Master Mathématiques, Informatique, Décision, Organisation (MIDO)
2ère année
Spécialités MIAGE-ID, MIAGE-IF et MIAGE-SITN et MIAGE-IF App.

ANNEE 2014 / 2015

Désignation de l'enseignement : Persistance Objet-Relationnel / Hibernate

Désignation du document : **TP JDBC**

Enseignement assuré par : Maude Manouvrier

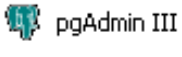
L'objectif de ce TP est de vous faire créer et manipuler manuellement une couche de persistance simple en utilisant JDBC au-dessus d'une base de données gérée par le SGDB *PostgreSQL*. La base de données sera créée et manipulée via l'interface *PgAdmin3* de *PostgreSQL*. La programmation se fera sous *Eclipse*.

Les documents et fichiers nécessaires au bon déroulement de ce TP sont disponibles à l'adresse :
http://www.lamsade.dauphine.fr/~manouvri/HIBERNATE/TP_JDBC/TP_JDBC.html
Ils sont également disponibles sur mycourse (https://mycourse.dauphine.fr/) sous le nom : M2 MIAGE ID/IF/IF-APP/SITN_2014-2015_Persistance objet-relationnel / Hibernate_Maude Manouvrier

Table des matières

1. Créer une base de données sous PostgreSQL

Tous les fichiers de scripts SQL nécessaires pour créer la base de *données sont disponibles à l'adresse : http://www.lamsade.dauphine.fr/~manouvri/HIBERNATE/TP_JDBC/TP_JDBC.html*
Ou sur mycourse (https://mycourse.dauphine.fr/) sous le nom : M2 MIAGE ID/IF/IF-APP/SITN_2014-2015_Persistance objet-relationnel / Hibernate_Maude Manouvrier

1. Créer votre base de données en allant à l'adresse : `https://manager.crio.dauphine.fr/mes-projets/`
2. Noter l'identifiant et le mot de passe indiqué une fois votre base créée.
3. Lancer  : Menu **Applications**, puis **Outils Système**, puis **PgAdmin III**.
4. Connecter vous à la base en cliquant sur le bouton représentant une prise de courant en haut à gauche. Une fenêtre apparaît.
5. Taper `postgres.dauphine.fr` dans le champ **Hôte** de la fenêtre: il s'agit de l'adresse du serveur sur lequel à été installé le SGBD PostgreSQL.
6. Taper `TP JDBC` dans le champ **Nom** : le bouton OK n'est plus grisé.
7. Taper le *login* transmis lors de la création de la base (étape 1) dans le champ **BD maintenance** : la base de données, sur laquelle vous allez travailler, a le même nom que votre nom d'utilisateur.
8. Taper le *login* transmis lors de la création de la base (étape 1) dans le champ **Nom Utilisateur** et le **Mot de Passe** transmis lors de la création de la base (étape 1) dans le champ correspondant et cliquer sur OK.
9. Cliquer sur le symbole + situé à coté du terme **Bases de Données** dans le menu à gauche, puis sur le symbole + situé à coté de la base de données correspondant à votre *nom d'utilisateur*.
10. Cliquer sur le bouton contenant le mot SQL situé dans la barre d'icônes en haut de la fenêtre de PgAdmin3 pour lancer l'interpréteur de requêtes `SQL`
11. Ouvrez le script de création nommé `Script_Creation_BDExemple.sql` (voir le contenu en annexe de ce document).

Il vous suffit de recopier le contenu du fichier dans la fenêtre du haut de l'interpréteur. Vous pouvez aussi sauvegarder le fichier dans votre répertoire et l'ouvrir dans l'interpréteur SQL.

Attention : L'interface de la version de PgAdmin3, installée au CRIO UNIX, a parfois un bug. Si vous ouvrez un fichier dans l'interpréteur SQL et que rien ne se passe, ouvrez alors les fichiers dans un éditeur de texte quelconque et copier le contenu dans l'interpréteur SQL.

- Exécuter le script SQL en cliquant sur le bouton représentant un triangle vert  en haut de l'interpréteur SQL
- De la même manière, exécuter le script d'insertion des nuplets dans les relations précédemment créées, à l'aide du script SQL nommé `Script_Insertion_BDExemple.sql` (voir le contenu en annexe de ce document).

2. Exécuter et tester le programme JDBC exemple

Le programme exemple est disponible à l'adresse : http://www.lamsade.dauphine.fr/~manouvri/HIBERNATE/TP_JDBC/

- Créer un nouveau projet Java sous Eclipse et ajouter, dans l'onglet Librairies (*Add External JARs*), le fichier `.jar` correspondant au pilote JDBC du SGBD PostgreSQL avec lequel vous allez travailler (ex. `postgresql-9.1-901.jdbc4.jar` récupérable à l'adresse http://jdbc.postgresql.org/download.html#jars).

- Récupérer le programme JDBC exemple (fichiers `TestJDBCPostgresql.java` + `ConfigConnection.java` + `FichierConnexion.txt`) et tester le sur la base exemple créée précédemment.

  Le fichier `TestJDBCPostgresql.java` est un exemple complet (il montre comment se connecter, comment exécuter des requêtes de mise à jour et de sélection). Le fichier `FichierConnexion.txt` contient les paramètres de connexion (nom du pilote et adresse de la base). Ce fichier permet de ne pas modifier le programme à chaque changement de base ou de SGBD (voir commentaire du fichier `TestJDBCPostgresql.java`).

  Le fichier `Departement.java` est un exemple de classe Java (*Active Record*) dont les objets sont persistants (i.e. sont récupérés à partir de données de la base de données ou dont les valeurs des attributs sont stockées dans la base). Le fichier `CreerDepartement.java` permet de tester cette classe.

  Le contenu de ces programmes est donné en annexe de ce document.

- Analyser les différentes instructions du programme.

