BD et SGBD

Une base de données

est un ensemble d'informations, structurées de façon à pouvoir être consultées (extraites), modifiées, ajoutées, supprimées.

Un Système de Gestion d'une Bases de Données (SGBD)

est un logiciel qui gère un ensemble de bases de données.

- Il permet
  - la mémorisation et la **représentation interne** des données.
  - la manipulation et la consultation de ces données par l'intermédiaire d'une **interface utilisateur**.
- Le plus souvent l'utilisateur manipule les données sans connaître l'implémentation interne, en tous cas pas dans ses détails.
- Quelques exemples de SGBD : Oracle database, PostgresQL, IBM DB2, Microsoft SQL server, MySQL

Bases relationnelles

Les modèles de données

- Il existe plusieurs principes pour structurer les données, **les modèles de données** : relationnel, objet, hiérarchique ....
- un SGBD est spécialisé dans un ou parfois deux modèles de données

Le modèle relationnel

Les données sont structurées en un **ensemble**

- de **relations** → selon une approche mathématique
- de **tables** → selon une approche plus « graphique »

Ce sont des quasi synomymes, mais dans des formalismes différents !

Exemple de relation / table

relation

- **Schéma de la relation** :
  $(Nom : Chaine,\ Prenom : Chaine,\ Groupe : Entier)$
- **Relation** :
  $\{(Dupont, Alfred, 2), (Smith, John, 1), (Durand, Alfred, 1)\}$

table

Chaîne	*Chaîne*	*Entier*
Nom	Prénom	Groupe
Dupont	Alfred	2
Smith	John	1
Durand	Alfred	1

Les relations

- **Schéma de la relation** : N-uple :
  $(Attribut_1 : Domaine_1,\ \ldots,\ Attribut_N : Domaine_N)$

$Domaine_1$	...	$Domaine_N$
$Attribut_1$	...	$Attribut_N$

  - **Attribut** ≡ Nom de colonne
  - **Domaine** de définition ≡ Type
- **Relation** : ensemble de N-uples (les « tuples »)
  $\{(a_1, ..., a_N), (b_1, ..., b_N), \ldots\}$

$Domaine_1$	...	$Domaine_N$
$Attribut_1$	...	$Attribut_N$
a_1	...	a_N
b_1	...	b_N
...	...	...

Opérations relationnelles (1/4)

$T \cup T'$: union de deux relations T et T'

- Il faut qu'elles soient définies sur le même schéma
- $T \cup T'$

$T - T'$: différence de deux relations T et T'

- Il faut qu'elles soient définies sur le même schéma
- $T - T' = \{$ éléments de T qui ne sont pas dans T' $\}$

Exemple :
$T = \{(Dupont, Alfred, 2), (Smith, John, 1), (Durand, Alfred, 1)\}$
$T' = \{(Smith, John, 1), (Smith, Arthur, 3)\}$
$T - T' = \{(Dupont, Alfred, 2), (Durand, Alfred, 1)\}$

Opérations relationnelles (2/4)

$T \times T'$: produit cartésien de deux relations T et T'

- T de schéma s et T' de schéma s'.
- $T \times T'$ défini sur le schéma $s.s'$ (concaténation).
- $T \times T'$ contient les tuples $(x_1, \ldots x_n, y_1, \ldots y_{n'})$
  où $(x_1, \ldots x_n) \in R$ et $(y_1, \ldots y_{n'}) \in R'$

$M =$

Nom	IP
lxh1	172.16.12.181
lxh2	172.16.12.182

$T \times M =$

Nom	Prénom	Groupe	Nom	IP
Dupont	Alfred	2	lxh1	172.16.12.181
Dupont	Alfred	2	lxh2	172.16.12.182
Smith	John	1	lxh1	172.16.12.181
Smith	John	1	lxh2	172.16.12.182
Durand	Alfred	1	lxh1	172.16.12.181
Durand	Alfred	1	lxh2	172.16.12.182

