Fil rouge : séance 1
Module FIP INF 211

Conception des données de l'application

Déroulement

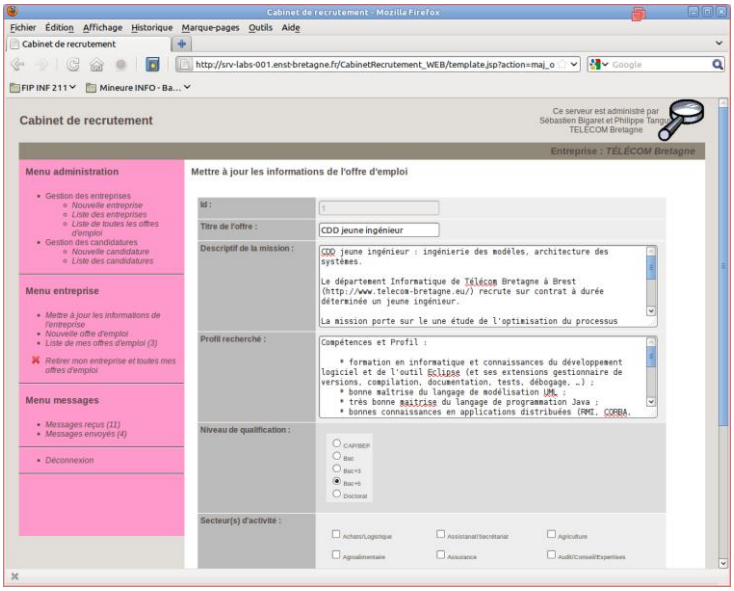
Cette première séance du fil rouge se divise en deux étapes principales :

1. La première étape vise à comprendre le sujet, à mettre en place l'environnement de travail et avoir une première vision de l'architecture de l'application. Celle-ci sera vue plus en détail au cours de l'enseignement *architectures applicatives*.
2. La seconde étape est centrée sur les données manipulées par l'application *cabinet de recrutement*, données déduites du cahier des charges. Le schéma conceptuel (diagramme de classes) des données de l'application est fourni, il faudra le dériver en un schéma logique et générer ensuite le script de création de la base de données.

Introduction : compréhension du sujet, mise en place de l'environnement de travail et présentation sommaire de l'architecture.

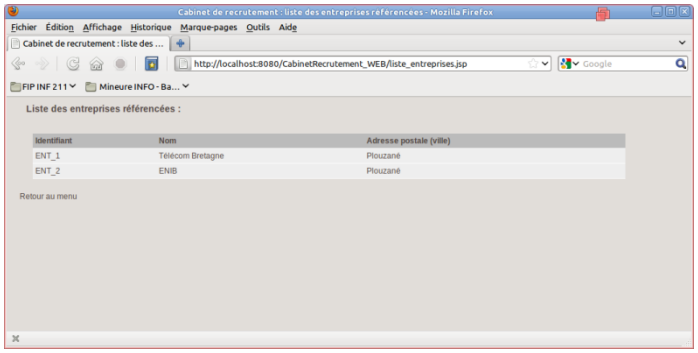
L'application

Rappel : comme énoncé dans la présentation du fil rouge, l'objectif de ce travail est essentiellement technologique. Le cahier des charges fourni présente l'intégralité des fonctionnalités à développer. En cas d'ambigüité dans la description d'une fonctionnalité, une implémentation de l'application Web est disponible. Il est possible d'y réaliser des tests : référencement d'une nouvelle entreprise, création d'une offre d'emploi, dépôt d'un CV, etc. La page 15 du cahier des charges liste les fonctionnalités par catégories et par ordre d'importance : vitale et importante (fonctionnalités obligatoires) et mineure (fonctionnalités optionnelles).



http://srv-labs-001.enst-bretagne.fr/CabinetRecrutement_WEB/

Une ébauche de l'application est déployée dans votre environnement de travail, elle implémente deux des fonctionnalités à développer : affichage de la liste des entreprises référencées et affichage des informations d'une entreprise donnée qui pourront servir d'exemple pour vos développements futurs.



