TD

version 17 avril 2018 (TD 1a-10a)

Cahier des charges : Achat de séjours de vacances par des clients

Une agence de voyages propose à ses clients des séjours dans des villages de vacances, qu'ils peuvent choisir, et payer grâce à un avoir.

Données La base suivante décrit les informations correspondantes.

Client(idc, nom, age, avoir) Village(idv, ville, activite, prix, capacite) Sejour(ids, idc, idv, jour)

Un client a un identifiant, un nom, un âge, et un avoir (entier).

Un village de vacances a un identifiant, une ville, l'activité qu'on peut y pratiquer, un prix par jour (entier) et un nombre de lits (capacité).

Un séjour a un identifiant, un identifiant d'acheteur, un identifiant de village, et une date. Pour simplifier, chaque séjour a une durée d'un jour, et on considère que tout à lieu dans l'année courante (non bissextile), une date sera donc simplement un jour : un entier entre 1 et 365.

Exemple *Riton* a *22* ans, et un avoir de *1700* euros. Il y a un village à *Rio*, de *250* lits, coût par jour *50* euros, où l'on pratique le *kitesurf*. Le client *2* a acheté un séjour dans le centre *11* pour le *31 décembre*.

Client(1, Riton, 23, 1700) *Village(10, Rio, kitesurf, 50, 250)* *Sejour(100, 2, 11, 365)*

Précisions et restrictions Les identifiants (entiers) sont uniques. Plusieurs clients peuvent avoir le même nom. L'agence fait un cadeau de bienvenue aux nouveaux clients, qui, en s'inscrivant, se voient créer automatiquement un avoir de 2000 euros, mais ensuite les avoirs négatifs ne sont pas autorisés. Plusieurs villages peuvent être dans la même ville. Il n'y a qu'une activité par village. Un client ne peut acheter un séjour que pour lui-même, donc un seul pour une date donnée. Il peut y avoir d'autres objets SGBD dans la base (tables, contraintes, etc.), mais on ne les connaît pas.

Accès aux données Les employés ont des comptes SGBD sur la base (les logins ne sont pas les noms), et il y a un unique compte client. Un employé peut créer des villages, les consulter et les modifier (sauf identifiant, ville et prix), consulter les séjours, et effectuer le traitement 3. Un client peut effectuer les traitements 1 et 2, consulter les villages pour lesquels il n'y a aucun séjour (mais sans la capacité), ainsi que consulter les lignes lui correspondant dans chaque table à partir de son identifiant (et pour *Client* il donne aussi son nom). Rien d'autre n'est possible.

Fonctionnement Les traitements ne vérifieront pas les contraintes, ni si les paramètres sont corrects.
Traitement 1 : L'inscription d'un client est faite par lui-même comme suit. Il donne son nom et son âge, une nouvelle ligne est alors créée dans *Client*, et il obtient son identifiant. Paramètre(s) : nom et âge. Retour(s) : identifiant client.
Traitement 2 : L'achat d'un séjour est fait par un client comme suit. Il arrive avec en tête une ville et un jour. Un village dans cette ville de prix journalier le plus cher est alors cherché (car plus le prix est élevé, meilleur est le village, et l'activité n'est pas prioritaire). S'il y en plusieurs l'un quelconque est choisi (par exemple le premier qui vient). Puis le séjour correspondant est créé et l'avoir du client décrémenté du prix journalier du village. S'il n'y en a aucun on ne fait rien. Paramètre(s) : identifiant client, ville et jour. Retour(s) : identifiants village et séjour, et activité ; sinon *-1, -1, néant* si pas de village dans cette ville.
Traitement 3 : La maintenance des séjours est faite par les employés comme suit. Un employé peut détruire tous les séjours de date strictement inférieure à une date donnée. Paramètre(s) : jour. Retour(s) : le nombre de séjours détruits.
Traitement 4 : L'archivage d'un séjour est effectué automatiquement (pas par les employés ni les clients) en cas de destruction, avec l'avoir du client à ce moment. Paramètre(s) : néant. Retour(s) : néant.

TD/TP 1a

1 Mises à jour et interrogations (traitements)

1. Donnez la partie mise à jour et interrogation complète de l'application (en mode interactif). (On ne traitera pas aujourd'hui dans cet exercice les autres problèmes BD : contraintes, etc.)

   Indication : procédez comme suit.

   (a) Donnez pour cette application la liste des actions du programmeur, et la liste des actions des autres acteurs (utilisateurs, etc.).

