



SYSTÈMES D'INFORMATION

FÉVRIER 2025

LICENCE 2 INFORMATIQUE

INF 214

(PAR DR. JUSTIN MOSKOLAI)
QUALIFIÉ CNU 27^E SESSION



Chapitre IV : Le Langage SQL

- Présentation générale
- recherche dans une BD
- Définition et modification de schémas
- Mise à jour de données

BREF APERÇU DE SQL



- ❑ **SQL (Structured Query Language)**
- ❑ **Bref historique**
 - langage créé par IBM en 1981,
 - dérivé de SEQUEL, dérivé de SQUARE
- ❑ **Structure du langage**

3 familles fonctionnellement distinctes

- LDD (langage de définition de données)
 - permet la description de la structure des données)
- LMD (langage de manipulation des données)
 - permet de rechercher, modifier, mettre à jour, ajouter des données
- LCD (langage de contrôle des données)
 - contient les primitives de gestion des transactions et des privilèges d'accès aux données

NOTATIONS



❑ Dans la suite

- $[a]$ signifie que l'élément a est optionnel
- $\{a\ b\}$ signifie que les éléments a et b sont considérés comme un élément unique
- $\{a\ |\ b\}$ signifie un choix possible entre l'alternative a ou b
- $\langle a \rangle$ signifie que a est un paramètre qui doit être remplacé par une valeur effective
- un exposant $^+$ signifie que l'élément qui précède peut être répété n fois ($n > 0$), chaque occurrence étant séparé de la précédente par une virgule
- un exposant multiplié $*$ indique que l'élément qui précède peut être répété n fois ($n \geq 0$), chaque occurrence étant séparé de la précédente par une virgule

LA RECHERCHE DE DONNÉES



❑ Principales requêtes

- projection
- sélection
- jointure
- sous-questions
- questions quantifiées
- union / intersection
- fonctions de calculs et de groupes (agrégats)
- fonctions de dates

LA PROJECTION



❑ Définition et remarques

- en algèbre relationnelle, cette opération effectue l'extraction des colonnes (attributs) spécifiées, puis élimine les tuples en doublon
- en SQL, les doublons ne sont pas éliminés, à moins que cela soit explicitement spécifié avec le mot-clé `DISTINCT`
- SQL permet en outre d'appliquer des fonction de calcul sur les colonnes (ex. fonctions arithmétiques d'addition, soustraction, multiplication...)

❑ Syntaxe SQL

```
SELECT [ALL|DISTINCT] <expression de valeurs>+  
FROM <nom de table> [nom de variable] ;
```

LES DONNÉES DES EXEMPLES À SUIVRE

□ Soient les relations suivantes



- etudiant (matricule, nom, prenom)
- stage (code, #matricule, entreprise, note)

etudiant	matricule	nom	prenom
stage	SJP1001	AKONO	Elie
	SJP1002	TEMGOUA	Marlène
	SJP1003	KEMMO	Paul
	SJP1004	AMADOU	Idrissou
	SJP1005	AKONO	Elie

etudiant	code	matricule	entreprise	note
stage	L1SJP13001	SJP1001	SONEL	10
	L1SJP13002	SJP1002	SONEL	11
	L1SJP13003	SJP1003	ALUCAM	15

EXEMPLES (PROJECTION)



❑ Exemples de requêtes

(donner les noms et prénoms des étudiants ; doublons possibles)

```
SELECT nom, prenom FROM etudiant ;
```

(donner les noms et prénoms des étudiants, sans les doublons)

```
SELECT DISTINCT nom, prenom FROM etudiant ;
```

(donner tous les attributs de la relation etudiant)

```
SELECT DISTINCT * FROM etudiant ;
```

(donner le matricule et la moyenne sur 10 des notes de stage)

```
SELECT matricule, note/2 as note_sur_dix FROM stage ;
```



```
1 SELECT nom, prenom FROM etudiant ;
```

⚙️ Favoris Historique		
nom	prenom	
AKONO	Elie	
TEMGOUA	Marlène	
KEMMO	Paul	
AMADOU	Idrissou	
AKONO	Elie	

```
1 SELECT DISTINCT nom, prenom FROM etudiant ;
```

⚙️ Favoris Historique		
nom	prenom	
AKONO	Elie	
TEMGOUA	Marlène	
KEMMO	Paul	
AMADOU	Idrissou	

```
1 SELECT DISTINCT * FROM etudiant ;
```

⚙️ Favoris Historique			
matricule	nom	prenom	
SJP1001	AKONO	Elie	
SJP1002	TEMGOUA	Marlène	
SJP1003	KEMMO	Paul	
SJP1004	AMADOU	Idrissou	
SJP1005	AKONO	Elie	

```
1 SELECT matricule, note/2 as note_sur_dix FROM stage ;
```