Opérations relationnelles (3/4)

un sous-schéma de $s = (X_1, \ldots X_n)$

$s' = (X_{i_1}, \ldots X_{i_{n'}})$ où $i_k \in [1, n]$
C'est à dire : ne contient que certaines composantes de s.
Ex : (Nom : $Chaine$, $Groupe$: $Entier$)

$Proj_{s'}(T)$: projection d'une relation T sur un sous-schéma s'

$\{(x_{i_1}, \ldots x_{i_{n'}})$ où $X_{i_k} \in s'$ et $(x_1, \ldots x_n) \in T\}$

$Proj_{(Nom, Groupe)}(R) =$

Nom	Groupe
Dupont	2
Smith	1
Durand	1

C'est donc un choix de « colonnes »

Opérations relationnelles (4/4)

Une qualification définie sur un schéma s

est une **expression booléenne** composée de constantes, de noms d'attributs, d'opérateurs de comparaison,d'opérateurs logiques
Exemple :

- Groupe<=1 et Prénom=Alfred
- Groupe=1 ou Groupe>= 4

Une restriction ou sélection de T selon une qualification Q

est l'ensemble des éléments de T satisfaisant Q.

Exemple

Restriction de T selon «Groupe<=1 et Prénom=Alfred»

Nom	Prénom	Groupe
Durand	Alfred	1

C'est donc un choix de « lignes »

Algèbre relationnelle

- Les 5 opérations que nous venons de voir définissent l'**algèbre relationnelle**.
- Elle forment des opérations de base permettant d'extraire de l'information et de définir de nouvelles relations à partir de relations existantes.
- Toutes les opérations relationnelles peuvent s'écrire à partir de celles-là.
- Cependant nous examinerons un peu plus tard d'autres opérations (comme la jointure).

Le langage SQL

Simple Query Langage (SQL)

- est un langage d'interrogation des SGBD relationnelles.
- Né dans les années 70-80, il a fait l'objet de plusieurs normes successives (86, 89, 92 [SQL2], 99 [SQL3]).
- Par ailleurs les différents SGBD peuvent implémenter tout ou partie ... ou des extensions de SQL.

Fonctionnalités de SQL :

- Définition, modification ou suppression de schémas de tables (DDL) : `create, alter, drop`.
- Manipulation de données (DML) : `select, update, insert, delete`.
- Administration et gestion des droits (DCL) : `grant, revoke`.

Select

Select : commande permettant de consulter (« extraire ») les données

- Sa syntaxe minimale est

  `select` *noms de colonnes* `from` *nom de table*

- Exemple : `select Nom,Groupe from R`
- Sous cette forme, c'est une **projection**. Les noms de colonnes forment le sous-schéma.
- Pour le sous-schéma, on peut utiliser la notation * qui désigne le schéma complet : l'expression `select * from R` désigne la projection « identité ».
- **Le résultat d'une requête « select » est une table**

Restriction(sélection) : `where`

La clause where

- permet de réaliser une **restriction** du résultat (choix de lignes)
- optionnelle
- est suivie d'une qualification.

```
select * from R where groupe<=1 and prenom='Alfred'
```

Nom	Prénom	Groupe
Durand	Alfred	1

Tris : `order by`

L'ordre des lignes obtenues est incertain

Sauf si l'on indique explicitement un ordre de classement par la clause **order by** . `select * from R order by groupe`

Smith	John	1
Durand	Alfred	1
Dupont	Alfred	2

`select * from R order by groupe desc`

Dupont	Alfred	2
Smith	John	1
Durand	Alfred	1

`select * from R order by groupe desc, nom asc`

Dupont	Alfred	2
Durand	Alfred	1
Smith	John	1

Du modèle à la pratique...

différences par rapport au modèle ensembliste strict

- Une table peut contenir des doublons. La clause **distinct** permet de les éliminer du résultat, si besoin :

`select prenom from R`

Alfred
Alfred
John

`select distinct prenom from R`

Alfred
John

- Les domaines sont des types. Quelques exemples :

