



François Jacquenet

Professeur d'Informatique

Faculté des Sciences

Laboratoire Hubert Curien – UMR CNRS 5516

18 rue Benoit Lauras

42000 Saint-Etienne

Tél : 04 77 91 58 07

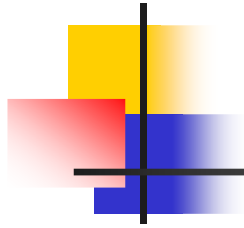
e-mail : Francois.Jacquenet@univ-st-etienne.fr

Web : <http://labh-curien.univ-st-etienne.fr/~fj/bd>

Licence de Sciences et Techniques

Unité d'enseignement BASES DE DONNEES

SQL



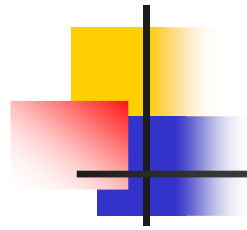
Dans ce cours nous verrons

- SQL = Langage de définition de données
 - CREATE TABLE
 - ALTER TABLE
 - DROP TABLE
- SQL = Langage de manipulation de données
 - INSERT INTO
 - UPDATE
 - DELETE FROM
- SQL = Langage de requêtes
 - SELECT ... FROM ... WHERE ...
 - Sélection
 - Projection
 - Jointure
 - Produit cartésien
 - Union
 - Intersection
 - Différence
 - Les agrégats



Introduction

- **SQL** =
- Inventé chez (centre de recherche d'Almaden en Californie), en 1974 par Astrahan & Chamberlin dans le cadre de System R
- Le langage SQL est standardisé
 - SQL92
 - SQL99
 - SQL2003
 - SQL2008
 - SQL2011
- C'est



SQL : Trois langages en un

- Langage de

- création de relations :
- modification de relations: ALTER TABLE
- suppression de relations:
- vues, index : CREATE VIEW ...

- Langage de


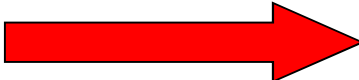

- insertion de tuples:
- mise à jour des tuples: UPDATE
- suppression de tuples:

- Langage de

- SELECT FROM WHERE

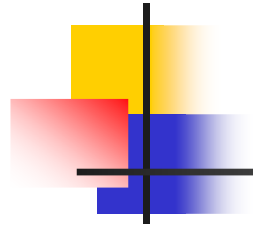


Terminologie

- Relation 
- Tuple 
- Attribut 

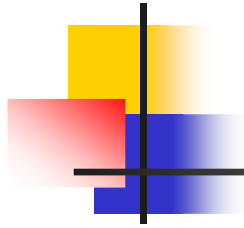
SQL

**Un langage de
définition de données**



Un langage de définition de données

- Commandes pour créer, modifier et supprimer les éléments du schéma
- **CREATE TABLE** : une table (une relation),
- **CREATE VIEW** : créer une vue particulière sur les données à partir d'un SELECT,
- **DROP {TABLE | VIEW}** : une table
ou une vue,
- **ALTER {TABLE | VIEW}** : une table
ou une vue.



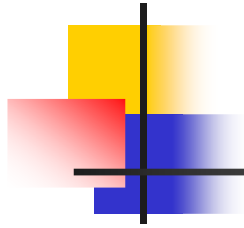
CREATE TABLE

Commande créant une table en donnant son nom, ses attributs et ses contraintes

```
CREATE TABLE nom_table  
  { ( nom-col  type-col  
    [DEFAULT val]  
    [ [CONSTRAINT] contrainte-col] ) *  
    [ [CONSTRAINT] contrainte-table]  
    | AS requête-SQL };
```

Légende :

- {a | b} : a ou b,
- [option];
- * : applicable autant de fois que souhaité;
- mots en capitale : mots-clé.



CREATE TABLE

CREATE TABLE nom_table

{ (nom-col type-col [DEFAULT val] [[CONSTRAINT] contrainte-col])*
[[CONSTRAINT] contrainte-table]
| AS requête-SQL };

Exemple 1

CREATE TABLE Doctorant

(

);

Exemple 2

CREATE TABLE Doctorant

AS SELECT

FROM

WHERE statut=

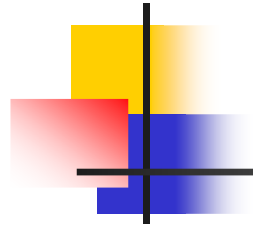
;



Type des colonnes (en MySQL)

■ Numériques

- : idem DECIMAL
- . Possibilité DECIMAL(M,D) M chiffres au total
- - TINYINT 1 octet (de -128 à 127)
 - SMALLINT 2 octets (de -32768 à 32767)
 - MEDIUMINT 3 octets (de -8388608 à 8388607)
 - INT 4 octets (de -2147483648 à 2147483647)
 - BIGINT 8 octets (de -9223372036854775808 à 9223372036854775807)
 - Possibilité de donner la taille de l'affichage : INT(6) => 674 s'affiche 000674
 - Possibilité de spécifier UNSIGNED
 - INT UNSIGNED => de 0 à 4294967296
- : 4 octets par défaut. Possibilité d'écrire FLOAT(P)
- : 8 octets
- : 8 octets



Type des colonnes (en MySQL)

■ Date et Heure



- AAAA-MM-JJ HH:MM:SS
- de 1000-01-01 00:00:00 à '9999-12-31 23:59:59



- AAAA-MM-JJ
- de 1000-01-01 à 9999-12-31



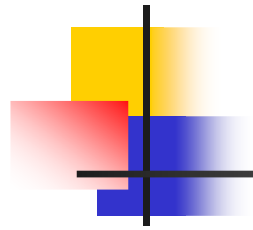
- Date sans séparateur AAAAMMJJHHMMSS



- HH:MM:SS (ou HHH:MM:SS)
- de -838:59:59 à 838:59:59



- YYYY
- de 1901 à 2155



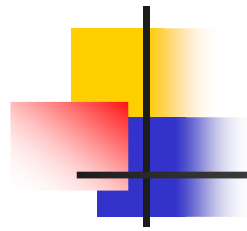
Type des colonnes (en MySQL)

