Intégrité des données

Informatique et Statistique 2A 3ème année

Olivier Caron[1]

[1]École d'ingénieurs Polytech Lille
Université de Lille

9 novembre 2023

Contenu

1. Sécurité des données
   - les activités du DBA
   - Exemple de fraude
   - Administration des bases de données Postgres
2. Définition de contraintes

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)
  - limiter l'accès des activités d'administration du serveur à un nombre minimal de machines du réseau.

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)
  - limiter l'accès des activités d'administration du serveur à un nombre minimal de machines du réseau.
  - auditer régulièrement les échecs de connexions via les logs, mise en place d'alerte

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)
  - limiter l'accès des activités d'administration du serveur à un nombre minimal de machines du réseau.
  - auditer régulièrement les échecs de connexions via les logs, mise en place d'alerte
  - mettre à jour régulièrement le SGBD

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)
  - limiter l'accès des activités d'administration du serveur à un nombre minimal de machines du réseau.
  - auditer régulièrement les échecs de connexions via les logs, mise en place d'alerte
  - mettre à jour régulièrement le SGBD
  - effectuer une veille technologique sur les techniques de **hacking**.

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, l'activité du **DBA** consiste à :
  - maintenir régulièrement les utilisateurs référencés (suivi du personnel)
  - définir un niveau d'exigence des mots de passe (durée de vie du mot de passe, complexité du mot de passe)
  - limiter l'accès des activités d'administration du serveur à un nombre minimal de machines du réseau.
  - auditer régulièrement les échecs de connexions via les logs, mise en place d'alerte
  - mettre à jour régulièrement le SGBD
  - effectuer une veille technologique sur les techniques de **hacking**.
  - informer, documenter les développeurs

Injection SQL

Définition

L'injection SQL directe est une technique où un pirate modifie une requête SQL existante pour afficher des données cachées, ou pour écraser des valeurs importantes, ou encore exécuter des commandes dangereuses pour la base.

Injection SQL

Définition

L'injection SQL directe est une technique où un pirate modifie une requête SQL existante pour afficher des données cachées, ou pour écraser des valeurs importantes, ou encore exécuter des commandes dangereuses pour la base.

- L'injection SQL s'applique à n'importe quel langage de programmation (PHP, Java, ...),

Injection SQL

Définition

L'injection SQL directe est une technique où un pirate modifie une requête SQL existante pour afficher des données cachées, ou pour écraser des valeurs importantes, ou encore exécuter des commandes dangereuses pour la base.

- L'injection SQL s'applique à n'importe quel langage de programmation (PHP, Java, ...),
- De nombreuses variantes sur le détournement de `select`, `update`, ...

Injection SQL

Définition

L'injection SQL directe est une technique où un pirate modifie une requête SQL existante pour afficher des données cachées, ou pour écraser des valeurs importantes, ou encore exécuter des commandes dangereuses pour la base.

- L'injection SQL s'applique à n'importe quel langage de programmation (PHP, Java, ...),
- De nombreuses variantes sur le détournement de `select`, `update`, ...
- Importance de vérifier les données transmises !

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

  `http://www.serv.fr/connect.php?login=admin&password=secret`

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

  `http://www.serv.fr/connect.php?login=admin&password=secret`

- La requête SQL va être construite par le code PHP suivant :

```
$req="select 1 from user where username='$login' and password='$password'";
```

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

  `http://www.serv.fr/connect.php?login=admin&password=secret`

- La requête SQL va être construite par le code PHP suivant :

```
$req="select 1 from user where username='$login' and password='$password'";
```

- Si pas de ligne en résultat, l'authentification est refusée.

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

  `http://www.serv.fr/connect.php?login=admin&password=secret`
- La requête SQL va être construite par le code PHP suivant :

  `$req="select 1 from user where username='$login' and password='$password'";`
- Si pas de ligne en résultat, l'authentification est refusée.
- Le hackeur saisit les chaînes `"' or '1'='1"` dans les champs `login` et `password`.