3. Travail à réaliser.

- Développer en Java 3 classes métiers[1] en implémentant au minimum une association de **manière bi-directionnelle**. (NB : Il s'agit de programmation objet, par conséquent les associations doivent être représentées par des objets ou des collections d'objets – pas par des attributs de type entier !!).
- Développer des DAO permettant de gérer la persistance des objets métiers précédemment créés.
  Vous pouvez pour cela vous inspirer de la classe `Departement.java`, permettant de rendre persistant les objets de la classe *Departement* à partir des nuplets de la relation *Departement* de la base exemple (attention aucun DAO n'est implémenté dans cet exemple – il s'agit d'un *Active Record* !!).

  Rappel : faites attention à la persistance des graphes d'objets !

Merci déposer votre travail (fichiers .java (commentés) uniquement DANS UN ZIP) sur *MyCourse* et d'envoyer par sécurité une copie de votre travail à (`manouvrier@lamsade.dauphine.fr,maude.manouvrier@gmail.com`)

4. Annexe 1 : Script de création de la base de données exemple

La base de données exemple contient plusieurs relations (tables). Toutes les relations (sauf `Enseignement`, `Salle`, `Candidature` et `Presence`) possèdent des clés primaires artificielles (de type SERIAL, permettant incrémentation automatique (i.e. par la SGBD) des clés primaires, lorsqu'elles sont mono-attribut et de type entier). L'insertion de nuplets dans ces relations se fait par conséquent en utilisant une séquence (dont le nom est indiqué dans les commentaires du script ci-dessous) – voir le script d'insertion en Annexe 2 (Section 5).

Les relations `Enseignement`, `Salle`, `Candidature` et `Presence` en revanche, possède des clés primaires métier composées de deux attributs.

Le script ci-dessous correspond au script SQL de création de la base. Le texte en gras correspond aux commandes SQL. Le texte non gras précédée de `--` correspond aux commentaires.

La script de création de la relation `Note` est donné en commentaire (un commentaire commence par les caractères `--`) à titre d'information, cette relation étant créée par le programme Java `TestJDBCPostgresql.java`.