VARCHAR(n)	Chaîne de *n* caractères maximum.
INTEGER	Entier signé, sur 32 bits
SMALLINT	Entier signé, sur 16 bits
FLOAT	Flottant
NUMERIC(n,d)	*n* chiffres **dont** *d* après la virgule
DATE	date `A-M-J`
TIME	temps `H:MN:S`

SQL : Les fonctions de regroupement

Exemple

Soit la table décompte :

date	numero	cout
2016-1-5	+33320434343	0.40
2016-1-5	+33320434343	0.05
2016-1-5	+33328778551	0.12
2016-2-29	+33320434343	0.60

`select sum(cout) from decompte`

réalise la somme de tous les attributs prix de la table decompte.

Le résultat comporte une seule ligne :

sum(cout)
1.17

SQL : Les fonctions de regroupement

Pour obtenir des totaux par numéro :

`select sum(cout) from decompte group by numero`

On obtient ici 2 lignes :

sum(cout)
1.05
0.12

On peut aussi attribuer un nom à la colonne avec as
il aurait également été utile de conserver le champ numéro.

```
select date,numero,sum(cout) as total
from decompte
group by date,numero
```

date	numero	total
2016-1-5	+33320434343	0.45
2016-1-5	+33328778551	0.12
2016-2-29	+33320434343	0.60

SQL : Les fonctions de regroupement

Il s'agit de fonctions **statistiques** qui s'appliquent sur des **groupes de lignes** de la table.

AVG	moyenne	numérique
SUM	somme	numérique
COUNT	dénombrement	quelconque
MAX	maximum	quelconque
MIN	minimum	quelconque

Une fonction est suivie du nom de l'attribut auquel on l'applique.
Par défaut, il existe un seul groupe de lignes qui les comprend toutes.
La clause **group by** permet de modifier les regroupements. Elle est suivie du nom de un ou plusieurs attributs. Les lignes sont dans le même groupe si elles ont les mêmes valeurs pour chacun des attributs cités.

Les fonctions de regroupement

la clause having

permet de faire une sélection portant sur le résultat d'une fonction de regroupement. La clause having est liée à une clause "group by".

```
select numero,sum(cout) from decompte
    group by numero having sum(cout)>1;
```

numero	sum(cout)
+33320434343	1.05

```
select numero,sum(cout) from decompte
    group by numero having count(*)=1;
```

numero	sum(cout)
+33328778551	0.12

Produit cartésien et qualification (1/2)

Concerts

salle	artiste
Biplan	Machin
Aéronef	Bidule
Biplan	Bidule

Artistes

nom	site
Bidule	bidule.free.fr
Chose	www.musique.zz/chose
Machin	www.machin.com
Truc	www.musique.zz/truc

On souhaite

Obtenir la liste de tous les concerts programmés, avec la salle, le nom de l'artiste et son site web.

Il faut faire intervenir les 2 tables

- Réaliser un produit cartésien des 2 tables
- Ne sélectionner que les lignes telles que Concerts.artiste = Artistes.nom
- Ne conserver que les colonnes qui nous intéressent

Produit cartésien et qualification (2/2)

Concerts × *Artistes*

salle	artiste	nom	site
Biplan	Machin	Bidule	bidule.free.fr
Biplan	Machin	Chose	www.musique.zz/chose
Biplan	Machin	Machin	www.machin.com
Biplan	Machin	Truc	www.musique.zz/truc
Aéronef	Bidule	Bidule	bidule.free.fr
Aéronef	Bidule	Chose	www.musique.zz/chose
... etc ...			