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

```
http://www.serv.fr/connect.php?login=admin&password=secret
```

- La requête SQL va être construite par le code PHP suivant :

```
$req="select 1 from user where username='$login' and password='$password'";
```

- Si pas de ligne en résultat, l'authentification est refusée.
- Le hackeur saisit les chaînes "' or '1'='1" dans les champs `login` et `password`.
- La requête SQL générée sera donc :

```
"select * from user where username='' or '1'='1' and password='' or '1'='1'"
```

Exemple d'injection SQL

- Soit un programme qui vérifie le login/password d'un formulaire :

  `http://www.serv.fr/connect.php?login=admin&password=secret`
- La requête SQL va être construite par le code PHP suivant :

  `$req="select 1 from user where username='$login' and password='$password'";`
- Si pas de ligne en résultat, l'authentification est refusée.
- Le hackeur saisit les chaînes `"' or '1'='1"` dans les champs `login` et `password`.
- La requête SQL générée sera donc :

  `"select * from user where username='' or '1'='1' and password='' or '1'='1'"`
- Conséquence : l'authentification est acceptée !

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

  trust les connexions sont autorisées sans condition.

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

  trust les connexions sont autorisées sans condition.

  reject les connexions sont refusées sans condition.

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

trust	les connexions sont autorisées sans condition.
reject	les connexions sont refusées sans condition.
password	mot de passe transmis en clair.

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

trust	les connexions sont autorisées sans condition.
reject	les connexions sont refusées sans condition.
password	mot de passe transmis en clair.
crypt	mot de passe transmis en chiffré

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

trust	les connexions sont autorisées sans condition.
reject	les connexions sont refusées sans condition.
password	mot de passe transmis en clair.
crypt	mot de passe transmis en chiffré

- Connexions distantes, mode supplémentaire :

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

  trust les connexions sont autorisées sans condition.

  reject les connexions sont refusées sans condition.

  password mot de passe transmis en clair.

  crypt mot de passe transmis en chiffré

- Connexions distantes, mode supplémentaire :

  ident authentification TCP/IP de l'utilisateur Unix.

Connexions à un serveur Postgres

- Connexions locales (`locahost`), plusieurs modes :

  trust les connexions sont autorisées sans condition.

  reject les connexions sont refusées sans condition.

  password mot de passe transmis en clair.

  crypt mot de passe transmis en chiffré

- Connexions distantes, mode supplémentaire :

  ident authentification TCP/IP de l'utilisateur Unix.

- Fichiers de configuration (`pg_hba.conf`) pour préciser machine(s) éligibles

La sécurité sous Postgres

- Plusieurs niveaux d'utilisateurs :

La sécurité sous Postgres

- Plusieurs niveaux d'utilisateurs :
  - L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...

La sécurité sous Postgres

- Plusieurs niveaux d'utilisateurs :
  - L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...
  - Les **utilisateurs-administrateurs** : peuvent créer des bases, peuvent autoriser d'autres utilisateurs à accéder à ces bases.
    Certains utilisateurs-administrateurs peuvent créer d'autres utilisateurs.

La sécurité sous Postgres

- Plusieurs niveaux d'utilisateurs :
  - L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...
  - Les **utilisateurs-administrateurs** : peuvent créer des bases, peuvent autoriser d'autres utilisateurs à accéder à ces bases.
    Certains utilisateurs-administrateurs peuvent créer d'autres utilisateurs.
  - Les **utilisateurs** peuvent accéder à des bases (selon les droits)

La sécurité sous Postgres

- Plusieurs niveaux d'utilisateurs :
  - L'utilisateur **"postgres"** dispose de tous les droits (root), création bases, utilisateurs, destruction,...
  - Les **utilisateurs-administrateurs** : peuvent créer des bases, peuvent autoriser d'autres utilisateurs à accéder à ces bases.
    Certains utilisateurs-administrateurs peuvent créer d'autres utilisateurs.
  - Les **utilisateurs** peuvent accéder à des bases (selon les droits)
- Niveau défini lors de la création d'utilisateur (`create user/role`) et évolutif (`alter user/role`)

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)
- `pg_user` est accessible à tous mais pas de mot de passe affiché.