⚙️ Favoris Historique		
matricule	note_sur_dix	
SJP1001	5.0000	
SJP1002	5.5000	
SJP1003	7.5000	

LA SÉLECTION



❑ Définition et remarque

- une sélection s'exprime comme une projection avec en plus une condition de recherche

❑ Syntaxe SQL

```
SELECT [ALL|DISTINCT] {<expression de valeurs>+ | *}  
FROM <nom de table> [nom de variable]  
WHERE <condition de recherche> ;
```

LA SÉLECTION



❑ Condition de recherche

- elle définit un critère de recherche à l'aide de prédicats
- un prédicat de restriction permet de comparer deux expressions de valeurs

❑ Exemples de prédicats

- de comparaison : `=`, `≠`, `<`, `>`, `≤`, `≥`
- d'intervalle : `BETWEEN`
(teste si la valeur est comprise entre 2 constantes)
- de comparaison de texte : `LIKE`
(teste si une chaîne de caractère contient 1 ou plusieurs sous-chaînes)
- de test de nullité : teste si un attribut est égal à `NULL`
(i.e. sa valeur est inconnue)
- d'appartenance : teste (avec `IN`) si la valeur d'un attribut appartient à une liste de constantes

EXEMPLES (SÉLECTION)



```
SELECT matricule FROM stage WHERE note < 15 ;
```

```
SELECT matricule FROM stage WHERE note BETWEEN 10 AND 12 ;
```

```
SELECT matricule FROM stage WHERE note IN (15, 20) ;
```

(le nom commence par MA, suivi de n'importe quels caractères)

```
SELECT nom, prenom FROM etudiant WHERE nom LIKE 'MA%' ;
```

(le prénom commence par P, suivi de n'importe quels caractères, très exactement au nombre de 4)

```
SELECT nom, prenom FROM etudiant  
WHERE prenom LIKE 'P- - - -' ;
```

(le prénom commence par n'importe quel caractère, très exactement un seul, suivi de n'importe quels caractères)

```
SELECT nom, prenom FROM etudiant  
WHERE prenom LIKE '-M%' ;
```

LA JOINTURE



❑ Définition et remarques

- un cas particulier de jointure sans qualification est le produit cartésien
- la jointure avec qualification peut s'exprimer comme la restriction du produit cartésien par un prédicat

❑ Syntaxe SQL

```
SELECT [ALL|DISTINCT] {<expression de valeurs>+ | *}  
FROM <nom de table_1> , ... , <nom de table_n>  
[WHERE <condition de jointure>] ;
```

EXEMPLES (JOINTURE)



```
SELECT nom, prenom, note
FROM etudiant, stage
WHERE etudiant.matricule = stage.matricule ;
```

```
SELECT nom, prenom, note
FROM etudiant, stage
WHERE etudiant.matricule = stage.matricule
      AND note > 13 ;
```

```
SELECT nom, prenom, note
FROM etudiant, stage
WHERE etudiant.matricule = stage.matricule
      AND note BETWEEN 15 AND 20 ;
```

```
SELECT etudiant.nom, prenom, note
FROM etudiant, stage
WHERE etudiant.matricule = stage.matricule
      AND etudiant.matricule LIKE 'SJP1%' ;
```

```
1 SELECT nom, prenom, note
   FROM etudiant, stage
   WHERE etudiant.matricule = stage.matricule ;
```

⚙️ ▾	Favoris ▾	Historique ▾
nom	prenom	note
AKONO	Elie	10
TEMGOUA	Marlène	11
KEMMO	Paul	15

```
1 SELECT nom, prenom, note
   FROM etudiant, stage
   WHERE etudiant.matricule = stage.matricule
      AND note > 13 ;
```

