L3/L'3 Janvier 2016

**Projet de Bases de données II**

Application de gestion de notes d'une école

# Instructions générales

Une école a sollicité votre groupe de projet pour lui réaliser une application de gestion du suivi des notes de ses élèves.

**Vous pouvez vous aider :**

* Du support de cours (incontournable)
* Des ressources disponibles sur Internet, notamment sur le site d'Oracle
* Outils AGL pour la conception de votre base de données (Exemple : AnalyseSI ou NextObjetcs, PowerAMC, Mega, ...)

**Critères d'évaluation**

* Ergonomie (vous avez carte blanche sur le choix du *look and feel*). Mais vos schémas doivent être visuellement agréables.
* Qualité du code (commentaires, operateurs SQL utilisés)

**Etapes de mise en œuvre**

* Avant tout, imprégnez-vous des exigences du cahier des charges : rédigez un mini document de conception technique avec le dictionnaire de données, le MCD, MLD et/ou MPD, diagramme de classes, ....

# Cahier des charges

## I – Introduction

Votre base de données est destinée à la direction des études d’une école d’ingénieurs pour mieux gérer les notes et jurys de ses différentes promotions et formations.

Elle permettra deux types d'actions :

* La gestion administrative simple des élèves
* La gestion académique des élèves

## II – Fonctionnalités techniques

* Un couplage faible avec le SGBD est impératif. Ceci permettra entre autres d'inter-changer de SGBD avec un minimum de changements et de paramétrage.
  1. Libre à vous de choisir l'un des SGBD suivants pour implémenter votre base. Pour tout autre SGBD, bien vouloir nous prévenir à l'avance : Oracle, MySQL, PostgreSQL, SQLServer, Apache Derby (alias Cloudscape ou Java DB), HSQLDB ou Access
* Gestion des droits d'accès aux différentes catégories d'utilisateurs ainsi que la gestion des accès concurrents.

## III – Gestion administrative des élèves

* **Général :**
* Matricule (4 chiffres représentant l’année de recrutement suivi de 4 chiffres représentant le numéro d’ordre lors du recrutement par année et il doit être non modifiable et si possible auto-incrémenté)
* Nom (champs obligatoire)
* Prénom (champs obligatoire)
* **Identité :**
* Date de naissance (champs obligatoire)
* Sexe
* Date d'inscription
* Etablissement précédent
* Photo
* **Coordonnées**
* N° et Nom de rue
* Code postal
* Ville
* Tél domicile
* Tél mobile (champs obligatoire)
* **Personnes(s) responsable(s)**

Prévoir **un** contact.

* Nom (champs obligatoire)
* Prénom
* Adresse
* Téléphone (champs obligatoire)
* Email
* **Santé**
* Médecin traitant (Nom et Prénom)
* Téléphone du médecin traitant
* Vaccinations
* Allergies
* Remarques médicales
* **Documents**

Possibilité de modifier et d'imprimer chacun de ces documents

* Convocation des parents
* Bulletin de notes (par élève)

## IV – Gestion académique des élèves

Attention on gère uniquement l’année scolaire en cours !

* Gestion des groupes d’élèves (création d’un nouveau groupe et mise à jour des élèves du groupe)
* Gestion des cours (création d’un cours, association d’un cours à un groupe et choix des professeurs du cours, renseignements sur le coefficient de la matière et des notes de la matière qui peuvent être DE, TP ou PRJ). Un cours est suivi par tous les élèves du groupe associé.
* Gestion des notes qui sont saisies par le professeur responsable du cours uniquement et peuvent être mises à jour uniquement par les personnes de la scolarité sur demande du professeur responsable.
* Génération de relevé de notes par élève.
* Login avec trois niveaux de privilèges : **administrateur, professeur** et **élève**.
  + L’administrateur aura des droits complets de mise à jour
  + Les professeurs peuvent mettre à jour les notes de leurs matières tant que le bulletin de notes n’a pas été édité et ils peuvent consulter les notes des élèves dont ils sont tuteur.
  + L’élève ne pourra utiliser l’application que pour consulter ses notes et imprimer son bulletin de notes une fois validé par l’administrateur.