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)
- `pg_user` est accessible à tous mais pas de mot de passe affiché.
- `pg_shadow` est accessible aux administrateurs.

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)
- `pg_user` est accessible à tous mais pas de mot de passe affiché.
- `pg_shadow` est accessible aux administrateurs.
- Impossible de connaître le mot de passe : le champ `pg_shadow.passwd` est crypté.

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)
- `pg_user` est accessible à tous mais pas de mot de passe affiché.
- `pg_shadow` est accessible aux administrateurs.
- Impossible de connaître le mot de passe : le champ `pg_shadow.passwd` est crypté.
- La vérification consiste à crypter un mot de passe saisi puis de le comparer à `pg_shadow.passwd`

Mot de passe des utilisateurs sous Postgres

- Gestion des utilisateurs (vue système `pg_user` de la table `pg_shadow`)
- `pg_user` est accessible à tous mais pas de mot de passe affiché.
- `pg_shadow` est accessible aux administrateurs.
- Impossible de connaître le mot de passe : le champ `pg_shadow.passwd` est crypté.
- La vérification consiste à crypter un mot de passe saisi puis de le comparer à `pg_shadow.passwd`
- L'utilisateur, son administrateur ou 'postgres' peuvent modifier le mot de passe (`alter user/role`).

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :
  - Un groupe d'utilisateurs est un rôle

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :
  - Un groupe d'utilisateurs est un rôle
  - Un utilisateur est un rôle

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :
  - Un groupe d'utilisateurs est un rôle
  - Un utilisateur est un rôle
  - mais un rôle n'est pas forcément un utilisateur (s'il ne dispose pas de droit de login)

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :
  - Un groupe d'utilisateurs est un rôle
  - Un utilisateur est un rôle
  - mais un rôle n'est pas forcément un utilisateur (s'il ne dispose pas de droit de login)
- Un utilisateur peut avoir plusieurs rôles

Les rôles sous Postgres

- On peut regrouper les utilisateurs par **rôle**.
- Vue système `pg_roles`.
- Sous Postgres, "tout est rôle" :
  - Un groupe d'utilisateurs est un rôle
  - Un utilisateur est un rôle
  - mais un rôle n'est pas forcément un utilisateur (s'il ne dispose pas de droit de login)
- Un utilisateur peut avoir plusieurs rôles
- Hiérarchie de rôles

Définition des rôles Postgres

- Un administrateur peut créer/supprimer des rôles (create/drop).

Définition des rôles Postgres

- Un administrateur peut créer/supprimer des rôles (create/drop).
- Syntaxe : `create role user [ [with] option { option }]`

Définition des rôles Postgres

- Un administrateur peut créer/supprimer des rôles (create/drop).
- Syntaxe : `create role user [ [with] option { option }]`
- Les options possibles sont :

Nom	Signification
`LOGIN`	Permet au rôle de se connecter à une base
`SUPERUSER`	Permet au rôle d'être super utilisateur
`CREATEDB`	Permet au rôle de créer des bases
`CREATEROLE`	Permet au rôle de créer des rôles
`PASSWORD`	assigne un mot de passe, exemple : `create role dupont password 'secret'`
`INHERIT` ou `NOINHERIT`	Permet ou pas d'hériter des droits des autres rôles

Définition des rôles Postgres

- Un administrateur peut créer/supprimer des rôles (create/drop).
- Syntaxe : `create role user [ [with] option { option }]`
- Les options possibles sont :

Nom	Signification
`LOGIN`	Permet au rôle de se connecter à une base
`SUPERUSER`	Permet au rôle d'être super utilisateur
`CREATEDB`	Permet au rôle de créer des bases
`CREATEROLE`	Permet au rôle de créer des rôles
`PASSWORD`	assigne un mot de passe, exemple : `create role dupont password 'secret'`
`INHERIT` ou `NOINHERIT`	Permet ou pas d'hériter des droits des autres rôles

- La commande `alter role` permet de modifier ces options.