[1] **Les 3 classes à implémenter vous seront affectées lors de la séance de TP.**

```
-----------------------------------------------------------------
-- Suppression des relations si elles sont déjà créées
-- (enlever les tirets de commentaires)
-- L'ordre de suppression des relations doit être respecté
-- pour ne pas violer les contraintes d'intégrité référentielles
-----------------------------------------------------------------

--DROP TABLE Stage;
--DROP TABLE Biblio;
--DROP TABLE Affectation;
--DROP TABLE Presence;
--DROP TABLE Seminaire;
--DROP TABLE Candidature;
--DROP TABLE Reservation;
--DROP TABLE Salle;
--DROP TABLE Note;
--DROP TABLE Preference;
--DROP TABLE Inscription;
--DROP TABLE Etudiant;
--DROP TABLE Enseignement;
--DROP TABLE Master;
--DROP TABLE Enseignant;
--DROP TABLE Departement;


-----------------------------------------------------------------
-- Script de création des relations
-----------------------------------------------------------------

CREATE TABLE Departement
(
 Departement_id     SERIAL,
 Nom_Departement   varchar(25) NOT NULL,
 CONSTRAINT UN_Nom_Departement UNIQUE (nom_departement), -- contrainte d'unicité sur le nom du Département
 CONSTRAINT PK_Departement PRIMARY KEY(Departement_ID) -- Définition de la clé primaire
);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE:  CREATE TABLE will create implicit sequence "departement_departement_id_seq" for serial column
"departement.departement_id"


CREATE TABLE Enseignant
(
 Enseignant_ID     SERIAL,
 Departement_ID    integer NOT NULL, -- departement de rattachement
 Nom              varchar(25) NOT NULL,
 Prenom            varchar(25) NOT NULL,
 Grade             varchar(25)
 CONSTRAINT CK_Enseignant_Grade
 CHECK (Grade IN ('Vacataire', 'Moniteur','ATER', 'MCF', 'PROF')), -- contrainte de domaine
 Telephone         varchar(10) DEFAULT NULL,
 Fax               varchar(10) DEFAULT NULL,
 Email             varchar(100) DEFAULT NULL,
 CONSTRAINT PK_Enseignant PRIMARY KEY (Enseignant_ID), -- Définition de la clé primaire
 CONSTRAINT "FK_Enseignant_Departement_ID" FOREIGN KEY (Departement_ID) REFERENCES Departement
(Departement_ID) ON UPDATE RESTRICT ON DELETE RESTRICT -- Définition d'une clé étrangère
 );
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE: CREATE TABLE will create implicit sequence "enseignant_enseignant_id_seq" for serial column
"enseignant.enseignant_id"


CREATE TABLE Master
(
 Master_ID     SERIAL, -- clé artificielle
 Nom_Master   varchar(25) NOT NULL,
 Responsable_ID integer NOT NULL,
 Departement_ID    integer NOT NULL,
 CONSTRAINT UN_Nom_Master UNIQUE (Nom_Master), -- Contrainte d'unicité sur le nom du Master
 CONSTRAINT PK_Master PRIMARY KEY(Master_ID), -- Définition de la clé primaire
 CONSTRAINT "PK_Master_Responsable_ID" FOREIGN KEY (Responsable_ID) REFERENCES Enseignant (Enseignant_ID)
ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "FK_Master_Departement_ID" FOREIGN KEY (Departement_ID) REFERENCES Departement
(Departement_ID) ON UPDATE RESTRICT ON DELETE RESTRICT -- Définition d'une clé étrangère

);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE:  CREATE TABLE will create implicit sequence "master_master_id_seq" for serial column
"master.master_id"
```

```
CREATE TABLE Etudiant
(
 Etudiant_ID    SERIAL, -- clé artificielle
 Nom           varchar(25) NOT NULL,
 Prenom        Varchar(25) NOT NULL,
 Date_Naissance date NOT NULL,
 Adresse       Varchar(50) DEFAULT NULL,
 Ville         Varchar(25) DEFAULT NULL,
 Code_Postal   Varchar(9) DEFAULT NULL,
 Telephone      Varchar(10) DEFAULT NULL,
 Fax            Varchar(10) DEFAULT NULL,
 Email          Varchar(100) DEFAULT NULL,
 Master_ID     integer,  -- Master où est inscrit l'étudiant
 CONSTRAINT PK_Etudiant PRIMARY KEY (Etudiant_ID), -- Définition de la clé primaire
 CONSTRAINT "PK_Etudiant_Master_ID" FOREIGN KEY (Master_ID) REFERENCES Master (Master_ID) ON UPDATE
RESTRICT ON DELETE RESTRICT -- Définition d'une clé étrangère

);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE: CREATE TABLE will create implicit sequence "etudiant_etudiant_id_seq" for serial column
"etudiant.etudiant_id"


CREATE TABLE Enseignement
(
  Enseignement_ID  integer NOT NULL,
  Master_ID   integer NOT NULL,
  Intitule         varchar(60) NOT NULL,
  Description      varchar(1000),
  Obligatoire      varchar(3) NOT NULL,
  Enseignant_ID      integer NOT NULL, -- enseignant responsable de l'enseigement
  CONSTRAINT PK_Enseignement PRIMARY KEY (Enseignement_ID, Master_ID), -- Définition de la clé primaire
métier
  CONSTRAINT "PK_Enseignement_Master_ID" FOREIGN KEY (Master_ID) REFERENCES Master (Master_ID) ON UPDATE
RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
  CONSTRAINT "FK_Enseignement_Enseignant" FOREIGN KEY (Enseignant_ID) REFERENCES Enseignant
(Enseignant_ID) ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
  CONSTRAINT CK_Enseignement_Obligatoire CHECK (Obligatoire IN ('OUI', 'NON')) -- contrainte de domaine
) ;
-- Pour information : un enseignement est identifié par un numéro et le numéro du master associé - clé
métier!!!

CREATE TABLE Preference
(
 Preference_ID SERIAL, -- clé artificielle
 Etudiant_ID     integer,
 Enseignement_ID integer,
 Master_ID integer,
 Preference integer DEFAULT 0,
 CONSTRAINT PK_Preference PRIMARY KEY (Preference_ID), -- Définition de la clé primaire
 CONSTRAINT "FK_Preference_Etudiant" FOREIGN KEY (Etudiant_ID) REFERENCES Etudiant (Etudiant_ID) ON UPDATE
RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "FK_Preference_Enseignement" FOREIGN KEY (Enseignement_ID,Master_ID) REFERENCES Enseignement
(Enseignement_ID,Master_ID) ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "UN_Preference" UNIQUE (Etudiant_ID,Enseignement_ID,Master_ID), -- pour vérifier que le
triplet est unique
 CONSTRAINT CK_Preference_Preference CHECK (Preference IN (0,1,2,3,4)) -- contrainte de domaine
);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE:  CREATE TABLE will create implicit sequence "preference_preference_id_seq" for serial column
"preference.preference_id"


CREATE TABLE Inscription
(
 Inscription_ID SERIAL, -- clé artificielle
 Etudiant_ID     integer,
 Enseignement_ID integer,
 Master_ID integer,
 CONSTRAINT PK_Inscription PRIMARY KEY (Inscription_ID), -- Définition de la clé primaire
 CONSTRAINT "FK_Inscription_Etudiant" FOREIGN KEY (Etudiant_ID) REFERENCES Etudiant (Etudiant_ID) ON
UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "FK_Inscription_Enseignement" FOREIGN KEY (Enseignement_ID,Master_ID) REFERENCES Enseignement