$Qualification_{artiste=nom}(Concerts \times Artistes)$

salle	artiste	nom	site
Biplan	Machin	Machin	www.machin.com
Aéronef	Bidule	Bidule	bidule.free.fr
Biplan	Bidule	Bidule	bidule.free.fr

Jointure

1ère syntaxe

```
select * from Concerts, Artistes
  where Concerts.artiste = Artistes.nom
```

2ème syntaxe

```
 select * from Concerts
  join Artistes on Concerts.artiste = Artistes.nom
```

avec choix des attributs

```
select Concerts.salle, Concerts.nom, Artistes.sites
  from Concerts, Artistes
  where Concerts.artiste = Artistes.nom
```

ou

```
select Concerts.salle, Concerts.nom, Artistes.sites
  from Concerts
  join Artistes on Concerts.artiste = Artistes.nom
```


Jointure

La combinaison d'un produit cartésien et d'une qualification est appelée jointure

C'est une opération essentielle dans les consultations de bases de données.

- Elle permet de croiser les informations de plusieurs tables
- On a rarement besoin de connaître l'ensemble du produit cartésien, mais seulement d'un sous-ensemble restreint de lignes.
- Les SGBD doivent, en principe, optimiser les requêtes de jointure pour ne pas construire tout le produit cartésien mais éliminer au fur et à mesure les entrées non sélectionnées

Les différents types de jointure.

Jointure interne : `inner join`

- Seules les lignes de chaque table qui répondent à la condition de jointure figurent dans le résultat
- C'est la jointure par défaut (le mot `inner` est optionel en SQL)

Jointures externes : `left join`, `right join`, `full join`

- **`left join`**
  - les lignes de la 1ère table (table de gauche) qui ne correspondent à aucune entrée de la 2ème table sont conservées.
  - avec la valeur NULL pour tous les attributs correspondant à la deuxième table
- **`right join`** : idem, symétriquement
- **`full join`** : idem, dans les 2 sens.

Exemple de jointures externe

```
select * from concerts
RIGHT JOIN artistes on artiste=nom
```

salle	artiste	nom	site
Aéronef	Bidule	Bidule	bidule.free.fr
Biplan	Bidule	Bidule	bidule.free.fr
NULL	*NULL*	Chose	www.musique.zz/chose
Biplan	Machin	Machin	www.machin.com
NULL	*NULL*	Truc	www.musique.zz/truc

Il faut utiliser la jointure.... quand c'est nécessaire !

Problème :exemple précédent (liste des concerts + sites web

solution 1 (bricolage , peu efficace)

1. Faire une requête sur la table concerts pour récupérer les noms des artistes (supposons qu'il y en a n)
   `select artiste from concerts`
2. puis, dans une boucle (PHP ?) , faire n requêtes SQL sur la table artistes pour obtenir les adresses web des sites.
   `select site from artistes where nom= ....`

solution 2 (efficace, avec jointure)

1. Faire une seule requête de jointure entre les 2 tables :

```
select artiste, site  from concerts,artistes
    where artiste=nom
```


Il faut utiliser la jointure.

La deuxième solution est bien meilleure :

- On réduit le nombre de requêtes.
- Le SGBD est censé optimiser les calculs de jointure. Il le fera plus efficacement que dans un programme externe (PHP ou autre).
- Il faut chercher à réduire le volume de données qui transite entre le SGBD et le programme ainsi que le volume des données gérées par le programme.

Opérations ensemblistes : union, différence, intersection

combiner 2 requêtes (ou plus)

union : **UNION**, différence : **EXCEPT**, intersection : **INTERSECTION**

condition

- les requêtes doivent avoir le même nombre de colonnes
- les domaines (types) des colonnes doivent être compatibles

(voir le cours sur l'algèbre relationnelle)

exemple

```
select nom from R union select prenom from R
```

- La première requête détermine les noms des colonnes
- Le mot clé **ALL** peut être ajouté pour conserver les doublons