⚙️ ▾	Favoris ▾	Historique ▾
nom	prenom	note
KEMMO	Paul	15

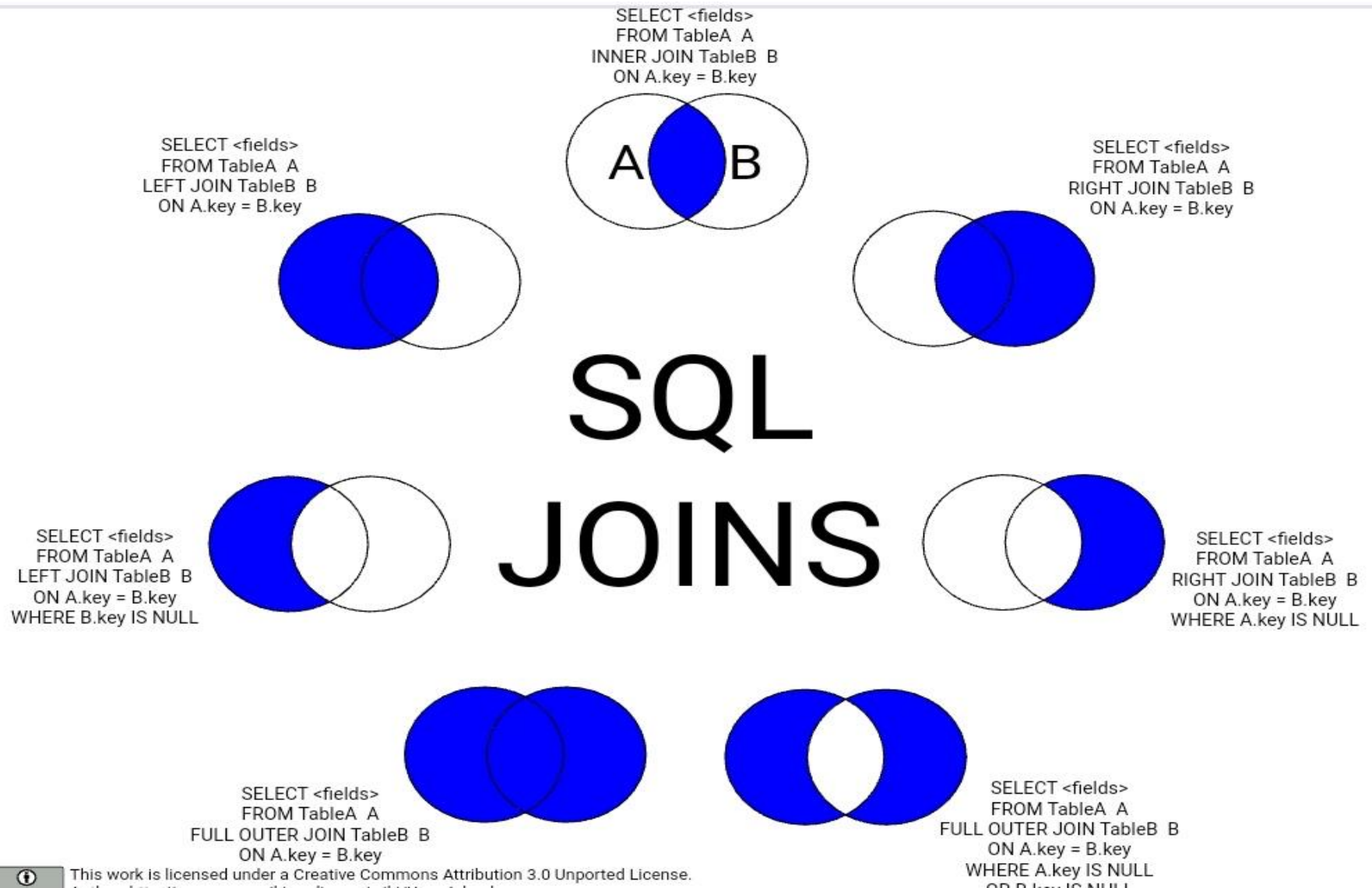
```
1 SELECT nom, prenom, note
   FROM etudiant, stage
   WHERE etudiant.matricule = stage.matricule
      AND note BETWEEN 10 AND 13 ;
```

⚙️ ▾	Favoris ▾	Historique ▾
nom	prenom	note
AKONO	Elie	10
TEMGOUA	Marlène	11

```
1 SELECT etudiant.matricule, nom, prenom, note
   FROM etudiant, stage
   WHERE etudiant.matricule = stage.matricule
      AND etudiant.matricule LIKE 'SJP1%' ;
```

2				
⚙️ ▾	Favoris ▾		Historique ▾	
matricule	nom	prenom	note	
SJP1001	AKONO	Elie	10	
SJP1002	TEMGOUA	Marlène	11	
SJP1003	KEMMO	Paul	15	

CAS PARTICULIERS DES JOINTURES



CAS PARTICULIER : AUTO-JOINTURE



❑ Définition

- une *auto-jointure* est une jointure d'une relation avec elle-même,
- très pratique lorsque la condition de jointure compare des attributs de la même relation, en particulier pour les relations ayant une structure hiérarchique (par ex. relation parent/enfant...)