## V – Fonctionnalités bonus

Ce défi rapportera **+3 pts sur la note finale** aux membres du groupe

Seule condition : **Toutes** les fonctionnalités précédentes doivent être mises en œuvre.

* Application C++, Java ou PHP utilisant votre base de données avec la mise en œuvre d’une gestion transparente et optimale des accès concurrents au SGBD.
* Automatisation de certains traitements en procédures stockées.

# Travail à realiser :

**1/ Séance 1 :**

1. utiliser un AGL(Atelier de Génie Logiciel) pour élaborer votre schéma conceptuel de données (MCD).
2. générer le schéma logique puis le schéma physique de votre base de données.
3. générer le code SQL pour la création de votre base de données .
4. Intégrer à votre base de données toutes les contraintes d'intégrité nécessaires pour le bon fonctionnement de votre application
5. proposer un jeu d'essai en remplissant quelques tables pour valider votre bonne conception.

**2/ Séances 2 et 3 (Gestion des vues, droits d'accès, transactions et concurrence sur le schéma fourni en Annexe-1)**

Considérons la base de données de gestion de notes d’étudiants dont le script de creation et d'instanciation est fourni en Annexe I.

Etudiant (**NumEtu**, Nom, Prenom, DateNaiss, Rue, CP, Ville)

Matiere (**CodeMat**, Libelle, Coef)

Epreuve (**NumEpreuve**, DateEpreuve, Lieu, #CodeMat)

Notation (**#NumEtu, #NumEpreuve,** Note)

**I – Gestion des vues**

1. Créer la vue renfermant tous les étudiants ayant eu des épreuves en Informatique ainsi que les notes obtenues.

2. Donner la moyenne et le nombre d’épreuves en informatique de chaque étudiant ayant passé au moins une épreuve dans cette matière.

3. Donner le nombre d’étudiants ayant eu au moins une moyenne de 10 en Informatique

4. Donner les noms d’étudiants ainsi que leur moyenne ayant eu une moyenne supérieure ou égale à 10 en Informatique et classés par ordre de mérite.

5. Donner les noms d’étudiants qui n’ont passé aucune épreuve en Informatique en utilisant les sous requêtes puis la jointure externe.

**II – Gestion de la confidentialité (Vues et Droits d’accès)**

1. Créer la vue EtudiantsLyonnais renfermant tous les étudiants lyonnais.

2. Donner la moyenne en informatique des étudiants lyonnais

3. L’étudiant Dupond a quitté l’établissement. Est-il possible de le supprimer depuis la vue EtudiantsLyonnais? Vérifier ?

4. L’étudiant Durand déménage lors de son stage à 1, Rue de Lyon Paris 75002. Est-il possible de mettre à jour ses informations depuis la vue EtudiantsLyonnais ? Vérifier ?

5. Est-il possible d’ajouter l’étudiant (700, Bill, Gates, , 01-09-1980, Rue de Paris, 69005, Lyon) ? Vérifier ?

6. Donner la vue MoyenneLyonnais contenant la moyenne par matière des étudiants lyonnais.

7. Est-il possible de mettre à jour une note de l’étudiant Dupont ?

8. Donnez un droit d’accès en consultation à la vue EtudiantsLyonnais à votre binôme. Vérifier le contenu de la vue depuis les 2 comptes.

9. Mettez à jour le nom de rue de Dupond qui passe à Rue de la Meuse, 69008, Lyon. Vérifier sur les 2 comptes si la mise à jour est faite. Le second compte essaye de remettre Dupond à son ancienne adresse. Que se passe –t-il ? Quoi faire ?

10. Votre binôme insère un nouvel étudiant lyonnais. Que doit-il faire pour que vous puissiez consulter ce nouvel étudiant dans la base de données ?

11. Une erreur de saisie entre 2 notes de 2 étudiants sur une même épreuve doit être corrigée. Comment procéder ?

12. Que doit-on faire pour éviter des mises à jour erronées des notes des étudiants par vous ou votre binôme (conflit de mise à jour) ?