PDO : accès aux bases de données depuis PHP

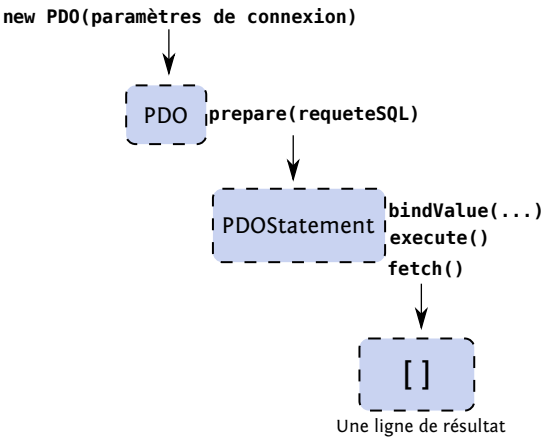
PHP Data Object (PDO)

- API permettant un accès uniformisé aux différentes bases de données
- Orientée objet

3 classes

- `PDO` : une instance de PDO représente la connexion à une base de données.
  ⇒ le plus souvent une seule instance de PDO par exécution de PHP
- `PDOStatement` : une instance de PDOStatement représente une requête vers la base.
  ⇒ permet de préparer la requête puis de consulter son résultat.
- `PDOException`

PDO : schéma



PDO : connexion

Création de la connexion : `new PDO(...)`

- Forme générale :

```
$connexion= new PDO($argDSN, $user, $passwd);
```

- Pour Postgres

```
$connexion= new PDO(
    "pgsql:host=nomdhôte;dbname=nomdebase",
    $user, $passwd);
```

Exemple de connexion avec utilisation de l'exception

```
try { $connexion= new PDO(
   "pgsql:host=localhost;dbname=nomLogin","nomLogin","XX");
} catch (PDOException $e) {
   echo("Erreur connexion" : $e->getMessage() );
   exit();
}
```

PDO : requêtes préparées

PDO::prepare

- méthode : `PDOStatement prepare($requeteSQL);`
- prépare la requête mais ne l'exécute pas
- bien adaptée pour les requêtes exécutées plusieurs fois.
- mais aussi pour **sécuriser les requêtes avec paramètres**
- l'objet `PDOStatement` permettra (ensuite) de lancer l'exécution.
-
```
$stmt = $connexion->prepare(
          "select nom,prenom from tab"
        );
// ....
$stmt->execute();
```
- **Une fois la requêtes exécutée,** l'objet `PDOStatement` permet d'explorer le résultat de la requête.

PDO : requêtes préparées paramétrées

PDO::prepare

- La requête peut comporter des "paramètres"...
- ... dont on fixe la valeur au moment de l'exécution
- Sécurité contre **l'injection de code**

```
$stmt = $connexion->prepare(
         "select nom,prenom from tab where nom=:name"
        );
$stmt->execute(array(':name'=>'Toto'));`
//...
$stmt->execute(array(':name'=>'Machin'));`
```

Voir aussi la méthode `bindValue`

PDO : exploration des résultats

PDOStatement::fetch

- `mixed fetch();` lecture **ligne par ligne** du résultat
- Par défaut renvoie un **tableau PHP** représentant une ligne.
- renvoie `false` si plus aucune ligne n'est disponible
-
```
   // pour recevoir des tableaux associatifs :
$stmt->setFetchMode(PDO::FETCH_ASSOC);
 // parcours des lignes du résultat
while ($ligne = $stmt->fetch()) {
  print_r($ligne);
}
```

PDOStatement::setFetchMode

`PDO::FETCH_ASSOC` : tableau indexé par les noms de colonne
`PDO::FETCH_NUM` : tableau indexé par des entiers
`PDO::FETCH_CLASS` : instance d'une classe à fournir en 2ème argument

PDO : exemple complet

```
try { $connexion= new PDO(
    "pgsql:host=localhost;dbname=nomLogin","nomLogin","XX");
} catch (PDOException $e) {
    echo("Erreur connexion" : $e->getMessage() );
    exit();
}
$stmt = $connexion->prepare(
          "select nom,prenom from tab"
         );
$stmt->execute();
$stmt->setFetchMode(PDO::FETCH_ASSOC);
echo "<table>";
while ($ligne = $stmt->fetch()) {
 echo "<tr><td>{$ligne['nom']}</td>".
      "<td>{$ligne['prenom']}</td></tr>";
}
echo"</table>";
```


Danger de l'injection SQL : exemple de cas

Une table d'utilisateurs

ident	nom	carte
user1	Premier	111222333444
user2	Patrick	222333444555
user3	Alphonse	333444555666

L'attribut **carte** est un numéro de carte bancaire, donc confidentiel.

exemple

Le but

Permettre de lister tous les noms d'utilisateurs commençant par un certain préfixe.

