

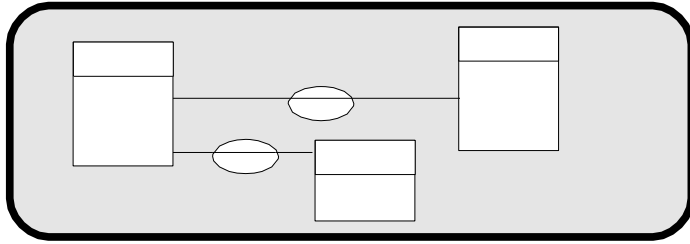
Bases de données et conception des SI

Plan :

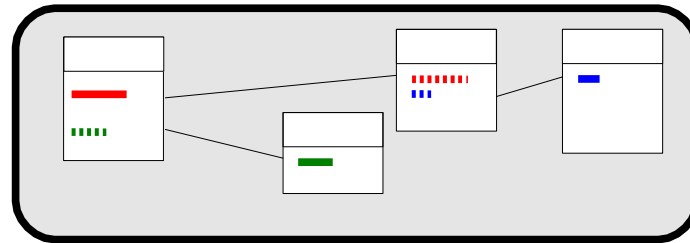
- ▶ Introduction
- ▶ Groupe de fonctions et jointures SQL
- ▶ Les sous requêtes
- ▶ Les vues
- ▶ Gestion des utilisateurs en MYSQL
- ▶ ...
- ▶ Présentation de MERISE
- ▶ ...

Modèle physique des données

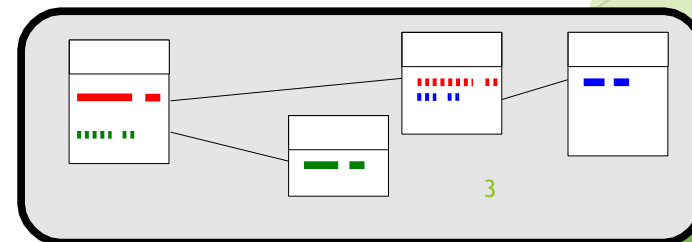
Modèle entité-association (modèle conceptuel des données)



Modèle logique des données



Modèle physique des données



SQL GROUP BY

- ▶ La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.
- ▶ Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de lister regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

```
SELECT colonne1, fonction(colonne2) FROM table GROUP BY colonne1
```

SQL GROUP BY

- une table « **achat** » qui résume les ventes d'une boutique :

id	client	tarif	date
1	Ahmed	102	2021-10-23
2	Sidi	47	2021-10-27
3	Mariem	18	2021-11-05
4	Mariem	20	2021-11-14
5	Ahmed	160	2021-12-03

- La fonction **SUM()** permet d'additionner la valeur de chaque tarif pour un même client.
- Pour obtenir le coût total de chaque client, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif) FROM achat  
GROUP BY client
```

- Le résultat sera donc le suivant :

client	SUM(tarif)
Ahmed	262
Sidi	47
Mariem	38

SQL GROUP BY

- L'utilisation de **GROUP BY** permet d'éviter de présenter plusieurs fois les mêmes lignes.

id	client	tarif	date
1	Ahmed	102	2021-10-23
2	Sidi	47	2021-10-27
3	Mariem	18	2021-11-05
4	Mariem	20	2021-11-14
5	Ahmed	160	2021-12-03

```
SELECT client, SUM(tarif) FROM achat
```

- Le résultat sera donc le suivant sans utiliser GROUP BY :

client	SUM(tarif)
Ahmed	262
Sidi	47
Mariem	38
Mariem	38
Ahmed	262

Utilisation d'autres fonctions de statistiques

Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements, il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :

- **AVG()** pour calculer la moyenne d'un set de valeur. Permet de connaître le prix du panier moyen pour de chaque client
- **COUNT()** pour compter le nombre de lignes concernées. Permet de savoir combien d'achats a été effectué par chaque client
- **MAX()** pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus cher
- **MIN()** pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d'un client
- **SUM()** pour calculer la somme de plusieurs lignes. Permet par exemple de connaître le total de tous les achats d'un client

SQL HAVING

- La condition **HAVING** en SQL est presque similaire à **WHERE** à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

```
SELECT colonne1, SUM(colonne2) FROM nom_table GROUP BY colonne1  
HAVING fonction(colonne2) operateur valeur
```

- **HAVING** est très souvent utilisé en même temps que GROUP BY bien que ce ne soit pas obligatoire.

SQL HAVING

- une table « **achat** » qui résume les ventes d'une boutique :

id	client	tarif	date
1	Ahmed	102	2021-10-23
2	Sidi	47	2021-10-27
3	Mariem	18	2021-11-05
4	Mariem	20	2021-11-14
5	Ahmed	160	2021-12-03

- Pour obtenir le coût total pour les clients qui ont commandé plus de 40 MRU, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif) FROM achat  
GROUP BY client  
HAVING SUM(tarif) > 40
```

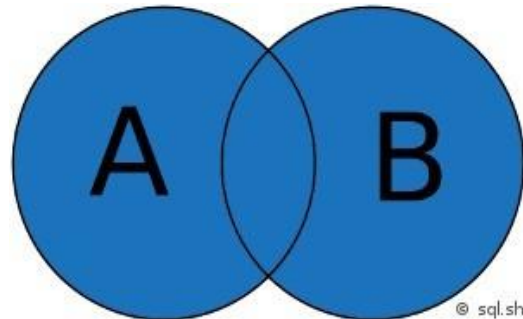
- Le résultat sera donc le suivant :

client	SUM(tarif)
Ahmed	262
Sidi	47

SQL UNION

- ▶ C'est donc une commande qui permet de concaténer les résultats de 2 requêtes ou plus.
- ▶ Pour l'utiliser il est nécessaire que chacune des requêtes à concaténer retournes le même nombre de colonnes, avec les mêmes types de données et dans le même ordre.

```
SELECT * FROM table1 UNION SELECT * FROM table2
```



SQL UNION

La table « magasin1_client » correspond au premier magasin :

prenom	nom	ville	date_naissance	total_achat
Ahmed	Ali	Nouakchott	01/02/2000	135
Mariam	Mohamed	Nouakchott	01/02/2001	75
Ali	Ahmed	Aleg	01/02/2005	27
Mohamed	Ahmed	Nouakchott	01/02/2003	39

La table « magasin2_client » correspond au deuxième magasin :

prenom	nom	ville	date_naissance	total_achat
Outhman	Mohamed	NDB	01/03/2000	285
Sara	Ahmed	NDB	01/02/1999	133
Mariam	Mohamed	Nouakchott	01/02/2001	75
Mohamed	Ahmed	Nouakchott	01/02/2003	39