http://localhost:8080/CabinetRecrutement_WEB/liste_entreprises.jsp

L'environnement de travail – les outils

L'ensemble des outils nécessaires à la réalisation du fil rouge se trouve au sein d'une machine virtuelle *VMware* dans laquelle un système *Linux Ubuntu 11.4* est installé. Pour lancer la machine virtuelle, double-cliquez sur le fichier *Ubuntu.vmx* situé dans le répertoire *Ubuntu_BD_PROG*. Une fois le système Linux lancé, connectez-vous avec l'utilisateur ayant pour login ***user*** et pour mot de passe ***password***. Cette machine virtuelle sert à la fois au fil rouge ainsi qu'aux TP de l'enseignement *architectures applicatives*.

Voir le document *description de l'environnement de travail* pour de plus amples informations sur la machine virtuelle utilisée.

Les outils à votre disposition :

EDI Eclipse	Package Eclipse utilisé : *Eclipse Java EE IDE for Web developers - Version Indigo* Eclipse est l'un des environnements de développement intégré (IDE) les plus populaires actuellement. C'est un outil professionnel largement utilisé. Il se lance à partir du bureau Ubuntu avec l'icône nommée *Eclipse - Indigo*. Cette version est adaptée au développement d'applications de type Java EE (*Enterprise Edition*). Elle comprend un ensemble d'outils additionnels adaptés à cette utilisation : gestion des projets JEE, gestion de connexions à des SGBD, pilotage de serveurs d'applications (GlassFish dans notre cas), coloration syntaxique pour les fichiers Web (HTML, CSS, etc.), les scripts SQL, etc.
Serveur d'applications GlassFish	GlassFish Server Open Source Edition 3.1.1 GlassFish est un serveur d'applications compatible Java EE 6 qui existe en deux produits : l'un open source (*GlassFish Server Open Source Edition* utilisée ici) et l'autre commercial (*Oracle GlassFish Server*) non utilisé ici. C'est au sein de ce serveur qu'est déployée notre application. Il offre différentes fonctionnalités : serveur Web, infrastructure d'exécution de composants logiciels, etc. Dans notre cas, le pilotage du serveur (démarrage/arrêt) se fera par l'intermédiaire d'Eclipse. Une fois le serveur lancé, une interface d'administration Web est disponible à l'URL : http://localhost:4848/common/index.jsf
SGBD PostgreSQL	Le SGBD PostgreSQL est un SGBD open source. La version installée est la version 8.4. Outre sa gratuité, l'un des ses avantages est sa proximité avec le SGBD Oracle concernant la syntaxe du langage SQL. PostgreSQL est livré avec un programme d'administration qui permet d'écrire et de tester des requêtes SQL : *pgAdmin3*. L'icône *PgAdmin* présente sur le bureau permet de lancer cet outil. Un autre outil est installé pour une utilisation similaire : *Aqua Data Studio*, icône *ADS 4.7*. À noter, une excellente documentation disponible en ligne à l'URL : http://docs.postgresqlfr.org/8.4/