(Enseignement_ID,Master_ID) ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "UN_Inscription" UNIQUE (Etudiant_ID,Enseignement_ID,Master_ID) -- pour vérifier que le
triplet est unique
);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE:  CREATE TABLE will create implicit sequence "inscription_inscription_id_seq" for serial column
"inscription.inscription_id"
```

```
------------------------------------------------------------
-- Relation Note créée via le programme Java
------------------------------------------------------------
--CREATE TABLE Note
--(
-- Note_ID SERIAL,
-- Etudiant_ID     integer,
-- Enseignement_ID integer,
-- Master_ID integer,
-- Note real,
-- CONSTRAINT PK_Notes PRIMARY KEY (Note_ID),
-- CONSTRAINT "FK_Notes_Etudiant" FOREIGN KEY (Etudiant_ID)
--  REFERENCES Etudiant (Etudiant_ID)
--   ON UPDATE RESTRICT ON DELETE RESTRICT,
-- CONSTRAINT "FK_Notes_Enseignement"
--  FOREIGN KEY (Enseignement_ID,Master_ID)
--   REFERENCES Enseignement (Enseignement_ID,Master_ID)
--    ON UPDATE RESTRICT ON DELETE RESTRICT
--);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE:  CREATE TABLE will create implicit sequence "note_note_id_seq" for serial column
"note.notes_id"

-- Tables utiles uniquement pour exécuter le programme exemple JDBC
CREATE TABLE Salle
(
 Batiment      varchar(1),
 Numero_Salle   varchar(10),
 Capacite      integer CHECK (Capacite >1), -- contrainte de domaine
 CONSTRAINT PK_Salle PRIMARY KEY (Batiment, Numero_Salle) -- Définition de la clé primaire métier!
);
-- Pour information : une salle est identifiée par le nom du Batiment (A, P, B, C, D) et par un numéro

CREATE TABLE Reservation
(
 Reservation_ID     SERIAL, -- clé artificielle
 Batiment           varchar(1) NOT NULL,
 Numero_Salle       varchar(10) NOT NULL,
 Enseignement_ID    integer NOT NULL,
 Master_ID     integer NOT NULL,
 Enseignant_ID      integer NOT NULL,
 Date_Resa          date NOT NULL DEFAULT CURRENT_DATE,
 Heure_Debut       time NOT NULL DEFAULT CURRENT_TIME,
 Heure_Fin         time NOT NULL DEFAULT '23:00:00',
 Nombre_Heures     integer NOT NULL,
 CONSTRAINT PK_Reservation PRIMARY KEY (Reservation_ID), -- Définition de la clé primaire
 CONSTRAINT "FK_Reservation_Salle" FOREIGN KEY (Batiment,Numero_Salle) REFERENCES Salle
(Batiment,Numero_Salle) ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "FK_Reservation_Enseignement" FOREIGN KEY (Enseignement_ID,Master_ID) REFERENCES Enseignement
(Enseignement_ID,Master_ID) ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT "FK_Reservation_Enseignant" FOREIGN KEY (Enseignant_ID) REFERENCES Enseignant (Enseignant_ID)
ON UPDATE RESTRICT ON DELETE RESTRICT, -- Définition d'une clé étrangère
 CONSTRAINT CK_Reservation_Nombre_Heures CHECK (Nombre_Heures >=1),
 CONSTRAINT CK_Reservation_HeureDebFin CHECK (Heure_Debut < Heure_Fin)
);
-- Pour information l'utilisation de SERIAL entraîne la création d'une séquence
-- NOTICE: CREATE TABLE will create implicit sequence "reservation_reservation_id_seq" for serial column
"reservation.reservation_id"

CREATE TABLE Candidature
(
Etudiant_id integer NOT NULL,
Master_id integer NOT NULL,
Preference integer NOT NULL,
Confirmee boolean DEFAULT false,
CONSTRAINT pk_Candidature PRIMARY KEY (etudiant_id, master_id), -- Définition de la clé primaire métier!
CONSTRAINT "FK_Candidature_Etudiant" FOREIGN KEY (etudiant_id) -- Définition d'une clé étrangère
  REFERENCES etudiant (etudiant_id),
CONSTRAINT "FK_Candidature_Master" FOREIGN KEY (master_id) -- Définition d'une clé étrangère
  REFERENCES master (master_id),
CONSTRAINT ck_candidature_preference CHECK (preference IN (1,2,3,4,0)), -- contrainte de domaine
CONSTRAINT un_candidature UNIQUE (etudiant_id,master_id) -- contrainte d'unicité
);
```

```
CREATE TABLE Seminaire
(
Seminaire_id SERIAL, -- clé artificielle,
Master_id integer NOT NULL,
Entreprise varchar(100) NOT NULL,
Confirmee boolean DEFAULT false,
CONSTRAINT pk_Seminaire PRIMARY KEY (seminaire_id), -- Définition de la clé primaire
CONSTRAINT "FK_Seminaire_Master" FOREIGN KEY (master_id) -- Définition d'une clé étrangère
  REFERENCES master (master_id)
);
--NOTICE:  CREATE TABLE créera des séquences implicites « seminaire_seminaire_id_seq » pour la colonne
serial «seminaire_seminaire_id »

CREATE TABLE Presence
(
Seminaire_id integer NOT NULL,
Etudiant_id int NOT NULL,
CONSTRAINT pk_presence PRIMARY KEY (seminaire_id, etudiant_id), -- Définition de la clé primaire métier!
CONSTRAINT "FK_Presence_Seminaire" FOREIGN KEY (seminaire_id) -- Définition d'une clé étrangère
  REFERENCES seminaire (seminaire_id),
CONSTRAINT "FK_Presence_Etudiant" FOREIGN KEY (etudiant_id) -- Définition d'une clé étrangère
  REFERENCES etudiant (etudiant_id)
);

CREATE TABLE Stage
(
Stage_id SERIAL, -- clé artificielle,
Etudiant_id int NOT NULL,
Descriptif varchar(100) NOT NULL,
Entreprise varchar(100) NOT NULL,
CONSTRAINT pk_Stage PRIMARY KEY (stage_id, etudiant_id), -- Définition de la clé primaire
CONSTRAINT "FK_Stage_Etudiant" FOREIGN KEY (etudiant_id) -- Définition d'une clé étrangère
  REFERENCES etudiant (etudiant_id)
);
--NOTICE:  CREATE TABLE créera des séquences implicites « stage_stage_id_seq » pour la colonne serial «
stage.stage_id »

CREATE TABLE Affectation
(
Affectation_id SERIAL, -- clé artificielle,
batiment varchar(1) NOT NULL,
numero_salle varchar(100) NOT NULL,
master_id int NOT NULL,
CONSTRAINT pk_Affectation PRIMARY KEY (affectation_id), -- Définition de la clé primaire
CONSTRAINT "FK_Affectation_Salle" FOREIGN KEY (batiment, numero_salle) -- Définition d'une clé étrangère
  REFERENCES salle (batiment, numero_salle),
CONSTRAINT "FK_Affectation_Master" FOREIGN KEY (master_id) -- Définition d'une clé étrangère
  REFERENCES master (master_id)
);
-- NOTICE:  CREATE TABLE créera des séquences implicites « affectation_affectation_id_seq » pour la
colonne serial « affectation.affectation_id »

CREATE TABLE Biblio
(
Biblio_id SERIAL, -- clé artificielle,
Enseignement_ID    integer NOT NULL,
Master_ID     integer NOT NULL,
ISBN_Livre varchar(20) NOT NULL,
CONSTRAINT pk_Biblio PRIMARY KEY (biblio_id), -- Définition de la clé primaire
CONSTRAINT "FK_Biblio_Enseignement" FOREIGN KEY (Enseignement_ID,Master_ID) REFERENCES Enseignement
(Enseignement_ID,Master_ID), -- Définition d'une clé étrangère
CONSTRAINT "FK_Biblio_Master" FOREIGN KEY (master_id) -- Définition d'une clé étrangère
  REFERENCES master (master_id)
);
```