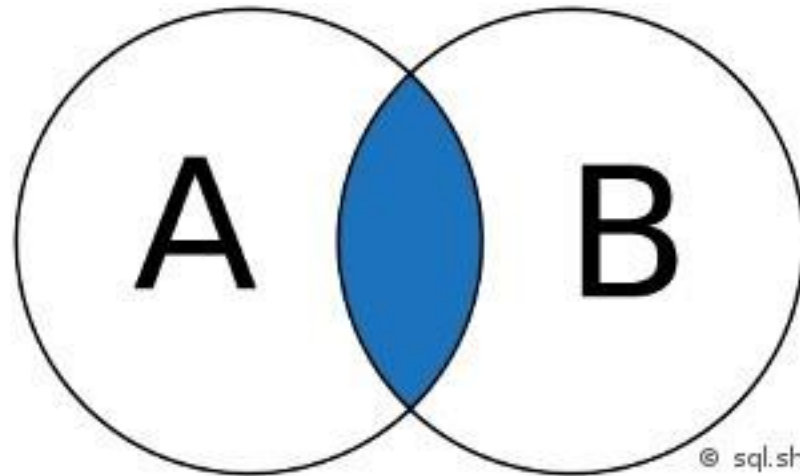
```
SELECT * FROM magasin1_client UNION  
SELECT * FROM magasin2_client
```

prenom	nom	ville	date_naissance	total_achat
Ahmed	Ali	Nouakchott	01/02/2000	135
Mariam	Mohamed	Nouakchott	01/02/2001	75
Ali	Ahmed	Aleg	01/02/2005	27
Mohamed	Ahmed	Paris	01/02/2003	39
Outhman	Mohamed	NDB	01/03/2000	285
Sara	Ahmed	NDB	01/02/1999	133

SQL INTERSECT

- La commande SQL **INTERSECT** permet d'obtenir l'intersection des résultats de 2 requêtes.

```
SELECT * FROM table1 INTERSECT  
SELECT * FROM table2
```



SQL INTERSECT

La table « magasin1_client » correspond au premier magasin :

prenom	nom	ville	date_naissance	total_achat
Ahmed	Ali	Nouakchott	01/02/2000	135
Mariem	Mohamed	Nouakchott	01/02/2001	75
Ali	Ahmed	Aleg	01/02/2005	27
Mohamed	Ahmed	Paris	01/02/2003	39

La table « magasin2_client » correspond au deuxième magasin :

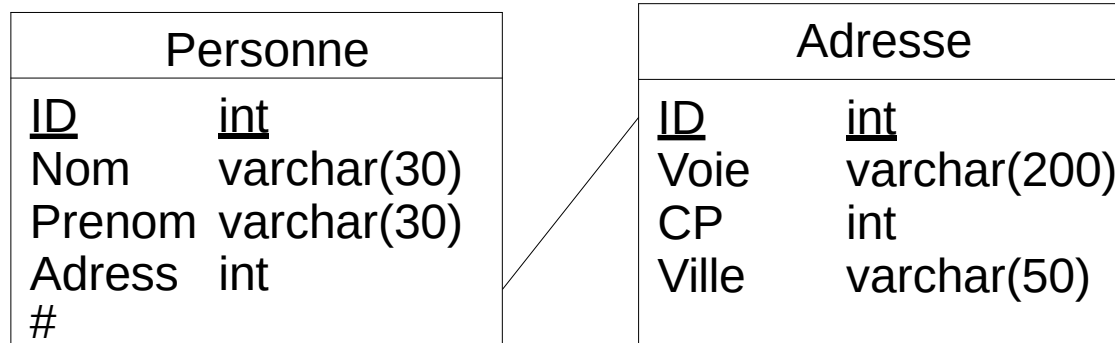
prenom	nom	ville	date_naissance	total_achat
Outhman	Mohamed	NDB	01/03/2000	285
Sara	Ahmed	NDB	01/02/1999	133
Mariem	Mohamed	Nouakchott	01/02/2001	75
Mohamed	Ahmed	Nouakchott	01/02/2003	39

```
SELECT * FROM magasin1_client UNION  
SELECT * FROM magasin2_client
```

prenom	nom	ville	date_naissance	total_achat
Mariem	Mohamed	Nouakchott	01/02/2001	75
Mohamed	Ahmed	Nouakchott	01/02/2003	39

Jointure SQL

- ▶ Les jointures en SQL permettent d'associer plusieurs tables dans une même requête.
- ▶ Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.



L'adresse correspondant à une personne

Types de jointures

Il y a plusieurs méthodes pour associer 2 tables ensemble. Voici la liste des différentes techniques qui sont utilisées :

- ▶ **INNER JOIN** : jointure interne pour retourner les enregistrements quand la condition est vrai dans les 2 tables. C'est l'une des jointures les plus communes.
- ▶ **CROSS JOIN** : jointure croisée permettant de faire le produit cartésien de 2 tables.
- ▶ **LEFT JOIN (ou LEFT OUTER JOIN)** : jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.
- ▶ **RIGHT JOIN (ou RIGHT OUTER JOIN)** : jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifiée dans l'autre table.
- ▶ **FULL JOIN (ou FULL OUTER JOIN)** : jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.
- ▶ **SELF JOIN** : permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.
- ▶ **NATURAL JOIN** : jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL
- ▶ **UNION JOIN** : jointure d'union

SQL INNER JOIN

- ▶ Dans le langage SQL la commande **INNER JOIN**, aussi appelée EQUIJOIN, est un type de jointures très communes pour lier plusieurs tables entre-elles.
- ▶ Cette commande retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.

```
SELECT *  
FROM table1  
INNER JOIN table2 ON table1.id = table2.fk_id
```

La jointure SQL peut aussi être écrite de la façon suivante :

```
SELECT *  
FROM table1  
INNER JOIN table2  
WHERE table1.id = table2.fk_id
```

SQL INNER JOIN

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2022/01/23	A00103	203.14
1	2022/01/14	A00104	124.00
2	2022/01/17	A00105	149.45
2	2022/01/21	A00106	235.35
5	2022/01/02	A00107	47.58

Table utilisateur :

id	prenom	nom	email	ville
1	Ahmed	Mohamed	ahmed.mohamed@example.com	NKC
2	Fatma	Ahmed	Fatma.ahmed@example.com	NDB
3	Sidi	Mohamed	Sidi.mohamed@example.com	BTT
4	Mohamed	Ahmed	Mohamed.ahmed@example.com	NKC

```
SELECT id, prenom, nom, date_achat, num_facture, prix_total FROM utilisateur  
INNER JOIN commande ON utilisateur.id = commande.utilisateur_id
```

Résultats :

id	prenom	nom	date_achat	num_facture	prix_total
1	Ahmed	Mohamed	2022/01/23	A00103	203.14
1	Ahmed	Mohamed	2022/01/14	A00104	124.00
2	Fatma	Ahmed	2022/01/17	A00105	149.45
2	Fatma	Ahmed	2022/01/21	A00106	235.35

SQL CROSS JOIN

- La commande CROSS JOIN est un type de jointure sur 2 tables SQL qui permet de retourner le produit cartésien.

```
SELECT *  
FROM table1  
CROSS JOIN table2
```

Méthode alternative pour retourner les mêmes résultats :

```
SELECT *  
FROM table1, table2
```

SQL CROSS JOIN

Table legume :

l_id	l_nom_fr_fr	l_nom_en_gb
45	Carotte	Carott
46	Oignon	Onion
47	Poireau	Leek

Table fruit :

f_id	f_nom_fr_fr	f_nom_en_gb
87	Banane	Banana
88	Kiwi	Kiwi
89	Poire	Pear

Toutes les combinaisons doivent être affichées. Pour cela il convient d'effectuer l'une ou l'autre des requêtes suivantes :

```
SELECT l_id, l_nom_fr_fr, f_id, f_nom_fr_fr  
FROM legume, fruit
```

ou

```
SELECT l_id, l_nom_fr_fr, f_id, f_nom_fr_fr FROM legume  
CROSS JOIN fruit
```

SQL CROSS JOIN

Résultats :

l_id	l_nom_fr_fr	f_id	f_nom_fr_fr
45	Carotte	87	Banane
45	Carotte	88	Kiwi
45	Carotte	89	Poire
46	Oignon	87	Banane
46	Oignon	88	Kiwi
46	Oignon	89	Poire
47	Poireau	87	Banane
47	Poireau	88	Kiwi
47	Poireau	89	Poire

Le résultat montre bien que chaque légume est associé à chaque fruit. Avec 3 fruits et 3 légumes, il y a donc 9 lignes de résultats ($3 \times 3 = 9$).