   (b) Donnez pour chaque action ce qui suit.

      i. Le ou les ordres SQL correspondants sur un exemple, en indiquant pour chaque ordre quel acteur le tape et sur quel compte.

      ii. Généralisez en donnant en pseudo-SQL le "modèle d'ordre" correspondant, en précisant le cas échéant les "paramètres" et leurs occurences, ainsi que les retours.

   (c) Vous disposez maintenant de l'application complète en SQL. Donnez et exécutez plusieurs scénarios (exemples) de déroulement de l'application. (Mais pour simplifier aujourd'hui c'est vous qui taperez tous les ordres de tous les acteurs.)

TD/TP 2a

2 Contraintes

1. On considère les contraintes naturelles sur le schéma du cahier des charges.
   a. (moitié des points) Donnez les contraintes SQL en pseudo-SQL de préférence (sinon en SQL). (Un bonus hors barème sera attribué si la présentation en pseudo-SQL du corrigé fourni en cours et sur la page du module est suivie.)
   b. (moitié des points) Donnez les contraintes non SQL en les énonçant rigoureusement en fonction des lignes et des colonnes des tables (en français ou de la manière de votre choix, n'hésitez pas à questionner un enseignant).
   Indications : il y a environ 27 contraintes SQL (dont l'une correspond à l'ensemble de tous les *"not null"*) et 5 non SQL.
   Dans le reste de l'épreuve, on ignore cet exercice et on se replace dans la situation du cahier des charges.

   Indication : procédez comme suit (sur table).

   (a) Contraintes SQL. Pour cette question, considérez chaque colonne de chaque table, pour chaque type de contrainte SQL, et donnez à chaque fois la (ou les) contrainte(s) correspondante(s) en pseudo-SQL. Plus précisément, pour chaque colonne de chaque table, dites si c'est une clé primaire. Puis faites de même pour chaque autre contrainte parmi *fk, (not) null, unique, check*. Pour *unique*, considérez aussi tous les ensembles de colonnes à l'intérieur de chaque table (éventuellement aussi pour *pk* et *fk*).

   (b) Contraintes non SQL. Pour cette question, considérez chaque colonne de chaque table, par rapport à toutes les autres colonnes de chaque table. Enoncez rigoureusement les contraintes non SQL en français (ou formellement), en fonction des lignes et des colonnes des tables. Montrez qu'il existe une contrainte supplémentaire si on ignore le traitement 3.

2. Sur machine. Etendez votre application complète SQL de l'exercice précédent (mises à jour et interrogation) en y intégrant ce qui suit.

   (a) Les ordres SQL correspondant aux contraintes SQL ci-dessus. (La programmation de la gestion des contraintes non SQL sera vue ultérieurement.)

   (b) L'outil *séquence* dans toutes les actions à chaque fois qu'il est pertinent (ordres SQL et/ou modèles d'ordre).

   (c) Testez une violation de chaque contrainte SQL.

   (d) Vous disposez maintenant d'une nouvelle version de l'application complète en SQL, incluant les outils mises à jour, interrogation, et contraintes. Donnez et exécutez plusieurs scénarios de déroulement de l'application. (A nouveau, pour simplifier aujourd'hui c'est vous qui taperez tous les ordres de tous les acteurs.)

TD/TP 3a

Ce TD/TP utilise tout le matériel vu jusqu'à présent dans l'UE BD2. Dans chaque exercice, partez à chaque fois du modèle d'ordre correspondant (fourni dans le corrigé sur la page).

3 PL/SQL : procédures stockées, les bases

1. Ecrivez la procédure *traitement3* implantant l'action correspondante du cahier des charges.

   Indications : procédez comme suit.

   (a) Sur table :
      i. Ouvrez devant vous la version papier des exemples et des transparents du cours.
      ii. Récupérez sur la page du module dans le corrigé du TD1a (mises et jour et interrogations) le modèle d'ordre correspondant à cette action. Ce modèle d'ordre est donc en fait simplement la procédure en "pseudo-PL/SQL", et l'objet du présent exercice est de l'adapter à la syntaxe concrète PL/SQL en s'inspirant des exemples PL/SQL du cours.
      iii. Choisissez parmi : procédure, fonction, paramètres *in* et/ou *out*, types.
      iv. Identifiez les variables intermédiaires pour échange entre ordres ou retour de *select*, et résultats.
      v. Ecrivez le corps de la procédure.
      vi. Ecrivez l'appel de la procédure sur quelques exemples.

