SCI6306 Bases de données documentaires (Automne 2023)

Christine Dufour, EBSI, UdeM

A2023 29 septembre 2023

Cours 4 : SQL (partie 2 de 3)

SCI6306

Christine Dufour, EBSI, UdeM

Table des matières

Cours 4 - SQL (partie 2 de 3)

- + Au programme aujourd'hui
- SQL (niveau 2)
    - Regroupements (Clause GROUP BY)
        - Logique du GROUP BY
    - Union de requête (fonction UNION)
    - Copies multiples d'une table
    - Quelques fonctions/opérations utiles
- Ressources en lien avec le cours

1. + Au programme aujourd'hui

- **SQL – partie 2 de 3** (*niveau 2*)
    - Clause GROUP BY et fonctions d'agrégation
    - Union de requêtes
    - Copies multiples d'une table dans une requête
    - Quelques fonctions/opérations utiles
- Laboratoire : **TP Requêtes SQL** – niveau 2

2. SQL (niveau 2)

2.1. Regroupements (Clause GROUP BY)

La clause **GROUP BY** permet de faire des regroupements d'enregistrements par rapport à certains champs – regroupements qui permettent entre autres des opérations statistiques sur les éléments regroupés. Par exemple, dans INSCRIP, on pourrait vouloir regrouper ensemble tous les cours suivis par un étudiant ou une étudiante pour calculer le nombre de cours qu'il ou elle suit. On pourrait suivre une logique similaire pour compter le nombre de cours enseignés par les professeurs et professeures. Un autre exemple consisterait à regrouper ensemble toutes les notes des étudiants et étudiantes dans un cours pour calculer la moyenne du cours. Tous ces exemples exploitent la logique du regroupement.

La forme générique d'une requête comportant un regroupement est la suivante :

```
SELECT nom(s) de champs, valeurs calculées/fonctions d'agrégation
FROM nom(s) de table
WHERE condition(s)
GROUP BY nom(s) de champs
```

a) Logique du GROUP BY

Exemple **: Calcul du nombre de cours suivis par les étudiant.e.s**

Reprenons l'exemple du **dénombrement des cours suivis par un.e étudiant.e**. Le *point de départ* pour ce besoin est la table présentant toutes les **inscriptions**, soit le résultat de l'équijointure entre la table ETUD et la table SUIT obtenu par la requête SQL suivante :

```
SELECT *
FROM etud, suit
WHERE etud.no_etud=suit.no_etud ;
```

L'observation de la table de résultats ci-dessous nous permet de facilement compter "à l'oeil" le nombre de cours suivis par exemple par Claire Bretécher (3 cours). Ce que l'on veut est bien entendu non pas d'avoir à le calculer nous-mêmes manuellement, mais de le faire faire par une requête SQL. C'est ce que nous permet une clause **GROUP BY**.

no_etud	nom	adresse	dat_nais	prog	no_cours	note	note_p
10001	Wagner, Richard	1200 de l'Opéra, Bayreuth	1980-01-01	Musique	20001	75.5	1
10002	Bretécher, Claire	2400 du Fou-rire, Paris	1950-01-01	Littérature	20001	0	0
10002	**3 cours suivis**	2400 du Fou-rire, Paris	1950-01-01	Littérature	20004	0	0
10002	Bretécher, Claire	2400 du Fou-rire, Paris	1950-01-01	Littérature	20005	98	1
10003	Tremblay, Michel	4800 St-Laurent, Montréal	1970-01-01	Sciences humaines	20002	80	1
10003	Tremblay, Michel	4800 St-Laurent, Montréal	1970-01-01	Sciences humaines	20004	0	0
10004	Asimov, Isaac	9600 du Futur, Los Angeles		Sciences	20001	0	1
10004	Asimov, Isaac	9600 du Futur, Los Angeles		Sciences	20003	38	1
10004	Asimov, Isaac	9600 du Futur, Los Angeles		Sciences	20005	79.5	1
10006	Smithwick, Dale	4300 Crescent, Montréal		Sciences	20004	0	0
10006	Smithwick, Dale	4300 Crescent, Montréal		Sciences	20007	62	1