5. Annexe 2 : Script d'insertion des nuplets

Le texte ci-dessous correspond aux commandes SQL nécessaires pour insérer des nuplets dans les relations de la base exemple :

```
INSERT INTO Departement VALUES (nextval('departement_departement_id_seq'),'MIDO');
INSERT INTO Departement VALUES (nextval('departement_departement_id_seq'),'LSO');
INSERT INTO Departement VALUES (nextval('departement_departement_id_seq'),'MSO');

INSERT INTO Enseignant VALUES(nextval('enseignant_enseignant_id_seq'),(SELECT Departement_ID FROM
Departement WHERE
Nom_Departement='MIDO'),'MANOUVRIER','Maude','MCF','4185','4091','manouvrier@lamsade.dauphine.fr');
INSERT INTO Enseignant VALUES(nextval('enseignant_enseignant_id_seq'),(SELECT Departement_ID FROM
Departement WHERE Nom_Departement='MIDO'),'OZTURK','Meltem','MCF','','4091','ozturk@lamsade.dauphine.fr');
```

```
INSERT INTO Enseignant VALUES(nextval('enseignant_enseignant_id_seq'),(SELECT Departement_ID FROM
Departement WHERE Nom_Departement='MIDO'),'MURAT','Cécile','MCF','','4091','murat@lamsade.dauphine.fr');
INSERT INTO Enseignant VALUES(nextval('enseignant_enseignant_id_seq'),(SELECT Departement_ID FROM
Departement WHERE Nom_Departement='MIDO'),'DEBECE','Gilles','Moniteur',NULL,NULL,NULL);

INSERT INTO Master VALUES (nextval('master_master_id_seq'),'ID',(SELECT Enseignant_ID FROM Enseignant
WHERE nom='OZTURK'),(SELECT Departement_ID FROM Departement WHERE Nom_Departement='MIDO'));
INSERT INTO Master VALUES (nextval('master_master_id_seq'),'MIAGE-IF', (SELECT Enseignant_ID FROM
Enseignant WHERE nom='MANOUVRIER'),(SELECT Departement_ID FROM Departement WHERE Nom_Departement='MIDO'));
INSERT INTO Master VALUES (nextval('master_master_id_seq'),'MIAGE-SITN', (SELECT Enseignant_ID FROM
Enseignant WHERE nom='MURAT'),(SELECT Departement_ID FROM Departement WHERE Nom_Departement='MIDO'));

INSERT INTO Etudiant VALUES (nextval('etudiant_etudiant_id_seq'),'DEBECE', 'Aude','1979/08/15','45, Avenue
des abeilles','PARIS','75012',NULL,NULL,NULL,(SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-IF'));
INSERT INTO Etudiant VALUES (nextval('etudiant_etudiant_id_seq'),'SUFFIT', 'Sam','1985/09/01','145,
Boulevard Raspail','PARIS','75014',NULL,NULL,NULL,(SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-
SITN'));

INSERT INTO Enseignement VALUES ('1',(SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-
IF'),'C++/C#/Python','C++ (18h)+ Python (6h) + C#(3h)','NON',(SELECT Enseignant_ID FROM Enseignant WHERE
Nom LIKE 'MANOUVRIER'));
INSERT INTO Enseignement VALUES ('2',(SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-
IF'),'Persistance ORM/HIBERNATE','TP JDBC en Java et projet Hibernate + Examen','NON',(SELECT
Enseignant_ID FROM Enseignant WHERE Nom LIKE 'MANOUVRIER'));

INSERT INTO Preference VALUES (nextval('preference_preference_id_seq'),(SELECT Etudiant_ID FROM Etudiant
WHERE nom ='SUFFIT'), 1, (SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-IF'), 4);
INSERT INTO Preference VALUES (nextval('preference_preference_id_seq'),(SELECT Etudiant_ID FROM Etudiant
WHERE nom ='SUFFIT'), 2, (SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-IF'), 0);

INSERT INTO Inscription VALUES (nextval('inscription_inscription_id_seq'),(SELECT Etudiant_ID FROM
Etudiant WHERE nom ='SUFFIT'), 1, (SELECT Master_ID FROM Master WHERE Nom_Master='MIAGE-IF'));


INSERT INTO Salle VALUES('B','020','15');
INSERT INTO Salle VALUES('B','022','15');

INSERT INTO Reservation VALUES (nextval('reservation_reservation_id_seq'),'B','022',(SELECT
Enseignement_ID FROM Enseignement WHERE Intitule LIKE '%C++%'),(SELECT Master_ID FROM Enseignement WHERE
Intitule LIKE '%C++%'),(SELECT Enseignant_ID FROM Enseignant WHERE Nom LIKE
'MANOUVRIER'),'2012/01/15','08:30:00','11:45:00','3');
INSERT INTO Reservation VALUES (nextval('reservation_reservation_id_seq'),'B','020',(SELECT
Enseignement_ID FROM Enseignement WHERE Intitule LIKE '%HIBERNATE%'),(SELECT Master_ID FROM Enseignement
WHERE Intitule LIKE '%HIBERNATE%'),(SELECT Enseignant_ID FROM Enseignant WHERE Nom LIKE
'MANOUVRIER'),'2011/11/17','13:45:00','17:00:00','3');

INSERT INTO Candidature VALUES ((SELECT Etudiant_ID FROM Etudiant WHERE nom ='SUFFIT'), (SELECT Master_ID
FROM Master WHERE Nom_Master='MIAGE-IF'),4,true);

INSERT INTO Seminaire VALUES (nextval('seminaire_seminaire_id_seq'),(SELECT Master_ID FROM Master WHERE
Nom_Master='MIAGE-IF'),'SG CIB', true);

INSERT INTO Presence VALUES ((SELECT Seminaire_ID FROM Seminaire WHERE seminaire_id=1), (SELECT
Etudiant_ID FROM Etudiant WHERE nom ='SUFFIT'));

INSERT INTO Stage VALUES (nextval('stage_stage_id_seq'),(SELECT Etudiant_ID FROM Etudiant WHERE nom
='SUFFIT'),'Stage de développement C++','SG CIB');

INSERT INTO Affectation VALUES (nextval('affectation_affectation_id_seq'),'B','022', (SELECT Master_ID
FROM Master WHERE Nom_Master='MIAGE-IF'));

INSERT INTO Biblio VALUES (nextval('biblio_biblio_id_seq'),(SELECT Enseignement_ID FROM Enseignement WHERE
Intitule LIKE '%C++%'),(SELECT Master_ID FROM Enseignement WHERE Intitule LIKE '%C++%'),'2 345 678 456');
```

6. Annexe 3 : Programme exemple JDBC

Pour exécuter un programme Java au-dessus d'un SGBD, il faut utiliser un driver. Pour utiliser PostgreSQl, le driver est : `org.postgresql.Driver`. Pour se connecter au SGBD, les paramètres sont le nom du driver, l'adresse du serveur de base de données ainsi que le login et le mot de passe de l'utilisateur.