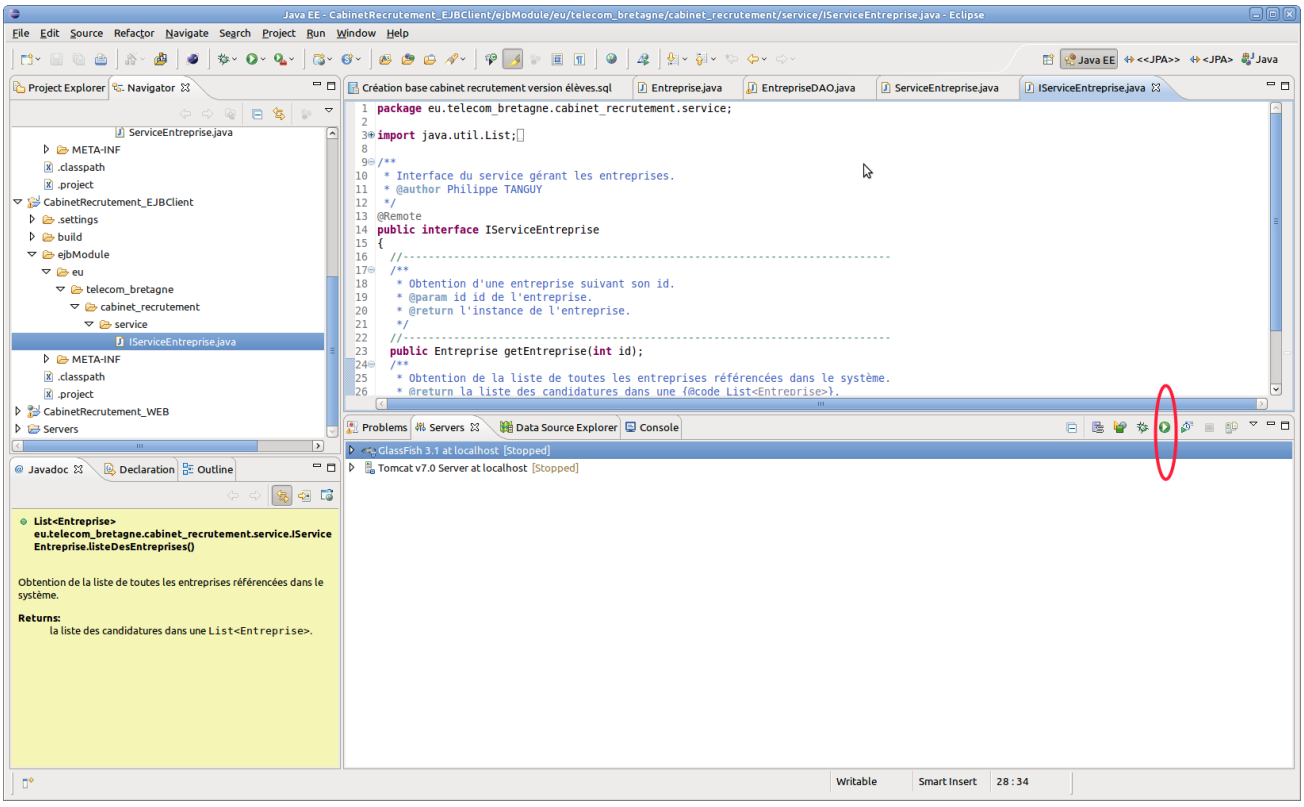
Mise en route de l'ébauche de l'application

Lancez Eclipse (icône *Eclipse - Indigo*). Le code de l'application est réparti sur quatre projets qu'il faut ouvrir séparément. Pour importer un projet dans Eclipse : menu *File* > *Import…*, choisir *Existing Projects into Workspace*, bouton *Next >*. Choisir alors le répertoire principal du projet. Les quatre projets se trouvent dans le répertoire : */home/user/workspace*. Ils se nomment :

- *CabinetRecrutement_EAR*
- *CabinetRecrutement_EJB*
- *CabinetRecrutement_EJBClient*
- *CabinetRecrutement_WEB*

L'interface d'Eclipse est divisée en plusieurs parties :