Résultat de l'équijointure entre la table ETUD et la table SUIT

On peut imaginer en quelque sorte que si on demande de regrouper la table résultant de l'équijointure sur la base du champ **etud.no_etud** (ce qui revient à ajouter à la requête la clause `GROUP BY etud.no_etud`), les valeurs des autres champs se regroupent dans une même cellule comme suit :

etud. no_etud	no_cours regroupés
10001	20001
10002	20001 20004 20005
10003	20002 20004
10004	20001 20003 20005
10006	20004 20007

Illustration de l'effet du regroupement sur le champ etud.no_etud

Il est alors possible d'appliquer différentes opérations - des **fonctions d'agrégation** par exemple - sur ces valeurs regroupées. On peut entre autres utiliser la fonction d'agrégation **COUNT()** qui permet de dénombrer les valeurs regroupées, ce qui est justement ce que l'on cherche! La requête SQL complète pour ce besoin serait ainsi :

```
1 SELECT etud.no_etud, nom, COUNT(no_cours) as 'Nbre cours'
2 FROM etud, suit
3 WHERE etud.no_etud=suit.no_etud
4 GROUP BY etud.no_etud;
```

Cette requête produira la table de résultats suivante :

no_etud	nom	Nbre cours
10001	Wagner, Richard	1
10002	Bretécher, Claire	3
10003	Tremblay, Michel	2
10004	Asimov, Isaac	3
10006	Smithwick, Dale	2

Table des résultats

À noter : la fonction `COUNT()` a été appliquée au champ `no_cours`. Il faut faire bien attention au champ utilisé. Si vous utilisiez un champ qui est facultatif, les lignes où le champ est vide ne seraient pas comptées. On peut aussi demander de compter non pas un champ, mais la ligne au complet en utilisant l'astérisque ainsi : `COUNT(*)`.

Exemple : Calcul de la moyenne globale obtenue par les étudiant.e.s pour la session

Prenons maintenant l'exemple du calcul de la **moyenne obtenue par les étudiant.e.s** pour la session. La table de résultats de base, avant le regroupement, peut s'obtenir simplement en exploitant la table SUIT (si on ne sent pas le besoin d'avoir les noms des étudiants et étudiantes) :

```
1 SELECT no_etud, note
2 FROM suit
3 WHERE note_p;
```

Remarquez la présence de la condition **`WHERE note_p`**. Cette dernière permet de s'assurer de ne prendre en considération que les **vraies notes** et ainsi d'omettre le "zéro" par défaut pour les cours dont les notes n'ont pas encore été saisies. Une condition devant retourner VRAI ou FAUX, il suffit ici d'aller vérifier la valeur du champ `note_p`.

Comme on veut obtenir la moyenne par étudiant et étudiante, le regroupement se fera sur la base du champ **no_etud** en ajoutant à la requête SQL la clause `GROUP BY no_etud`. Les notes seraient ainsi mises ensemble pour chaque étudiant et étudiante :

no_etud	*note regroupées*
10001	*75,50*
10002	*98,00*
10003	*80,00*
10004	*0,00* *38,00* *79,50*
10006	*62,00*

Illustration de l'effet du regroupement sur le champ no_etud

Maintenant que voilà les notes regroupées, il est possible d'y appliquer la fonction `AVG()` qui permet de calculer une **moyenne** (*average*). La requête SQL finale se lirait ainsi :

```
1 SELECT no_etud, AVG(note)
2 FROM suit
3 WHERE note_p
4 GROUP BY no_etud ;
```

La table de résultats retournée serait la suivante :

no_etud	*AVG(note)*
10001	*75,50*
10002	*98,00*
10003	*80,00*
10004	*39,16666666664*
10006	*62,00*