Le fichier `FichierConnexion.txt` contient le nom du driver et l'adresse du serveur - pensez à modifier le `nom_base` par le nom de votre base (i.e. votre login).
Son contenu est le suivant :

```
driver = org.postgresql.Driver
url = jdbc:postgresql://database.etud.dauphine.fr/nom_base
```

Le fichier `TestJDBCPostgresql.java` montre comment se connecter (en faisant saisir le nom de la source de données, le login et le mot de passe par l'utilisateur (partie commentée du programme) ou à partir des informations contenues dans un fichier contenant les paramètres de connexion : `FichierConnexion.txt`), comment créer une relation via un programme JAVA, exécuter des requêtes de mise à jour et de sélection . Son contenu est le suivant :

```
/*
 * From http://www.fankhausers.com/postgresql/jdbc/#driver_download
 * and from http://deptinfo.unice.fr/~grin/mescours/minfo/bdavancees/tp/tpjdbc1/index.html
 * and adaptated by M. Manouvrier to the database example
 */

import java.sql.*;   // All we need for JDBC
import java.text.*; // To format date
import java.util.*;

public class TestJDBCPostgresql
{
  Connection       db=null;   // A connection to the database
  Statement        sql=null;  // Our statement to run queries with
  DatabaseMetaData dbmd;      // This is basically info the driver delivers
                              // about the DB it just connected to.
                              // Just used to get the DB version to confirm the
                              // connection in this example.

  public TestJDBCPostgresql(String argv[])
    throws ClassNotFoundException, SQLException, java.io.IOException
  {
    /* Put the following lines instead of the four first lines
     * of the try block in order to not use the ConfigConnection class*/
    /*String database = argv[0];
    String username = argv[1];
    String password = argv[2];
    Class.forName("org.postgresql.Driver"); //load the driver
    db = DriverManager.getConnection("jdbc:postgresql:"+database,
                                     username,
                                     password); //connect to the db */

        try {
                //Code using the ConfigConnection class
                String username = argv[0];
                String password = argv[1];
                String fichierProp = argv[2];
                db = ConfigConnection.getConnection(fichierProp,username,password);

                /* AUTOCOMMIT: If set to true, PostgreSQL will automatically do
                 * a COMMIT after each successful command that is not inside
                 * an explicit transaction block (that is, unless a BEGIN with no
                 * matching COMMIT has been given).
                 * If set to false, PostgreSQL will commit only upon receiving
                 * an explicit COMMIT command.
                 * The default is true, for compatibility with historical
                 * PostgreSQL behavior. However, for maximum compatibility
                 * with the SQL specification, set it to false.
                 * cf. http://www.postgresql.org/docs/7.3/static/runtime-config.html
                 */
                db.setAutoCommit(false);

                dbmd = db.getMetaData(); //get MetaData to confirm connection

                System.out.println("Connection to SGBD "+ dbmd.getDatabaseProductName()+ " version "+
                        dbmd.getDatabaseProductVersion()+ " database " +
                        dbmd.getURL()+ " \nusing " + dbmd.getDriverName() + " version "+
                        dbmd.getDriverVersion()+ " " + "successful.\n");

                sql = db.createStatement(); //create a statement that we can use later

                // Creation of a table Note
                String sqlText = "CREATE TABLE Note ( Note_ID SERIAL, " +
                                                "Inscription_ID integer," +
                                                "Note real," +
                                                "CONSTRAINT PK_Notes PRIMARY KEY (Note_ID)," +
                                                " CONSTRAINT FK_Notes_Inscription " +
```

```
                                        " FOREIGN KEY (Inscription_ID) " +
                                        " REFERENCES Inscription (Inscription_ID) " +
                                " );";

System.out.println("Executing this command: "+sqlText+"\n");
sql.executeUpdate(sqlText);
// Don't forget command COMMIT!!!
db.commit();

// Insertion into table Inscription
sqlText = "INSERT INTO Inscription VALUES " +
" (nextval('inscription_inscription_id_seq')," +
" (SELECT Etudiant_ID FROM Etudiant WHERE Nom='GAMOTTE')," +
" (SELECT Enseignement_ID FROM Enseignement " +
   " WHERE Intitule LIKE '%HIBERNATE%')," +
" (SELECT Departement_ID FROM Enseignement " +
   " WHERE Intitule LIKE '%HIBERNATE%'));";
System.out.println("Executing this command: "+sqlText+"\n");
sql.executeUpdate(sqlText);

// Insertion into table Note
sqlText = "INSERT INTO Note VALUES " +
" (nextval('note_note_id_seq')," +
" (SELECT Inscription_ID FROM Inscription i, Enseignement e, Etudiant et " +
  "WHERE Intitule LIKE '%HIBERNATE%' " +
  "AND e.Enseignement_ID=i.Enseignement_ID " +
  "AND et.Etudiant_ID=i.Etudiant_ID " +
  "AND et.Nom='SUFFIT' " +
  "AND e.Master_ID=i.Master_ID)," +
"14);";
System.out.println("Executing this command : "+sqlText+"\n");
sql.executeUpdate(sqlText);
sql.executeUpdate(sqlText);
// Don't forget command COMMIT!!!
db.commit();

//Update of table Salle
// SELECT before update
sqlText="SELECT capacite FROM Salle " +
                      "WHERE Batiment ='B' " +
                      "AND Numero_Salle='020' FOR UPDATE";
System.out.println("Executing this command : "+sqlText+"\n");
ResultSet rset = sql.executeQuery(sqlText);
// To print current result record before update
if (rset.next()) {
        //Column numbered from 1 (not from zero)
        System.out.println("Capacité de la salle B020 avant MàJ : "
            + rset.getInt(1) + "\n");

        // UPDATE