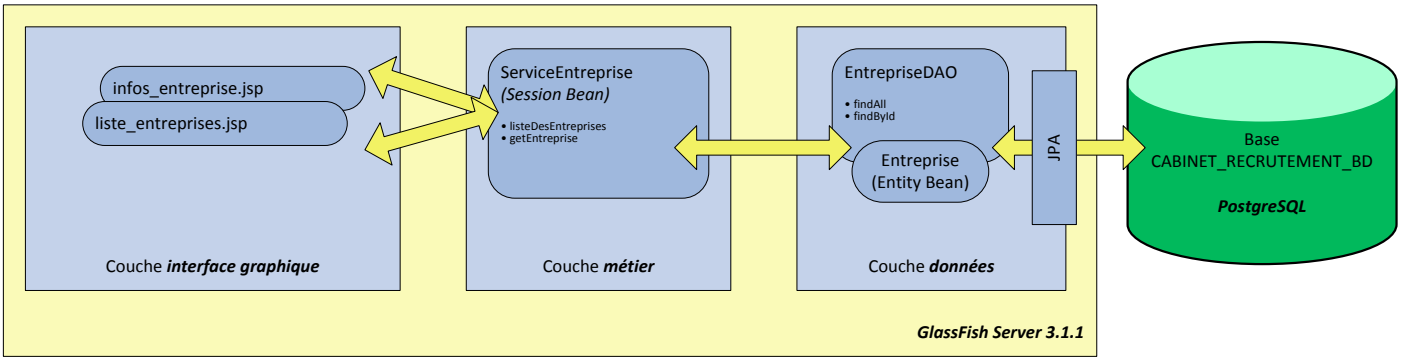
- Sur la gauche apparaît la liste des projets dans l'onglet *Navigator* ou *Project Explorer*.
- À droite, partie supérieure : la partie éditeur de code. Chaque fichier en cours d'édition s'ouvre au sein d'un onglet différent.
- À droite, partie inférieure : différents onglets parmi lesquels se situe l'onglet *Servers*, voir copie d'écran ci-dessous. C'est à partir de celui-ci que sera piloté le serveur GlassFish où est hébergée l'application. Sélectionnez le référencement du serveur (*GlassFish 3.1 at localhost*) :
    1. Ajoutez le projet au serveur : clic droit > *Add and Remove…*, l'application *CabinetRecrutement_EAR* apparaît dans la partie gauche, sélectionnez-la et cliquez sur le bouton *Add >*. L'application est maintenant référencée dans le serveur.
    2. Le lancement du serveur se fait à l'aide du bouton vert (cerclé en rouge sur la copie d'écran). Au bout de quelques secondes, le serveur est prêt : son statut passe de *[Stopped]* à *[Started, Republish]*. Il est alors possible de lancer l'application dans un navigateur Web à l'URL : http://localhost:8080/CabinetRecrutement_WEB/
    Note : il peut arriver, lors d'une mauvaise manipulation, que cet onglet soit fermé. Pour le réafficher : menu *Window* > *Show View* > *Servers*.



IDE Eclipse

Architecture de l'application

Cette ébauche de l'application met déjà en œuvre l'architecture complète que vous verrez plus en détail au cours de l'enseignement *architectures applicatives*. Cette architecture est présentée ici succinctement pour vous permettre de vous retrouver dans le code source fourni qui est réparti dans les différents sous-projets.



Architecture simplifiée de l'application cabinet de recrutement

L'application est codée au sein de deux parties :

- La base de données hébergée dans un SGBD PostgreSQL : la base *CABINET_RECRUTEMENT_BD*. Cette base est, pour le moment, simplissime. Elle ne comporte qu'une seule table qui stocke les informations des entreprises, la table *entreprise*. Le script de création de la base est disponible au sein du projet *CabinetRecrutement_EJB*, répertoire *BD*.
  L'outil pgAdmin3 permet la connexion à la base, il est alors possible de l'interroger :
  - Utilisateur : *cabinet_recrutement_role*
  - Mot de passe : *abc*

  La séance 1 du fil rouge est centrée sur la base de données.
- L'application au dessus de la base de données qui est hébergée au sein du serveur GlassFish, écrite suivant la technologie Java EE.

Cette application respecte une architecture répartie sur trois niveaux (*3-tier*) :

1. La couche ***données*** :
   Quand la couche métier a besoin de manipuler des données (lecture, écriture, effacement ou modification) elle passe par cette couche qui sert d'intermédiaire entre celle-ci et la base de données. Elle comporte plusieurs éléments :
   a. Les classes qui permettent la manipulation de nos données au sein de l'application Java. Pour le moment, seule la classe qui représente les données des entreprises est présente : la classe *Entreprise*. Chaque fois que l'application récupère un tuple (une ligne) de la table *entreprise*, les données sont représentées sous la forme d'une instance de cette classe. Dans l'architecture EJB, ces classes sont appelées des *Entity Beans*, *EJB Entities*, *classes entités*, etc.
   b. Les classes qui assurent les fonctionnalités minimales autour des classes entités (création, récupération, mise à jour, effacement) sont nommées *DAO* (*Data Access Object*). Les méthodes de ces classes sont appelées par la couche supérieure. Ce type de classe n'est représenté ici que par une seule classe : la classe *EntrepriseDAO*.
   c. La correspondance entre les classes (et leurs attributs) et les tables (et leurs colonnes) est assurée par la technologie *JPA* (*Java Persistence API*). À voir dans la classe *Entreprise*, les annotations qui permettent de gérer cette correspondance. La technologie JPA fera l'objet ultérieurement d'un TP.