- Chaînes

- $1 \leq n \leq 255$
- $1 \leq n \leq 255$

Exemple :

| | CHAR(4) | | VARCHAR(4) | |
|----------|---------|----------|------------|----------|
| Valeur | Stockée | Taille | Stockée | Taille |
| " | ' ' | 4 octets | " | 1 octets |
| 'ab' | 'ab ' | 4 octets | 'ab' | 3 octets |
| 'abcd' | 'abcd' | 4 octets | 'abcd' | 5 octets |
| 'abcdef' | 'abcd' | 4 octets | 'abcd' | 5 octets |



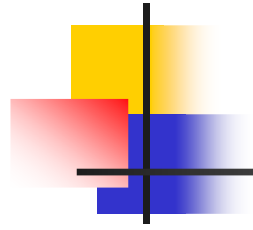
Type des colonnes (en MySQL)

- Chaînes

- TINYBLOB Taille < 2⁸ caractères
- Taille < 2¹⁶ caractères
- MEDIUMBLOB Taille < 2²⁴ caractères
- Taille < 2³² caractères

- TINYTEXT Taille < 2⁸ caractères
- Taille < 2¹⁶ caractères
- MEDIUMTEXT Taille < 2²⁴ caractères
- Taille < 2³² caractères

Les tris faits sur les BLOB tiennent compte de la casse, contrairement aux tris faits sur les TEXT.



Type des colonnes (en MySQL)



- Enumération
- `ENUM("un", "deux", "trois")`
- Valeurs possibles : "", "un", "deux", "trois"
- Au plus 65535 éléments

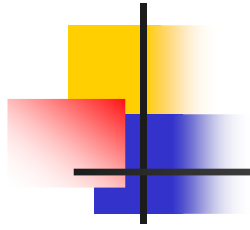


- Ensemble
- `SET("un", "deux")`
- Valeurs possibles : "", "un", "deux", "un,deux"
- Au plus 64 éléments



Contraintes

- Contraintes sur une colonne (un attribut)
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - REFERENCES nom_table [(nom-col)] [action]
 - CHECK (condition)
- Contraintes sur une table
 - UNIQUE (nom-col)*
 - PRIMARY KEY (nom-col)*
 - FOREIGN KEY (nom-col)* REFERENCES nom_table [(nom-col)*] [action]
 - CHECK (condition)



Contraintes NOT NULL / UNIQUE



- Après un nom de colonne (d'attribut)
- CREATE TABLE Pays (nom VARCHAR(20) NOT NULL, ...



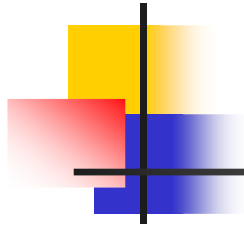
- Clé secondaire, pas deux lignes avec la même valeur, éventuellement NULL

Exemple 1

```
CREATE TABLE Etudiant (  
    num_etudiant INT UNIQUE,  
    ... );
```

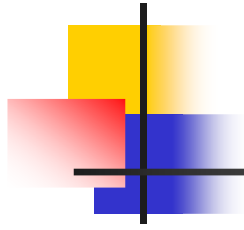
Exemple 2

```
CREATE TABLE Etudiant (  
    num_etudiant INT CONSTRAINT c_num_etu UNIQUE,  
    ... );
```

Contraintes PRIMARY KEY

- **PRIMARY KEY** : IDENTIFIANT de la relation
 - PRIMARY KEY ↔
 - Exemple 1
 - CREATE TABLE Departement
(numero_departement
 - Exemple 2
 - CREATE TABLE Employé
(nom VARCHAR(30),
prénom VARCHAR(30),
adresse VARCHAR(60), ...,
CONSTRAINT *c_cle_employe*
 - **AUTO_INCREMENT**
- **UNIQUE et PRIMARY KEY sont incompatibles**



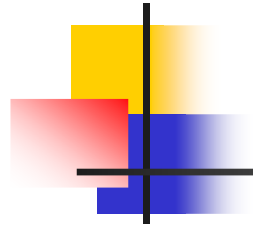
Contraintes d'intégrité référentielle

■ Foreign key :

- référence soit une soit une .

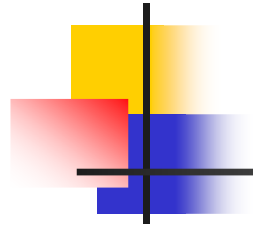
- Exemples:

- CREATE TABLE Dept
(numero_departement INT(4) PRIMARY KEY,
numero_manager INT(4) **REFERENCES** EMPLOYE(numero_employe)
, ...)
- CREATE TABLE Departement
(numero_departement INT(4) PRIMARY KEY,
numero_manager INT(4),
...,
CONSTRAINT **fk_mgr** **FOREIGN KEY** (numero_manager)
REFERENCES EMPLOYE(numero_employe),
...) ;



Actions déclenchées

- **REFERENCES** nom_table [(nom-colonne)] [**action**]
 - Que se passe-t-il quand on détruit/met à jour une clé primaire ou un attribut de type *unique* qui est référencé par un attribut (foreign key) d'une autre table?
 - Exemple :
 - CREATE TABLE Departement
(numero_departement INT(4) PRIMARY KEY,
numero_manager INT(4) REFERENCES EMPLOYE(**numero_employe**),
...) ;
 - Si on a l'enregistrement (numero_departement=1, numero_manager=21, ...) dans la table Departement, que se passe-t-il si on détruit ou met à jour l'employé d'identifiant 21 dans la table Employé?
 - (Voir schéma au tableau)



Actions déclenchées

- Deux circonstances



- Trois possibilités d'actions



- : valeur par défaut si elle existe, sinon NULL



- : on répercute la mise à jour

- Exemple :

- CREATE TABLE Departement

- (numero_departement INT(4) PRIMARY KEY,

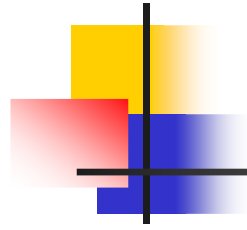
- numero_manager INT(4) **REFERENCES** EMPLOYE(numero_employe)

- ON DELETE SET NULL**

- ON UPDATE CASCADE**

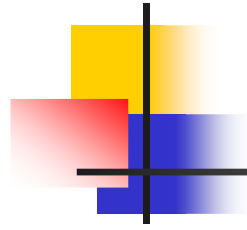
- ...) ;