        sqlText = "UPDATE Salle SET capacite = 30 WHERE Batiment ='B' AND
                                                        Numero_Salle='020'";
        System.out.println("Executing this command: "+sqlText+"\n");
        sql.executeUpdate(sqlText);
        System.out.println (sql.getUpdateCount()+
          " rows were update by this statement\n");
        // Don't forget command COMMIT!!!
        db.commit();
        // SELECT after UPDATE
        rset = sql.executeQuery("SELECT Capacite FROM Salle " +
               "WHERE Batiment ='B' " +
               "AND Numero_Salle='020'");
        // To print current result record before update
        while (rset.next()) {
               System.out.println("Capacité de la salle B020 après MàJ : "
                      + rset.getInt(1) + "\n");
        }
}
else {
        db.rollback(); // free lock created by  "SELECT ... FOR UPDATE"
        System.out.println("Pas de salle B020 : " +
                      "Libération du verrou posé lors du SELECT ...FOR UPDATE. \n" );
}

rset = sql.executeQuery("SELECT Date_Resa FROM Reservation " +
                        "WHERE Batiment='B' " +
                        "AND Numero_Salle='022'");
while (rset.next()) {

        // To print meta data
        // !!There is a bug , table name is empty
```

```
                System.out.println("Nom de la table : (généralement vide bug du driver)"
                + rset.getMetaData().getTableName(1));
                System.out.println("Type de la colonne  : "
                + rset.getMetaData().getColumnTypeName(1));
                System.out.println("Nom de la colonne : "
                + rset.getMetaData().getColumnName(1) + "\n");

                // To print current result record
                System.out.println("Date de reservation du nuplet résultat: "
                    + rset.getDate(1) + "\n");

                // Example of code for managing Date
                java.sql.Date dateResa = rset.getDate(1);
                if (! rset.wasNull()) {
              String dateResaF =
              DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.FRANCE).format(dateResa);
              SimpleDateFormat formateur =
              (SimpleDateFormat)DateFormat.getDateInstance(DateFormat.DEFAULT, Locale.FRANCE);
              formateur.applyPattern("MMMM"); // configure le formateur pour avoir
                                              // le mois en lettres (avec plus de
                                              // 3 lettres (car plus de 3 M)
              String mois = formateur.format(dateResa);
              System.out.println("Affichage de la date de manière formaté:");
              System.out.println("Réservation pour le : " + dateResaF);
              System.out.println("Au mois de " + mois);
            }
            else
              System.out.println();
          }

            // Creation of a PreparedStatement object for sending
            // parameterized SQL statements to the database
            System.out.println("\n\nNow demostrating a prepared statement...");
            sqlText = "INSERT INTO Salle VALUES (?,?,?)";
            System.out.println("The Statement looks like this: "+sqlText+"\n");
            System.out.println("Looping several times filling in the fields...\n");
            PreparedStatement ps = db.prepareStatement(sqlText);

            // Execute the parameterized SQL statements
            // Seting the specified parameter to the given string value
            String [] NumBatiment = {"A", "B", "C", "P", "D"};
            String [] NumSalle = {"208", "026", "405", "340", "120"};
            int lenNB = NumBatiment.length;
            for (int i=0, c=30 ; (i<lenNB) && (c<35) ;c++,i++)
            {
                    System.out.println(i+" " + NumBatiment[i]+ " " + NumSalle[i]+ "...\n");
                    ps.setString(1,NumBatiment[i]); //set column one (Batiment) to ith string of
                                                                                    NumBatiment
                    ps.setString(2,NumSalle[i]);; //set column two (Numero_Salle) to ith string of
                                                                                    NumSalle
                    ps.setInt(3,c); //set column three (Capacite) to c
                    ps.executeUpdate();
            }
            // Don't forget command COMMIT!!!
            db.commit();
            ps.close();

            System.out.println("Now executing the command: "+
                    "SELECT * FROM Salle");
            ResultSet results = sql.executeQuery("SELECT * FROM Salle");
            System.out.println("Affichage des nuplets de la relation Salle après insertion :");
            if (results != null)
            {
                    while (results.next())
                    {
                            // To print the current result record
                            System.out.println("Batiment =" +results.getString(1)+
                                    "; Numero de salle = "+results.getString("Numero_Salle")+
                         "; Capacité = "+results.getInt("Capacite")+"\n");
                    }
            }
            results.close();

        }
    catch (SQLException ex)
     {
       System.out.println("***Exception:\n"+ex);
       ex.printStackTrace();
       System.out.println("\nROLLBACK!\n"+ex); // ROLLBACK updates
       if (db != null) db.rollback();
     }
```

```
        finally {
                  System.out.println("Deconnexion");
              if (sql != null) sql.close();
              if (db != null) db.close();
        }
  }

  public static void correctUsage()
  {
    System.out.println("\nIncorrect number of arguments.\nUsage:\n "+
                       "java   \n");
    System.exit(1);
  }

  public static void main (String args[])
  {
    if (args.length != 3) correctUsage();
    try
    {
      new TestJDBCPostgresql(args);
    }
    catch (Exception ex)
    {
      System.out.println("***Exception:\n"+ex);
      ex.printStackTrace();
    }
  }
}
```