   La séance 2 est centrée sur le développement de cette couche.
2. La couche ***métier***
   Au sein de cette couche se trouve toute la gestion (minimale…) de notre application : liste des entreprises et

affichage des informations détaillées d'une entreprise particulière.
Cette gestion métier de l'application est confiée à un (ou plusieurs) *Session Bean(s)*. Pour le moment, une seule classe : *ServiceEntreprise*. Cette classe (et les futures classes à venir) offre ses services (d'où son nom) à la couche supérieure (*interface graphique*) et se sert de la couche inférieure (*données*) sans jamais accéder directement à la base de données.

3. La couche ***interface graphique*** (appelée aussi couche ***IHM***)
   Cette couche est responsable de la génération des interfaces Web de l'application. Elle est réalisée à l'aide de la technologie *JSP* (*Java Server Pages*) que l'on peut comparer à la technologie PHP. Comme en PHP, les pages JSP mêlent au sein du même fichier à la fois du code HTML pour la représentation des pages au sein du navigateur Web et du code (ici du code Java : tests conditionnels, boucles, appels vers les Session Beans, etc.).
   Trois JSP existent dans l'état actuel du projet : *index.jsp*, *liste_entreprises.jsp* et *infos_entreprise.jsp*.
   Les séances 3 et 4 sont centrées sur le développement des couches métier et IHM.

L'organisation en package du code Java de l'application est calquée sur cette architecture, on trouve les trois packages qui correspondent aux trois couches :

- *eu.telecom_bretagne.cabinet_recrutement.data* : classes de la couche *données*.
  Ce package se divise lui-même en 2 sous-packages :
  - *dao* : classes DAO
  - *model* : classe entité.
- *eu.telecom_bretagne.cabinet_recrutement.service* : classes de la couche *métier*.
- *eu.telecom_bretagne.cabinet_recrutement.front* : classes nécessaires à la couche *IHM*. Ce package contient un sous-package (package *utils*) contenant des classes utilitaires nécessaires aux pages JSP de la couche *IHM*.

Cette organisation du code n'est pas obligatoire (il vous est possible de choisir une autre organisation…), elle permet cependant de s'y retrouver plus facilement dans le développement de l'application. Ce code est réparti dans l'ensemble des projets Eclipse.

Organisation des projets de développement dans Eclipse

Les développements de l'application sont organisés autour des quatre projets ouverts précédemment qui recoupent en partie (en partie seulement, ce serait trop simple…☺) l'architecture présentée ci-dessus. Les types de projets nécessaires :

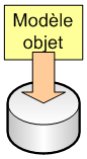
- Un projet de type *Enterprise Application Project* (***CabinetRecrutement_EAR***) :
  C'est un projet « conteneur » qui ne contient pas directement le code de l'application mais qui référence les trois autres sous-projets. Il produit, pour le déploiement de l'application un fichier *Enterprise Archive* ou *EAR* : ce type de projet s'appelle aussi *EAR Project*.
- Le codage de la logique métier de l'application se fait au sein de deux projets. Le premier est un projet type *EJB Project* (***CabinetRecrutement_EJB***) et le second est un projet automatiquement ajouté au premier lors de la création de celui-ci : projet de type *EJB Client Project* (***CabinetRecrutement_EJBClient***).
  - Le projet *CabinetRecrutement_EJB* contient le code de la couche *données* et de la couche *métier* (répertoire *ejbModule*). Il contient aussi le script de création de la base de données au sein du répertoire *BD* (fichier *Création base cabinet recrutement version élèves.sql*) à la racine du projet.
  - Le projet *CabinetRecrutement_EJBClient* contient une partie du code de la couche métier : les interfaces (au sens Java) des classes Session Beans.
- Le projet client qui utilisera la logique métier développée dans les projets précédents. Le choix opéré ici est de développer un client Web utilisant la technologie JSP qui nécessite un projet de type *Dynamic Web Project* (***CabinetRecrutement_WEB***). Les classes utilitaires sont présentes dans ce projet (répertoire *src* à la racine du projet).