(Voir le schéma au tableau)



Contrainte CHECK

- Condition que chaque ligne de la table doit vérifier
- En tant que contrainte de colonne : porte uniquement sur cette colonne
- En tant que contrainte de table : peut porter sur plusieurs colonnes.
- Exemples de contraintes sur des colonnes
 - **CREATE TABLE Divisions**
(
 num_div INT **CHECK** (num_div BETWEEN 10 AND 99),
 nom_div VARCHAR(9) **CHECK** (nom_div = UPPER(div_name)),
 bureau VARCHAR(10) **CHECK** (bureau IN ('Lyon', 'Paris', 'Lille'))
);



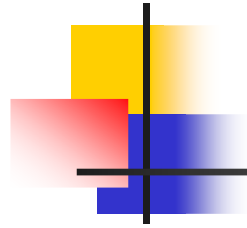
Contrainte CHECK

- Exemple de contraintes faisant intervenir plusieurs colonnes

```
CREATE TABLE Employe (  
    numero_employe INT(4) PRIMARY KEY,  
    nom_employe VARCHAR(10),  
    nom_job VARCHAR(9),  
    numero_manager INT(4),  
    salaire DECIMAL(7,2),  
    commission DECIMAL(7,2),  
    numero_departement SMALLINT(2),  
    CONSTRAINT check_salaire_et_commission
```

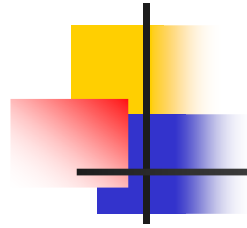
CHECK (salaire + commission <= 5000));

- Si on cherche à entrer des valeurs pour les attributs salaire et commission ne vérifiant pas la contrainte, **elles sont rejetées.**



Statut des contraintes

- Comment et quand vérifier une contrainte?
 - (la contrainte est activée immédiatement ou pas)
 - (on active ou désactive la contrainte)
 - (on valide ou pas les données déjà saisies)
- Peut être spécifié après chaque contrainte
- ENABLE / DISABLE
 - Activation/désactivation d'une contrainte dont on a donné le nom dans un CREATE TABLE (ou ALTER TABLE, cf plus loin)
 - Si la contrainte n'est pas vérifiée sur certaines valeurs, alors on ne peut pas l'activer
 - On ne peut pas activer une contrainte de clé étrangère qui référence un attribut auquel est associée une contrainte non active



Statut des contraintes

■ ENABLE

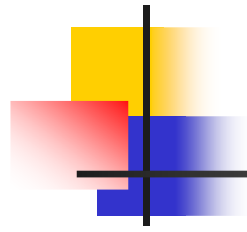
- VALIDATE : la contrainte est activée et contrôle que les données de la table vérifient la contrainte (c'est l'option par défaut de ENABLE)
- NOVALIDATE : la contrainte est activée, seules les nouvelles données entrées dans la base de données devront vérifier la contrainte.

■ DISABLE

- VALIDATE : on cherche à désactiver la contrainte, si les données ne sont pas valides, erreur.

Après DISABLE, on ne peut plus entrer, modifier ou supprimer des données de la table.

- NOVALIDATE : on peut faire n'importe quelle opération y compris entrer des données non conformes à la contrainte



DROP TABLE

- **DROP TABLE :**

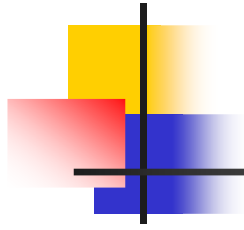


- **DROP TABLE** nom_table [**CASCADE CONSTRAINTS**];

- **CASCADE CONSTRAINTS**

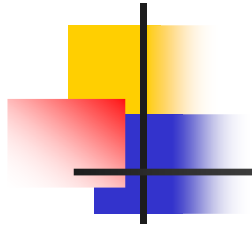
- Supprime toutes les contraintes référençant une clé primaire (primary key) ou une clé unique (UNIQUE) de cette table
 - Si on cherche à détruire une table dont certains attributs sont référencés sans spécifier CASCADE CONSTRAINT, on a un message d'erreur.

(voir exemple au tableau)



ALTER TABLE

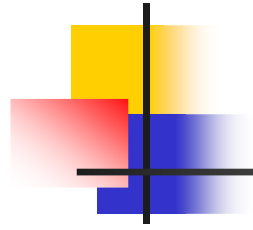
- Modifier la définition d'une table:
 - Changer le nom de la table
mot clé :
 - Ajouter une colonne ou une contrainte
mot clé :
 - Modifier une colonne ou une contrainte
mot clé :
 - Supprimer une colonne ou une contrainte
mot clé :
 - renommer une colonne ou une contrainte
mot clé :



ALTER TABLE

Syntaxe :

```
ALTER TABLE nom-table  
{ RENAME TO nouveau-nom-table  
  | ADD (( nom-col type-col [DEFAULT valeur] [contrainte-col]))*  
  | MODIFY (nom-col [type-col] [DEFAULT valeur] [contrainte-col])*  
  | DROP COLUMN nom-col [CASCADE CONSTRAINTS]  
  | RENAME COLUMN old-name TO new-name  
};
```

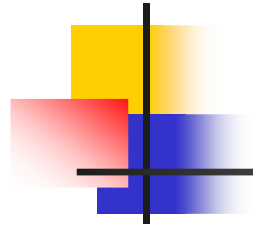


Renommer une table

- ... **RENAME TO** nouveau-nom-table

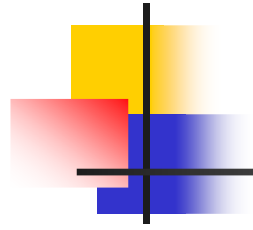
- Exemple :

ALTER TABLE country **RENAME TO** pays



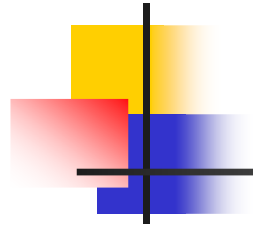
Ajouter une colonne

- ... **ADD** ((nom-col type-col
[DEFAULT valeur]
[contrainte-col])*)
- Exemple :
ALTER TABLE pays
 ADD (
 taxe DECIMAL(2,2) DEFAULT 19.6 CHECK (taxe < 25.0),
 visa_necessaire VARCHAR(3)
);