SQL LEFT JOIN

- ▶ la commande **LEFT JOIN** (aussi appelée LEFT OUTER JOIN) est un type de jointure entre 2 tables.
- ▶ Cela permet de lister tous les résultats de la table de gauche (left = gauche) même s'il n'y a pas de correspondance dans la deuxième tables.

```
SELECT *  
FROM table1  
LEFT JOIN table2 ON table1.id = table2.fk_id
```

SQL LEFT JOIN

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2022/01/23	A00103	203.14
1	2022/01/14	A00104	124.00
2	2022/01/17	A00105	149.45
2	2022/01/21	A00106	235.35
5	2022/01/02	A00107	47.58

Table utilisateur :

id	prenom	nom	email	ville
1	Ahmed	Mohamed	ahmed.mohamed@example.com	NKC
2	Fatma	Ahmed	Fatma.ahmed@example.com	NDB
3	Sidi	Mohamed	Sidi.mohamed@example.com	BTT
4	Mohamed	Ahmed	Mohamed.ahmed@example.com	NKC

SELECT * FROM utilisateur

LEFT JOIN commande ON utilisateur.id = commande.utilisateur_id

id	prenom	nom	date_achat	num_facture
1	Ahmed	Mohamed	2022/01/23	A00103
1	Ahmed	Mohamed	2022/01/14	A00104
2	Fatma	Ahmed	2022/01/17	A00105
2	Fatma	Ahmed	2022/01/21	A00106
3	Sidi	Mohamed	NULL	NULL
4	Mohamed	Ahmed	NULL	NULL

SQL RIGHT JOIN

- ▶ La commande **RIGHT JOIN** (ou RIGHT OUTER JOIN) est un type de jointure entre 2 tables qui permet de retourner tous les enregistrements de la table de droite (right = droite) même s'il n'y a pas de correspondance avec la table de gauche.
- ▶ S'il y a un enregistrement de la table de droite qui ne trouve pas de correspondance dans la table de gauche, alors les colonnes de la table de gauche

```
SELECT *  
FROM table1  
RIGHT JOIN table2 ON table1.id = table2.fk_id
```

SQL RIGHT JOIN

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2022/01/23	A00103	203.14
1	2022/01/14	A00104	124.00
2	2022/01/17	A00105	149.45
3	2022/01/21	A00106	235.35
5	2022/01/02	A00107	47.58

Table utilisateur :

id	prenom	nom	email	ville
1	Ahmed	Mohamed	ahmed.mohamed@example.com	NKC
2	Fatma	Ahmed	Fatma.ahmed@example.com	NDB
3	Sidi	Mohamed	Sidi.mohamed@example.com	BTT
4	Mohamed	Ahmed	Mohamed.ahmed@example.com	NKC

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture FROM utilisateur  
RIGHT JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Ahmed	Mohamed	1	2022/01/23	A00103
1	Ahmed	Mohamed	1	2022/01/14	A00104
2	Fatma	Ahmed	2	2022/01/17	A00105
3	Sidi	Mohamed	3	2022/01/21	A00106
NULL	NULL	NULL	5	20130302	A00107

SQL FULL JOIN

- ▶ la commande **FULL JOIN** (ou FULL OUTER JOIN) permet de faire une jointure entre 2 tables.
- ▶ L'utilisation de cette commande permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

```
SELECT *  
FROM table1  
FULL JOIN table2 ON table1.id = table2.fk_id
```

SQL FULL JOIN

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2022/01/23	A00103	203.14
1	2022/01/14	A00104	124.00
2	2022/01/17	A00105	149.45
3	2022/01/21	A00106	235.35
5	2022/03/02	A00107	47.58

Table utilisateur :

id	prenom	nom	email	ville	actif
1	Ali	Sidi	Ali.sidi@example.com	Nouakchott	1
2	Fatma	Ahmed	Fatma.ahmed@example.com	Nouakchott	0
3	Aicha	Brahim	Aicha.brahim@example.com	NDB	1
4	Ahmed	Mohamed	Ahmed.mohamed@example.com	Rosso	1

SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture FROM utilisateur
FULL JOIN commande ON utilisateur.id = commande.utilisateur_id

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Ali	Sidi	1	2022/01/23	A00103
1	Ali	Sidi	1	2022/01/14	A00104
2	Fatma	Ahmed	2	2022/01/17	A00105
3	Aicha	Brahim	3	2022/01/21	A00106
4	Ahmed	Mohamed	NULL	NULL	NULL
NULL	NULL		5	2022/03/02	A00107

SQL SELF JOIN

- ▶ En SQL, un SELF JOIN correspond à une jointure d'une table avec elle-même.
- ▶ Ce type de requête n'est pas si commun mais très pratique dans le cas où une table lie des informations avec des enregistrements de la même table.

```
SELECT `t1`.`nom_colonne1`, `t1`.`nom_colonne2`, `t2`.`nom_colonne1`,  
`t2`.`nom_colonne2` FROM `table` as `t1`  
INNER JOIN `table` as `t2` ON `t2`.`fk_id` = `t1`.`id`
```

SQL SELF JOIN

Table utilisateur :

id	prenom	nom	email	manager_id
1	Ali	Sidi	a.sidi@example.com	NULL
2	Fatma	Ahmed	f.ahmed@example.com	NULL
3	Aicha	Brahim	a.brahim@example.com	1
4	Ahmed	Mohamed	a.mohamed@example.com	2

```
SELECT `u1`.`u_id`, `u1`.`u_nom`, `u2`.`u_id`, `u2`.`u_nom` FROM `utilisateur` as `u1`  
LEFT JOIN `utilisateur` as `u2` ON `u2`.`u_manager_id` = `u1`.`u_id`
```

u1_id	u1_prenom	u1_nom	u1_email	u1_manager_id	u2_prenom	u2_nom
1	Ali	Sidi	a.sidi@example.com	NULL	NULL	NULL
2	Fatma	Ahmed	f.ahmed@example.com	NULL	NULL	NULL
3	Aicha	Brahim	a.brahim@example.com	1	Ali	Sidi
4	Ahmed	Mohamed	a.mohamed@example.com	2	Fatma	Ahmed

SQL NATURAL JOIN

- ▶ la commande NATURAL JOIN permet de faire une jointure naturelle entre 2 tables. Cette jointure s'effectue à la condition qu'il y ai des colonnes du même nom et de même type dans les 2 tables.
- ▶ Le résultat d'une jointure naturelle est la création d'un tableau avec autant de lignes qu'il y a de paires correspondant à l'association des colonnes de même nom.

```
SELECT *  
FROM table1  
NATURAL JOIN table2
```