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant is to john ;`
  "john" (r2) devient membre du rôle "is" (r1) et obtient tous les droits attribués à "is"

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant is to john ;`
  "john" (r2) devient membre du rôle "is" (r1) et obtient tous les droits attribués à "is"
- Opération duale : retrait des droits :
  Exemple :`revoke is from john ;`

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant is to john ;`
  "john" (r2) devient membre du rôle "is" (r1) et obtient tous les droits attribués à "is"
- Opération duale : retrait des droits :
  Exemple :`revoke is from john ;`
- Possibilité de hiérarchies de rôles :
  Rappel : tout est rôle (membre = user = rôle)
  Exemple :`grant is to is2a3 ;`

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant is to john ;`
  "john" (r2) devient membre du rôle "is" (r1) et obtient tous les droits attribués à "is"
- Opération duale : retrait des droits :
  Exemple :`revoke is from john ;`
- Possibilité de hiérarchies de rôles :
  Rappel : tout est rôle (membre = user = rôle)
  Exemple :`grant is to is2a3 ;`
- Rôle particulier : `public` (tout le monde)

Affectation des droits d'une table à un rôle Postgres

- Syntaxe :
  `grant ⟨liste_droits⟩ on ⟨objet⟩ to ⟨role⟩`

Affectation des droits d'une table à un rôle Postgres

- Syntaxe :

  `grant ⟨liste_droits⟩ on ⟨objet⟩ to ⟨role⟩`

- Exemples :

```
grant select on etudiant to is;
grant select,update,delete on etudiant to secretariat_is;
grant select,update(note) on etudiant to profs_is;
```

Affectation des droits d'une table à un rôle Postgres

- Syntaxe :

  `grant ⟨liste_droits⟩ on ⟨objet⟩ to ⟨role⟩`

- Exemples :

```
grant select on etudiant to is;
grant select,update,delete on etudiant to secretariat_is;
grant select,update(note) on etudiant to profs_is;
```

- ⟨objet⟩ désigne une table

Affectation des droits d'une table à un rôle Postgres

- Syntaxe :

  `grant` ⟨`liste_droits`⟩ `on` ⟨`objet`⟩ `to` ⟨`role`⟩
- Exemples :

```
grant select on etudiant to is;
grant select,update,delete on etudiant to secretariat_is;
grant select,update(note) on etudiant to profs_is;
```

- ⟨`objet`⟩ désigne une table
- Possibilité de préciser une colonne ($3^{ième}$ exemple)

Quelques droits usuels de table (1/2)

SELECT Autorise `SELECT` sur toutes les colonnes, ou sur les colonnes listées spécifiquement, de la table, vue ou séquence indiquée.

Quelques droits usuels de table (1/2)

SELECT Autorise `SELECT` sur toutes les colonnes, ou sur les colonnes listées spécifiquement, de la table, vue ou séquence indiquée.

INSERT Autorise `INSERT` d'une nouvelle ligne dans la table indiquée. Si des colonnes spécifiques sont listées, seules ces colonnes peuvent être affectées dans une commande INSERT, (les autres colonnes recevront par conséquent des valeurs par défaut)

Quelques droits usuels de table (2/2)

UPDATE Autorise `UPDATE` sur toute colonne de la table spécifiée, ou sur les colonnes spécifiquement listées. (En fait, toute commande `UPDATE` non triviale nécessite aussi le droit `SELECT` car elle doit référencer les colonnes pour déterminer les lignes à mettre à jour et/ou calculer les nouvelles valeurs des colonnes.)

Quelques droits usuels de table (2/2)

UPDATE Autorise `UPDATE` sur toute colonne de la table spécifiée, ou sur les colonnes spécifiquement listées. (En fait, toute commande `UPDATE` non triviale nécessite aussi le droit `SELECT` car elle doit référencer les colonnes pour déterminer les lignes à mettre à jour et/ou calculer les nouvelles valeurs des colonnes.)