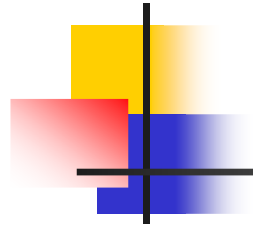
Modifier une colonne

- ...**MODIFY** ((nom-col [type-col]
[DEFAULT valeur]
[contrainte-col])*)
- Exemple :
ALTER TABLE pays
MODIFY (taxe DECIMAL(3,2));
- **Important :**



Supprimer une colonne

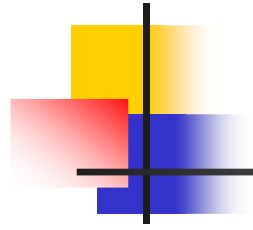
- ...DROP COLUMN nom-col
[CASCADE CONSTRAINTS]
- Exemple :



Renommer une colonne

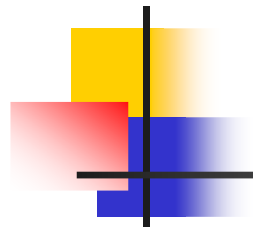
- ... `RENAME COLUMN` ancien_nom
 TO nouveau_nom

- Exemple :



ALTER TABLE sur les contraintes

```
ALTER TABLE nom_table
{ ADD contrainte_table
  | DROP { PRIMARY KEY
            | UNIQUE ( nom_colonne)*
            | CONSTRAINT nom_contrainte }
  [ CASCADE CONSTRAINTS ]
  | RENAME CONSTRAINT ancien TO nouveau
  | MODIFY [CONSTRAINT] nom_contrainte
                statut_contrainte
};
```



Ajouter une contrainte

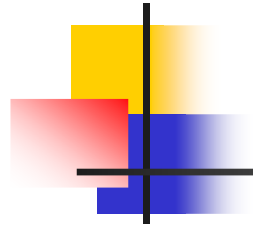
- ... **ADD** contrainte-table

Exemple 1 : on veut ajouter une contrainte à la table Employe

```
ALTER TABLE Employe  
  ADD CONSTRAINT seuil_commission  
  CHECK ((commission / salaire) <= 1);
```

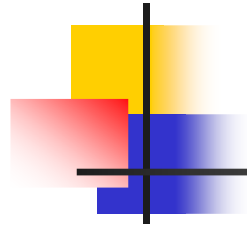
Exemple 2 : on veut définir une clé primaire dans la table Pays
(en supposant qu'elle n'ait pas été définie auparavant)

```
ALTER TABLE Pays  
  ADD CONSTRAINT cle_pays PRIMARY KEY (nom);
```



Supprimer une contrainte

- DROP {
 PRIMARY KEY
 | UNIQUE (nom-colonne)*
 | CONSTRAINT nom-contrainte }
 [CASCADE CONSTRAINTS]
- Exemple :



Modifier le statut d'un contrainte

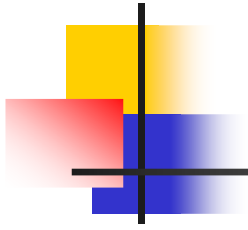
- ... **MODIFY** [**CONSTRAINT**] nom-contrainte
statut_contrainte

Exemple :

```
ALTER TABLE Pays  
  MODIFY CONSTRAINT cle_pays DISABLE
```

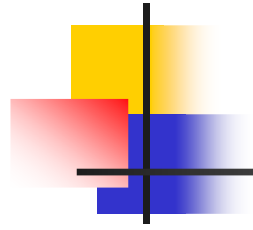
Équivalent à:

```
ALTER TABLE pays DISABLE PRIMARY KEY
```



SQL

**Un langage de
manipulation de données**



Manipulation des données

- **INSERT INTO :**

- **UPDATE :**

- **DELETE FROM :**

INSERT INTO

| JO | | |
|-------|----------|--------|
| Année | Lieu | Pays |
| 1896 | Athènes | Grèce |
| 1900 | Paris | France |
| 1904 | St Louis | USA |
| 1908 | Londres | UK |

INSERT INTO JO
VALUES (1996, 'Atlanta', 'U.S.A')

INSERT INTO JO
(Année, Lieu)
VALUES (1996, 'Atlanta')

| JO | | |
|-------|----------|--------|
| Année | Lieu | Pays |
| 1896 | Athènes | Grèce |
| 1900 | Paris | France |
| 1904 | St Louis | USA |
| 1908 | Londres | UK |
| 1996 | Atlanta | USA |

| JO | | |
|-------|----------|--------|
| Année | Lieu | Pays |
| 1896 | Athènes | Grèce |
| 1900 | Paris | France |
| 1904 | St Louis | USA |
| 1908 | Londres | UK |
| 1996 | Atlanta | NULL |



INSERT INTO

- Syntaxe :

INSERT INTO

{nom_table | nom_vue}

[(nom_col (, nom_col)*)]

{ VALUES (valeur (, valeur)*) | sous-requête };



UPDATE

- Syntaxe :

- **UPDATE** {nom_table | nom_vue}
SET { (nom_col)* = (sous-requête)
 | nom_col = { valeur | (sous-requête) } }*
WHERE condition;

- Exemples :

- **UPDATE** Pays
SET langue = 'English'
WHERE nom = 'Ireland'
- **UPDATE** Infos
SET esp_vie = esp_vie+2 , poids = poids * 2
WHERE continent = 'Amérique' AND esp_vie < 80



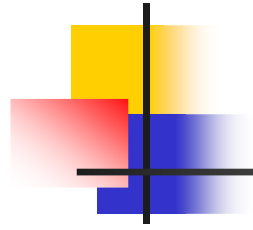
DELETE FROM

- Syntax :
 - **DELETE FROM** {nom_table | nom_vue}
WHERE condition;
- Exemple :
 - **DELETE FROM** Pays
WHERE population > 300

| Pays | | | | |
|----------|------------|------------|---------|-----------|
| Nom | Capitale | Population | Surface | Continent |
| Irlande | Dublin | 5 | 70 | Europe |
| Autriche | Vienne | 9 | 83 | Europe |
| UK | Londres | 53 | 244 | Europe |
| Suisse | Berne | 8 | 41 | Europe |
| USA | Washington | 314 | 441 | Amérique |