❑ Syntaxes SQL d'une auto-jointure

```
SELECT [ALL|DISTINCT] {<expression de valeurs>+ | *}  
FROM <nom de table_gauche>  
      INNER JOIN <nom de table_droite>  
      ON <condition de jointure>  
[WHERE <condition de restriction>] ;
```

ou

```
SELECT [ALL|DISTINCT] {<expression de valeurs>+ | *}  
FROM <nom de table> <variable 1>, <nom même table>  
      <variable 2>  
WHERE <condition de jointure> [AND <condition de  
restriction>] ;
```

CAS PARTICULIER : JOINTURES EXTERNES



❑ Rappel

- une *jointure externe* permet de récupérer les enregistrements des relations correspondant à la condition de jointure, mais aussi ceux pour lesquels il n'existe pas de correspondance

❑ Syntaxe SQL d'une jointure externe

```
SELECT [ALL|DISTINCT] {<expression de valeurs>+ | *}  
FROM <nom de table_gauche>  
[LEFT OUTER | RIGHT OUTER | FULL OUTER] JOIN  
<nom de table_droite> ON <condition de jointure>  
[WHERE <condition de restriction>] ;
```

EXEMPLES (AUTO-JOINTURE ET JOINTURES EXTERNES)



(noms, prénoms, notes des étudiants, y compris ceux qui n'ont pas effectué de stage → il s'agit peut-être d'un oubli de saisie !)

```
SELECT nom, prenom, note
FROM etudiant LEFT OUTER JOIN stage
      ON etudiant.matricule = stage.matricule ;
```

(noms, prénoms, notes des étudiants dont la note de stage > 13)

```
SELECT nom, prenom, note
FROM etudiant LEFT OUTER JOIN stage
      ON etudiant.matricule = stage.matricule
WHERE note > 13;
```

```
1 SELECT nom, prenom, note
   FROM etudiant LEFT OUTER JOIN stage
   ON etudiant.matricule = stage.matricule ;
```

⚙️	Favoris	Historique
nom	prenom	note
AKONO	Elie	10
TEMGOUA	Marlène	11
KEMMO	Paul	15
AMADOU	Idrissou	NULL
AKONO	Elie	NULL

```
1 SELECT nom, prenom, note
   FROM etudiant LEFT OUTER JOIN stage
   ON etudiant.matricule = stage.matricule
   WHERE note > 13;
```

⚙️	Favoris	Historique
nom	prenom	note
KEMMO	Paul	15

LES SOUS-QUESTIONS




❑ Le principe : requêtes imbriquées

- SQL permet l'imbrication de sous-questions dans la clause `WHERE`

sous-question dont le résultat peut être :

- une valeur simple
- ou un ensemble de valeurs



❑ Syntaxe

`SELECT ... FROM ... WHERE ... SELECT ...`

❑ Remarques

- une sous-question peut être considérée argument des prédicats (de comparaison ou d'appartenance à une liste)
- une sous-question elle-même invoquer de sous-question

EXEMPLES (REQUÊTES IMBRIQUÉES)



(noms et prénoms des étudiants ayant effectué un stage à la SONEI)

```
SELECT nom, prenom FROM etudiant
WHERE matricule IN (
SELECT matricule FROM stage
WHERE entreprise = 'SONEL') ;
```




(auto-jointure → matricules et notes des étudiants dont la note de stage est supérieure à la moyenne des notes de tous les étudiants)

```
SELECT s1.matricule, s1.note FROM stage s1
INNER JOIN stage s2
ON s1.matricule = s2.matricule
WHERE s1.note > (SELECT AVG(note) FROM stage) ;
```

```

1 SELECT nom, prenom FROM etudiant
   WHERE matricule IN (
     SELECT matricule FROM stage
       WHERE entreprise = 'SONEL' ) ;

```

 Favoris  Historique 	
nom	prenom
AKONO	Elie
TEMGOUA	Marlène

```

1 SELECT s1.matricule, s1.note FROM stage s1
   INNER JOIN stage s2
     ON s1.matricule = s2.matricule
   WHERE s1.note > (
     SELECT AVG(note) FROM stage) ;

```

 Favoris  Historique 	
matricule	note
SJP1003	15

LES QUESTIONS QUANTIFIÉES



❑ Le contexte

- on peut être amené à comparer la valeur d'un attribut :
 - à tous les résultats d'une sous-question,
 - ou seulement à l'une quelconque des valeurs trouvées
- il est également possible de vouloir tester si la valeur d'un attribut existe dans le(s) résultat(s) d'une sous-question

❑ La syntaxe en SQL

- SQL propose l'écriture de sous-questions quantifiées avec les mots-clés suivants, précédant la sous-question : `ALL`, `ANY`, `EXISTS`

❑ Signification

- `ALL` permet de vérifier si la comparaison est vraie pour tous les résultats de la sous-question
- `ANY` vérifie si elle est vraie pour au moins un résultat de la sous-question
- `EXISTS` teste l'existence de la valeur d'un attribut dans le résultat

EXEMPLES (QUESTIONS QUANTIFIÉES)



(matricules des étudiants ayant une note supérieure à la note de tous les étudiants i.e. le ou les étudiants ayant la meilleure note de stage)

```
SELECT matricule FROM stage
      WHERE note >= ALL
                SELECT note FROM stage ;
```