DELETE Autorise `DELETE` d'une ligne sur la table indiquée. (En fait, toute commande `DELETE` non triviale nécessite aussi le droit `SELECT` car elle doit référencer les colonnes pour déterminer les lignes à supprimer.)

Affectation des droits

- Il existe bien d'autres attributions de droits sur les éléments suivants :

Affectation des droits

- Il existe bien d'autres attributions de droits sur les éléments suivants :
  - vue, séquence, base, fonctions, langages de procédure, schéma.

Affectation des droits

- Il existe bien d'autres attributions de droits sur les éléments suivants :
  - vue, séquence, base, fonctions, langages de procédure, schéma.
- La commande `grant` dispose de l'option `with grant option` : celui qui reçoit le droit peut le transmettre à son tour

Affectation des droits

- Il existe bien d'autres attributions de droits sur les éléments suivants :
  - vue, séquence, base, fonctions, langages de procédure, schéma.
- La commande `grant` dispose de l'option `with grant option` : celui qui reçoit le droit peut le transmettre à son tour
- Quelques contrôles effectués par le SGBD : cycle d'héritage des rôles.

Tables et vues, pas les mêmes droits (1/3)

- Utilisateur "carono", propriétaire de la base :

```
select current_user ;
current_user
--------------

carono
(1 row)

select * from etudiant ;
 login  | adresse
--------+---------
 carono | lille
 demo   | paris
(2 rows)
```

Tables et vues, pas les mêmes droits (2/3)

- Utilisateur "carono", propriétaire de la base :

```
create view my_etudiant as
  select * from etudiant where login=current_user ;
CREATE VIEW

select * from my_etudiant ;
 login  |  adresse
--------+-----------
 carono | lille
(1 row)

grant select on my_etudiant to demo ;
```

Tables et vues, pas les mêmes droits (3/3)

- Utilisateur "demo" :

```
select * from my_etudiant ;
 login  |  adresse
--------+-----------
 demo   |  paris
(1 row)

select * from etudiant ;
ERROR:  permission denied for table etudiant
```

Exercice

Exercice 1

Quelles sont les commandes pour créer un utilisateur `u1` et deux rôles `r1` et `r2`, attribuer le droit `select` d'une table `t` au rôle `r1`, faire hériter les droits de `r1` à `r2`, faire hériter les droits de `r2` à `u1`.

Exercice

Exercice 1

Quelles sont les commandes pour créer un utilisateur `u1` et deux rôles `r1` et `r2`, attribuer le droit `select` d'une table `t` au rôle `r1`, faire hériter les droits de `r1` à `r2`, faire hériter les droits de `r2` à `u1`.

Exercice 2

L'utilisateur `u1` essaye de faire un `select` sur la table `t`. Que se passe-t-il ?

Les types de contraintes

- Normalisation SQL-92

Les types de contraintes

- Normalisation SQL-92
- Les contraintes de domaine
  définissent les valeurs prises par un attribut.

Les types de contraintes

- Normalisation SQL-92
- Les contraintes de domaine
  définissent les valeurs prises par un attribut.
- Les contraintes d'intégrité d'entité
  précisent la clé primaire de chaque table

Les types de contraintes

- Normalisation SQL-92
- Les contraintes de domaine
  définissent les valeurs prises par un attribut.
- Les contraintes d'intégrité d'entité
  précisent la clé primaire de chaque table
- Les contraintes d'intégrité référentielle
  assurent la cohérence entre les clés primaires et les clés étrangères

Les types de contraintes

- Normalisation SQL-92
- Les contraintes de domaine
  définissent les valeurs prises par un attribut.
- Les contraintes d'intégrité d'entité
  précisent la clé primaire de chaque table
- Les contraintes d'intégrité référentielle
  assurent la cohérence entre les clés primaires et les clés étrangères
- Les assertions
  spécifient des contraintes plus générales entre attributs quelconques.