**III – Gestion de la concurrence d’accès, des transactions et reprise sur panne :**

**3.1 – Les transactions sous Oracle**

1. Une transaction est une séquence d’ordres SQL exécutés tous ou aucun[[1]](#footnote-1).

2. Le début d’une transaction est le premier ordre qui suit :

(a) connexion au serveur, ou

(b) fin de transaction.

Les fins de transaction :

(a) annulation

(b) confirmation (*transaction dite alors validée*).

3. listes des confirmations :

(a) explicite : *commit*

(b) implicites :

i. tous ordres de mise à jour de schéma

ii. commande de déconnexion du mode interactif (ou application)

iii. *grant* (ordre de gestion de la confidentialité)

iv. en mode de confirmation automatique : toutes mises à jour de l’instance[[2]](#footnote-2)

effet : confirme toutes mises à jour de l’instance depuis le début de la transaction

4. listes des annulations :

(a) explicite : *rollback*

(b) implicite : déconnexion anormale (autre qu’ordre explicite de déconnexion interactif ou application)

effet : annule toutes mises à jour de l’instance depuis le début de la transaction

5. remarques :

(a) au cours d’une transaction : “table virtuelle locale”, différente de la table réelle visible par

tous clients

(b) un ordre peut échouer sans annuler la transaction (ex : insertion avec colonnes incorrectes)

6. rapport avec la persistance : une mise à jour[[3]](#footnote-3) persiste si et seulement si elle fait partie d’une transaction validée

7. recommandations :

(a) ne pas mettre dans la même transaction des mises à jour non corrélées logiquement

(b) expliciter toutes les confirmations et annulations par des *commit* ou des *rollback*.

**3.2 – Analyse de programme**

On s’intéresse ici à prévoir le contenu de la base à chaque instant au cours du déroulement d’une séquence d’ordres SQL[[4]](#footnote-4).

On considère une table T(A INTEGER, B INTEGER) contenant à l’origine le(s) n-uplet(s) indiqué(s) au début de chaque tableau. Tout ordre dessus est autorisé à tout utilisateur. On considère un client p1 qui lance les séquences d’ordres suivantes, immédiatement après sa connexion, avec le mode de confirmation automatique non activé. Un autre client quelconque est représenté par p2. Compléter les tableaux suivants. Indiquez le début et la fin de chaque transaction. Vérifiez ensuite sur machine avec SQLPLUS.

|  |  |  |
| --- | --- | --- |
| Ordres | T ds trans p1 | T vue par p2 |
|  | (1,2) | (1,2) |
| INSERT (3,4) |  |  |
| INSERT (1,2) |  |  |
| ROLLBACK |  |  |
| COMMIT |  |  |
| INSERT (1,2) |  |  |
| EXIT |  |  |

|  |  |  |
| --- | --- | --- |
| Ordres | T ds trans p1 | T vue par p2 |
|  | (1,2) | (1,2) |
| ROLLBACK |  |  |
| INSERT (1,2) |  |  |
| ALTER TABLE T ADD (C INTEGER) |  |  |
| UPDATE T SET B = 3 WHERE A = 1 |  |  |
| *panne : tapez ctl-\* |  |  |

|  |  |  |
| --- | --- | --- |
| ordres | T ds trans p1 | T vue par p2 |
|  | (1,2) | (1,2) |
| INSERT (3,4) |  |  |
| INSERT (5,6) |  |  |
| SET AUTOCOMMIT ON |  |  |
| INSERT (7,8) |  |  |
| ROLLBACK |  |  |
| INSERT (1,2,3,4) |  |  |
| DELETE FROM T WHERE A = 1 |  |  |
| *panne : tapez ctl-\* |  |  |

**3.3 – Construction de programme**

Il s’agit ici de gérer vous-même la reprise sur panne en plaçant les débuts et fins de transaction aux bons endroits pour préserver la cohérence de votre base en cas de panne.

