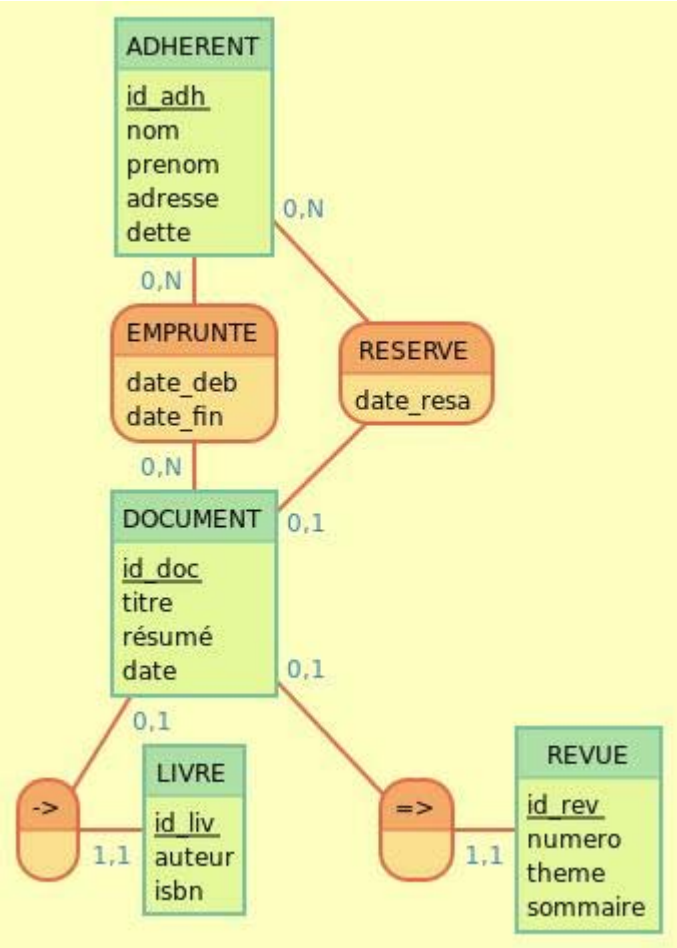
EISTI Ecole Internationale des Sciences du Traitement de l'Information	**EXAMEN DE** **BASES DE DONNEES AVANCEES** **Durée : 2 heures** **2 feuilles recto verso manuscrites sont autorisées**
Rédigé par : L'équipe pédagogique	Ref : *BDDA*
A l'intention de : Élèves ingénieurs 2ème année	Créé le : 29/11/2017

On étudie dans cette épreuve les possibilités d'amélioration d'une base de données permettant de gérer les documents, les adhérents et les emprunts dans une bibliothèque.

Vous trouverez ci-dessous le MCD de la base, construite dans un schéma Oracle nommé « biblio » précédemment créé. Vous trouverez en annexe le script SQL de création de cette base. « biblio » possède tous les droits d'administration sur l'ensemble de son schéma. On suppose que vous avez la possibilité de vous connecter en tant que « system ». Si vous devez changer d'utilisateur pour effectuer des actions, vous l'indiquerez auparavant.

En vous aidant du code fourni en annexe, répondez aux questions qui suivent.



I. Administration Base de Données (8 points)

1. Écrire les requêtes SQL pour effectuer les actions suivantes :
    a. Créer deux utilisateurs : Irma Pince (bibliothécaire, de schéma « **ipince** ») et Bastien Ende (adhérent, de schéma « **bende** »). Ils ont comme mot de passe leur nom de schéma.
    b. Créer les rôles suivants :
        i. Rôle « **bibliothecaire** » : qui peut consulter toutes les tables, ajouter/supprimer des documents, modifier les prêts et réservations.
        ii. Rôle « **adherent** » : qui peut consulter toutes les tables, ajouter/supprimer les prêts et réservations.
    c. Affecter les rôles aux utilisateurs.
2. Répondez aux questions suivantes :
    a. Est-ce que l'administrateur de la base de données peut voir les données en train d'être supprimées dans une transaction par Bastien Ende ?
    b. La bibliothécaire a ouvert deux sessions avec le même utilisateur Irma Pince. Dans la première session, elle ajoute un nouveau document de la table DOCUMENT. Est-ce que dans la deuxième session, connectée avec le même utilisateur, elle peut voir le document ajouté dans la première session ?
    c. Irma Pince peut-elle annuler partiellement une transaction ?
    d. Citez les commandes SQL qui valident automatiquement une transaction.
3. Créez les vues suivantes :
    a. Écrire une vue « **doc_libres** » qui permet de ne proposer que les documents qui ne sont ni empruntés, ni réservés.
    b. Écrire une vue « **emprunts_adh** » qui fournit, pour l'ensemble des adhérents de la bibliothèque, leur nombre de documents empruntés.

II. PL/SQL (6 points)

1. Écrire un trigger qui gère la clé primaire d'un document à chaque ajout. Il sera nécessaire pour cela de créer une séquence. Vous en écrirez donc la requête SQL correspondante auparavant.
2. Écrire un trigger qui, lors de l'ajout d'un prêt, positionne la date de rendu à un mois plus tard.
3. Écrire un trigger qui, si le nombre de documents déjà empruntés par un utilisateur est égal à 10, empêche un nouvel emprunt en générant une exception.
4. Écrire une fonction « **infoISBN** » qui prend en entrée un ISBN et retourne une chaîne contenant le titre, l'auteur et le résumé du document.