7. Annexe 4 : Exemple d'*Active Record*

Le programme `Departement.java` implémente un *active record* correspondant à la relation `Departement` de la base exemple. Un attribut booléen permet notamment de savoir si l'objet correspond ou non à un nuplet de base de données. Seules les méthodes `save` et `delete` ont été implémentées. Les méthodes `equals` et `hashCode` ont été redéfinies.

```
/* Repris de http://www.lamsade.dauphine.fr/rigaux/mysql/ et adapté pour créer et manipuler un objet
Departement à partir d'un nuplet de la relation Departement */

import java.sql.*;


public class Departement {
    private int id;
    private String nom;
    private boolean _builtFromDB;
    private static String _query = "SELECT * FROM Departement";

    private String _update() {
            return "UPDATE Departement SET Nom_Departement='" + nom +
             "' WHERE Departement_ID=" + id;
    }

    private String _insert() {
            return "INSERT INTO Departement"
            + " VALUES(nextval('departement_departement_id_seq'), '" +
            nom + "')";
    }


    private String _delete() {
        return "DELETE FROM Departement"
        + " WHERE Departement_ID = " + id ;
    }

    public Departement() {
        _builtFromDB=false;
    }

    public Departement(String nom) {
        this.nom=nom; id=-1; _builtFromDB=false;
    }

    public String toString() {
        return "Département " + id + " : " + nom ;
      }
```

```
    public Departement(ResultSet rs) throws SQLException {
        id = rs.getInt("Departement_ID");
        nom = rs.getString("Nom_Departement");
        _builtFromDB = true;
    }

    public int getId() {return id;}

    public void setId(int id) {this.id=id;}

    public String getNom() {return nom;}

    public void setNom(String nom) {this.nom=nom;}

    public boolean equals(Object other) {
       if (this == other) return true;
       if ( !(other instanceof Departement) ) return false;
       final Departement obj = (Departement) other;
          if ( obj.getNom()!=getNom() )
            return false;
       return true;
    }

    public int hashCode(){
       return nom.hashCode() ;
       }
    public void save(Connection cx) throws SQLException {
        Statement s = cx.createStatement();
        if(_builtFromDB) {
               System.out.println("Executing this command: "+_update()+"\n");
            s.executeUpdate(_update());
        }
        else {
               // Récupération de la clé générée par la séquence

               // Code ne fonctionnant pas avec le driver JDBC de PostgreSQL
               // Mais apparemment avec celui de MySQL OUI.
            /*s.executeUpdate(_insert(), Statement.RETURN_GENERATED_KEYS);
            ResultSet r = s.getGeneratedKeys();
            while(r.next())
                id = r.getInt(1);
                */


               System.out.println("Executing this command: "+_insert()+"\n");
               s.executeUpdate(_insert());
               _builtFromDB=true;

               // Pour récupérer la clé générée sous PostgreSQL
               ResultSet rset =s.executeQuery("SELECT last_value FROM departement_departement_id_seq");
               if (rset.next()) {
                       //Column numbered from 1 (not from zero)
                       id = rset.getInt(1);
               }
        }
    }

    public void delete(Connection cx) throws SQLException {
       Statement s = cx.createStatement();
        if(_builtFromDB) {
               System.out.println("Executing this command: "+_delete()+"\n");
            s.executeUpdate(_delete());
        }
        else System.out.println("Objet non persistant!");
    }

    public String getQuery() {
        return _query;
    }
}
```

8. Annexe 5 : Exemple de programme utilisant un *Active Record*

Le fichier `CreerDepartement.java` permet de tester la classe `Departement` présentée dans l'annexe précédente.

```
/* Repris de http://www.lamsade.dauphine.fr/rigaux/mysql/ et adapté pour créer et manipuler un objet
Departement  à partir d'un nuplet de la relation Departement */


import java.sql.*;
import java.util.*;

public class CreerDepartement {

    static public void main (String[] argv)
       throws ClassNotFoundException, SQLException, java.io.IOException
    {
       Connection _cx=null;
        DatabaseMetaData dbmd;

       // Code using the ConfigConnection class
              String username = argv[0];
              String password = argv[1];
              String fichierProp = argv[2];

        try {
            // Obtention de la connexion
            _cx = ConfigConnection.getConnection(fichierProp,username,password);

            _cx.setAutoCommit(false);

            dbmd = _cx.getMetaData(); //get MetaData to confirm connection

               System.out.println("Connection to SGBD "+ dbmd.getDatabaseProductName()+ " version "+
                           dbmd.getDatabaseProductVersion()+ " database " +
                           dbmd.getURL()+ " \nusing " + dbmd.getDriverName() + " version "+
                           dbmd.getDriverVersion()+ " " + "successful.\n");

               // Insertion d'un nouveau département
            Departement d = new Departement("DEP");
            d.save(_cx);
            _cx.commit();
            System.out.println("Département créé et persistant : " + d.toString());

            // Récupération des Départements de la base
            Statement sql=null;
            sql = _cx.createStatement();

            String sqlText="SELECT * FROM Departement";
               System.out.println("Executing this command: "+sqlText);
            ResultSet rset = sql.executeQuery(sqlText);

            /*ArrayList<Departement> plante avec la version Java 1.4 du CRIO UNIX
              Donc mettre en commentaire la ligne suivante
              et enlever les commentaires de la ligne d'après */
            ArrayList<Departement> liste = new ArrayList<Departement>() ;
            //ArrayList liste = new ArrayList() ;
            while(rset.next()){
             liste.add(new Departement(rset));
            }
            rset.close();

            // Affichage
            System.out.println("\nDépartements : ");
            for(int i=0; i<liste.size(); i++){
               System.out.println(liste.get(i).toString());;
            }

            d.setNom("Dép. Education Permanente");
            d.save(_cx);
            _cx.commit();
```

```
            // Mise à jour
            sqlText=d.getQuery() + " WHERE Departement_ID=" + d.getId();
               System.out.println("Executing this command: "+sqlText);
            rset = sql.executeQuery(sqlText);
            if (rset.next()) {
               Departement d2 = new Departement(rset);
                System.out.println("Département modifié : " + d2.toString() + "\n");
                       if(d2.equals(d)) System.out.println("Attention les objets d et d2 représentent le
                                                              même nuplet!\n");
            }
            rset.close();

            // Suppression
            d.delete(_cx);
            _cx.commit();
            System.out.println("Executing this command: "+sqlText);
            rset = sql.executeQuery(sqlText);
            if (!rset.next())
                      System.out.println("Département non persistant!\n");
            rset.close();

        }
        catch (SQLException ex)
        {
          System.out.println("***Exception:\n"+ex);
          ex.printStackTrace();
          // ROLLBACK updates
          System.out.println("\nROLLBACK!\n"+ex);
          if (_cx != null) _cx.rollback();
        }

        finally {
               System.out.println("Deconnexion");
            try { if (_cx != null) _cx.close(); } catch (SQLException ex) { }
        }
    }
}
```