Génération de la base de données

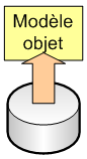
Les données persistantes de l'application connaissent des états différents :

- Elles sont stockées dans la base de données sous la forme de valeurs au sein de colonnes qui font partie d'une table.
- Quand elles sont manipulées par l'application, elles sont alors présentes sous la forme de valeurs que peuvent prendre les attributs de certaines classes (les *classes entités*).

Cette double structuration – le modèle objet du langage Java et le modèle relationnel du SGBD – coexiste mais doit rester complètement cohérent car ils sont interdépendants. Deux approches sont possibles :



- L'approche ***top-down*** (du haut vers le bas) :
  Le modèle objet existe (ou est imposé) et l'on souhaite générer modèle relationnel. Dans ce cas, le schéma de la base est réalisé à partir d'un ensemble de classes Java.



- L'approche ***bottom-up*** (du bas vers le haut) :
  Le modèle de stockage existe (ou est imposé) et l'on souhaite générer le modèle objet. L'approche est ici inverse de la précédente. Le code des classes et la correspondance avec la base (appelée aussi *mapping objet/relationnel*) est généré après examen de la base de données. Ce travail fera l'objet de la deuxième séance du fil rouge.

Dans les deux cas, des outils permettent de générer automatiquement le double du modèle. Dans le fil rouge l'approche choisie est l'approche *bottom-up*, l'approche *top-down* sera vue durant le TP JPA.