   (b) Sur machine :
      i. Lancez un client interactif.
      ii. Tapez la commande SQL*Plus *set serveroutput on*.
      iii. Tapez dans votre fichier de TD votre procédure.
      iv. Copiez-collez-la sous SQL*Plus : cela la compile et la stocke.
      v. En cas d'erreurs, consultez-les (*show errors*), choisissez uniquement la première, traduisez et comprenez le message, et corrigez-la.
      vi. Exécutez votre procédure, avec des valeurs correctes, mais aussi avec des incorrectes pour voir ce qui se passe.
      vii. (Vous procéderez de même pour les codes PL/SQL suivants du TD.)
      viii. Délogez-vous de SQL*Plus. Reconnectez-vous et la *première* chose que vous faites est l'appel de cette procédure (donc sans la recompiler/restocker). Les procédures *stockées* persistent-elles ?

2. Ecrivez à nouveau la procédure *traitement3* implantant l'action correspondante du cahier des charges, mais cette fois exceptionnellement comme une *procédure* (pas une fonction) dont la valeur de retour sera dans une variable *out*.

3. Ecrivez la procédure *traitement1* implantant l'action correspondante du cahier des charges.

4. Ecrivez la procédure *traitement4* implantant l'action correspondante du cahier des charges (même si on ne sait pas encore "qui" l'appellera ni comment) ; elle prend en paramètre une ligne de *Séjour* (représentant "la ligne détruite" indiquée dans le cahier des charges), que vous déclarerez de type *rowtype*. Appelez-la normalement, sur une ligne que vous récupérerez par un *select-into*.

5. Exécutez, en utilisant les ordres et les codes déjà implantés, un exemple de déroulement de l'application, c'est à dire, à partir de la création des tables vides, une suite réaliste des actions des acteurs.

TD/TP 4a

Ce TD/TP utilise tout le matériel vu jusqu'à présent dans l'UE BD2. Dans chaque exercice, partez à chaque fois du modèle d'ordre correspondant (fourni dans le corrigé sur la page).

4 PL/SQL : curseurs

1. Ecrivez la procédure *authentification* (action 9 des clients).
2. Ecrivez la procédure *consulter_informations* (action 9 des clients). Vous afficherez les séjours avec la forme de curseur dite "simplifiée" (*for-x-in*) ; et les villages avec la forme dite "générale" (*open-fetch-while*).
3. Ecrivez la procédure *traitement2* implantant l'action correspondante du cahier des charges.
4. Optimisation de la requête *"embedded"* jointure (sur *Village* et *Séjour*) de la procédure *consulter_informations*. Donnez les deux manières possibles d'inclure cette jointure dans la procédure PL/SQL, et expliquez pourquoi l'une est à préférer systématiquement. C'est bien sûr celle-là que vous implanterez.
5. *Entraînement personnel.* Ré-écrivez *consulter_informations*, mais avec les deux autres formes de curseur.
6. *Entraînement personnel.* Ecrivez les procédures pour les actions restantes du cahier des charges.
7. *Entraînement personnel. Récapitulatif.* Exécutez, en utilisant les ordres et les codes déjà implantés, un exemple de déroulement de l'application, c'est à dire, à partir de la création des tables vides, une suite réaliste des actions des acteurs.
8. *Travail personnel obligatoire, rappel.* Pour chaque TD/TP, lorsque vous n'avez pas fini de tester sur machine avec succès tous les exercices du TD/TP, vous devez le faire avant le cours suivant.

TD/TP 5a

Ce TD/TP utilise tout le matériel vu jusqu'à présent dans l'UE BD2. Dans chaque exercice, partez à chaque fois du modèle d'ordre correspondant (fourni dans le corrigé sur la page).

5 PL/SQL : triggers

1. Programmez le traitement 4 du CC.

   Indication : Procédez comme suit.

   (a) Ce traitement est-il effectué par un acteur ? Comment effectuer automatiquement un traitement déclenché par un type d'événement particulier ?

   (b) Appliquez la méthodologie donnée en cours, en identifiant :

      i. ce traitement nécessite-t-il la création d'une table supplémentaire ? Si oui faites-le. (Pour simplifier les tests machine, ne déclarez aucune contrainte sur cette table supplémentaire.)

      ii. tables et ordres SQL concernés, éventuellement colonnes ;

      iii. déclenchement global ou pour chaque ligne ;

      iv. condition logique supplémentaire de déclenchement : clause *when*, et éventuellement partie non exprimable par *when* ;

      v. corps du trigger.

   (c) Codez entièrement le(s) trigger(s) correspondant(s).

   (d) Testez. A partir des points ci-dessus b.ii, iii, iv, donnez des tests détaillés et effectuez-les.