SQL NATURAL JOIN

Table utilisateur :

user_id	user_prenom	user_ville	pays_id
1	Jérémie	Paris	1
2	Ali	NDB	2
3	Sophie	Kidalle	NULL
4	Yann	Lille	9999
5	Léa	Paris	1

Table pays :

pays_id	pays_nom
1	France
2	Mauritanie
3	Mali
4	Suisse

```
SELECT *  
FROM utilisateur NATURAL JOIN pays
```

pays_id	user_id	user_prenom	user_ville	pays_nom
1	1	Jérémie	Paris	France
2	2	Ali	NDB	Mauritanie
NULL	3	Sophie	Kidalle	NULL
9999	4	Yann	Lille	NULL
1	5	Léa	Paris	France

SQL Sous-requête

- ▶ Dans le langage SQL une sous-requête (aussi appelé « requête imbriquée » ou « requête en cascade ») consiste à exécuter une requête à l'intérieur d'une autre requête.
- ▶ Une requête imbriquée est souvent utilisée au sein d'une clause WHERE ou de HAVING pour remplacer une ou plusieurs constantes.

```
SELECT *  
FROM `table`  
WHERE `nom_colonne` = ( SELECT `valeur` FROM `table2` LIMIT 1  
)
```

```
SELECT *  
FROM `table`  
WHERE `nom_colonne` IN ( SELECT `colonne` FROM `table2`  
WHERE `cle_etrangere` = 36  
)
```

SQL Sous-requête

Table Question

q_id	q_date_ajout	q_titre	q_contenu
1	2021/03/24 12:54:32	Comment réparer un ordinateur?	Bonjour, j'ai mon ordinateur de cassé, comment puis-je procéder pour le réparer?
2	2021/03/26 19:27:41	Comment changer un pneu?	Quel est la meilleur méthode pour changer un pneu facilement ?
3	2021/04/18 20:09:56	Que faire si un appareil est cassé?	Estil préférable de réparer les appareils électriques ou d'en acheter de nouveaux?
4	2021/04/22 17:14:27	Comment faire nettoyer un clavier d'ordinateur?	Bonjour, sous mon clavier d'ordinateur il y a beaucoup de poussière, comment fautil procéder pour le nettoyer? Merci.

Table Réponse

r_id	r_fk_question_id	r_date_ajout	r_contenu
1	1	2021/03/27 07:44:32	Bonjour. Pouvez vous expliquer ce qui ne fonctionne pas avec votre ordinateur? Merci.
2	1	2021/03/28 19:27:11	Bonsoir, le plus simple consiste à faire appel à un professionnel pour réparer un ordinateur. Cordialement,
3	2	2021/05/09 22:10:09	Des conseils son disponible sur internet sur ce sujet.
4	3	2021/05/24 09:47:12	Bonjour. Ça dépend de vous, de votre budget et de vos préférence vis à vis de l'écologie. Cordialement,

Avec une telle application, il est peut-être utile de connaître la question liée à la dernière réponse ajoutée sur l'application.

SQL Sous-requête

```
SELECT *  
FROM `question` WHERE q_id = (  
SELECT r_fk_question_id FROM `reponse`  
ORDER BY r_date_ajout DESC LIMIT 1  
)
```

q_id	q_date_ajout	q_titre	q_contenu
3	2021/04/18 20:09:56	Que faire si un appareil est cassé?	Est-il préférable de réparer les appareils électriques ou d'en acheter de nouveaux?

Imaginons maintenant que l'on souhaite obtenir les questions liées à toutes les réponses comprises entre 2 dates.

SQL Sous-requête

Ces questions peuvent être récupérée par la requête SQL suivante :

```
SELECT *  
FROM `question` WHERE q_id IN (  
SELECT r_fk_question_id FROM `reponse`  
WHERE r_date_ajout BETWEEN '2021/01/01' AND '2021/12/31'  
)
```

q_id	q_date_ajout	q_titre	q_contenu
1	2021/03/24 12:54:32	Comment réparer un ordinateur?	Bonjour, j'ai mon ordinateur de cassé, comment puis-je procéder pour le réparer?
2	2021/03/26 19:27:41	Comment changer un pneu?	Quel est la meilleur méthode pour changer un pneu facilement ?
3	2021/04/18 20:09:56	Que faire si un appareil est cassé?	Estil préférable de réparer les appareils électriques ou d'en acheter de nouveaux?

SQL EXISTS

- La commande **EXISTS** s'utilise dans une clause conditionnelle pour savoir s'il y a une présence ou non de lignes lors de l'utilisation d'une sous-requête.

```
SELECT nom_colonne1 FROM `table1`  
WHERE EXISTS (  
  SELECT nom_colonne2 FROM `table2`  
  WHERE nom_colonne3 = 10  
)
```

Table commande :

c_id	c_date_achat	c_produit_id	c_quantite_produit
1	2022/01/08	2	1
2	2022/01/24	3	2
3	2022/02/14	8	1
4	2022/03/23	10	1

Table produit :

p_id	p_nom	p_date_ajout	p_prix
2	Ordinateur	2021/11/17	799.9
3 SUPNUM 2022-2023	Clavier	2021/11/27	49.9
4	Souris	2021/12/04	15
5	Ecran	2021/12/15	250

Il est possible d'effectuer une requête SQL qui affiche les commandes pour lesquels il y a effectivement un produit.

SQL Sous-requête

```
SELECT *  
FROM commande WHERE EXISTS (  
SELECT *  
FROM produit  
WHERE c_produit_id = p_id  
)
```

Table commande :

c_id	c_date_achat	c_produit_id	c_quantite_produit
1	2022/01/08	2	1
2	2022/01/24	3	2
3	2022/02/14	8	1
4	2022/03/23	10	1

Table produit :

p_id	p_nom	p_date_ajout	p_prix
2	Ordinateur	2021/11/17	799.9
3	Clavier	2021/11/27	49.9
4	Souris	2021/12/04	15
5	Ecran	2021/12/15	250

c_id	c_date_achat	c_produit_id	c_quantite_produit
1	2022/01/08	2	1
2	2022/01/24	3	2

SQL ALL

- ▶ Avec cette requête, si nous supposons que dans table1 il y a un résultat avec la valeur 10, voici les différents résultats de la conditions selon le contenu de table2 :
 - ▶ La condition est vrai (cf. TRUE) si table2 contient {-5,0,+5} car toutes les valeurs sont inférieure à 10
 - ▶ La condition est fausse (cf. FALSE) si table2 contient {12,6,NULL,-100} car au moins une valeur est inférieure à 10
 - ▶ La condition est non connue (cf. UNKNOWN) si table2 est vide

```
SELECT *  
FROM table1  
WHERE condition > ALL ( SELECT *  
FROM table2 WHERE condition2  
)
```

SQL ALL

- La commande ALL permet de comparer une valeur dans l'ensemble de valeurs d'une sous-requête.
- En d'autres mots, cette commande permet de s'assurer qu'une condition est « égale », « différente », « supérieure », « inférieure », « supérieure ou égale » ou « inférieure ou égale » pour tous les résultats retourné par une sous-requête.

```
SELECT *  
FROM table1  
WHERE condition > ALL ( SELECT *  
FROM table2 WHERE condition2  
)
```