Schéma conceptuel (diagramme de classes) des données de l'application

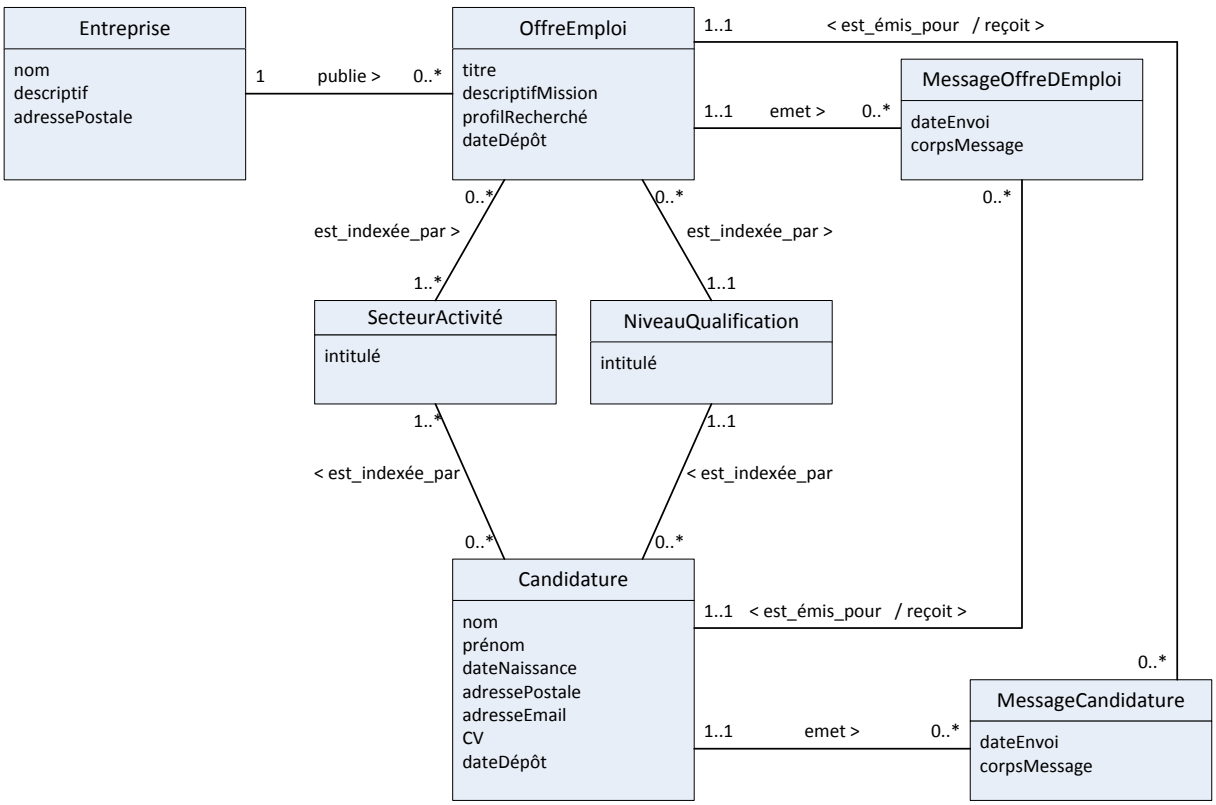


Schéma conceptuel des données de l'application

Les données de ce schéma conceptuel sont issues du cahier des charges. En cas de besoin, s'y référer pour une meilleure compréhension.

Note : les classes *MessageOffreDEmploi* et *MessageCandidature* sont liées à des fonctionnalités optionnelles, il est possible de ne pas les prendre en compte à ce stade du travail.

Travail à réaliser

Le processus à réaliser ici est similaire au travail réalisé au cours de la PC *modélisation conceptuelle et dérivation en un schéma logique*. Il se divise en deux étapes :

1. Dérivation du schéma conceptuel en un schéma logique.
2. Génération du script de création de la base de données à partir du schéma logique.
   Note : les valeurs des secteurs d'activité et des niveaux de qualification ne changent pas. Prévoir l'insertion de ces valeurs directement dans le script SQL.

Aide pour la création du schéma logique

- NOTER LES NOM ET PRÉNOM DES DEUX MEMBRES DU BINÔME SUR LE SCHÉMA.
- Il n'est pas nécessaire d'utiliser un quelconque outil informatique à ce stade, un schéma réalisé avec un crayon et une feuille de papier est amplement suffisant à condition évidemment qu'il soit propre et lisible… ☺.
- À partir des entités du diagramme de classes, identifiez les relations (tables).
- Identifiez les clés primaires de ces relations.
- Prendre en compte les associations suivant les cardinalités définies au niveau conceptuel (cas des associations 1-N, N-M, etc.). Cette prise en compte se fait à l'aide du mécanisme *clé référentielle -> clé primaire*.
- Identifier les éventuelles contraintes du diagramme de classes, impossibles à prendre en compte à ce niveau. Ces contraintes sont cependant importantes, envisager une manière possible de les prendre en compte.

Aide pour la création du script

- NOTER LES NOM ET PRÉNOM DES DEUX MEMBRES DU BINÔME DANS LE SCRIPT.
- La syntaxe de PostgreSQL est proche de la syntaxe d'Oracle. Il est possible de s'inspirer des scripts Oracle existants sur Moodle pour réaliser celui-ci. Les principaux éléments syntaxiques qui diffèrent sont donnés plus bas. Voir aussi plus haut l'URL vers la documentation en ligne de PostgreSQL.
- Une ébauche du script existe, il se situe dans le projet *CabinetRecrutement_EJB* au sein du répertoire *BD*. Quelques conseils :
  1. Il est préférable d'utiliser le script existant et de le compléter.
  2. Il est aussi préférable d'utiliser un seul fichier pour tous les objets de la base (tables, etc.) plutôt que de les répartir en plusieurs fichiers.
  3. L'utilisation d'Eclipse vous apportera plus de confort, il assure la coloration syntaxique du code SQL, ce que ne fera pas un éditeur de texte basique.
- Conservez la structure du script :
  1. Destruction des objets : ***drop table …***
     Au cours de vos différents tests, vous rejouerez plusieurs fois le script. Les tables existantes mais modifiées doivent être détruites avant d'être recréées. Cette première partie du script est là pour faire le « ménage » !
  2. Création des tables : ***create table …***
     La création de toutes les tables se situe dans cette partie. Attention, la présence d'une clé référentielle d'une table A vers une table B implique que la table B préexiste à la table A. Cet ordre est à inverser dans la destruction (partie précédente).
  3. Insertion de données de test : ***insert into …***
- L'exécution du script se fait à l'aide de l'outil *pdAdmin3* ou, meilleur choix, *Aqua Data Studio*. Une fois connecté (voir paramètres du compte), ouvrir une fenêtre de requêtes (icône SQL) et copier/coller le script à partir d'Eclipse. D'autres manières de faire sont possibles mais celle-ci est la plus simple.