2. *Synthèse : gestion des contraintes non SQL par trigger et levée d'exception*

   Dans cet exercice on considère la contrainte non SQL du cahier des charges portant sur les achats des clients. On veut la gérer en PL/SQL de sorte que toute violation provoque exactement les mêmes effets que pour une contrainte SQL.
   Codez en PL/SQL une solution complète et autonome.

   Indication : Procédez comme suit. On va programmer deux solutions : une avec un trigger niveau table, et une avec un trigger niveau ligne. Pour simplifier, on ne gère comme événement déclencheur que le cas d'insertion dans la table *Sejour*.

   (a) Rappelez ces effets d'une violation d'une contrainte SQL.

   (b) Récupérez sur la page du module le corrigé de cette contrainte non SQL.

   (c) On considère maintenant la solution par le trigger niveau table. Dans cette solution, on va exécuter une requête globale sur la base, dont le résultat sera vide si, et seulement si la contraintes est satisfaite.

      i. Donnez cette requête. Indication : faites une jointure incluant l'avoir, les séjours et leur prix, puis groupez par client et conservez (*having*) ceux dont les dépenses dépassent la borne. Testez-la dans votre client interactif.

      ii. Choisissez les bonnes clauses dans le trigger (reprenez la "méthodologie" de l'exercice 5.1.1).

      iii. Synthétisez tout cela dans une solution complète. Indication : Exécutez votre requête et voyez si elle est vide.

iv. Concevez un jeu de test systématique. Utilisez-le pour tester. Vérifiez que votre solution est correcte, en particulier concernant l'effet des mises à jour comme pour les contraintes SQL.

(d) On va maintenant adapter la solution ci-dessus en un trigger niveau ligne. Dans cette solution, au lieu d'une approche globale, on va vérifier pour chaque ligne concernée par la mise à jour si une violation de la contrainte découle de la mise à jour.

Ecrivez-le et testez-le (reprenez la "méthodologie" de l'exercice 5.1.1, et inspirez-vous en partie de la solution globale ci-dessus).

3. Généralisez la solution ci-dessus à toutes les mises à jour concernées, et pas seulement l'insertion dans *Sejour*.

(a) Identifiez les états de la base enfreignant le CC, et les ordres SQL, avec leurs tables et colonnes, susceptibles d'y mener à partir d'un état correct. Déduisez-en les cas de déclenchement. Remarquez que cette question est indépendante du choix entre niveau table et niveau ligne.

(b) *Facultatif.* Codez comme ci-dessus deux solutions, une niveau table et une niveau ligne, en adaptant vos solutions de l'Exercice 5.2.1 ci-dessus. Assurez-vous que votre solution ne duplique *aucune* partie de code ; pour cela factorisez les parties de code communes dans des fonctions comme d'habitude.

TD/TP 6a

6 Indépendance des niveaux

6.1 L'outil vue

1. *Fonctionnement.* On s'intéresse à l'action 8 du CC (consulter les villages pour lesquels il n'existe aucun séjour).
   (a) Créer une vue *vue_ village_ sans_ sejour* contenant ces villages.
   (b) Testez : interrogez les tables *Village* et *Séjour*, puis la vue : celle-ci contient-elle les bonnes lignes ?
   (c) Ajoutez un séjour à un village qui n'en avait pas, grâce au traitement 2 (il est disponible dans le corrigé du TD4a), puis réinterrogez tables et vue. Concluez sur le contenu de la vue à chaque instant par rapport aux tables dont elle est issue.

6.2 L'outil dictionnaire de données

Dans cet exercice on utilise l'outil dictionnaire de données uniquement pour interroger les méta-données, présentes dans les tables "système". (On ne considérera pas la gestion de l'indépendance du niveau logique par rapport aux mises à jour du niveau physique.)

1. *Fonctionnement.* Donnez les informations sur tous les objets utilisés dans le module que vous possédez ou auxquels vous avez accès :
   (a) autour des tables : tables, colonnes et types, contraintes, vues, etc. ;
   (b) impliquant du code PL/SQL : procédures stockées et leur code, triggers, etc ;
   (c) droits d'accès.

6.3 Indépendance des niveaux externe-logique

Dans cette section on considère l'indépendance du niveau externe par rapport au niveau logique.

1. On considère le traitement 3 du cahier des charges.
   a. Donnez en pseudo-SQL et/ou en SQL un exemple *concret* complet de problème d'indépendance des niveaux (hors renommage) pour ce traitement. (Il est *interdit* d'écrire des phrases en français : donnez simplement ce que vous tapez dans un client interactif, et ce qui est affiché d'important en réponse.)
   b. Donnez les ordres SQL *concrets* (en pseudo-SQL et/ou en SQL) pour gérer le problème concret précis que vous venez de donner. (Il est *interdit* d'écrire plus de deux ou trois phrases en français : comme dans la question ci-dessus, donnez ce que vous tapez dans le client interactif et ses réponses importantes.)
   Dans les deux questions vous indiquerez aussi la chronologie, les acteurs impliqués et les réactions concrètes du SGBD.
   (Rappel : il n'est pas demandé d'écrire une procédure PL/SQL.)