(matricules des étudiants ayant une note supérieure à la note d'au moins un étudiant)

```
SELECT matricule FROM stage
      WHERE note > ANY
                SELECT note FROM stage ;
```

(matricules, noms et prénoms des étudiants ayant effectué un stage à la SONEI)

```
SELECT DISTINCT etudiant.matricule, nom, prenom
      FROM stage, etudiant
      WHERE EXISTS
        (SELECT * FROM stage
         WHERE entreprise = 'SONEI'
           AND stage.matricule = etudiant.matricule) ;
```


rappel

code	matricule	entreprise	note
L1SJP13001	SJP1001	SONEL	10
L1SJP13002	SJP1002	SONEL	11
L1SJP13003	SJP1003	ALUCAM	15

1	SELECT matricule FROM stage
2	WHERE note >= ALL
3	(SELECT note FROM stage) ;
⚙ Favoris Historique	
matricule	
SJP1003	

1	SELECT matricule FROM stage
2	WHERE note > ANY
3	(SELECT note FROM stage) ;
⚙ Favoris Historique	
matricule	
SJP1002	
SJP1003	

1	SELECT DISTINCT etudiant.matricule, nom, prenom
2	FROM stage, etudiant
3	WHERE EXISTS
4	(SELECT * FROM stage WHERE entreprise = 'SONEL'
5	AND stage.matricule = etudiant.matricule) ;
6	

	Favoris	Historique
---	---------	------------

matricule	nom	prenom
SJP1001	AKONO	Elie
SJP1002	TEMGOUA	Marlène

L'UNION



❑ Syntaxe SQL

- utilisation du mot-clé `UNION` entre les requêtes relatives à l'opération d'union

❑ Exemple

(matricules des étudiants ayant effectué leur stage à la SONEI, plus les matricules de ceux ayant une note > 14)

```
SELECT matricule FROM stage WHERE entreprise = 'SONEI'  
UNION  
SELECT matricule FROM stage WHERE note > 14 ;
```

1	SELECT matricule FROM stage WHERE entreprise = 'SONEI'
2	UNION
3	SELECT matricule FROM stage WHERE note > 14;
4	

⚙	Favoris ▼	Historique ▼
matricule		
SJP1001		
SJP1002		
SJP1003		

L'INTERSECTION



❑ Syntaxe SQL

- utilisation du mot-clé `INTERSECT` entre les requêtes relatives à l'opération d'intersection

❑ Exemple

(matricules des étudiants ayant effectué leur stage à la SONEI et ayant une note > 10)

```
SELECT matricule FROM stage WHERE entreprise = 'SONEI'  
INTERSECT  
SELECT matricule FROM stage WHERE note > 10 ;
```

LES FONCTIONS DE CALCULS ET DE GROUPE



❑ Les principales fonctions de calcul

- COUNT : pour compter le nombre de valeurs d'un ensemble
- SUM : pour sommer le nombre de valeurs d'un ensemble
- AVG : pour calculer la valeur moyenne d'un ensemble
- MAX : pour calculer la valeur maximum d'un ensemble
- MIN : pour calculer la valeur minimum d'un ensemble

❑ Utilisation

- ces fonctions peuvent être utilisées dans la clause SELECT
(par ex. pour afficher la moyenne générale des notes de stage)
- elle sont également utilisées pour effectuer des calculs sur des groupes avec **GROUP BY** <spécification des colonnes>+
(par ex. pour calculer les moyennes de notes par entreprise)
- il est possible d'appliquer une sélection sur une groupe, avec **HAVING** <condition de sélection>, après la clause **GROUP BY**
(par ex. pour calculer les moyennes de notes par entreprise, mais uniquement pour les notes supérieures à 10)

EXEMPLES (CALCULS ET GROUPES)



(moyenne générale des notes de stage)

```
SELECT AVG(note) as MOYENNE_NOTE FROM stage ;
```

(moyennes des notes de stage par entreprise, avec le nom de l'entreprise concernée)

```
SELECT entreprise, AVG(note) as moyenne FROM stage  
      GROUP BY entreprise ;
```

(idem que précédemment, mais uniquement pour s'il y a plus d'une note dans la groupe i.e. l'entreprise a reçu au moins 2 étudiants)

```
SELECT entreprise, AVG(note) as moyenne FROM stage  
      GROUP BY entreprise HAVING COUNT(matricule) > 1 ;
```

rappel

code	matricule	entreprise	note
L1SJP13001	SJP1001	SONEL	10
L1SJP13002	SJP1002	SONEL	11
L1SJP13003	SJP1003	ALUCAM	15

```
1 SELECT AVG(note) as MOYENNE_NOTE
2 FROM stage;
3
```