Danger de l'injection SQL : exemple de cas

solution 1

```
<?php
 function resultToHTML($stmt){
   // construit une table HTML avec le résultat
    ...
 }
 require ('lib/connexion.php');
 $pref = $_GET['pref'];
 $sql1 = "select nom from users where nom like '$pref%'";
 $stmt = $connexion->prepare($sql1);
 $stmt->execute();
 $stmt->setFetchMode(PDO::FETCH_ASSOC);
?>
<html> ... <?php echo resultToHTML($stmt);?> ...</html>
```

Où est le problème ?

Danger de l'injection SQL : exemple de cas

Le (gros) problème

Si on envoie comme variable **pref** la chaîne

```
' union select carte from users--
```

la requête devient

```
select nom from users where nom like ''
union
select carte from users--%'
```

- le 1er select ne donne aucun résultat
- le 2ème select envoie tous les numéros de carte !

Danger de l'injection SQL : exemple de cas

la bonne solution

```
  ...
$sql2 =
 "select nom  from users"
 ." where nom like :prefixe ||'%' ";
$stmt = $connexion->prepare($sql2);
$stmt->bindValue(":prefixe",$pref);
$stmt->execute();
$stmt->setFetchMode(PDO::FETCH_ASSOC);
```

Requêtes préparées

Lors de l'association pseudo-variable <-> valeur, un filtrage est réalisé (échappement de caractères) pour empêcher l'injection SQL

Ajout de données : commande `insert`

`insert` permet d'ajouter des tuples dans une table

- Forme simple :

  insert into *nomTable* values (*liste valeurs*)

  ```
  insert into Concerts values ('Zénith','Chose')
  ```

  Fournir **toutes** les valeurs, **dans l'ordre** des attributs.
- On peut spécifier les attributs :

  insert into *nomTable* (*liste attrs*) values (*liste valeurs*)

  ```
  insert into Concerts (salle,artiste)
         values ('Zénith','Chose')
  insert into Concerts (artiste,salle)
         values ('Chose','Zénith')
  ```

  On peut omettre certains attributs (leur valeur sera NULL).

Ajout de données : `insert ... select`

Les valeurs à ajouter peuvent résulter d'un select.

insert into *nom de table* (*liste d'attributs*)
select *suite du select*

- Le select doit renvoyer autant de colonnes qu'il y a d'attributs dans la liste.
- Les types des données doivent être compatibles
- Cette forme de commande insert permet d'ajouter plusieurs tuples d'un coup (chaque ligne résultant du select)
- Permet de faire simplement une recopie des données d'une table.

Suppression de données : commande `delete`

supprimer une ou des lignes

delete from *nom de table* where *qualification*

(la clause where est optionnelle).
Supprime de la table tous les tuples satisfaisant la qualification.
Exemples

- `delete from Concerts where salle='Biplan'`

  détruit toutes les lignes correspondantes.
- `delete from Concerts`

  vide la table...

Cette opération est irréversible !

Modification de données : commande `update`

update *nom de table* set *attribut* = *expression* where *qualification*

(la clause where est optionnelle).
Exemple : remplacement du coût TTC par le coût HT

```
update decompte set cout=cout/1.2
```

Là aussi, l'opération est irréversible.

```
update decompte set cout=0
```

Création de table : commande `create`

commande `create`

Syntaxe simplifiée :

`create table` *nom de table* (*attribut type, ...*)

Ex :

```
create table etudiants
  ( nom varchar(20),
    prenom varchar(30),
    groupe smallint,
    naissance date
  )
```