SQL

**Un langage de
requêtes**



Structure générale d'une requête

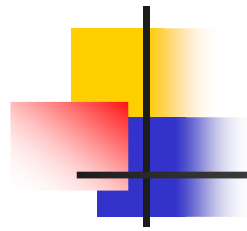
- Structure d'une requête formée de trois clauses:

SELECT

FROM

WHERE

- **SELECT** définit le format du résultat cherché
- **FROM** définit à partir de quelles tables le résultat est calculé
- **WHERE** définit les prédicats de sélection du résultat



Exemple de requête

SELECT * FROM pays

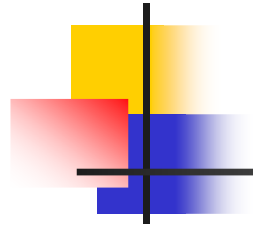
Afficher toutes les valeurs de tous les attributs de tous les tuples dans la table “pays”

| Pays | | | |
|-------------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| Royaume-Uni | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |

SQL

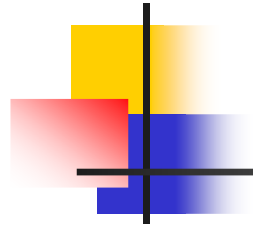
et

Algèbre relationnelle



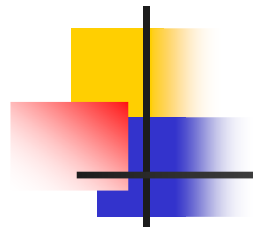
SQL / Algèbre : Identité

- En algèbre : $\text{Id}(\mathbf{R})$
- En SQL :



SQL / Algèbre : Sélection

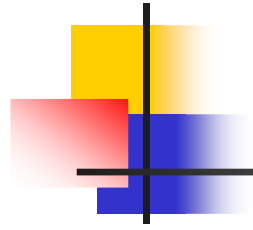
- En algèbre : σ [condition] R
- En SQL :



SQL / Algèbre : Sélection

- **SELECT** * **FROM** pays **WHERE** population < 20

| Pays | | | |
|-------------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| Royaume-Uni | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |



SQL / Algèbre : Projection

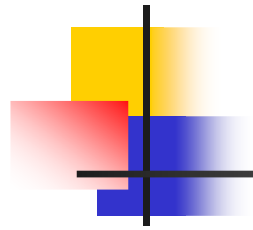
- En algèbre : $\pi [A1, A2, ..., An] R$
- En SQL :



SQL / Algèbre : Projection

SELECT nom, capitale **FROM** pays ;

| Pays | | | |
|-------------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| Royaume-Uni | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |



SQL / Algèbre : Sélection + Projection

$\pi[\text{nom, capitale, population}](\sigma[\text{population} < 20]\text{pays})$

SELECT nom, capitale, population
FROM pays
WHERE population < 20

| Pays | | | |
|-------------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| Royaume-Uni | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |

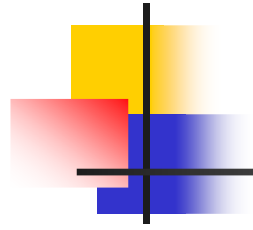


SQL / Algèbre : Renommage

- En algèbre : $\alpha (A_1:B_1, \dots, A_n:B_n) R$
- En SQL : Impossible de renommer des attributs. Il faut faire des “copies logiques” des relations.

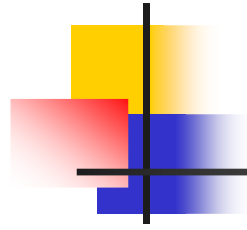
SELECT
FROM
WHERE

- R2 peut être vue comme une copie logique de R



SQL / Algèbre : Produit cartésien

- En algèbre : $R \times S$
- En SQL :



SQL / Algèbre : Jointure

- En algèbre : $R \bowtie S$

- En SQL :

SELECT *

FROM R, S

WHERE R.A₁ = S.A'₁

AND R.A₂ = S.A'₂

...

AND R.A_n = S.A'_n

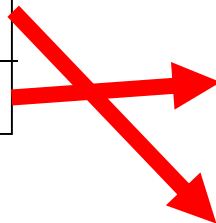
- A₁, ..., A_n étant tous les attributs R et A'₁, ..., A'_n étant tous les attributs S sur lesquels on souhaite effectuer la jointure

SQL / Algèbre : Jointure

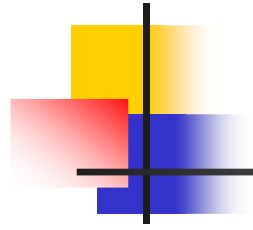
```
SELECT année, lieu, pays, capitale  
FROM JO, Pays  
WHERE JO.pays = Pays.nom;
```

| JO | | |
|-------|----------|--------|
| Année | Lieu | Pays |
| 1896 | Athènes | Grèce |
| 1900 | Paris | France |
| 1904 | St Louis | USA |
| 1908 | Londres | UK |

| Pays | | | |
|----------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 10 | 83 |
| UK | Londres | 50 | 244 |
| Suisse | Berne | 7 | 41 |
| USA | Washington | 350 | 441 |



| Année | Lieu | Pays | Capitale |
|-------|----------|------|------------|
| 1908 | Londres | UK | Londres |
| 1904 | St Louis | USA | Washington |

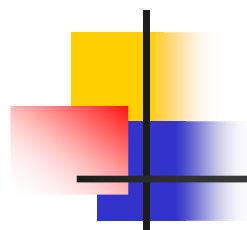


Jointure sur la même table

- Comment comparer les populations des pays?
- Exemple :

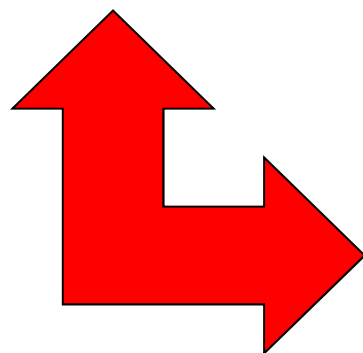
```
SELECT P1.nom, P1.population, P2.nom, P2.population  
FROM Pays AS P1, Pays AS P2  
WHERE P1.population > P2.population ;
```