Contrainte de domaine : NOT NULL

```
CREATE TABLE personnel (
    nom TEXT NOT NULL,
    prenom TEXT
) ;

INSERT INTO personnel(nom) VALUES ('dupont')

INSERT INTO personnel(prenom) VALUES('henri')   -> ERREUR
```

Contrainte de domaine : DEFAULT

```
CREATE TABLE article (
   num  INT  NOT NULL,
   quantite INT DEFAULT 1,
   date_creation DATE DEFAULT now()
) ;
```

Contrainte de domaine : UNIQUE

- Éviter les redondances :

```
CREATE TABLE article (
  num  INT  NOT NULL UNIQUE,
  nom TEXT  ...
```

- L'unicité peut être constituée de plusieurs attributs :

```
CREATE TABLE reserve_par (
  num_client INT NOT NULL,
  num_livre  INT NOT NULL,
 UNIQUE (num_client, num_livre)
) ;
```

Contrainte de domaine : CHECK (1/2)

- But : spécifier une contrainte qui doit être vérifiée à tout moment par les tuples de la table :

```
CREATE TABLE personnel (
  num  INT  NOT NULL UNIQUE,
  age INT CHECK (age >= 18),
  sexe CHAR DEFAULT 'F' CHECK (sexe IN ('M','F')),
  ageFuturePromotion INT CHECK (ageFuturePromotion >= age)
) ;
```

Contrainte de domaine : CHECK (2/2)

- La clause CHECK peut se placer après la définition de tous les attributs.
  - Il est préférable de nommer la contrainte (facultatif)
- Utilisation de sous-requêtes SQL

```
CONSTRAINT moy_age
 CHECK ((select avg(age) from personnel) > 35))
```

Déclaration d'un domaine

- Plusieurs attributs ont le même type et les mêmes contraintes
- Syntaxe :

```
CREATE DOMAIN nom [AS] type_donnees
  [DEFAULT expression ] [ contrainte [ ...] ]
```

- Exemple :

```
CREATE DOMAIN entier_positif
   INT DEFAULT 0 CHECK (VALUE >=0) ;

CREATE TABLE personnel(
   num  INT  NOT NULL UNIQUE,
   age entier_positif
   ...
```

Les contraintes d'intégrité d'entité

- Permet de spécifier la clé primaire
- Analogue à NOT NULL UNIQUE
- Génère un index
- Peut être spécifiée à part lorsque la clé est constituée de plusieurs attributs (idem clause UNIQUE)

```
CREATE TABLE personnel (num INT PRIMARY KEY
    ...
```

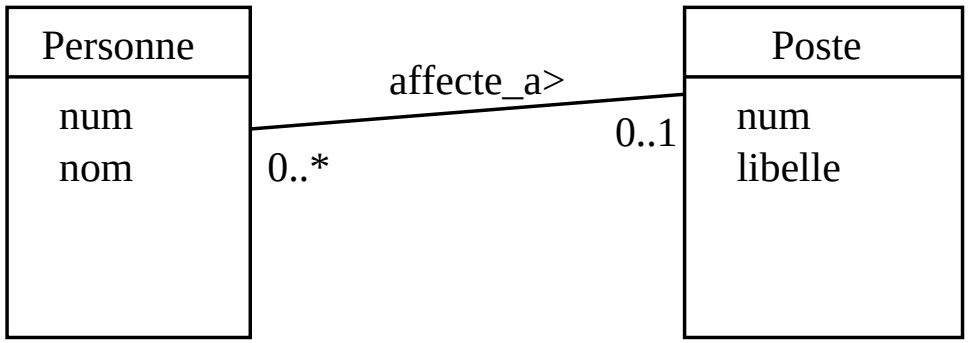
Les contraintes d'intégrité référentielle

- Définies dans la norme SQL-92

Les contraintes d'intégrité référentielle

- Définies dans la norme SQL-92
- Permettent d'assurer la cohérence des associations issues de la conception.