Table de résultats

¶ *Syntaxe* : Principales fonctions d'agrégation

SQL offre plusieurs fonctions d'agrégation, les principales étant les suivantes :

Principales fonctions d'agrégation

Fonction d'agrégation	Description
COUNT(*nom de champ*)	Pour compter le **nombre de valeurs d'un champ** pour un regroupement d'enregistrements; `COUNT(DISTINCT nom de champ)` calcule le nombre de valeurs distinctes
MIN(*nom de champ*)	Pour identifier le **minimum** parmi les valeurs d'un champ pour un regroupement d'enregistrements
MAX(*nom de champ*)	Pour identifier le **maximum** parmi les valeurs d'un champ pour un regroupement d'enregistrements
SUM(*nom de champ*)	Pour obtenir la **somme** des valeurs d'un champ pour un regroupement d'enregistrements
AVG(*nom de champ*)	Pour calculer la **moyenne** des valeurs d'un champ pour un regroupement d'enregistrements
GROUP_CONCAT(*nom de champ*)	Pour **concaténer** dans une même valeur tous les éléments d'un groupe; il est possible de préciser un *séparateur* `GROUP_CONCAT(nom de champ SEPARATOR '; ')`

Une liste complète des fonctions d'agrégation peut être consultée à l'URL https://dev.mysql.com/doc/refman/8.0/en/aggregate-functions.html.[1]

⌕ *Exemple* : Liste des cours, des étudiant.e.s inscrit.e.s, et du nombre d'étudiant.e.s inscrit.e.s, en ordre décroissant du nombre d'étudiant.e.s inscrit.e.s

Maintenant que la logique des regroupements a été présentée, nous ferons un dernier exemple impliquant son utilisation. Le besoin ici est pour obtenir, pour chacun.e des étudiant.e.s, la liste des cours qu'il ou elle suit ainsi que le nombre de cours suivis, le tout en ordre décroissant de nombre de cours suivis.

Décomposition du besoin :

- *Information affichée* : Les champs pour décrire l'étudiant.e (no_etud et nom), les champs pour décrire les cours suivis (no_cours et titre), le nombre de cours suivis
- *Source de données* : Table SUIT, table COURS, table ETUD

[1] https://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html

- *Jointures* :
    - entre la table SUIT et la table COURS sur la base du champ NO_COURS
    - entre la table SUIT et la table ETUD sur la base du champ NO_ETUD
- *Conditions* : aucune (uniquement les conditions de jointure décrites ci-dessus)
- *Regroupement* : sur la base du numéro d'étudiant.e
- *Tri* : en ordre décroissant du nombre de cours suivis

Requête SQL :

```
SELECT CONCAT(etud.no_etud, ' ', nom) as 'Étudiant.e',
GROUP_CONCAT(CONCAT(cours.no_cours, ' ', titre) SEPARATOR '\n') AS 'Cours suivis',
COUNT(cours.no_cours) AS 'Nombre de cours suivis'
FROM suit, cours, etud
WHERE cours.no_cours=suit.no_cours AND suit.no_etud=etud.no_etud
GROUP BY etud.no_etud
ORDER BY COUNT(cours.no_cours) DESC;
```

Cette requête donnera le résultat suivant :

Étudiant.e	Cours suivis	Nombre de cours suivis
10002 Bretécher, Claire	20001 Histoire 101 20004 Français 304 20005 Mathém...	3
10004 Asimov, Isaac	20001 Histoire 101 20003 Philosophie 813 20005 Mat...	3
10003 Tremblay, Michel	20002 Éthique 302 20004 Français 304	2
10006 Smithwick, Dale	20004 Français 304 20007 Chimie 007	2
10001 Wagner, Richard	20001 Histoire 101	1

Table de résultats

À noter :

