Administration & apports SQL 3

Olivier Caron

Polytech Lille
Avenue Paul Langevin
Cité Scientifique, Univ. Lille
59655 Villeneuve d'Ascq cedex

http://ocaron.polytech-lille.net
Olivier.Caron@polytech-lille.fr

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, une activité du **DBA** est :

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, une activité du **DBA** est :
  - Maintenir régulièrement les utilisateurs référencés (suivi du personnel)

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, une activité du **DBA** est :
  - 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, une activité du **DBA** est :
  - 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, une activité du **DBA** est :
  - 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, une activité du **DBA** est :
  - 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, une activité du **DBA** est :
  - 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
  - Veille technologique sur les techniques de **hacking**.

Gestion des utilisateurs, bonnes pratiques

- Chaque SGBD a sa propre base d'utilisateurs, une activité du **DBA** est :
  - 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
  - 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 :

```
$req="select * from user where username='$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 :

  ```
  $req="select * from user where username='$login' and
  password='$password'";
  ```
- Si l'exécution SQL ne fournit pas de ligne, 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 :

  `$req="select * from user where username='$login' and password='$password'";`
- Si l'exécution SQL ne fournit pas de ligne, 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 :

  ```
  $req="select * from user where username='$login' and
  password='$password'";
  ```
- Si l'exécution SQL ne fournit pas de ligne, 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 :

  ```
  $req="select * from user where username='$login' and
  password='$password'";
  ```
- Si l'exécution SQL ne fournit pas de ligne, 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	Permet ou pas d'hériter
`NOINHERIT`	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	Permet ou pas d'hériter
`NOINHERIT`	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 gis to john ;`
  ”john” (r2) devient membre du rôle ”gis” (r1) et obtient tous les droits attribués à ”gis”

Affectation des rôles Postgres

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

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant gis to john ;`
  ”john” (r2) devient membre du rôle ”gis” (r1) et obtient tous les droits attribués à ”gis”
- Opération duale : retrait des droits :
  Exemple :`revoke gis from john ;`
- Possibilité de hiérarchies de rôles :
  Rappel : tout est rôle (membre = user = rôle)
  Exemple :`grant gis3 to gis ;`

Affectation des rôles Postgres

- Assigner les droits du rôle `r1` au rôle `r2` :
  Exemple :`grant gis to john ;`
  ”john” (r2) devient membre du rôle ”gis” (r1) et obtient tous les droits attribués à ”gis”
- Opération duale : retrait des droits :
  Exemple :`revoke gis from john ;`
- Possibilité de hiérarchies de rôles :
  Rappel : tout est rôle (membre = user = rôle)
  Exemple :`grant gis3 to gis ;`
- 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 gis;
grant select,update,delete on etudiant to
secretariat_gis;
grant select,update(note) on etudiant to
profs_gis;
```

Affectation des droits d'une table à un rôle Postgres

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

```
grant select on etudiant to gis;
grant select,update,delete on etudiant to
secretariat_gis;
grant select,update(note) on etudiant to
profs_gis;
```

- ⟨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 gis;
grant select,update,delete on etudiant to
secretariat_gis;
grant select,update(note) on etudiant to
profs_gis;
```

- `⟨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.

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 ?

2ne partie :du monde relationnel au monde objet

- Les SGBD**OR** (Objet-Relationnel)

2ne partie :du monde relationnel au monde objet

- Les SGBD**OR** (Objet-Relationnel)
- La norme SQL 3

Le SGBD Objet-Relationnel Postgres

- Identité des T-uples

Le SGBD Objet-Relationnel Postgres

- Identité des T-uples
- Des attributs tableaux

Le SGBD Objet-Relationnel Postgres

- Identité des T-uples
- Des attributs tableaux
- Héritage de table

Identité de T-uple

- Chaque t-uple a une identité gérée par postgres (oid)
- Exemple :

```
create table personne (num integer primary key,
  nom varchar(20) not null) with oids;
CREATE TABLE 1
insert into personne values (1,'dupont') ;
INSERT 0 1

select oid, * from personne ;
  oid  | num |  nom
-------+-----+--------
 18829 |   1 | dupont
(1 row)
```

Des attributs tableaux

- Possibilité de définir des tableaux à plusieurs dimensions
- On peut fixer ou pas la taille des tableaux
- Exemple :