- => Toutes les paires de pays telles que le premier pays a une population plus grande que le deuxième pays
- NB: La table **Pays** est renommée en **P1** et **P2** (alias)

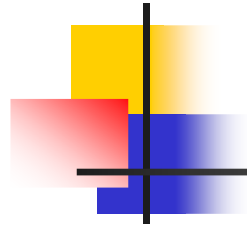


Jointure sur la même table

| pays | | | |
|----------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| UK | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |



| pays | | | |
|----------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| UK | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |

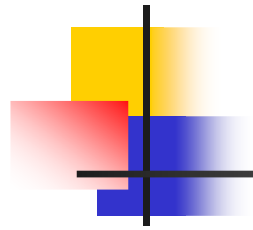


Jointure sur la même table

| P1 | | | |
|----------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| UK | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |

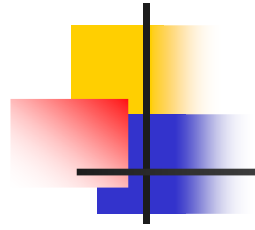
| P2 | | | |
|----------|------------|------------|---------|
| Nom | Capitale | Population | Surface |
| Irlande | Dublin | 5 | 70 |
| Autriche | Vienne | 9 | 83 |
| UK | Londres | 53 | 244 |
| Suisse | Berne | 8 | 41 |
| USA | Washington | 314 | 441 |

```
SELECT
  P1.nom,
  P1.population,
  P2.nom,
  P2.population
FROM
  Pays AS P1,
  Pays AS P2
WHERE
  P1.population >
  P2.population ;
```



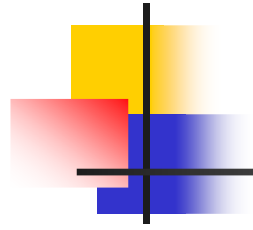
Jointure sur la même table

| P1.Nom | P1.Population | P2.Nom | P2.Population |
|-----------------|----------------------|-----------------|----------------------|
| Autriche | 9 | Irlande | 5 |
| Autriche | 9 | Suisse | 8 |
| UK | 53 | Irlande | 5 |
| UK | 53 | Autriche | 9 |
| UK | 53 | Suisse | 8 |
| | | | |



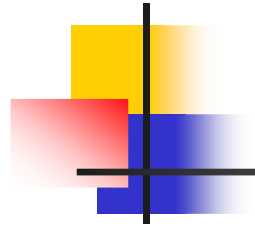
SQL / Algèbre : Union

- En algèbre :
- En SQL :
- Exemple :
- Les tuples en double sont éliminés du résultat



SQL / Algèbre : Intersection

- En algèbre :
- En SQL :
- Exemple :
- Les tuples en double sont éliminés du résultat



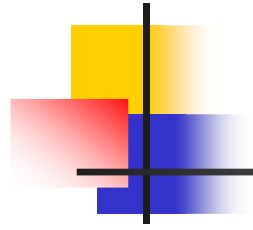
SQL / Algèbre : Différence

- En algèbre :
- En SQL :
- Exemple :
- Les tuples en double sont éliminés du résultat



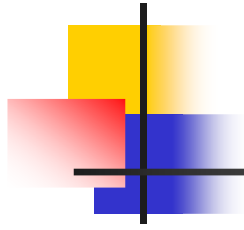
Remarques

- Le résultat d'une requête peut contenir plusieurs occurrences d'un tuple,
 - pour avoir une seule occurrence de chaque n-uplet dans une relation : **DISTINCT**
 - Exemple : select **DISTINCT** nom FROM Personne
- Le résultat d'une requête peut être trié
- Il existe une valeur spéciale dite indéfinie (**NULL**) utilisée pour remplir un champ dont on ne connaît pas la valeur.



Attention : NULL \neq 0

- **NULL** =
- SELECT **nom** FROM **Pays** WHERE **montagne** **IS** NULL
-> Pays Bas
- SELECT **nom** FROM **Pays** WHERE **montagne** **IS NOT** NULL
-> Autriche, Suisse
- NULL = Pas de valeur



Opérations sur NULL

- NULL dans les conditions:
 - (population > 0) ?
 - si population est NULL, le résultat est "unknown" donc "false"
 - (population = NULL) ?
 - le test retourne toujours "false":
 - La syntaxe correcte est: (population **IS** NULL)

- NULL dans une expression arithmétique:
 - (population + NULL) retourne NULL



Opérations sur NULL

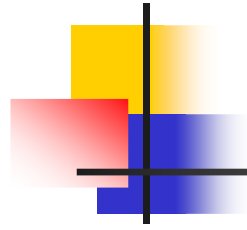
- NULL dans une fonction SQL
 - Concat (nom, '-', prenom) retourne NULL si prénom est NULL
 - Concat (nom, '-', NVL(prenom, ' ')) retourne 'dupont-'
 - NVL(attribut,valeur_de_replacement): retourne la valeur de remplacement lorsque que la valeur de l'attribut est NULL.

- Fonctions d'agrégation ignorent les NULL:
 - Moyenne(1000, NULL, NULL, NULL, 2000) = (1000+2000)/2



Remarques

- En SQL, le produit cartésien est possible sans renommer les attributs communs.
 - Exemple : schéma($R \times S$) = A (de R), B (de R), B (de S), C (de S).
- En SQL, si plusieurs attributs ont le même nom, pour résoudre l'ambiguïté, on spécifie la relation auquel l'attribut appartient.
 - Exemple : `SELECT A, R.B, C FROM R, S`



Opérateurs de comparaison

- - WHERE surface = 200

- - WHERE capitale <> 'Paris'

- - WHERE population > 8

- - WHERE population >= 8

- - WHERE surface < 83

- - WHERE surface <= 83



Opérateurs logiques

■ AND

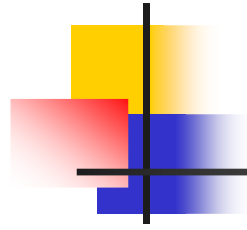
- WHERE population<10 AND surface<500

■ OR

- WHERE population<10 OR surface<500

■ Négation de la condition : NOT

- SELECT P1.nom, P2.nom, P1.capitale
FROM Pays P1, Pays P2
WHERE P1.capitale = P2.capitale
AND NOT P1.nom = P2.nom ;