- La fonction **CONCAT** a été utilisée pour des questions de *lisibilité* afin de présenter dans une même colonne le numéro de l'étudiant.e et son nom (de même pour le numéro du cours et son titre).
- La fonction **GROUP_CONCAT** a aussi été utilisée pour des questions de *lisibilité* comme elle permet de mettre ensemble les différentes valeurs regroupées dans une même cellule. Un retour de ligne, représenté dans phpMyAdmin par `\n`, sépare les différentes valeurs sur des lignes distinctes. Si vous testez cette requête dans *KeSQiL fait?*, vous remarquerez qu'il ne comprend pas cette chaîne comme un retour de ligne et l'affiche à l'écran. Cela vient du fait que l'affichage est géré en HTML. Pour représenter le retour de ligne, il faudrait plutôt utiliser la balise `<br/>` !
- Des **alias** ont été utilisés pour s'assurer d'avoir des entêtes de colonne *compréhensibles*.
- Le **tri** se fait sur la *fonction d'agrégation* et non à partir de son alias.

2.2. Union de requête (fonction UNION)

Il est possible d'unir les résultats de plusieurs requêtes en utilisant la fonction UNION. La **seule exigence** est que chacune des requêtes que l'on veut ainsi unir doit avoir le **même nombre de colonnes** dans sa table de résultats (peu importe la nature des données dans ces colonnes).

Exemple : Liste des cours avec des inscriptions ou ayant un.e professeur.e attitré.e

Pour certains besoins, on entrevoit parfois comment le faire en deux temps. Le besoin ici est d'avoir dans une même liste les cours avec des inscriptions ainsi que les cours ayant un.e professeur.e associé.e. Dans cet exemple, on sait par exemple comment identifier les **cours avec des inscriptions** :

```
1 SELECT DISTINCT cours.no_cours, titre
2 FROM cours, suit
3 WHERE cours.no_cours=suit.no_cours
4 ORDER BY cours.no_cours;
```

Cette requête permet d'obtenir la table de résultats suivante :

no_cours ▲ 1	titre
20001	Histoire 101
20002	Éthique 302
20003	Philosophie 813
20004	Français 304
20005	Mathématique 307
20007	Chimie 007

Table de résultats

On sait aussi comment trouver les **cours ayant un.e professeur.e associé.e** :

```
1 SELECT no_cours, titre
2 FROM cours
3 WHERE no_prof is not NULL
4 ORDER BY no_cours;
```

Cette requête donne la table de résultats suivante :

no_cours ▲ 1	titre
20001	Histoire 101
20002	Éthique 302
20003	Philosophie 813
20005	Mathématique 307
20006	Piano 110

Table de résultats

Ne reste ainsi qu'à les unir! Chacune des requêtes est mise **entre parenthèses** en ajoutant le mot-clé `UNION` entre les deux. De plus, comme on veut **globalement** trier les résultats, la clause `ORDER BY` sera sortie des requêtes. Finalement, une troisième colonne sera ajoutée à chacune des requêtes afin d'indiquer l**a nature des enregistrements** sinon, une fois les deux tables de résultats mises ensemble, il sera difficile de savoir quels cours correspondent à des cours avec des étudiant.e.s inscrit.e.s et lesquels ont des professeur.e.s assigné.e.s. Un dernier petit détail : le prédicat DISTINCT dans la première requête peut être enlevé; il devient en effet inutile comme une requête UNION le fait automatiquement. La requête SQL obtenue est la suivante :

```
1 (SELECT cours.no_cours, titre, 'Cours avec inscription' AS 'Statut du cours'
2 FROM cours, suit
3 WHERE cours.no_cours=suit.no_cours)
4 UNION
5 (SELECT no_cours, titre, 'Cours avec professeur attitré'
6 FROM cours
```

```
7 WHERE no_prof is not NULL)
8 ORDER BY no_cours;
```