MOYENNE_NOTE
12.0000

```
1 SELECT entreprise, AVG(note) as moyenne FROM stage
2 GROUP BY entreprise;
3
4
```

entreprise	moyenne
ALUCAM	15.0000
SONEL	10.5000

```
1 SELECT entreprise, AVG(note) as moyenne FROM stage
2 GROUP BY entreprise HAVING COUNT(matricule) > 1 ;
```

entreprise	moyenne
SONEL	10.5000

FONCTIONS DE DATES



❑ Fonctions de base

```
SELECT NOW() ;  
(retourne la date courante au format 'YYYY-MM-DD')
```

```
SELECT DAY('date_deb') FROM stage ;  
Idem avec DAYOFMONTH()  
(retourne le jour d'une date : 1, 2, ... 31)
```

```
SELECT DAYNAME('date_deb') FROM stage ;  
(retourne le nom du jour de la semaine d'une  
date : Monday, Tuesday, ... Sunday)
```

```
SELECT DAYOFWEEK('date_deb') FROM stage ;  
(retourne l'index du jour de la semaine d'une  
date : 1 pour Monday, 2 pour Tuesday, ... 7 pour  
Sunday)
```

```
SELECT DAYOFYEAR('date_deb') FROM stage ;  
(retourne le jour d'une date dans un  
intervalle de 1 à 366)
```

```
SELECT MONTH('date_deb') FROM stage ;  
(retourne le numéro de mois d'une date : 1,  
2, ... 12)
```

```
SELECT MONTHNAME('date_deb') FROM stage ;  
(retourne les noms de mois des dates de début  
des stages : January, February, ... December)
```

❑ Autres fonctions

```
WEEK()  
YEAR()  
...
```


FONCTIONS DE DATES



```
SELECT entreprise FROM stage
      WHERE TO_DAYS(NOW()) - TO_DAYS(date_deb) <= 30 ;
```

(liste des entreprises au sein desquelles les stages ont débuté depuis moins de 30 jours)

❑ Remarques

- La fonction `TO_DAYS` convertit une date en nombre de jours
 - Elle accepte les champs de type `DATETIME` et ignore la partie horaire
- La fonction `NOW()` calcule la date ou l'heure actuelle (celle de l'horloge du système)

- ❑ ***PRÉSENTATION GÉNÉRALE***
- ❑ ***RECHERCHE DANS UNE BD***
- ❑ ***DÉFINITION ET MODIFICATION DE SCHÉMAS***
- ❑ ***MISE À JOUR DE DONNÉES***

CRÉATION DE TABLES



❑ Syntaxe SQL

CREATE TABLE <nom de table> (élément de table>⁺)

- un nom de table peut être un nom simple (par ex. `etudiant`) ou un nom composé d'un nom de schéma (i.e. une BD) suivi par le nom de la table (par ex. `sjp1.etudiant`)
- un élément de table est
 - soit une définition de colonne (par exemple :
<nom d'un attribut> : <type de donnée>, par ex. `note : FLOAT(n,d)`),
 - soit une définition de contrainte (par ex. `matricule PRIMARY KEY` pour définir le matricule comme clé primaire)

❑ Contraintes d'intégrité

- **les contraintes de colonnes** : permettent de spécifier différentes contraintes d'intégrité portant sur un seul attribut
- **les contraintes de relations** : peuvent porter sur plusieurs attributs

LES CONTRAINTES D'INTÉGRITÉ



❑ Les contraintes de colonnes

- valeur nulle impossible : `NOT NULL`
- unicité des valeurs prises par l'attribut : `UNIQUE`
- contrainte référentielle :
`REFERENCES <table référencée> [colonne référencée]`
(le nom de la colonne référencée est optionnel s'il est identique à celui de la colonne référençante)
- contrainte générale : `CHECK <condition>`
(la condition est une condition pouvant spécifier des plages ou des listes de valeurs possibles)

LES CONTRAINTES D'INTÉGRITÉ



- ❑ **Les contraintes de relation**
(contraintes d'unicité, référentielles ou générales)
 - unicité : `UNIQUE <attribut>+`
 - contrainte référentielle (pour spécifier quelles colonnes référencent celles d'une autre table) :
`FOREIGN KEY (<colonne référençante>+) REFERENCES <table
référéncée> [(<colonne référéncée>+)]`
 - contrainte générale : `CHECK <condition>`

PRINCIPAUX TYPES



❑ Chaînes de caractères

- CHAR(longueur) : l'attribut doit avoir un nb. De caractères exactement égal au nombre spécifié
- VARCHAR (longueur) : le nombre de caractère est variable, et est au maximum, égal à la valeur spécifiée entre parenthèse

❑ Types numériques

- TINYINT (n) : entiers de -128 à +127 ou de 0 à 255
- SMALLINT (n) : entiers de -32768 à +32768 ou de 0 à 65535
- INT (n) : entiers de -2147483648 à 2147483647 ou de 0 à 4294967295
- FLOAT (n, d) : nombres à virgule flottante, en simple précision
- DOUBLE (n, d) : nombres à virgule flottante, en double précision