SQL ANY / SOME

- ▶ La commande **ANY (ou SOME)** permet de comparer une valeur avec le résultat d'une sous-requête.
- ▶ Il est ainsi possible de vérifier si une valeur est « égale », « différente », « supérieur », « supérieur ou égale », « inférieur » ou « inférieur ou égale » pour au moins une des valeurs de la sous-requête.

```
SELECT colonne1 FROM table1  
WHERE colonne1 > ANY ( SELECT colonne1 FROM table2  
)
```

SQL ANY / SOME

- ▶ Supposons que la table1 possède un seul résultat dans lequel colonne1 est égal à 10.
 - ▶ La condition est vrai (cf. TRUE) si table2 contient {21,14,7} car il y a au moins une valeur inférieure à 10
 - ▶ La condition est fausse (cf. FALSE) si table2 contient {20,10} car aucune valeur est strictement inférieure à 10
 - ▶ La condition est non connue (cf. UNKNOWN) si table2 est vide

```
SELECT colonne1 FROM table1
WHERE colonne1 > ANY ( SELECT colonne1 FROM table2
)
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

1. Liste des pilotes pilotant le même avion que le pilote numéro 4.
2. Liste des pilotes GAGNANT plus que tous les pilotes nicois.
3. Liste des pilotes ayant le MEME SALAIRE que le pilote 'PENAULD'.
4. Liste des pilotes ayant la MEME VILLE et le MEME SALAIRE que le pilote 'PENAULD'.
5. Liste des avions qui n'ont jamais volé.
6. Afficher les 3 pilotes les mieux payés.

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

1. Liste des pilotes pilotant le même avion que le pilote numéro 4.

```
SELECT P.PLNUM,P.PLNOM  
FROM PILOTE P, VOL V  
WHERE P.PLNUM=V.PLNUM AND  
V.AVNUM IN (SELECT AVNUM FROM VOL  
WHERE PLNUM=4);
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

2. Liste des pilotes GAGNANT plus que tous les pilotes nicois.

```
SELECT PLNUM,PLNOM,PLPRENOM,SALAIRE  
FROM PILOTE  
WHERE SALAIRE>ALL(SELECT SALAIRE FROM PILOTE  
WHERE VILLE='NICE');
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

3. Liste des pilotes ayant le MEME SALAIRE que le pilote 'PENAULD'.

```
SELECT PLNUM,PLNOM,PLPRENOM,SALAIRE  
FROM PILOTE  
WHERE SALAIRE=(SELECT SALAIRE FROM PILOTE  
WHERE PLNOM=PENAULD');
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

4. Liste des pilotes ayant la MEME VILLE et le MEME SALAIRE que le pilote 'PENAULD'.

```
SELECT PLNUM,PLNOM,PLPRENOM,SALAIRE  
FROM PILOTE  
WHERE (VILLE, SALAIRE)=(SELECT VILLE,SALAIRE  
FROM  
PILOTE  
WHERE PLNOM='PENAULD'  
);
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

5. Liste des avions qui n'ont jamais volé.

```
SELECT AVNUM, AVNOM  
FROM AVION  
WHERE NOT EXISTS (SELECT * FROM VOL  
WHERE VOL.AVNUM=AVION.AVNUM);
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

6. Afficher les 3 pilotes les mieux payés par ordre décroissant des salaires.

```
SELECT SALAIRE  
FROM PILOTE  
ORDER BY SALAIRE DESC LIMIT 3;
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

7. Le numéro et le nom de l'avion qui a volé le plus.

```
SELECT A.AVNUM, V.AVNOM  
FROM AVION A, VOL V WHERE A.AVNUM=V.AVNUM  
GROUP BY A.AVNUM  
HAVING COUNT(*)>=ALL (SELECT COUNT(*) FROM VOL GROUP BY  
AVNUM );
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

8. Créer une autre table pilote2 qui a la même structure que la table pilote.

```
CREATE TABLE PILOTE2 LIKE PILOTE;
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

9. Insérer dans la table pilote2 tous les pilotes qui n'ont jamais volé, en utilisant une sous-requête;

```
INSERT INTO PILOTE2  
SELECT * FROM PILOTE  
WHERE NOT EXISTS (SELECT * FROM VOL WHERE  
VOL.PLNUM=PILOTE.PLNUM);
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

10. Créer une autre table pilote3 qui contient les memes données que la table pilote, en utilisant une sous-requête;

```
CREATE TABLE PILOTE3 AS SELECT * FROM PILOTE;
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

11. Supprimer tous les pilotes de la table pilote3 qui sont dans la table pilote2, en utilisant une sous-requête;

```
DELETE FROM PILOTE3  
WHERE EXISTS (SELECT 1 FROM PILOTE2 WHERE  
PILOTE2.PLNUM=PILOTE3.PLNUM);
```

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

12. Que fait cette requete ?

```
UPDATE PILOTE3 SET salaire=(SELECT MAX(salaire)  
FROM PILOTE) WHERE plnum=9;
```

Elle attribue au pilote numero 9, le salaire le plus grand.

Requêtes

Nous avons les pilote, vol et avion de la base de données vols.

13. Que fait cette requete ?

```
DELETE FROM PILOTE3  
WHERE not exists (SELECT 1 FROM PILOTE3 WHERE  
PILOTE3.plnum=VOL.plnum);
```

Elle supprime les pilotes qui n'ont jamais vole'.

Les vues

Les vues

- ▶ Une vue est une table virtuelle qui est le résultat d'une requête (**SELECT**) à laquelle on a donné un nom.
- ▶ Le nom d'une vue peut être utilisé partout où on peut mettre le nom d'une table : **SELECT, UPDATE, DELETE, INSERT, GRANT, ...**

Création d'une vue : syntaxe

CREATE [OR REPLACE]

VIEW **nom-de-vue** [(attr1, ..., attrn)] AS **requête**

[WITH CHECK OPTION]

[WITH READ ONLY]

Utilisation d'une vue :

comme si elle était une table