CIR : une cardinalité "0..1" (1/3)



CIR : une cardinalité "0..1" (2/3)

- Réalisation des tables :

```
CREATE TABLE poste (
    num INT PRIMARY KEY,
    libelle TEXT NOT NULL UNIQUE
) ;
CREATE TABLE personne (
    num INT PRIMARY KEY,
    nom TEXT NOT NULL,
    num_poste INT REFERENCES poste) ;
```

CIR : une cardinalité "0..1" (3/3)

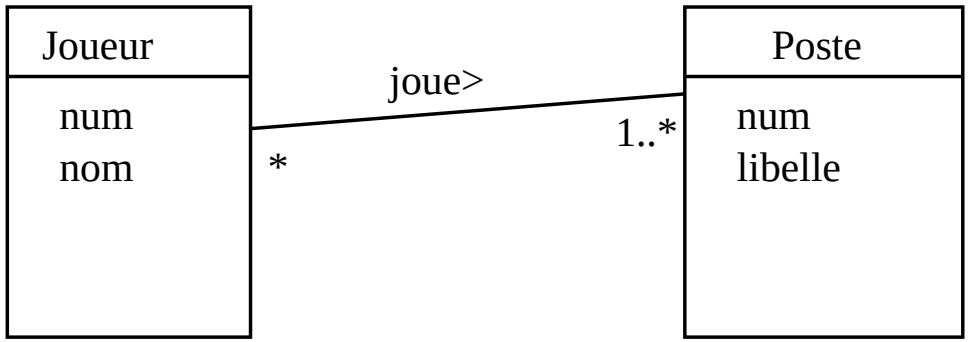
- table poste :

num	libelle
1	'directeur'
2	'ingenieur'
3	'agent'

- exécution :

```
INSERT INTO personne VALUES (1,'dupont', 1);      -> OK
INSERT INTO personne VALUES (2,'durant', 4);      -> ERREUR
DELETE FROM poste  WHERE num=1;                   -> ERREUR
UPDATE personne SET num_poste=NULL WHERE num=1;   -> OK
DELETE FROM poste WHERE num=1;                    -> OK
```

CIR : une cardinalité "*" (1/3)



CIR : une cardinalité "*" (2/3)

- Réalisation des tables :

```
CREATE TABLE poste (
   num INT PRIMARY KEY,
   libelle TEXT NOT NULL UNIQUE
) ;
CREATE TABLE joueur (
 num INT PRIMARY KEY, nom TEXT NOT NULL
);
CREATE TABLE joue (
   num_joueur INT NOT NULL REFERENCES joueur,
   num_poste INT NOT NULL REFERENCES poste,
   PRIMARY KEY (num_joueur, num_poste)
);
```

CIR : une cardinalité "*" (3/3)

- Les tables joueur et poste :

joueur	
num	nom
1	'Varane'
2	'Areola'

poste	
num	libelle
1	'goal'
2	'defenseur'
3	'milieu'
4	'attaquant'

- Exécution (en rouge : erreur) :

```
INSERT INTO joue(num_joueur,num_poste) VALUES (1,2);
INSERT INTO joue(num_joueur,num_poste) VALUES (1,3);
INSERT INTO joue(num_joueur,num_poste) VALUES (2,1);
```

```
INSERT INTO joue(num_joueur,num_poste) VALUES (2,1)
INSERT INTO joue(num_joueur,num_poste) VALUES (2,5)
DELETE FROM poste where num=3
DELETE FROM poste where num=4
```

Contraintes et clés étrangères

- Plusieurs modes possibles.

Contraintes et clés étrangères

- Plusieurs modes possibles.
- Objectifs communs : préserver la cohérence de la base

Contraintes et clés étrangères

- Plusieurs modes possibles.
- Objectifs communs : préserver la cohérence de la base
- Mode par défaut :
  Refuser l'opération si contrainte non respectée