❑ Types temporels

- DATE : date au format 'YYYY-MM-DD'
- DATETIME : combinaison de date et heure, au format 'YYYY-MM-DD HH:MM:SS'

EXEMPLE (CRÉATION DE TABLES)



❑ Création en SQL de la table associée à la relation STAGE

```
CREATE TABLE etudiant (  
matricule  VARCHAR (10) PRIMARY KEY,  
nom        VARCHAR (80) NOT NULL,  
prenom     VARCHAR (80)  
) ;
```

```
CREATE TABLE stage (  
code          VARCHAR (10) PRIMARY KEY,  
matricule     VARCHAR (10) NOT NULL,  
entreprise    VARCHAR (80),  
note          INT (2),  
FOREIGN KEY matricule REFERENCES etudiant,  
CHECK (note BETWEEN 0 AND 20)  
) ;
```

LA DÉFINITION DE SCHÉMAS



❑ Définition des vues

- une vue est une table virtuelle calculée à partir des tables de base par une question, et dont la syntaxe de création est :

```
CREATE VIEW <nom de la vue> [(<nom de colonne>+)] AS  
<spécification de la question> [WITH CHECK OPTION] ;
```

- exemple (stagiaires de SONEI, avec leurs notes) :

```
CREATE VIEW stagiaires_soneli AS  
SELECT matricule, note FROM stage  
WHERE entreprise = 'SONEL' ;
```

❑ Suppression de tables

```
DROP TABLE <nom de la table>  
exemple : DROP TABLE stages ;
```


LA MODIFICATION D'UN SCHÉMA



❑ Le principe

- changer la structure d'une table en spécifiant la nature du changement

❑ Principales spécifications

- ajout d'une colonne
- suppression d'une colonne
- suppression d'une contrainte
- ajout d'une contrainte
- modification de la définition d'une colonne (changement du type)
- renommage d'une table
- renommage d'une colonne

❑ Syntaxe générale SQL

ALTER TABLE <nom de table> <spécification de modification>⁺

AJOUT DE COLONNE



❑ Syntaxe SQL

```
ALTER TABLE <nom de table>
  ADD {[COLUMN] <nom de colonne> <type de la colonne>}+
  [FIRST | AFTER <nom de colonne>] ;
```

❑ Remarque

- le mot-clé `COLUMN` est optionnel ; toutefois, en le spécifiant, la valeur par défaut de la colonne est définie comme `NULL`

Exemples :

```
ALTER TABLE etudiant
  ADD COLUMN email VARCHAR(80) ;
```

```
ALTER TABLE etudiant
  ADD serie_bac VARCHAR(5) AFTER prenom ;
```

```
ALTER TABLE etudiant
  ADD (adresse VARCHAR(160), ville VARCHAR (30)) ;
```

SUPPRESSION DE COLONNE



❑ Syntaxe SQL

```
ALTER TABLE <nom de table>  
    DROP [COLUMN] <nom de colonne>  
        [RESTRICT | CASCADE] ;
```

❑ Remarques

- l'option [RESTRICT | CASCADE] permet de spécifier une suppression en cascade (ou pas), notamment lorsque la colonne en question est référencée dans une autre table
- la suppression n'est possible que si la colonne considérée ne fait pas partie d'une vue, d'un index ou d'une contrainte (ex. clé)

Exemple :

```
ALTER TABLE etudiant DROP email ;
```

SUPPRESSION ET AJOUT DE CONTRAINTE



❑ Syntaxe SQL de quelques cas de suppression de contrainte

```
ALTER TABLE <nom de table> DROP PRIMARY KEY ;
```

```
ALTER TABLE <nom de table> DROP  
    FOREIGN KEY <nom de la clé> ;
```

```
ALTER TABLE <nom de table> DROP  
    INDEX <nom de l'index> ;
```

❑ Syntaxe SQL de quelques cas d'ajout de contrainte

```
ALTER TABLE <nom de table> ADD  
    [CONSTRAINT] PRIMARY KEY <nom de colonne> ;
```

```
ALTER TABLE <nom de table> ADD  
    [CONSTRAINT] UNIQUE <nom de colonne> ;
```

```
ALTER TABLE <nom de table> ADD  
    [CONSTRAINT] {FOREIGN KEY  
        <nom de colonne> REFERENCES <nom de  
table>}+ ;
```

MODIFICATION DÉFINITION D'UNE COLONNE



❑ Syntaxe SQL de quelques cas de modification

ALTER TABLE <nom de table>

ALTER

{

[COLUMN] <nom de colonne> <type> |

[COLUMN] <nom de colonne> SET DEFAULT <valeur> |

[COLUMN] <nom de colonne> DROP DEFAULT |

[COLUMN] <nom de colonne> {SET | DROP} NOT NULL

}⁺ ;