```
SELECT ... FROM nom-de-vue  
WHERE ...
```

Suppression d'une vue :

- La suppression d'une vue n'entraîne pas la suppression des données.

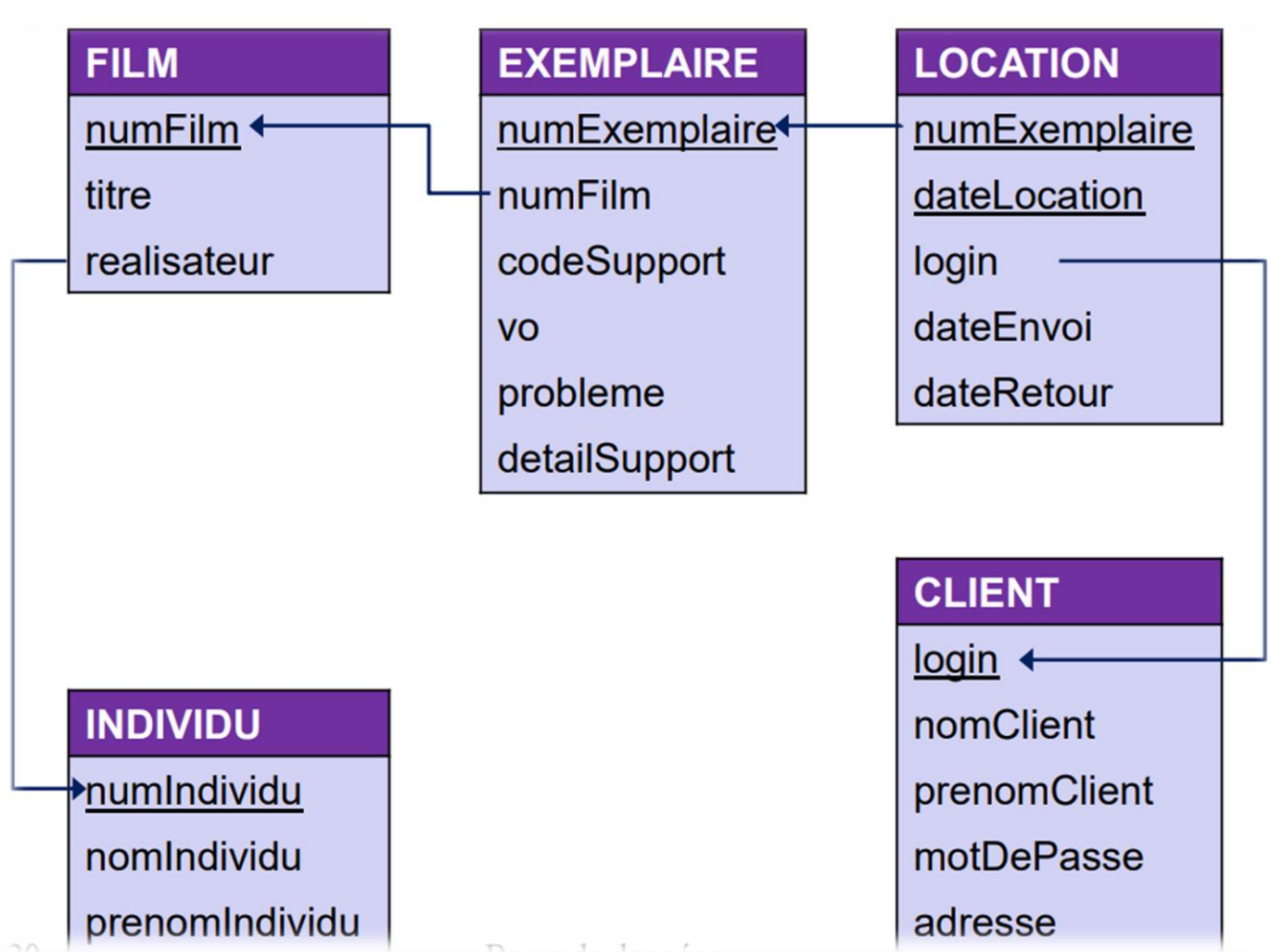
DROP VIEW nom-de-vue

Renommer une vue :

- La suppression d'une vue n'entraîne pas la suppression des données.

RENAME ancien-nom **TO** nouveau-nom

Exemple : vidéothèque



Exemple (2)

```
CREATE OR REPLACE VIEW  
exemplairePlus (num, vo, titre, real,  
support) AS SELECT numExemplaire, vo,  
titre, nomIndividu, codesupport FROM  
Exemplaire E, Film F, Individu WHERE  
E.numFilm = F.numFilm AND realisateur =  
numIndividu AND probleme IS NULL;
```

Exemple (2)

```
SELECT num, titre, dateLocation, login FROM  
exemplairePlus, Location WHERE num =  
numExemplaire AND real = 'SAUTET' AND  
dateRetour IS NULL;
```

Quatre raisons d'utiliser des vues

- 1) Effet macro : remplacer une requête compliquée par des requêtes plus simples
- 2) Confidentialité
- 3) Contraintes d'intégrité
- 4) Augmenter l'indépendance logique

1) Effet macro

- ▶ Remplacer une requête compliquée par des requêtes plus simples

2) Confidentialité :

► exemple

```
CREATE VIEW emprunteurRestreint AS  
SELECT login, nomClient, prenomClient  
FROM client
```

3) Contraintes d'intégrité (CHECK OPTION) :

```
CREATE VIEW anciensExemplaires  
AS SELECT * FROM Exemple WHERE numExemple <  
2000
```

WITH CHECK OPTION;

```
UPDATE anciensExemplaires SET  
= 3812 WHERE numExemple = 1318;
```

Sans 'WITH CHECK OPTION', c'est possible.

Avec 'WITH CHECK OPTION', c'est impossible.

4) Augmenter l'indépendance logique

Les applications utilisant les tables de la base ne doivent pas être modifiées si on change le schéma de la base

Conditions de mise à jour pour les vues

- Pour **UPDATE**, **DELETE**, **INSERT** la vue ne doit pas contenir :
 - Un opérateur ensembliste (**UNION**, **MINUS**, **INTERSECT**)
 - Un opérateur **DISTINCT**
 - Une fonction d'agrégation comme attribut
 - Une clause **GROUP BY**
 - Une jointure (la vue doit être construite sur une seule table)

Conditions de mise à jour pour les vues

- Pour **UPDATE**, **DELETE**, **INSERT** la vue ne doit pas contenir :
 - Les colonnes résultats de l'ordre **SELECT** doivent être des colonnes réelles d'une table de la base et non des expressions
 - Si la vue est construite à partir d'une autre vue, cette dernière doit elle-même vérifier les conditions ci-dessus

Gestion des comptes utilisateurs

Gestion des comptes utilisateurs

- ▶ La gestion des utilisateurs de **MySQL** se fait entièrement grâce à des tables dédiées qui appartiennent au schéma système **mysql**
- ▶ Grâce à des commandes SQL permettant de créer, modifier et supprimer les comptes utilisateurs et leurs droits.
- ▶ La table **user** contient la liste des comptes utilisateurs.
 - ▶ Les colonnes **host** (le nom d'hôte) et **user** (le nom) (colonnes représentent la clé primaire de la table.)
 - ▶ **password** (le mot de passe du compte)
 - ▶ Les droits globaux de l'utilisateur, comme le droit de faire des insertions (**INSERT**), des lectures (**SELECT**)...

Gestion des comptes utilisateurs

- ▶ La table **db** contient les droits spécifiques à une base de données (ou schémas) de chaque utilisateur.
- ▶ La table **tables_priv** contient les droits spécifiques à une table ou à une vue.
- ▶ La table **columns_priv** contient les droits spécifiques à une colonne.
- ▶ La table **procs_priv** contient les droits pour les routines stockées, c'est-à-dire les procédures et les fonctions stockées.

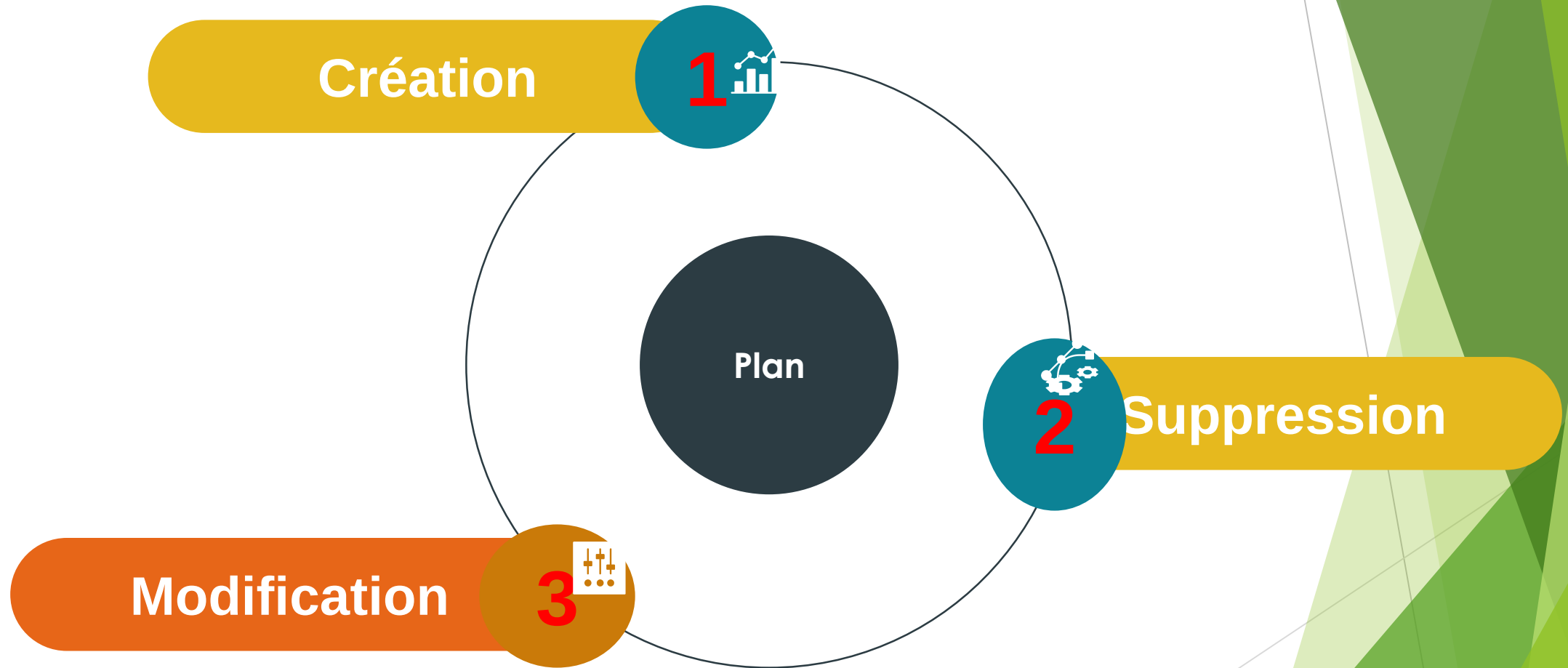
1. **Information_schema** : stocke les informations sur toutes les bases de données. Les tables, les colonnes, le type des colonnes, etc.
2. **performance_schema** : stocke des informations sur les actions effectuées sur le serveur (temps d'exécution, temps d'attente dus aux verrous, etc.)
3. **test** : base de test automatiquement créée (vide au début)
4. **MySQL** : contient de nombreuses informations sur le serveur. Elle stocke **les utilisateurs (user) et leurs privilèges**. (db, *tables_priv*, *columns_priv*, *proc_priv*),

Allez S.V.P à la console
MySQL !



ANALYSE
LOGIQUE
E

Création et modification des utilisateurs



Création et suppression

-- Création