Quelques éléments syntaxiques pour PostgreSQL

- Comme Oracle, PostgreSQL ne différencie pas les caractères minuscules et majuscules dans les mots clés du langage (*create*, *insert*, *into*, etc.), dans les noms des tables et dans les noms des colonnes. Par contre il fait une différence dans les valeurs textuelles :
  - *select * from entreprise* est équivalent à *SELECT * FROM ENTREPRISE*
  - *select * from candidature where nom='toto'* ne donnera pas le même résultat que *select * from candidature where nom='Toto'*
- Le séparateur des chaînes de caractères est le simple quote : '.
- La création d'une table est identique à Oracle. Exemple :

```
create table "nom_table" (
        colonne_1 type ,
        colonne_2 type ,
        …
) ;
```

- La création d'une clé primaire avec une séquence est légèrement différente : dans une syntaxe abrégée (suffisante ici) il suffit d'utiliser le type *serial* et d'ajouter la contrainte *primary key* (voir table *entreprise*). À l'exécution du script, PostgreSQL générera automatiquement une séquence qui sera nommée de la façon suivante : *nomTable_nomColonnePK_seq*. Par exemple, pour la table entreprise, la séquence se nomme *entreprise_id_seq*. Voir aussi dans le script, la partie insertion de données pour l'utilisation des séquences.
- La création d'une clé référentielle est similaire à Oracle. Dans une syntaxe abrégée (la contrainte n'est alors pas nommée), il suffit d'indiquer pour la colonne qui porte la contrainte :

  nom_colonne type_colonne [not null] references nom_table

  La partie entre crochets est optionnelle.

  exemple : *ref_entreprise integer not null references "entreprise"*

  Les doubles quotes sont optionnelles.
- Une contrainte de non nullité se fait en ajoutant *not null* après le type de la colonne. Exemple :

  nom varchar(50) not null
- Types des données utiles :
  - *serial* : type entier (*integer*) qui permet la génération automatique d'une séquence
  - *integer* : type entier sur 4 octets (largement suffisant pour la dimension de notre système d'information…
  - *varchar(n)* : chaîne de *n* caractères max
  - *text* : chaîne de caractères non bornée en taille. Typiquement le CV d'une candidature est de ce type, ce qui n'est pas le cas du nom et du prénom par exemple.
  - *date* : stockage d'une date (sans le stockage de l'heure, inutile ici). Pour le stockage de l'heure uniquement, il faut utiliser le type *time* et pour le stockage de la date avec l'heure, il faut utiliser le type *timestamp*.
    Pour insérer une date dans une colonne de ce type, il faut insérer une chaîne de caractères (entourée de simples quotes) au format *'2011-09-02'* soit le 02 septembre 2011.

Livrables

Le schéma logique (un scan de la version manuscrite est suffisant) ainsi que le script de création de la base de données seront à déposer sur Moodle avant la séance 2 du fil rouge (dépôt avant le 19 septembre). Faire une archive contenant les deux parties du livrable (au format zip, 7z, tar, bz2, etc.) qui porte IMPÉRATIVEMENT LES NOMS DES DEUX MEMBRES DU BINÔME (je me répète…).

Les modalités pour le dépôt du fichier vous seront communiquées ultérieurement.