Vous arrivez dans une SSII dans laquelle on vous confie la séquence d’ordres suivante, censée gérer une situation d’achat de billet sur les tables Voyage(client, destination, km), et Banque(client, montant) :

INSERT INTO VOYAGES VALUES(’Jules’, ’Saint Lucien’, 100) ;

UPDATE BANQUE SET MONTANT -= 100 WHERE CLIENT = ’Jules’ ;

1. Concrétisez l’exemple en considérant un ou deux n-uplets dans chaque table.
2. Enoncez la contrainte que doivent satisfaire les données de votre exemple.
3. Ce programme est-il correct ? Sinon, donnez tous les contre-exemples (pannes), en donnant à chaque fois le contenu de la base, et le déroulement du programme. Expliquez. Testez sur machine en simulant les pannes (lesquelles ? comment ?).
4. Réécrivez ce programme pour qu’il soit robuste aux pannes. Donnez toutes les variantes possibles. Testez sur machine en simulant les pannes.
5. Considérons mieux votre programme. Est-il possible qu’un des ordres échoue sans faire échouer la transaction? Votre programme est-il donc fiable ? Indiquez intuitivement ce qu’il faudrait faire pour qu’il le devienne. Est-ce le même problème pour toutes les variantes ?

**3.4 – Introduction au contrôle de concurrence**

Oracle gère des verrous pour contrôler les processus concurrents, au niveau des tables ou des n-uplets. Certains sont posés implicitement (sur les n-uplets lors des mises à jour d’instance, d’autres explicitement (sur les tables). Il n’est pas possible que deux processus possèdent des verrous globaux sur la même table ou locaux sur le même n-uplet. Les verrous sont relâchés en fin de transaction.

On s’initie ci-dessous à faire de l’intuition sur déroulement des ordres posant des verrous, puis à en poser nous-même là où il faut.

**3.4.1 – Analyse de programme**

On considère une table T(A integer, B integer) contenant au début les n-uplets (1,1) et (2,2). Tout ordre dessus est autorisé à tout utilisateur. On considère deux processus clients p1 et p2 qui effectuent les séquences d’ordres suivantes, immédiatement après leur connexion. Un autre client quelconque est représenté par p3.

Remplir les tableaux dans l’ordre dans lequel les ordres sont effectivement exécutés. Puis vérifiez sur machine.

1. 1 (p1) : DELETE T WHERE B=1;

2 (p1) : COMMIT;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Client | ordre | T ds trans p1 | T ds trans p2 | T vue par p3 |
|  |  | 11  22 | 11  22 | 11  22 |
|  |  |  |  |  |
|  |  |  |  |  |

2. 1 (p1) : DELETE T WHERE B=1;

2 (p2) : UPDATE T SET B=B-1;

3 (p1) : COMMIT;

4 (p2) : COMMIT;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Client | ordre | T ds trans p1 | T ds trans p2 | T vue par p3 |
|  |  | 11  22 | 11  22 | 11  22 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

3. 1 (p2) : UPDATE T SET B=B-1;

2 (p1) : DELETE T WHERE B=1;

3 (p2) : COMMIT;

4 (p1) : COMMIT;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Client | ordre | T ds trans p1 | T ds trans p2 | T vue par p3 |
|  |  | 11  22 | 11  22 | 11  22 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

4. 1 (p1) : UPDATE T SET A=1 WHERE B=2;

2 (p2) : DELETE T WHERE A=1;

3 (p1) : COMMIT;

4 (p2) : COMMIT;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Client | ordre | T ds trans p1 | T ds trans p2 | T vue par p3 |
|  |  | 11  22 | 11  22 | 11  22 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**3.4.2 – Demandes de verrous mutuellement bloquantes**

La table T(A integer, B integer) contient au départ les n-uplets (1 1) et (2 2). On considère les deux séquences d’ordres SQL+ suivantes :

|  |  |
| --- | --- |
| s1 : | s2 : |
| UPDATE T SET A=A+1 WHERE (A=1); | UPDATE T SET A=A+1 WHERE (A=2); |
| UPDATE T SET A=A+1 WHERE (A=2); | UPDATE T SET A=A+1 WHERE (A=1); |
| COMMIT ; | COMMIT ; |

Détailler les problèmes liés à l’exécution concurrente des séquences s1 et s2 par des utilisateurs autorisés différents. Quelles sont les solutions que propose Oracle pour gérer ce type de problème.

# Consignes d’organisation

* S’organiser en groupes de 2 à 3 membres par groupe (liste à remettre à la première séance du projet)
* Rendu de la première partie du projet à la fin de la première séance du projet (Partie I).
* Rendu final par dépôt sur ecampus ou par email selon consigne du professeur de Tp après la troisième et dernière séance en format pdf ou fichier SQL.