```
CREATE USER 'login'@'hote' [IDENTIFIED BY 'mot_de_passe'];
```

-- Suppression

```
DROP USER 'login'@'hote';
```

Exemple de création

```
CREATE USER 'etudiant'@'localhost' IDENTIFIED BY 'cheikh';
```

Exemple de suppression

```
DROP USER 'etudiant'@'localhost';
```

Création et suppression

Utilisateur

L'utilisateur est donc défini par **deux éléments** :

- son login ;
- l'hôte à partir duquel il se connecte.

Login

Le login est un simple identifiant. Vous pouvez le choisir comme vous voulez. Il n'est pas obligatoire de l'entourer de guillemets, sauf s'il contient des caractères spéciaux comme — ou @

Hôte

L'hôte est l'**adresse à partir de laquelle l'utilisateur va se connecter**. Si l'utilisateur se connecte à partir de la machine sur laquelle le serveur MySQL se trouve, on peut utiliser 'localhost'. Sinon, on utilise en général une adresse IP ou un nom de domaine.

Exemples :

```
CREATE USER 'ahmed'@'localhost' IDENTIFIED BY 'ahmedmed';  
CREATE USER 'fatma'@'194.28.12.4' IDENTIFIED BY 'fatou1';  
CREATE USER 'sidi'@'iscae.mr' IDENTIFIED BY 'iscae';
```

Connexion à plusieurs hôtes différents (le joker %)

- Possibilité de se connecter à partir de plusieurs hôtes différents
- Sans devoir créer un utilisateur par hôte
- On peut préciser des noms d'hôtes partiels
- Ou permettre à l'utilisateur de se connecter à partir de n'importe quel hôte.

Exemples :

-- mohamed peut se connecter à partir de n'importe quel hôte dont l'adresse IP commence par 194.28.12.

```
CREATE USER 'mohamed'@'194.28.12.%' IDENTIFIED BY 'iscae2020';
```

-- mariem peut se connecter à partir de n'importe quel hôte du domaine iscae.mr

```
CREATE USER 'mariem'@'%.iscae.mr' IDENTIFIED BY 'adminiscae';
```

-- amadou peut se connecter à partir de n'importe quel hôte

