE (id) ON Enseignant TO user1234;
```

- Transfert de privilèges

```
GRANT SELECT ON Enseignant TO user1234 WITH GRANT OPTION;
```

```
REVOKE SELECT ON Enseignant FROM user123, user567 CASCASE;
```

```
REVOKE SELECT ON Enseignant FROM user123, user567 RESTRICT;
```


Déclencheurs

(triggers)

Gestion « active » des données

- **Gestion des contraintes** : quand une opération viole une contrainte
  - On annule l'opération
  - Ou on essaie de « réparer » les données
  - *Exemple* : correction après une violation de contrainte d'intégrité
- **Surveillance des données** : quand quelque chose se passe, exécuter une certaine action
  - *Exemple* : quand le prix de l'action passe au dessus de X €, on vend
  - *Exemple* : quand le nombre d'inscrits à un cours dépasse la capacité, envoyer un e-mail au secrétariat.

*Déclencheurs (*triggers*)*

- **Événement – Condition – Action (ECA)** :
  - Quand un **événement** arrive, tester la **condition**.
  - Si la **condition** est vérifiée, exécuter l'**action**

- *Exemple* :
  - **Événement** : quand un nouvel étudiant est ajouté à la base…
  - **Condition** : si son score au TOEIC blanc est inférieur à 500…
  - **Action** : on l'inscrit au cours d'anglais renforcé !

Exemple

```
CREATE TRIGGER trig_etudiant

AFTER INSERT ON Etudiant

REFERENCING NEW ROW AS nouvelEtudiant

FOR EACH ROW
WHEN (nouvelEtudiant.TOEIC < 500)

INSERT INTO Cours VALUES
            (nouvelEtudiant.EID, 'Anglais Renforcé')
```

Événement

Condition

Action

déclencheur au niveau de la ligne

Exemple

```
CREATE TRIGGER trig_etudiant

AFTER INSERT ON Etudiant

REFERENCING NEW TABLE AS nouvelEtudiant

FOR EACH STATEMENT
INSERT INTO Cours (SELECT EID, 'Anglais Renforcé'
             FROM nouvelEtudiant WHERE TOEIC < 500)
```

déclencheur au niveau de l'action entière

Options possibles

- Les événements possibles :
  - **`INSERT ON`**
  - **`DELETE ON`**
  - **`UPDATE ON`**
- Temporalité : les actions peuvent être exécutées
  - Après ou avant l'événement (**`BEFORE`** ou **`AFTER`**)
  - À la place de l'événement (**`INSTEAD OF`**)

Options possibles

- Granularité : les déclencheurs peuvent être activés :
  - Pour chaque ligne modifiée (**`FOR EACH ROW`**)
  - Pour chaque action qui modifie quelque chose (**`FOR EACH STATEMENT`**)
- Pourquoi deux granularités sont-elles nécessaires ?
  - Un déclencheur sur l'ensemble de la table peut être la seule possible
    - ex : si la moyenne des nouveaux étudiants au TOEIC est inférieure à 600, alors…
  - Un déclencheur sur une ligne est plus simple à implémenter
    - Mais attention, peut être inefficace si appliqué à de nombreuses modifications

Variables existantes

- Variables :
  - **`OLD ROW`** : la ligne modifiée telle qu'elle était avant la modification
  - **`NEW ROW`** : la ligne modifiée telle qu'elle est après la modification
  - **`OLD TABLE`** : une table en lecture seule contenant toutes les lignes modifiées telles qu'elles étaient avant la modification
  - **`NEW TABLE`** : une table en lecture seule contenant toutes les lignes modifiées telles qu'elles sont après la modification
- Cas variables n'ont de sens que dans un certain contexte :
  - **`AFTER INSERT`** + déclencheur sur une action → **`NEW TABLE`**
  - **`BEFORE DELETE`** + déclencheur sur des lignes → **`OLD ROW`**
  - …

Exemple

```
CREATE TRIGGER trig_salaire

BEFORE UPDATE OF salaire ON Employe

REFERENCING OLD ROW AS old, NEW ROW AS new

FOR EACH ROW
WHEN (new.salaire > 1.5 * old.salaire)

SET new.salaire = 1.5 * old.salaire
```


Exemple

```
CREATE TRIGGER trig_salaire_scoreTOEIC

AFTER UPDATE OF TOEIC ON Etudiant

REFERENCING OLD TABLE AS old, NEW TABLE AS new

FOR EACH STATEMENT
WHEN (NOT EXISTS(SELECT * FROM old, new
              WHERE old.EID = new.EID
              AND old.TOEIC >= new.TOEIC)

UPDATE Enseignant
SET salaire = salaire + 1000 WHERE Matiere='Anglais'
```


Questions diverses

- Attention à l'effet domino
  - L'action d'un déclencheur peut causer d'autres déclenchements
  - Peut conduire à une boucle infinie
    - Certains systèmes savent gérer (Oracle, BD2…)
    - D'autres vous laissent vous débrouiller (PostgreSQL…)
- Attention à l'interaction avec les contraintes
  - Attention aux effets en cascade dus à des déclenchements mal maîtrisés

Déclencheurs et SGBD

- Fonctionnalités, implémentation et syntaxe sont TRÈS différents selon les systèmes
  - Ne pas compter sur la syntaxe présentée aux pages précédentes
  - Il faut se documenter sur votre SGBD préféré.