Dans le reste de l'épreuve, on ignore cet exercice et on se replace dans la situation du cahier des charges.

Indication : Procédez comme suit.

(a) Dans cet exercice, on considère comme programmes du niveau externe les actions du CC, implantées par des séquences d'ordres SQL en mode interactif. Rappelez le programme implantant le traitement ci-dessus. Rappel : On ne gérera pas l'indépendance des niveaux pour les ordres de mise à jour (cf. cours), ce que vous indiquerez simplement en commentaire dans les codes concernés. Identifiez un ordre *select* dans ce traitement.

(b) Rappel : tapez tous vos ordres dans votre unique fichier de TP.

(c) Donnez une réorganisation (hors renommage) des tables impliquées qui soit SPI par rapport à la requête, et telle que ce programme ne fonctionne plus. Procédez comme suit.

i. Donnez une base concrète avant la réorganisation (rappel : le programme fonctionne dessus).

ii. Effectuez la réorganisation et donnez la base obtenue.

iii. Montrez que la réorganisation est bien SPI en exhibant la nouvelle requête. (Rappel : si elle n'est pas SPI ce n'est pas un problème d'indépendance des niveaux.)

iv. Donnez le déroulement du programme et pointez dedans son arrêt en erreur : c'est le problème d'indépendance des niveaux.

v. Testez (on suppose que le client effectuant le traitement est votre voisin).

(d) Revenez en arrière dans le temps au début de la programmation de l'application, et programmez ce traitement pour qu'il ait la propriété d'indépendance par rapport au niveau logique. Puis gérez au cours du temps les réorganisations SPI lorsqu'elles surviennent. Procédez comme suit.

i. Créez une vue adéquate et utilisez-là. Déroulez l'application.

ii. Effectuez (sans prévenir votre voisin) la réorganisation ci-dessus et gérez le problème qu'elle posait, en adaptant la vue de manière adéquate.

iii. Testez.

(e) Approfondissement.

i. Donnez d'autres réorganisations SPI pour cet ordre *select* créant un problème d'indépendance des niveaux (mais sans justifier, et ne les gérez pas).

ii. Donnez d'autres réorganisations SPI mais ne créant pas de problème d'indépendance des niveaux.

iii. Donnez plusieurs réorganisations non SPI créant un problème ; est-ce un problème d'indépendance des niveaux ?

iv. Donnez un exemple concret complet de problème d'indépendance des niveaux pour l'ordre *udpate* de ce traitement (même si on ne le gérera pas).

(f) *Entraînement personnel.* On considère maintenant l'application complète, c'est à dire les autres actions. Gérez pour chacune l'indépendance des niveaux.

2. *PL/SQL.* On reprend exactement l'énoncé ci-dessus, mais cette fois on considère comme programmes du niveau externe les actions du CC bien sûr, mais implantées maintenant par des procédures PL/SQL.

Indication : Procédez comme suit.

(a) Reprenez la totalité des questions en indication de l'exercice précédent.

(b) Les problèmes sont-ils différents ? La solution est-elle différente ? Concluez : l'indépendance des niveaux dépend-elle de l'environnement (mode interactif, PL/SQL) contenant les ordres SQL ?

TD/TP 7a

Ce TD/TP utilise tout le matériel vu jusqu'à présent dans l'UE BD2. Dans chaque exercice, partez à chaque fois du modèle d'ordre correspondant (fourni dans le corrigé sur la page).

7 Confidentialité

1. *Manipulation des outils.* Travaillez avec votre voisin : vous êtes le programmeur, votre voisin est un employé.

   (a) *Outils SQL : comptes utilisateurs et grants sur tables.* Donnez à votre voisin le droit d'insérer dans votre table *Village* (action 1 du CC). Testez ensemble, avant et après. Retirez-lui ce droit. Testez à nouveau.

   (b) *Encapsulation en PL/SQL.*

      i. *Outil SQL : grant procedure.* Donnez à votre voisin le droit d'exécuter votre procédure *traitement2*. Testez avant et après.

      ii. *Encapsulation.* A qui appartiennent les tables qui ont été modifiées ? Votre voisin a-t-il le droit d'effectuer les ordres SQL utilisés dans le code de votre procédure ? A-t-il accès aux tables correspondantes ? La table *Séjour* est-elle encore encapsulée si vous lui donnez aussi les droits en insertion dessus ?