Création de table ET insertion de données : `create ... as select`

Duplication

Il est possible de créer simplement une table par duplication avec la commande

`create table` *nom de table* **as select** *suite du select*

Exemple

```
create table copie_etudiants
     as select * from etudiants;

create table jeunes
     as select * from etudiants
         where naissance > '1995-1-1';
```


Les types de données

Les attributs peuvent être de type :

CHAR(n)	Chaîne de n caractères exactement.
VARCHAR(n)	Chaîne de n caractères maximum.
INTEGER	Entier signé, sur 32 bits
SMALLINT	Entier signé, sur 16 bits
BIGINT	Entier signé, sur 64 bits
FLOAT	Flottant
NUMERIC(n,d)	n chiffres **dont** d après la virgule
DATE	date `A-M-J`
TIME	temps `H:MN:S`
TIMESTAMP	temps (estampille)
BLOB	donnée binaire $\leq 2^{16} = 64K$ octets

Conception : définition de contraintes et intégrité

On peut demander au SGBD de vérifier un certain nombre de contraintes portant sur les valeurs des tables.

- **Contraintes portant sur chaque ligne : Restriction du type d'un attribut**. Seules certaines valeurs sont autorisées (ex valeur maxi, valeur mini ...). Interdire NULL
- **Contraintes portant sur une table dans son ensemble : Unicité / Primalité**. Vérifier qu'une valeur d'attribut ou une combinaison des valeurs d'attributs figure au plus une fois dans la table.
- **Contraintes portant sur plusieurs tables : Intégrité référentielle**. Vérifier qu'une valeur d'attribut ou une combinaison des valeurs d'attributs existe dans une **autre** table.

Restriction du type

Exemple

- aucun attribut ne peut contenir NULL
- le prix (type numeric) doit être > 0.
- la date (type date) doit être en 2004.

```
create table prixdate (
    ref character(5) NOT NULL,
    prix numeric(6,2) NOT NULL,
    date date NOT NULL,
    check (prix >0),
    check (extract(year from date)=2004)
);
```


Unicité

Un attribut vérifie la contrainte d'unicité si ...

chaque valeur figure dans **au plus 1 ligne**.

- Exemple :
  On pourra demander à un attribut « Num INSEE » de verifier l'unicité : un numéro donné ne pourra alors apparaître qu'une fois dans la table.
- On peut définir une contrainte d'unicité portant sur un n-uple d'attributs
  Exemple : Une table contient des attributs nom et prénom et l'on veut vérifier que chaque couple (nom, prénom) ne concerne qu'au plus un enregistrement.

Primalité

On peut le plus souvent choisir une **clé primaire** dans une table. Cette clé peut être composée de 1 ou plusieurs attributs.
Une clé doit

- Vérifier le principe d'**unicité**.
- Ne pas valoir **null**

La présence d'une clé primaire permet au SGBD de structurer la représentation de la relation en l'indexant sur la clé primaire. Il facilite ainsi les recherches dans la table qui se fondent sur cette clé.
Il est souhaitable de définir une clé primaire dans une table dès lors qu'une clé est naturellement candidate à le devenir.

Intégrité référentielle

- Quand toutes les valeurs d'un attribut d'une table doivent apparaître dans un attribut d'une autre table, cet attribut constitue une **clé étrangère**.
- Exemple : Une base de donnée contient une table d'**étudiants** (nom, prénom, groupe, date de naissance) dont la clé primaire est le couple (nom, prénom). Par ailleurs elle contient une table d'**évaluations** d'étudiants (nom, prénom, module, note...).
- Si la relation évaluations faisait intervenir un couple nom et prénom absent de la table d'étudiants cela représenterait une incohérence.
- Dans la table évaluations, la clé (nom, prenom) est une clé étrangère qui doit coïncider avec une clé primaire de la table étudiants.

Résumé des contraintes

Une table **peut** contenir :

- Une seule clé primaire.
- Aucune, une ou plusieurs autres clés "uniques".
- Aucune, une ou plusieurs clés étrangères.