```
create table groupe (numg integer primary key,
  nom  varchar(20) not null unique,
  membres integer []) ;
  insert into groupe values (1,'clients', '{2, 4, 6}') ;

  select membres[1] as premier_membre from groupe ;
  select membres[2:3] as deux_derniers_membres from groupe ;
```

L'héritage (1/2)

- Le premier SGBD !

L'héritage (1/2)

- Le premier SGBD !
- Possibilité d'héritage simple de table

L'héritage (1/2)

- Le premier SGBD !
- Possibilité d'héritage simple de table
- Possibilité d'héritage multiple !

L'héritage (1/2)

- Le premier SGBD !
- Possibilité d'héritage simple de table
- Possibilité d'héritage multiple !
- Prise en compte de l'extension des tables héritées

L'héritage (1/2)

- Le premier SGBD !
- Possibilité d'héritage simple de table
- Possibilité d'héritage multiple !
- Prise en compte de l'extension des tables héritées
- Distinction des types

L'héritage (2/2)

```
create table personnel(
  num int primary key,
  nom text not null,
  salaire float) ;

create table chef
 (prime float)
 inherits (personnel) ;

insert into personnel values
 (1,'dupont', 7000.0) ;
insert into chef values
 (2, 'durant', 10000.0, 2000.0) ;
```

```
select * from personnel ;
 num |  nom   | salaire
-----+--------+---------
   1 | dupont |    7000
(1 row)

select * from personnel* ;
 num |  nom   | salaire
-----+--------+---------
   1 | dupont |    7000
   2 | durant |   10000
(2 rows)
```

SQL3 / code Oracle

- Définitions de types et de sous-types.

SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.

SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.
- Pointeurs (attributs REF).

SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.
- Pointeurs (attributs REF).
- Collections :

SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.
- Pointeurs (attributs REF).
- Collections :
  - Tables imbriquées (Nested Tables)

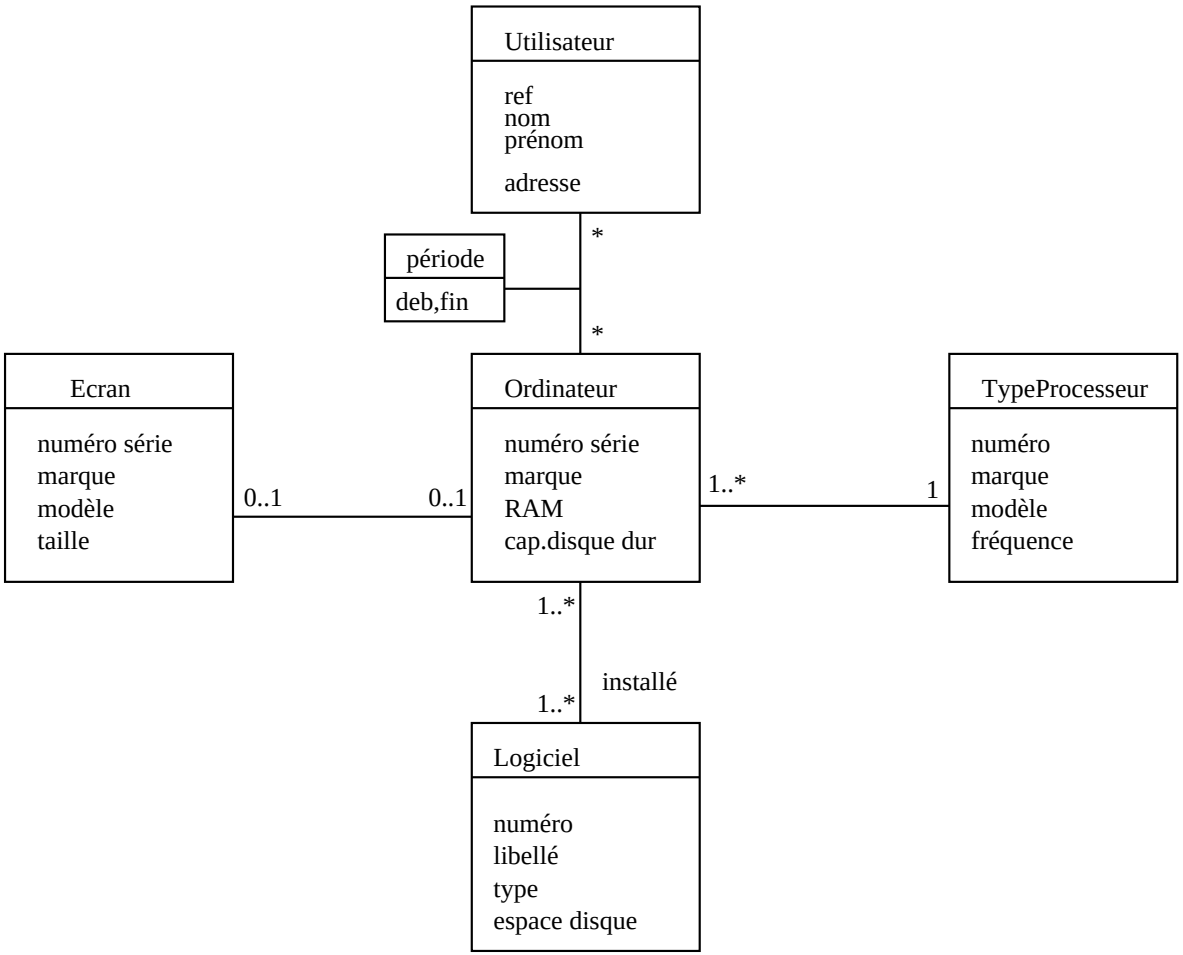
SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.
- Pointeurs (attributs REF).
- Collections :
  - Tables imbriquées (Nested Tables)
  - Tableaux prédimensionnés (VARRAY)

SQL3 / code Oracle

- Définitions de types et de sous-types.
- Types abstraits de données (ADT) = types d'objets.
- Pointeurs (attributs REF).
- Collections :
  - Tables imbriquées (Nested Tables)
  - Tableaux prédimensionnés (VARRAY)
- Pas d'héritage (ni sur les ADT, ni sur les tables).

Exemple


Utilisateur
ref
nom
prénom
adresse
période
deb,fin
*
*
Ecran
numéro série
marque
modèle
taille
0..1
0..1
Ordinateur
numéro série
marque
RAM
cap.disque dur
1..*
1
TypeProcesseur
numéro
marque
modèle
fréquence
1..*
installé
1..*
Logiciel
numéro
libellé
type
espace disque

Association 1-1