III. JDBC (6 points)

A l'aide du code source fourni en annexe (dont on ne donne que la déclaration des fonctions utiles et 2 exemples complets d'autres fonctions), réalisez les opérations suivantes :

1. Écrire une fonction « **nbDocAuteur** » qui prend en entrée un auteur et retourne le nombre de documents associés.
2. Écrire une procédure « **rendreDoc** » qui permet de rendre un livre/une revue passé en paramètre, à savoir que rendre un document, c'est supprimer l'emprunt associé.
3. Écrire une fonction « **calcPenalité** » qui prend un adhérent en paramètre et qui calcule le montant de la pénalité de retard (1€/document/semaine).
4. Écrire une procédure « **updateDette** » qui met à jour la dette de tous les adhérents de la bibliothèque.

ANNEXE : Script SQL de création de la base de données BIBLIO :

```sql
CREATE TABLE "ADHERENT" (
  "id_adh" VARCHAR(42),
  "nom" VARCHAR(42),
  "prenom" VARCHAR(42),
  "adresse" VARCHAR(42),
  "dette" VARCHAR(42),
  PRIMARY KEY ("id_adh")
);

CREATE TABLE "EMPRUNTE" (
  "id_adh" VARCHAR(42),
  "id_doc" VARCHAR(42),
  "date_deb" VARCHAR(42),
  "date_fin" VARCHAR(42),
  PRIMARY KEY ("id_adh", "id_doc")
);

CREATE TABLE "DOCUMENT" (
  "id_doc" VARCHAR(42),
  "titre" VARCHAR(42),
  "résumé" VARCHAR(42),
  "date" VARCHAR(42),
  "id_adh" VARCHAR(42),
  "date_resa" VARCHAR(42),
  PRIMARY KEY ("id_doc")
);

CREATE TABLE "LIVRE" (
  "id_liv" VARCHAR(42),
  "auteur" VARCHAR(42),
  "isbn" VARCHAR(42),
  "id_doc" VARCHAR(42),
  PRIMARY KEY ("id_liv")
);

CREATE TABLE "REVUE" (
  "id_rev" VARCHAR(42),
  "numero" VARCHAR(42),
  "theme" VARCHAR(42),
  "sommaire" VARCHAR(42),
  "id_doc" VARCHAR(42),
  PRIMARY KEY ("id_rev")
);

ALTER TABLE "EMPRUNTE" ADD FOREIGN KEY ("id_doc") REFERENCES "DOCUMENT" ("id_doc");
ALTER TABLE "EMPRUNTE" ADD FOREIGN KEY ("id_adh") REFERENCES "ADHERENT" ("id_adh");
ALTER TABLE "DOCUMENT" ADD FOREIGN KEY ("id_adh") REFERENCES "ADHERENT" ("id_adh");
ALTER TABLE "LIVRE" ADD FOREIGN KEY ("id_doc") REFERENCES "DOCUMENT" ("id_doc");
ALTER TABLE "REVUE" ADD FOREIGN KEY ("id_doc") REFERENCES "DOCUMENT" ("id_doc");
```

ANNEXE : Code source Java utile

```
public class MyDataSourceFactory {

    // Récupération d'une instance de la classe
    public static MyDataSourceFactory getInstance();

    // Recuperation du DataSouce
    public static DataSource getDataSource();
}

// On suppose le mapping de chaque table de la base de données fait sur le même modèle
// que la table Adherent suivante :
public class Adherent {

      int id_adh;
      String nom;
      String prenom;
      String adresse;
      int dette;

      public Adherent(int id_ahd, String nom, String prenom, String adresse, int
dette);
      public int getId_adh();
      public void setId_adh(int id_adh);
      public String getNom();
      public void setNom(String nom);
      public String getPrenom();
      public void setPrenom(String prenom);
      public String getAdresse();
      public void setAdresse(String adresse);
      public int getDette();
      public void setDette(int dette);
      public String toString();
      public boolean equals(Object obj);
}
```

```java
// Ci-dessous 2 exemples de fonction de CRUD
// Méthode qui insère un Adhérent
public static int addAdherent(Adherent adh) {
	int retour = 0;
	String ordreInsert = "INSERT INTO Adherent (nom, prenom, adresse, dette) VALUES
(?,?,?,?)" ;
	try (DataSource ds = MyDataSourceFactory.getDataSource());
		conn = ds.getDataSource();
		PreparedStatement insertAdhOrder = conn.prepareStatement(ordreInsert)) {

		insertAdhOrder.setString(1, adh.getNom());
		insertAdhOrder.setString(2, adh.getPrenom());
		insertAdhOrder.setString(3, adh.getAdresse());
		insertAdhOrder.setString(4, adh.getDette());
		retour = insertAdhOrder.executeUpdate();
	} catch (SQLException e) {
		System.err.println(ex);
	}
	return retour;
}

// Méthode de récupération des adhérents dans un ArrayList
public static ArrayList<Adherent> getAdherents() {
	ArrayList<Adherent> listAdherents = new ArrayList<Adherent>();
	String ordreSelect = "SELECT * FROM Adherent";
	try (DataSource ds = MyDataSourceFactory.getDataSource());
		conn = ds.getDataSource();
		Statement selectListeAdherent = conn.createStatement();
		ResultSet result = selectListeAdherent.executeQuery(ordreSelect)) {

		while (result.next()) {
			int id_ahd = result.getInt(1);
			String nom = result.getString(2);
			String prenom = result.getString(3);
			String adresse = result.getString(4);
			int dette = result.getInt(1);
			Adherent adherent = new Adherent(id_adh, nom, prenom, adresse,dette);
			listAdherents.add(adherent);
		}
		return listAdherents;
	} catch(Exception ex){
		System.err.println("Echec de lecture " + ex);
	}
	return null;
}
```