La table de résultats suivante sera retournée :

no_cours ▲ 1	titre	Statut du cours
20001	Histoire 101	Cours avec professeur attitré
20001	Histoire 101	Cours avec inscription
20002	Éthique 302	Cours avec professeur attitré
20002	Éthique 302	Cours avec inscription
20003	Philosophie 813	Cours avec professeur attitré
20003	Philosophie 813	Cours avec inscription
20004	Français 304	Cours avec inscription
20005	Mathématique 307	Cours avec professeur attitré
20005	Mathématique 307	Cours avec inscription
20006	Piano 110	Cours avec professeur attitré
20007	Chimie 007	Cours avec inscription

Table de résultats

Exemple : Cours ayant des informations manquantes (local non assigné, professeur non associé, titre non défini)

Nous aborderons un dernier exemple où l'union peut être bien pratique. Le besoin ici est d'afficher la liste des cours pour lesquels **certaines informations sont manquantes** (soit le local, soit le professeur, soit le titre). Comme on sait bien comment aller chercher chacun des morceaux individuellement, la logique d'une union s'applique ici très bien.

Décomposition du besoin :

- *Information affichée* : Les champs pour identifier le cours (no_cours et titre), le type d'information qui manque
- *Source de données* :
  - Première requête : table COURS
  - Deuxième requête : table COURS
  - Troisième requête : table COURS
- *Jointure* : aucune
- *Conditions* :
  - Première requête : le local doit être manquant (valeur NULL)
  - Deuxième requête : le numéro du professeur ou de la professeure doit être manquant (valeur NULL)
  - Troisième requête : le titre du cours doit être manquant (valeur NULL)
- *Regroupement* : aucun
- *Tri* : en ordre décroissant du numéro de cours
- *Autre* : utilisation de UNION entre les trois requêtes

Requête SQL :

```
(SELECT cours.no_cours, cours.titre, 'Local' AS 'Information manquante' FROM cours WHERE cours.local is NULL)
UNION
(SELECT cours.no_cours, cours.titre, 'Professeur' FROM cours WHERE cours.no_prof is NULL)
UNION
(SELECT cours.no_cours, cours.titre, 'Titre' FROM cours WHERE cours.titre is NULL)
ORDER BY no_cours;
```

Cette requête donnera comme résultat la table suivante :

no_cours ▲ 1	titre	Information manquante
20003	Philosophie 813	Local
20004	Français 304	Professeur
20007	Chimie 007	Local
20007	Chimie 007	Professeur

Table de résultats

2.3. Copies multiples d'une table

Et si l'on veut trouver **les étudiant.e.s qui suivent à la fois deux cours en particulier** (par exemple. '20001' et '20003')? On voudrait bien pouvoir faire une intersection au lieu d'une union, mais « INTERSECT » n'existe pas... Ajouter dans la clause WHERE une condition du type « `suit.no_cours= '20001' AND suit.no_cours='20003'` » ne fonctionnera pas non plus comme chaque ligne de la table COURS correspond à une seule inscription. Un.e étudiant.e suivant ces deux cours aura ainsi deux lignes pour ces deux cours; et les conditions dans une requête sont appliquées sur une seule ligne à la fois.

Ce que l'on voudrait, c'est avoir sur une même ligne dans la table servant de source de données pour une requête SQL pour un.e étudiant.e, deux des cours qu'il ou elle suit pour pouvoir faire une condition du type « suit1.no_cours='20001' AND suit2.no_cours='20003' »...

Comment faire? Utiliser **deux copies de la table SUIT**!