RENOMMER UNE COLONNE OU UNE TABLE



❑ Syntaxe SQL

```
ALTER TABLE <nom de table>
RENAME {
    COLUMN <ancien nom> TO <nouveau nom de colonne> |
    TO <nouveau nom de table>
}
```

❑ Exemples

```
ALTER TABLE stage
    RENAME COLUMN note TO note_rapport ;
```

```
ALTER TABLE stage
    RENAME TO stage_sjp1 ;
```

- ❑ ***PRÉSENTATION GÉNÉRALE***
- ❑ ***RECHERCHE DANS UNE BD***
- ❑ ***DÉFINITION ET MODIFICATION DE SCHÉMAS***
- ❑ ***MISE À JOUR DE DONNÉES***

LA MISE À JOUR DE DONNÉES



❑ Principales requêtes

- insertion de tuples
- mise à jour de tuples
- suppression de tuples

INSERTION DE TUPLES



❑ Cas d'utilisation

- une insertion de tuples (ajout de nouveaux enregistrements) peut être réalisée en mentionnant les tuples à insérer → directe
- elle peut également se faire à partir des résultats d'une question (requête) → via une question

❑ Syntaxe SQL

```
INSERT INTO <nom de table> [(<nom de colonne>+)]  
{VALUES (<constante>+) | <commande de recherche>} ;
```

- dans le cas où la liste de colonne n'est pas spécifiée, les valeurs insérées seront attribuées aux attributs de la table dans l'ordre dans lequel ils ont été définis dans le schéma (si seulement certaines valeurs sont spécifiées, les autres attributs auront une valeur nulle)

EXEMPLE (INSERTION DE TUPLES)

❑ Insertion directe

```
INSERT INTO etudiant (matricule, nom, prenom)
VALUES ('SJP1006', 'Noundou', 'Lucrèce') ;
```

```
INSERT INTO etudiant (matricule, nom, prenom)
VALUES ('SJP1007', 'Demgne', 'Carole'), ('SJP1008',
'Kidba', 'André');
```

❑ Insertion via une question

- on suppose que la table `etudiant` contient tous les étudiants de SJP
- on souhaite à présent séparer les données dans des tables distinctes

```
INSERT INTO etudiant_sjp2
SELECT matricule, nom, prenom FROM etudiant
WHERE matricule LIKE 'SJP2%' ;
```

MISE À JOUR DE TUPLES



❑ Le principe

- elle permet de changer des valeurs d'attributs des tuples existants
- elle peut être réalisée,
 - soit en indiquant directement les valeurs à modifier
 - soit modifiant une valeurs à partir du résultat d'une question
- si aucune condition n'est spécifiée (dans la clause **WHERE**) tous les tuples seront modifiés !!!

d'où l'intérêt
des vues

❑ Syntaxe SQL

```
UPDATE <nom de table>  
SET {<nom de colonne> = {<expression de valeur> |  
NULL} }+  
[WHERE {<condition de recherche>}] ;
```

EXEMPLE (MISE À JOUR DE TUPLES)



- ❑ **Modification de tous les tuples d'une table
(attention danger !!!)**

```
UPDATE stage  
SET note = 18 ;
```

- ❑ **Modification de tuples vérifiant une condition
(à utiliser avec précaution, si condition mal écrite)**

```
UPDATE stage  
SET note = 18  
WHERE entreprise = 'SONEL' ;
```

```
UPDATE stage  
SET note = 18  
WHERE matricule= 'SJP1001' ;
```

MISE À JOUR AVEC RECHERCHE DANS LA MÊME TABLE



☐ Écriture incorrecte

```
UPDATE stage
SET note = note * 1.1
WHERE matricule IN
  SELECT matricule FROM stage
WHERE note > SELECT AVG(note) FROM stage ;
```

☐ Nécessité de passer par des tables temporaires

```
UPDATE stage s1
SET s1.note = s1.note * 1.1
WHERE s1.matricule IN
  SELECT matricule FROM (SELECT * FROM stage) s2
WHERE s2.note >
  (
    SELECT AVG(s3.note) FROM (SELECT * FROM stage) s3
  ) ;
```

SUPPRESSION DE TUPLES



❑ Le principe

- elle permet d'enlever des tuples existants d'une relation
- si aucune condition n'est spécifiée (dans la clause WHERE) tous les tuples seront supprimés !!!

❑ Syntaxe SQL

```
DELETE FROM <nom de table>  
[WHERE {<condition de recherche>}] ;
```

❑ Exemple

```
DELETE FROM etudiant  
WHERE matricule NOT LIKE 'SJP%' ;
```

SYNTHÈSE