Exemple (suite) : la base contient une table des modules (nom, coefficient) dont le nom constitue la clé primaire.
La table évaluations contient une deuxième clé étrangère : module qui doit coïncider avec la clé modules.nom .
Par ailleurs le triplet nom, prenom, module pourrait constituer une clé primaire de la table évaluations.

Contrainte simple en SQL

La clause **check** *condition* peut figurer dans la liste des définitions de colonne. Il peut y avoir plusieurs clauses check.
La clause **not null** peut compléter une définition de colonne.

```
create table etudiants
  ( nom varchar(20) not null,
    prenom varchar(30) not null,
    groupe smallint,
    naissance date,
    check (groupe between 1 and 4),
    check (naissance < '2004-1-1')
  )
```


Clé primaire en SQL

La définition de clé primaire figure dans la liste des définition d'attributs par la clause :
primary key (*liste d'attributs*)

```
create table etudiants
  ( nom varchar(20) not null,
    prenom varchar(30) not null,
    groupe smallint,
    naissance date,
    check (groupe between 1 and 4),
    check (naissance < '2004-1-1'),
    primary key (nom, prenom)
  )
```


Clé étrangère en SQL

De la même manière une clé étrangère est définie par **foreign key** (*liste d'attributs*)
references*nom de table*(*liste d'attributs*)

```
create table evaluations
 ( nom varchar(20),
   prenom varchar(30),
   module varchar(12),
   note smallint, check (note between 0 and 20),
   foreign key (nom, prenom)
      references etudiants(nom,prenom),
   foreign key (module)
      references modules(nom)
 )
```

Synthèse. Destruction de données

La commande de destruction de donnée (delete) n'est pas réversible. Son usage direct est donc dangereux.
Prenons comme exemple la suppression de tous les étudiants des groupes 1 et 2. la commande directe serait

```
delete from etudiants where groupe<=2
```

Comment peut on agir plus sûrement ?

- Tester par avance la qualification utilisée. Un commande permet, de plus de copier les données selectionnées.

```
create table sauvegarde
  as select * from etudiants where groupe<=2
```

- Une petite vérification

```
select * from sauvegarde
```


Synthèse. Destruction de données(2/2)

- On peut maintenant supprimer :

```
delete from etudiants where groupe<=2
```

- Si l'on veut restaurer :

```
insert into etudiants
   select * from sauvegarde;
```

On pourrait ne faire qu'une restauration partielle en complétant par une qualification adaptée :

```
insert into etudiants
  select * from sauvegarde where groupe=1;
```

- Si l'on décide de perdre la sauvegarde :

```
drop table sauvegarde;
```


Synthèse : exemple des notes(1/3)

Faire la liste des étudiants avec la note globale obtenue :

```
select evaluations.nom, prenom, sum(coeff*note)
   from evaluations,modules
   where evaluations.module=modules.nom
   group by evaluations.nom, prenom
```

Problème :

- On ne connait pas le nombre de modules évalués pour chaque étudiant.
- On affiche même les étudiants qui n'ont pas tout passé.

Synthèse : exemple des notes(2/3)

Faire la liste des étudiants ayant des évaluations pour une somme de 10 coefficients. On affiche la moyenne obtenue :

```
select evaluations.nom, prenom,
       sum(coeff*note)/sum(coeff) as moyenne
   from evaluations,modules
   where evaluations.module=modules.nom
   having sum(coeff)=10
```


Synthèse : exemple des notes(3/3)

Affichage avec le numéro de groupe, classé par groupe, puis par moyennes décroissantes

```
select groupe, etudiants.nom, etudiants.prenom,
       sum(coeff*note)/sum(coeff) as moyenne
   from evaluations,modules,etudiants
   where evaluations.module=modules.nom
     and evaluations.nom=etudiants.nom
     and evaluations.prenom=etudiants.prenom
   group by evaluations.nom, prenom
   having sum(coeff)=10
   order by groupe asc, moyenne desc
```