- Une pour vérifier l'inscription au cours 20001
- L'autre pour vérifier l'inscription au cours 20003

Il faut donc commencer par construire la requête qui nous permettra d'obtenir cette table. On y retrouvera deux versions de la table SUIT, l'une nommée SUIT1 et l'autre SUIT2 ainsi qu'une version de la table ETUD. Nous travaillons avec une seule version de la table ETUD comme on souhaite obtenir, pour une même personne, deux de ses cours. Il faut bien entendu "attacher" la table ETUD à ces deux tables SUIT sur la base du champ NO_ETUD :

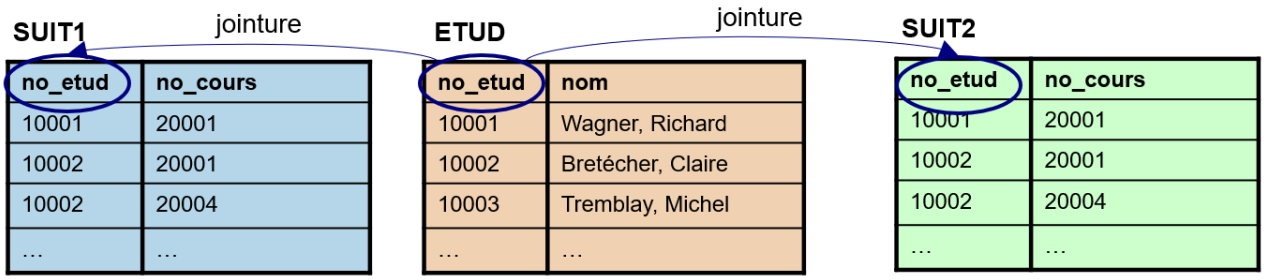
Et si l'on veut trouver **les professeur.e.s qui enseignent à la fois deux cours** (par exemple. '20001' et '20003')? On voudrait bien pouvoir faire une intersection au lieu d'une union, mais « INTERSECT » n'existe pas... Ajouter dans la clause WHERE une condition du type « `cours.no_cours= '20001' AND cours.no_cours='20003'` » ne fonctionnera pas non plus comme chaque ligne de la table COURS correspond à une seule inscription. Un.e professeur.e donnant ces deux cours aura ainsi deux lignes pour ces deux cours; et les conditions dans une requête sont appliquées sur une seule ligne à la fois.

Ce que l'on voudrait, c'est avoir sur une même ligne dans la table servant de source de données pour une requête SQL pour un.e professeur.e, deux des cours qu'il ou elle enseigne pour pouvoir faire une condition du type « cours1='20001' AND cours2='20003' »...

Comment faire? Utiliser **deux copies de la table COURS**!

- Une pour vérifier l'enseignement du cours 20001
- L'autre pour vérifier l'enseignement du cours 20003

Il faut donc commencer par construire la requête qui nous permettra d'obtenir cette table. On y retrouvera deux versions de la table COURS, l'une nommée COURS1 et l'autre COURS2 ainsi qu'une version de la table PROF. Nous travaillons avec une seule version de la table PROF comme on souhaite obtenir, pour une même personne, deux de ses cours. Il faut bien entendu "attacher" la table PROF à ces deux tables COURS sur la base du champ NO_PROF :



no_etud	no_cours
10001	20001
10002	20001
10002	20004
…	…

no_etud	nom
10001	Wagner, Richard
10002	Bretécher, Claire
10003	Tremblay, Michel
…	…

no_etud	no_cours
10001	20001
10002	20001
10002	20004
…	…

Utilisation de deux tables SUIT

La requête SQL pour obtenir ce résultat est la suivante :

```sql
SELECT suit1.no_cours as "premier cours suivi", etud.no_etud, etud.nom, suit2.no_cours as "deuxième cours suivi"
FROM etud, suit AS suit1, suit AS suit2
WHERE etud.no_etud=suit1.no_etud
AND etud.no_etud=suit2.no_etud;
```