Expressions logiques

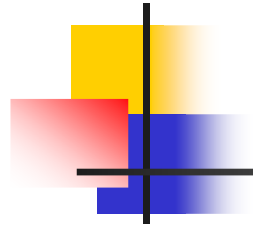
On peut bien sûr combiner les opérateurs :

WHERE

(ensoleillement > 80 **AND** pluviosité < 200)
OR température > 30

WHERE

ensoleillement > 80
AND (pluviosité < 200 **OR** température > 30)



Appartenance à un ensemble : IN

WHERE monnaie = 'Pound'

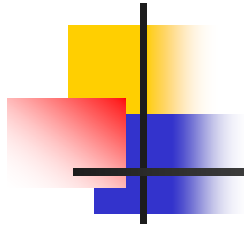
OR monnaie = 'Schilling'

OR monnaie = 'Euro'

Équivalent à:

WHERE monnaie IN ('Pound', 'Schilling', 'Euro')

NOT IN: non appartenance à un ensemble



Comparaison à un ensemble : ALL

```
SELECT * FROM Employe  
WHERE salaire > ALL ( SELECT salaire FROM Employe  
                       WHERE statut=« administratif »);
```

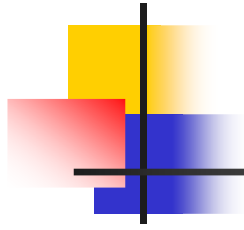
| Nom | Prénom | Salaire | Statut |
|---------|--------|---------|---------------|
| Bonnot | Jean | 2100 | Ingénieur |
| Smith | John | 1800 | Administratif |
| Afeu | Pierre | 1500 | Administratif |
| Lafleur | Marie | 1700 | Technicien |
| Rose | Sylvie | 2500 | DRH |

Équivalent à:

```
SELECT * FROM Employe  
WHERE salaire > 1800  
AND salaire > 1500 ;
```



| Nom | Prénom | Salaire | Statut |
|--------|--------|---------|-----------|
| Bonnot | Jean | 2100 | Ingénieur |
| Rose | Sylvie | 2500 | DRH |



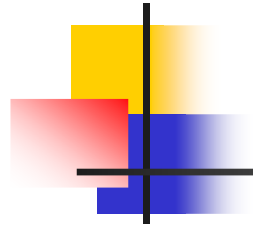
Valeur dans un intervalle : BETWEEN

WHERE population \geq 50 AND population \leq 60

Équivalent à:

WHERE population BETWEEN 50 AND 60

NOT BETWEEN



Conditions partielles (joker)

- **%** : un ou plusieurs caractères

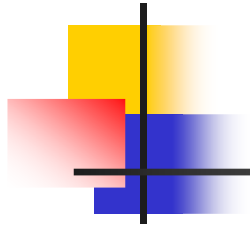
- WHERE pays LIKE '%lande'
- -> Irlande, Islande, Finlande, Hollande

- WHERE pays LIKE '%ran%'
- -> Iran, France

- **_** : exactement un caractère

- WHERE pays LIKE 'I_lande'
- -> Irlande, Islande

- **NOT LIKE**



Valeurs calculées

- SELECT **nom**, **population**, **surface**, **natalité**
FROM **Pays**
WHERE (**population** * 1000 / **surface**) < 50
AND (**population** * **natalité** / **surface**) > 0
- SELECT **nom**, (**population** * 1000 / **surface**)
FROM **Pays**

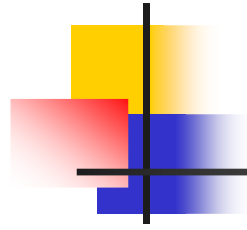
Requêtes

avec blocs emboîtés



BD exemple

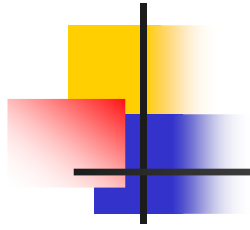
- Produit(**np**, **nomp**, **couleur**, **poids**, **prix**) *les produits*
- Usine(**nu**, **nomu**, **ville**, **pays**) *les usines*
- Fournisseur(**nf**, **nomf**, **type**, **ville**, **pays**) *les fournisseurs*
- Livraison(**np**, **nu**, **nf**, **quantité**) *les livraisons*
 - **np** référence *Produit.np*
 - **nu** référence *Usine.nu*
 - **nf** référence *Fournisseur.nf*



Jointure par blocs emboîtés

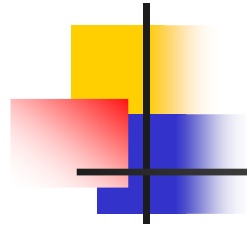
Nom et couleur des produits livrés par le fournisseur 1

- Solution 1 : la jointure déclarative
- Solution 2 : la jointure procédurale (emboîtement)



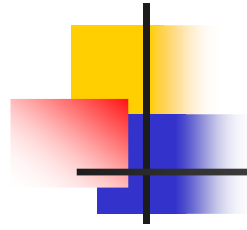
Jointure par blocs emboîtés

- SELECT **nomp**, **couleur** FROM **Produit**
WHERE **np** **IN**
 (SELECT **np** FROM **Livraison**
 WHERE **nf** = 1) ;
- **IN** compare chaque valeur de **np** avec l'ensemble (ou multi-ensemble) de valeurs retournés par la sous-requête
- **IN** peut aussi comparer un tuple de valeurs:
SELECT **nu** FROM **Usine**
WHERE (**ville**, **pays**)
 IN (SELECT **ville**, **pays** FROM **Fournisseur**) 80



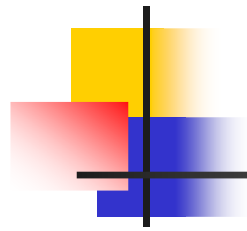
Composition de conditions

Nom des fournisseurs qui approvisionnent une usine de Londres ou de Paris en un produit rouge



Composition de conditions

*Nom des fournisseurs qui approvisionnent une usine de
Londres ou de Paris en un produit rouge (**en imbriqué**)*