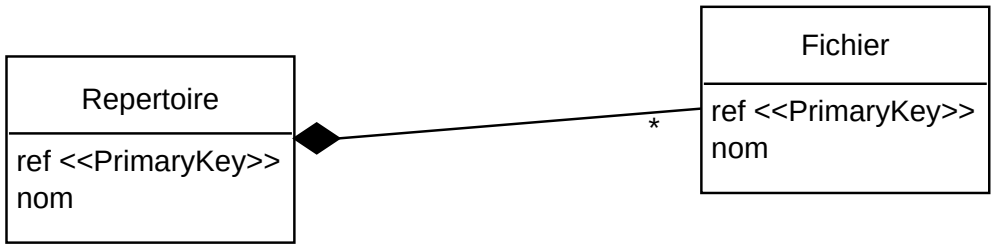
Contraintes et clés étrangères

- Plusieurs modes possibles.
- Objectifs communs : préserver la cohérence de la base
- Mode par défaut :
  Refuser l'opération si contrainte non respectée
- Autre mode :
  Accepter l'opération et modification en cascade pour préserver la cohérence.

Contraintes et clés étrangères

- Plusieurs modes possibles.
- Objectifs communs : préserver la cohérence de la base
- Mode par défaut :
  Refuser l'opération si contrainte non respectée
- Autre mode :
  Accepter l'opération et modification en cascade pour préserver la cohérence.
- Définition des modes lors de la définition de contraintes.

Contraintes sur clés étrangères : un exemple



- Valeurs initiales des tables :

Table repertoire		Table fichier		
ref	nom	ref	nom	ref_rep
0	'/'	1	'fic1'	1
1	'/rep1'	2	'fic2'	2
2	'/rep2'	3	'fic3'	1

Contraintes sur clé étrangère (1/5)

- Mode par défaut
- Le plus restrictif : refus de l'opération
- Exemple :

```
CREATE TABLE fichier (
    ref INTEGER PRIMARY KEY, nom TEXT NOT NULL,
    ref_rep INTEGER REFERENCES repertoire
            ON DELETE RESTRICT
) ;
...
DELETE FROM repertoire where ref=1 ;
```

- Résultat : la suppression est refusée

Contraintes sur clé étrangère (2/5)

- Le mode permissif
- Exemple :

```
CREATE TABLE fichier (
    ref INTEGER PRIMARY KEY, nom TEXT NOT NULL,
    ref_rep INTEGER REFERENCES repertoire
            ON DELETE CASCADE
) ;
...
DELETE FROM repertoire WHERE ref=1 ;
```

- Résultat : '/rep1' supprimé, 'fic1' et 'fic3' également !

Contraintes sur clé étrangère (3/5)

- Valeurs par défaut
- Exemple :

```
CREATE TABLE fichier (
    ref INTEGER PRIMARY KEY, nom TEXT NOT NULL,
    ref_rep INTEGER DEFAULT 0 REFERENCES repertoire
            ON DELETE SET DEFAULT
) ;
DELETE FROM repertoire WHERE ref=1 ;
```

- Résultat : supprime '/rep1', 'fic1' et 'fic3' sont désormais dans '/'
- Si la valeur par défaut est incohérente, la suppression est refusée

Contraintes sur clé étrangère (4/5)

- Clause `SET NULL`
- Exemple :

```
CREATE TABLE fichier (
    ref INTEGER PRIMARY KEY, nom TEXT NOT NULL,
    ref_rep INTEGER REFERENCES repertoire
            ON DELETE SET NULL
) ;
...
DELETE FROM repertoire WHERE ref=1 ;
```

- Résultat : supprime '/rep1', 'fic1' et 'fic3' ne sont pas affecté à un répertoire

Contraintes sur clé étrangère (5/5)

- Les contraintes sont également valables pour la mise à jour
- Syntaxe identique
- Exemple :

```
CREATE TABLE fichier (
    ref INTEGER PRIMARY KEY, nom TEXT NOT NULL,
    ref_rep INTEGER REFERENCES repertoire
            ON DELETE CASCADE ON UPDATE CASCADE
) ;
...
```

Conclusion

- Mécanisme de contraintes très développé
- Largement utilisé depuis SQL-92
- Syntaxe classique (contraintes nommées) :

  CONSTRAINT nom [**UNIQUE** | **NOT NULL** | ...]

- Mise à jour de contraintes via `ALTER TABLE`