Les deux versions de la table SUIT sont nommées à l'aide d'alias dans la clause FROM. On obtiendrait ainsi la table de tous les couples de cours suivis par les étudiant.e.s :

premier cours suivi	no_etud	nom	deuxième cours suivi
20001	10001	Wagner, Richard	20001
20001	10002	Bretécher, Claire	20001
20001	10002	Bretécher, Claire	20004
20001	10002	Bretécher, Claire	20005
20004	10002	Bretécher, Claire	20001
20004	10002	Bretécher, Claire	20004
20004	10002	Bretécher, Claire	20005
20005	10002	Bretécher, Claire	20001
20005	10002	Bretécher, Claire	20004
20005	10002	Bretécher, Claire	20005
20002	10003	Tremblay, Michel	20002
20002	10003	Tremblay, Michel	20004
20004	10003	Tremblay, Michel	20002
20004	10003	Tremblay, Michel	20004
20001	10004	Asimov, Isaac	20001
20001	10004	Asimov, Isaac	20003
20001	10004	Asimov, Isaac	20005
20003	10004	Asimov, Isaac	20001
20003	10004	Asimov, Isaac	20003
20003	10004	Asimov, Isaac	20005
20005	10004	Asimov, Isaac	20001
20005	10004	Asimov, Isaac	20003
20005	10004	Asimov, Isaac	20005
20004	10006	Smithwick, Dale	20004
20004	10006	Smithwick, Dale	20007
20007	10006	Smithwick, Dale	20004
20007	10006	Smithwick, Dale	20007

Table de résultats

Ainsi, si on veut identifier les étudiant.e.s qui suivent les cours 20001 et 20004, il ne reste qu'à ajouter les conditions nécessaires à la requête :

```
SELECT suit1.no_cours as "premier cours suivi", etud.no_etud, etud.nom, suit2.no_cours as "deuxième cours suivi"
FROM etud, suit AS suit1, suit AS suit2
WHERE etud.no_etud=suit1.no_etud
AND etud.no_etud=suit2.no_etud
AND suit1.no_cours=20001
AND suit2.no_cours=20004;
```

Une seule ligne serait alors retournée, comme une seule étudiante suit en même temps ces deux cour :

premier cours suivi	no_etud	nom	deuxième cours suivi
20001	10002	Bretécher, Claire	20004

Table de résultats

2.4. Quelques fonctions/opérations utiles

Fonctions/opérations utiles

Fonction/opération	Description
Troncature (%)	• S'utilise à droite, à gauche, au milieu • Représente 0 ou plusieurs caractères • Exemple : `SELECT * FROM prof WHERE nom LIKE 'Asi%';`
Masque (_)	• S'utilise à droite, à gauche, au milieu • Représente 1 caractère • Exemple : `SELECT * FROM prof WHERE nom LIKE '_simov%';`
Concaténation (**CONCAT**)	• Pour accoler des chaînes de caractères dans une même colonne (besoins de lisibilité) • Exemple : `SELECT CONCAT(no_cours, ' - ', titre) FROM cours;`
BETWEEN ... AND ...	• Expression comprise entre 2 valeurs • Exemple : `SELECT * FROM etud WHERE dat_nais BETWEEN '1970-01-01' AND '1990-01-01';`
IF(*condition***,** *valeur si vraie***,** *valeur si faux***)**	• Expression conditionnelle • Exemple : `SELECT etud.no_etud, nom, cours.no_cours, titre, IF(note>=60,'Réussite','Échec') AS 'Réussite du cours' FROM etud, suit, cours WHERE etud.no_etud=suit.no_etud AND suit.no_cours=cours.no_cours AND note_p;`

Fonction/opération	Description
IFNULL(*champ/expression***,** *valeur à afficher si NULL***)**	• Pour remplacer les valeurs nulles par autre chose • Exemple : `SELECT no_etud, nom, IFNULL(dat_nais,'Inconnue') AS 'Date de naissance' FROM etud;`
IN	• Pour vérifier si la valeur du champ est comprise dans un ensemble de valeurs • Exemple : `SELECT * FROM etud WHERE no_etud IN ('10001','10002','10003');`

3. Ressources en lien avec le cours

Matériel de cours

- *Notes de cours [cf. sci6306_cours4_notes.pdf]*