```
CREATE USER 'amadou'@'%' IDENTIFIED BY 'amda$123';
```

Remarque :

Notez que, si vous ne précisez pas d'hôte, c'est comme si vous autorisiez tous les hôtes
'cheikh'@'%' est équivalent à 'cheikh'

Renommer un utilisateur

Pour modifier l'identifiant d'un utilisateur (login et/ou hôte), on peut utiliser **RENAME USER** ancien_utilisateur **TO** nouvel_utilisateur

Exemple :

RENAME USER cheikh **TO** dhib

Mot de passe

- Le mot de passe de l'utilisateur est donné par la clause **IDENTIFIED BY**
- N'est pas obligatoire, auquel cas l'utilisateur peut se connecter sans donner de mot de passe.
- Du point de vue de la sécurité, il est conseiller d'**Évitez au maximum les utilisateurs sans mot de passe.**
- N'est pas stocké tel quel dans la table *mysql.user*. Il est d'abord crypté (hashé), et c'est cette valeur cryptée qui est stockée.

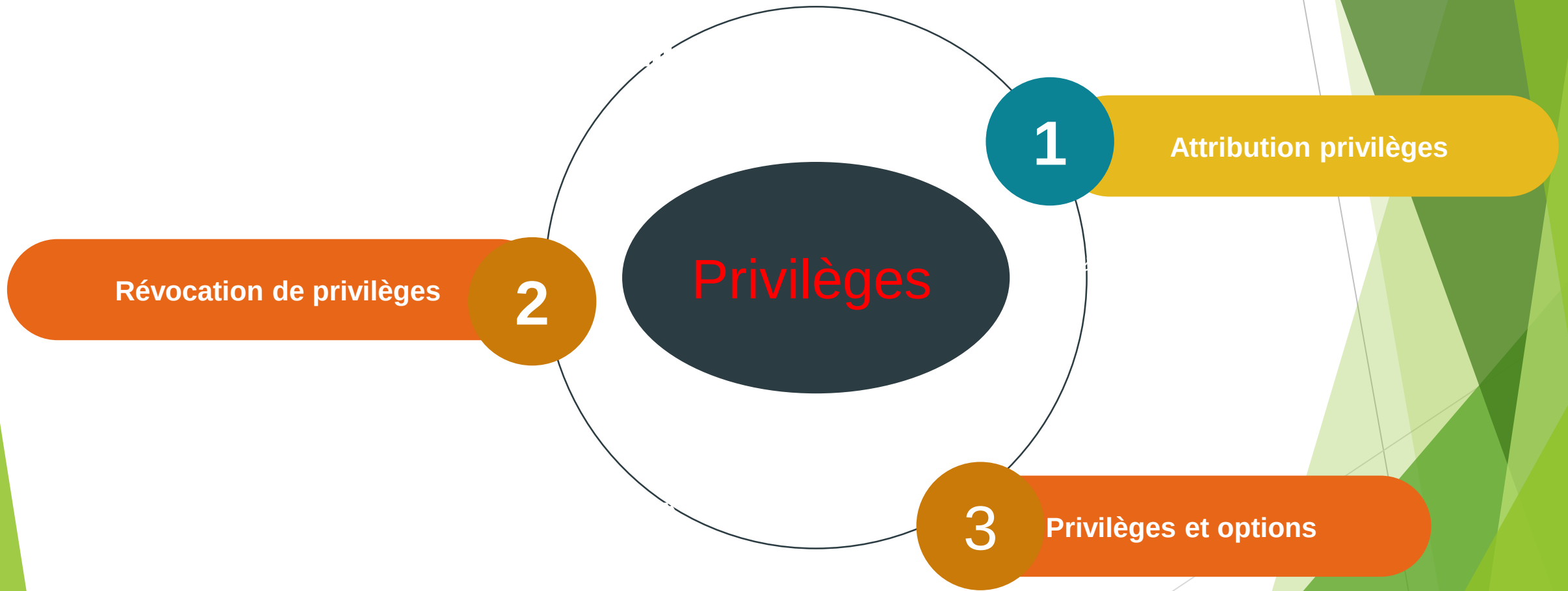
Modifier le mot de passe

Pour modifier le mot de passe d'un utilisateur, on peut utiliser la commande ***SET PASSWORD. Il faut soit***

- avoir les privilèges nécessaires, ou
- Etre connecté avec l'utilisateur dont on veut changer le mot de passe.
- Ne crypte pas le mot de passe automatiquement. Il faut donc utiliser la fonction **PASSWORD()**

Exemple : SET PASSWORD FOR 'etudiant'@'localhost' = PASSWORD('ig2igfp2');

Gestion des utilisateurs – Privilèges



Privilèges des utilisateurs - Introduction

- On se connecte sur MySQL avec un utilisateur (exemple : **root**)
- Chaque utilisateur possède un ensemble de privilèges relatifs aux données stockées sur le serveur
- Les privilèges peuvent être la **sélection** des données, la **modification**, la **création** des objets, etc.
- Les privilèges peuvent être attribué par niveau : **base de données**, **tables**, **colonnes**, **procédures**, etc.

Privilèges des utilisateurs : stockage

- *User* : on y stocke les privilèges globales
- *db* : privilèges au niveau des bases de données.
- *tables_priv* : privilèges au niveau des tables.
- *columns_priv* : privilèges au niveau des colonnes.
- *proc_priv* : privilèges au niveau des routines (procédures et fonctions stockées).

Privilèges des utilisateurs : types de droits

- Lors de la création, pas de droits (sauf la connexion)
- Donc pas de possibilité de faire des actions
- **CRUD : Insert, select, update et delete**
- Privilèges sur les bases de données, les tables et les vues : create table, create temporary table, create view ((il faut également avoir le privilège SELECT sur les colonnes sélectionnées par la vue), ALTER (modification d'une table avec alter table), DROP (suppression des tables, vues et bases de données)
- Utilisateurs : CREATE USER, drop user, rename user, set password)
- Autres : gestion des procédures, déclencheurs, etc.

Privilèges des utilisateurs – Niveaux

Commande	Niveau d'application	Table de stockage
* * .	Toutes les bases de données, tous les objets	Mysql.user
*	1. Sans sélectionner une bases : équivalent à *.* 2. Avec la sélection d'une base : sur tous les objets de cette base	1. MySQL.user 2. Mysql.db
Nom_base.*	Privilège de base de données : s'applique à tous les objets de la base nom_bdd	mysql.db
nom_bdd.nom_table	Privilège de table	mysql.tables_priv
nom_table (Privilège de table)	s'applique à la table nom_table de la base de données dans laquelle on se trouve, sélectionnée au préalable avec USE nom_bdd	mysql.tables_priv
Privilège sur colonnes	Sur des colonnes d'une table	mysql.columns_priv
Nom_bdd.routine	Pour les procédures, fonctions ,etc.	Mysql.procs_priv

Privèges des utilisateurs : attribution et suppression des privilèges

1. Attribution : commande GRANT

- Pour attribuer une privilège, l'utilisateur doit avoir le privilège **Grant option**
- **Syntaxe :**
GRANT privilege [(liste_colonnes)] [, privilege [(liste_colonnes)], ...] **ON** [type_objet] niveau_privilege **TO** utilisateur [IDENTIFIED BY mot_de_passe];

- ❑ privilege : le privilège à accorder à l'utilisateur (SELECT, CREATE VIEW, EXECUTE...);
- ❑ (liste_colonnes) : facultatif - liste des colonnes auxquelles le privilège s'applique ;
- ❑ niveau_privilege : niveau auquel le privilège s'applique (*.*, nom_bdd.nom_table...);
- ❑ type_objet : en cas de noms ambigus, il est possible de préciser à quoi se rapporte le niveau, TABLE ou PROCEDURE

```
CREATE USER 'cheikh'@'moi' IDENTIFIED BY  
'MeToo';  
GRANT SELECT,  
    UPDATE (PLNOM, PLPRENOM),  
    DELETE,  
    INSERT  
ON vols.pilote  
TO 'moi'@'localhost';
```

```
GRANT SELECT ON TABLE vols.avion TO  
'moi'@'localhost';
```

Privilèges des utilisateurs-Suppression de privilèges

Révocation: commande **REVOKE**

- ☐ On utilise la commande **REVOKE**
- ☐ On peut retirer plusieurs privilèges à la fois
- ☐ Syntaxe :

REVOKE privilege [, privilege, ...]
ON niveau_privilege
FROM utilisateur;

Exemple :

REVOKE DELETE
ON vols.pilote
FROM 'moi'@'localhost';

Privilèges des utilisateurs : Les privilèges ALL, USAGE et GRANT OPTION

ALL (ALL PRIVILEGES): sert à accorder tous les droits à l'utilisateur (Il faut évidemment préciser le niveau auquel tous les droits sont accordés)

Usage : n'attribue aucun privilège, peut être utile pour modifier l'utilisateur avec GRANT

GRANT OPTION : sert à accorder tous les droits à l'utilisateur (Il faut évidemment préciser le niveau auquel tous les droits sont accordés)

```
GRANT ALL ON vols.vol TO 'moi'@'localhost';
```

```
GRANT SELECT, UPDATE, INSERT, DELETE,  
GRANT OPTION ON vols.* TO 'moi'@'localhost'  
IDENTIFIED BY 'cheikh';
```

```
GRANT USAGE ON *.* TO 'moi'@'localhost';
```

```
GRANT SELECT, UPDATE, INSERT, DELETE  
ON vols.* TO 'moi'@'localhost' IDENTIFIED BY  
'cheikh'  
WITH GRANT OPTION;
```