```
create type ordi_type ;

create type ecran_type as object(
   numserie NUMBER(2),
   marque VARCHAR2(20),
   modele VARCHAR2(20),
   taille NUMBER(2),
   ordi REF ordi_type
)
```

```
create type ordi_type as object(
   numserie NUMBER(2),
   marque VARCHAR2(20),
   RAM NUMBER(4),
   capDisqueDur NUMBER(3),
   ecran REF ecran_type,
   ...)
```

Association 1-*

```
create type refOrdi_type as OBJECT (refOrdi REF ordi_type)

create type ensOrdi_type as TABLE of refOrdi_type
```

```
create type typeProc_type
  as object(
    marque VARCHAR2(20),
    modele VARCHAR2(20),
    frequence NUMBER(5),
    lesOrdi ensOrdi_type
  )
```

```
create type ordi_type as object(
  ...
  typeProc REF typeProc_type,
  ...)
```

Association *-* sans propriété

```
create type logiciel_type

create type reflogi_type as OBJECT (refLogi REF logiciel_type)

create type enslogi_type as TABLE of reflogi_type
```

```
create type logiciel_type
  as OBJECT(
  numero NUMBER(2),
  libelle VARCHAR2(50),
  espaceDisque NUMBER,
  installeSur ensOrdi_type
)
```

```
create type ordi_type as object(
  ...
  logiciels enslogi_type,
  ...)
```

Association *-* avec propriété (1/2)

Création d'une classe pour l'association "utilise".

```
create type utilisateur_type ;

create type periodeUtil_type as OBJECT (
  debut date, fin date,
  utilisateur REF utilisateur_type,
  ordi REF ordi_type
)

create type refperiodeUtil_type
  as OBJECT (periode REF periodeUtil_type)

create type ensPeriodeUtils_type
  as TABLE of refperiodeUtil_type;
```

Association *-* avec propriété (2/2)

```
create type adresse_type
  as object(
  batiment VARCHAR2(20),
  bureau VARCHAR2(10)
)

create type utilisateur_type
  as OBJECT (
  numero NUMBER(2),
  nom VARCHAR2(20),
  prenom VARCHAR2(20),
  adresse adresse_type,
  utilise ensPeriodeUtils_type
)
```

```
create type ordi_type as object(
  ...
  utilisePar ensPeriodeUtils_type)
  /
```

Création des tables et des contraintes (1/2)

```
create table ecran of ecran_type(
  constraint ecran_pkey primary key(numserie),
  constraint marque_ecran_non_null marque not null,
  constraint modele_ecran_non_null modele not null
);

create table typeProc of typeProc_type (
  constraint marque_proc_non_null marque not null,
  constraint modele_proc_non_null modele not null
) NESTED TABLE lesOrdi STORE AS tab_lesOrdi ;

create table logiciel of logiciel_type (
  constraint libelle_log_non_null libelle not null
) NESTED TABLE installeSur STORE AS tab_installeSur ;
```

Création des tables et des contraintes (2/2)

```
create table utilise of periodeUtil_type ;
create table utilisateur of utilisateur_type(
   constraint utilisateur_pkey primary key(numero)
) NESTED TABLE utilise STORE AS tab_utilise ;

create table ordi of ordi_type(
   constraint ordi_pkey primary key(numserie),
   constraint marque_ordi_non_null marque not null
) NESTED TABLE logiciels STORE AS tab_logiciels ,
   NESTED TABLE utilisePar STORE AS tab_utilisePar ;
```