2. *Application de l'outil grant sur table au cahier des charges.*
   Dans cet exercice, on suppose qu'on n'a pas de procédures PL/SQL, et on revient donc au matériel du Bloc 1a (ordres SQL).
   Pour chaque action du cahier des charges (décrite par son modèle d'ordre), le programmeur donne aux acteurs correspondants les autorisations minimales leur permettant d'effectuer cette action. (Pour simplifier dans cet exercice on ne cherchera pas à vérifier si ces "autorisations minimales" ne donnent pas le droit de faire des choses interdites, mais ce point sera vu dans un prochain exercice.)

3. *Synthèse : vues, grants sur tables, dictionnaire de données, encapsulation et expressivité.*

   On se place à un moment quelconque de la vie de l'application, pour lequel on ne sait pas quels sont les objets présents dans la base (sauf les tables dont on sait qu'elles existent) ni les droits d'accès en cours. On veut maintenant faire respecter exactement les droits d'accès du cahier des charges. Mettez en œuvre cette politique (en pseudo-SQL et/ou pseudo-PL/SQL), mais avec l'outil le plus simple possible (le moins puissant) à chaque fois, et de la manière la plus précise et la plus efficace, et en indiquant quel acteur du cahier des charges tape quel ordre : consultez les droits existants, créez éventuellement des objets nécessaires, retirez et/ou donnez exactement les droits nécessaires. Si vous avez besoin d'un traitement en PL/SQL, indiquez sans ambiguïté en pseudo-code son résultat, mais il n'est pas demandé de détailler plus ; s'il s'agit de l'un des traitements 1 à 4 du cahier des charges, pour son contenu donnez simplement son numéro.
   Dans le reste de l'épreuve on ignore cet exercice et on se replace dans la situation du cahier des charges.

   Indication : Procédez comme suit. Afin d'arriver exactement aux droits souhaités, pour simplifier on va retirer tous les droits sur l'application du CC, pour remettre exactement ceux souhaités. On utilisera pour cela chaque outil en commençant avec le moins puissant. (Rappel : Les autorisations d'action exprimables avec les *grant sur table* ont déjà été effectuées au TD précédent.)

(a) Rappelez la liste complète des actions du CC avec leur acteur : ordres SQl, détails, etc. (utilisez le corrigé de l'exercice sur les mises à jour).

(b) Donnez les comptes utilisateurs SGBD adaptés à l'application décrite par le CC.

(c) En pseudo-SLQ, identifiez exactement les droits existants concernant les acteurs et les objets du CC (tables, procédurs, etc.), hors programmeur ; puis retirez-les. Attention : n'en retirez pas d'autre. Rappel : gérez bien *public*. (Remarque : Il serait possible d'obtenir le même résultat avec des ordres SQL moins précis, mais pour des raisons pédagogiques on ne le fait pas.)

(d) A ce stade, les acteurs du CC n'ont aucun droit sur les objets du CC, sauf le programmeur (qui possède ces objets). Pour chaque action du CC, essayez d'implanter *exactement* les droits du CC avec les outils : comptes utilisateurs, *grant* sur les tables, en pseudo-SQL. Plus précisément, procédez comme suit.

   i. Donnez le (ou les) *grant* nécessaire pour cette action. L'action est-elle possible ? (Montrez d'ailleurs qu'il est toujours possible de cette manière d'autoriser toutes les actions du CC.)

   ii. Donnez si c'est possible une action concrète rendue possible par ce *grant* mais interdite par le CC.

   iii. Concluez pour cette action : soit l'outil *grant* ne permet que des actions du CC (on dit que l'autorisation de cette action est exprimable dans ce sous-langage), soit il permet aussi une action interdite, et l'outil *grant* ne permet pas de programmer le CC.

(e) Pour chaque action du CC qui n'a pas pu être programmée correctement avec l'outil : *grant* sur les tables, essayez d'implanter *exactement* les droits du CC avec en plus les vues, utilisées comme outil de gestion de la confidentialité (cf. cours). Plus précisément, procédez comme suit.

   i. Rappelez pourquoi dans notre cadre une vue ne peut servir à gérer une action du CC que si celle-ci est composée d'un unique ordre, qui est un *select* sans paramètres. Identifiez ces actions du CC. Pour chacune de ces actions, programmez la gestion de la confidentialité avec une vue comme suit.

   ii. Créez la vue. Donnez dessus les droits adéquats. L'action du CC est-elle rendue possible ?

   iii. Donnez si c'est possible une action concrète rendue possible par cette vue mais interdite par le CC.

   iv. Concluez pour cette action si l'outil vue permet ou non de programmer le CC.