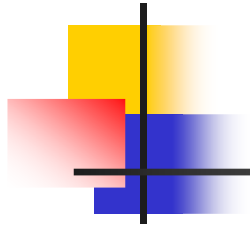


Quantificateur ALL

- *Numéros des fournisseurs qui ne fournissent que des produits rouges*

```
SELECT nf FROM Fournisseur  
WHERE 'rouge' = ALL  
  (SELECT couleur FROM Produit  
   WHERE np IN  
     (SELECT np FROM Livraison  
      WHERE Livraison.nf = Fournisseur.nf) ) ;
```

- *La requête imbriquée est ré-évaluée pour chaque tuple de la requête (ici pour chaque *nf*)*
- **ALL**: tous les éléments de l'ensemble doivent vérifier la condition



Condition sur des ensemble : **EXISTS**

- Test si l'ensemble n'est pas vide ($E \neq \emptyset$)
- Exemple : *Noms des fournisseurs qui fournissent au moins un produit rouge*

SELECT nomf

FROM Fournisseur

WHERE **EXISTS**

(SELECT *

FROM Livraison, Produit

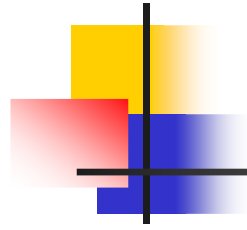
WHERE Livraison.nf = Fournisseur.nf

AND Livraison.np = Produit.np

AND Produit.couleur = 'rouge');

ce fournisseur

*Le produit fourni
est rouge*



Blocs emboîtés - récapitulatif

SELECT ...

FROM ...

WHERE ...

attr **IN** requête

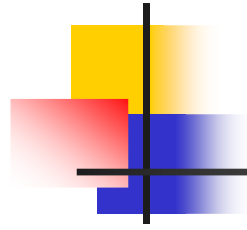
attr **NOT IN** requête

attr opérateur **ALL** requête

EXISTS requête

NOT EXISTS requête

Traitement des résultats



Fonctions sur les colonnes

- Attributs calculés

- Exemple : `SELECT nom, population*1000/surface FROM Pays`

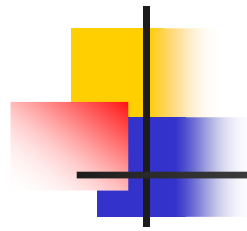
- Opérateurs sur attributs numériques

- -

- Opérateurs sur tous types d'attributs

- -
 -

Opérateurs
d'agrégation



Opérateurs d'agrégation

| Pays | | | | |
|----------|------------|------------|---------|-----------|
| Nom | Capitale | Population | Surface | Continent |
| Irlande | Dublin | 5 | 70 | Europe |
| Autriche | Vienne | 9 | 83 | Europe |
| UK | Londres | 53 | 244 | Europe |
| Suisse | Berne | 8 | 41 | Europe |
| USA | Washington | 314 | 441 | Amérique |

```
SELECT MIN(population), MAX(population), AVG(population),  
       SUM(surface), COUNT(*)  
FROM Pays WHERE continent = 'Europe'
```

Donne le résultat :

| MIN (population) | MAX (population) | AVG (population) | SUM (surface) | COUNT (*) |
|-------------------------|-------------------------|-------------------------|----------------------|------------------|
| | | | | |



DISTINCT

| Pays | | | | |
|----------|------------|------------|---------|-----------|
| Nom | Capitale | Population | Surface | Continent |
| Irlande | Dublin | 5 | 70 | Europe |
| Autriche | Vienne | 9 | 83 | Europe |
| UK | Londres | 53 | 244 | Europe |
| Suisse | Berne | 8 | 41 | Europe |
| USA | Washington | 314 | 441 | Amérique |

Suppression des doubles

```
SELECT DISTINCT continent  
FROM Pays
```

Donne le résultat :

| Continent |
|-----------|
| |
| |



ORDER BY

Tri des
tuples
du résultat

| Pays | | | | |
|----------|------------|------------|---------|-----------|
| Nom | Capitale | Population | Surface | Continent |
| Irlande | Dublin | 5 | 70 | Europe |
| Autriche | Vienne | 9 | 83 | Europe |
| UK | Londres | 53 | 244 | Europe |
| Suisse | Berne | 8 | 41 | Europe |
| USA | Washington | 314 | 441 | Amérique |

```
SELECT continent, nom, population
FROM Pays
WHERE surface > 60
ORDER BY continent, nom ASC
```

| Continent | Nom | Population |
|-----------|----------|------------|
| Amérique | USA | 350 |
| Europe | Autriche | 10 |
| Europe | Irlande | 5 |
| Europe | Suisse | 7 |
| Europe | UK | 50 |

2 possibilités : **ASC** / **DESC**



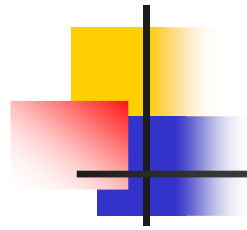
GROUP BY

Partition de l'ensemble des tuples en groupes homogènes

| Pays | | | | |
|----------|------------|------------|---------|-----------|
| Nom | Capitale | Population | Surface | Continent |
| Irlande | Dublin | 5 | 70 | Europe |
| Autriche | Vienne | 9 | 83 | Europe |
| UK | Londres | 53 | 244 | Europe |
| Suisse | Berne | 8 | 41 | Europe |
| USA | Washington | 314 | 441 | Amérique |

```
SELECT continent, MIN(population), MAX(population), AVG(population),
SUM(surface), COUNT(*)
FROM Pays GROUP BY continent ;
```

| Continent | MIN(population) | MAX(population) | AVG(population) | SUM(surface) | COUNT(*) |
|-----------|-----------------|-----------------|-----------------|--------------|----------|
| Europe | 5 | 50 | 18.75 | 438 | 4 |
| Amérique | 350 | 350 | 314 | 441 | 1 |



Renommage des attributs : AS

```
SELECT MIN(population) AS min_pop,  
       MAX(population) AS max_pop,  
       AVG(population) AS avg_pop,  
       SUM(surface) AS sum_surface,  
       COUNT(*) AS count  
FROM Pays  
WHERE continent = 'Europe' ;
```

| min_pop | max_pop | avg_pop | sum_surface | count |
|---------|---------|---------|-------------|-------|
| 5 | 50 | 18 | 438 | 4 |