(f) Pour chaque action du CC qui n'a pas pu être programmée correctement avec les outils : *grant* sur les tables et vues, essayez d'implanter *exactement* les droits du CC avec en plus l'encapsulation par une procédure PL/SQL. Plus précisément, procédez comme suit.

   i. Prenez l'un des traitements 1 à 3 ou créez une nouvelle procédure PL/SQL. Donnez dessus les droits adéquats. L'action du CC est-elle rendue possible ?

   ii. Donnez si c'est possible une action concrète rendue possible par cette procédure mais interdite par le CC.

   iii. Concluez pour cette action si l'outil encapsulation par des procédures PL/SQL permet ou non de programmer le CC. Montrez d'ailleurs qu'il est toujours possible de programmer la confidentialité d'une action du (et donc de tout le CC) avec uniquement l'encapsulation (sans les *grants* sur table ni les vues). L'encpasulation permet donc de simuler dans ce contexte les vues et les *grant* sur table, comme les vues permettent de simuler les *grant select* sur table. (Dans le cours on utilise les trois outils, mais c'est pour bien comprendre les possibilités et limitations de chaque outil.)

(g) *Magistère et double-licence.* Précisez pour chaque action la preuve de si elle est exprimable ou non par les *grants sur tables*.

(h) *Récapitulatif et déroulement global de l'application.*
Le programmeur, après avoir retiré précisément les autorisations existantes concernant les objets du CC, a programmé pour chaque action sa confidentialité avec l'outil le moins puissant possible à chaque fois : *grant* sur les tables, vues, encapsulation. Il a donc programmé exactement la confidentialité du CC.

i. Groupez-vous maintenant par trois pour programmer et utiliser l'application : un programmeur, un employé et un client. Chacun tape toutes les actions lui correspondant. Procédez par ordre "chronologique" de l'application.

(i) *Facultatif. Jeu de la confidentialité.* Ce jeu se joue à trois : un défenseur (le programmeur) et deux attaquants (client et employé). Le programmeur donne sur ses données et traitements des droits aux deux attaquants en fonction du cahier des charges (prenez par exemple *Bilbiothèque* ou les annales). Puis la partie commence. Un attaquant marque un point s'il ne peut pas effectuer une action du cahier des charges (donc erreur du programmeur), ou effectuer une action qui n'y est pas (encore erreur du programmeur). Les deux attaquants peuvent discuter ensemble. Le défenseur gagne si la somme des points marqués par les deux défenseurs est inférieure ou égale à 3. Jouez trois parties en permutant les rôles (chacun devra avoir joué chaque rôle). Remarquez qu'il s'agit *exactement* du présent exercice.

TD/TP 8a

8 Rappels et précisions Internet, HTML, PHP

8.1 HTML : pages statiques et formulaire de saisie avec action déclenchement de "programme HTML"

1. Ecrivez dans un fichier *ex1.html* sur les machines du PUIO un programme (une page) HTML comme suit.

   - Couleur de fond de page au choix (sauf blanc).
   - Votre prénom et votre surnom en grands caractères (taille au choix).
   - Une liste non ordonnée avec vos hobbies.

   Exécutez ce programme dans votre navigateur. (Demandez à votre grand-mère de Rio ou NY de faire de même ; sinon faites-le vous-même depuis l'extérieur du PUIO après la fin du TD.)

   Indication : Inspirez-vous fortement des exemples du cours, et suivez rigoureusement les étapes vues en cours du fichier *Environnement* de la page du cours indiquant comment créer et exécuter une page web.

2. Ecrivez dans un fichier *menu.html* un programme HTML contenant un formulaire comme suit.

   - Il permettra de saisir un "entier".
   - Le nom de variable associé à la chaîne saisie sera *salaire*.
   - L'action dans le formulaire consistera à appeler votre programme HTML *ex1.html* de l'exercice précédent.
   - (Remarque : Comme on l'a vu en cours, un programme HTML appelé par l'action d'un formulaire n'utilise pas la variable passée en paramètre, mais un programme PHP si, comme on le verra au prochain bloc.)

   Testez.

3. Ajoutez dans *menu.html* un deuxième formulaire saisissant un entier et une chaîne et appelant un fichier texte *non HTML ex2.txt* au choix très simple que vous écrirez.

8.2 PHP

1. *Création et exécution d'un programme PHP.*

   (a) Ecrivezdans un fichier *ex1.php* un programme PHP prenant en paramètre une variable n contenant un entier et renvoyant un fichier éphémère HTML contenant ce qui suit : couleur au choix, messages au choix, n, le carré de n.

   (b) Appelez *ex1.php* sans passer par un formulaire.

   (c) Appelez *ex1.php* au moyen d'un formulaire que vous insérerez dans votre menu HTML du dernier TD.

2. Saisissez deux entiers et affichez dans une table HTML tous les entiers entre les deux (toujours dans votre menu, et de même dans le prochain TD).

TD/TP 9a

9 Accès au SGBD en mode programme : PHP (1/2) : outils

1. *Structure du TD.*

   (a) Chaque exercice du TD consistera ci-dessous à écrire un programme PHP effectuant une certaine action du cahier des charges. Pour chacun vous ajouterez un nouveau formulaire dans votre menu HTML du dernier TD. Il contiendra la lecture des éventuels paramètres de l'action, puis appellera sur ces paramètres le fichier *.php* contenant le programme PHP implantant l'action correspondante, qui affichera les valeurs de retour. Pour ne pas conserver une connexion ouverte, vous effectuerez une nouvelle connexion en début de programme juste avant chaque action, et vous la refermerez immédiatement après dans le même programme. Vous n'écrirez aucune fonction PL/SQL, mais vous appellerez si c'est demandé dans l'énoncé celles des TDs précédents.

   (b) Pour simplifier vous ne gérerez pas les erreurs.

   (c) Agrémentez vos pages HTML et PHP de quelques balises HTML en vous inspirant des exemples du cours, ou autres.

   (d) Testez bien sûr chaque action, y compris en vérifiant sous un client interactif.

2. *Pb MP : interface, connexion et exécution d'ordre SQL (non curseur).*
   Ecrivez un programme PHP qui effectue l'action des employés : *traitement 3* (n'appelez pas la procédure PL/SQL correspondante).
   Indication : Pour simplifier, codez en dur dans votre programme les paramètres de connexion. Vous utiliserez la fonction PHP *oci_num_rows($ordre)*, qui renvoie le nombre de lignes traitées par l'ordre exécuté *$ordre* pour *insert, update, delete*.

3. *Pb MP : exécution d'ordre SQL : curseur (1/2).* Ecrivez le programme PHP effectuant l'action des clients : authentification (paramètre : identifiant et nom de ce client), (mais attention, sans appeler la procédure PL/SQL correspondante). Rappel : comme en examen, il est interdit dans ce cas d'utiliser *rownum, limit 1, count, etc.* ou tout autre équivalent.

4. *Pb MP : exécution d'ordre BD non SQL : appel de procédure stockée (1/2).*

   (a) Ecrivez le programme qui appelle la *fonction* PL/SQL *traitement3*.

   (b) Ecrivez le programme qui appelle la *procédure* PL/SQL *traitement3_out*. Rappel : il est *interdit* de lier les paramètres *in* d'un ordre BD par *ocibindbyname*, qui est reservé dans le module BD2 aux paramètres *out*. Les paramètres *in* doivent *obligatoirement* être concaténés.

5. *Pb MP : curseur (2/2).* Ecrivez le programme qui effectue l'action 10 des clients : afficher ses villages à partir de l'identifiant du client (mais attention, sans appeler la procédure PL/SQL correspondante). (Partez du modèle d'ordre et de la procédure PL/SQL du corrigé du TD 4a.)

6. *Gestion des séquences SQL.* Ecrivez le programme qui effectue l'action des clients : traitement 1 (mais attention, sans appeler la procédure PL/SQL correspondante). (Partez du modèle d'ordre et de la procédure PL/SQL du corrigé du TD 4a.)

TD/TP 10a

10 Accès au SGBD en Mode Programme : PHP (2/2) : application

Comme dans le TD 9a, chaque exercice dans le présent TD 10a fera l'objet d'un formulaire dans votre menu appelant un nouveau fichier *.php*.

1. Les étudiants n'ayant pas fini les exercices du TD 9a doivent les faire en priorité car les outils correspondants sont considérés acquis pour le présent TD 10a.
2. Ecrivez le programme qui appelle la *procédure* PL/SQL *traitement2*.
3. Ecrivez le programme effectuant l'action des clients : traitement 2 (mais attention, sans appeler la procédure PL/SQL correspondante). (Partez du modèle d'ordre et de la procédure PL/SQL du corrigé du TD 4a.)
4. *Entraînement personnel.* Ecrivez le programme qui appelle la *fonction* PL/SQL *traitement1*.