# Rendus attendus

1. Rapport de projet en format pdf portant les noms de tous les membres du groupe ainsi que le groupe de TD comme nom de fichier et vos noms doivent aussi être mentionnés dans le document.
2. Scripts SQL contenant vos commandes de création de schéma et requêtes.
3. Vos prises d'écran montrant les fonctionnalités de votre application + vos codes sources et scripts Sql. **Annexes**

**I - Script SQL de création des tables :**

-- ============================================================

-- Nom de la base : BD\_Notation

-- Nom de SGBD : ORACLE

-- Date de création : Janvier 2016

-- ============================================================

-- ============================================================

ALTER SESSION SET NLS\_DATE\_FORMAT = 'DD-MM-YYYY' ;

-- ============================================================

-- Destruction des tables

DROP TABLE Etudiant CASCADE CONSTRAINTS;

DROP TABLE Matiere CASCADE CONSTRAINTS;

DROP TABLE Epreuve CASCADE CONSTRAINTS;

DROP TABLE Notation CASCADE CONSTRAINTS;

-- Creation de tables

CREATE TABLE Etudiant (

NumEtu NUMBER(8) PRIMARY KEY,

Nom VARCHAR2(20),

Prenom VARCHAR2(20),

DateNais DATE,

Rue VARCHAR2(50),

CP CHAR(5),

Ville VARCHAR2(25)

);

CREATE TABLE Matiere (

CodeMat CHAR(3) PRIMARY KEY,

Libelle VARCHAR2(20),

Coef NUMBER(3,2)

);

CREATE TABLE Epreuve (

NumEpreuve NUMBER(10) PRIMARY KEY,

DateEpreuve DATE,

Lieu VARCHAR2(10),

CodeMat CHAR(3)

);

CREATE TABLE Notation (

NumEtu NUMBER(8),

NumEpreuve NUMBER(10),

Note NUMBER(4,2),

PRIMARY KEY (NumEtu,NumEpreuve)

);

ALTER TABLE Epreuve ADD CONSTRAINT FkEpreuve1 FOREIGN KEY(CodeMat) REFERENCES Matiere(CodeMat);

ALTER TABLE Notation ADD CONSTRAINT FkNotation1 FOREIGN KEY(NumEtu) REFERENCES Etudiant(NumEtu);

ALTER TABLE Notation ADD CONSTRAINT FkNotation2 FOREIGN KEY(NumEpreuve) REFERENCES Epreuve(NumEpreuve);

-- Remplissage des tables

INSERT INTO Etudiant VALUES (110,'Dupont','Albert','01-06-1990','RuedeCrimee','69001','Lyon');

INSERT INTO Etudiant VALUES (222,'West','James','03-09-1993','Studio',NULL,'Hollywood');

INSERT INTO Etudiant VALUES (300,'Martin','Marie','05-06-1998','RuedesAcacias','69130','Ecully');

INSERT INTO Etudiant VALUES (421,'Durand','Gaston','15-11-1990','RuedelaMeuse','69008','Lyon');

INSERT INTO Etudiant VALUES (575,'Titgoutte','Justine','28-02-1995','CheminduChâteau','69630','Chaponost');

INSERT INTO Etudiant VALUES (667,'Dupond','Noémie','18-09-1997','RuedeDôle','69007','Lyon');

INSERT INTO Etudiant VALUES (999,'Phantom','Marcel','30-01-1990',NULL,NULL,NULL);

INSERT INTO Matiere VALUES ('STA','Statistique',0.4);

INSERT INTO Matiere VALUES ('INF','Informatique',0.4);

INSERT INTO Matiere VALUES ('ECO','Econometrie',0.2);

INSERT INTO Epreuve VALUES(11031,'15-12-2003','Salle191L','STA');

INSERT INTO Epreuve VALUES(11032,'01-04-2004','AmphiG','STA');

INSERT INTO Epreuve VALUES(21031,'30-10-2003','Salle191L','INF');

INSERT INTO Epreuve VALUES(21032,'01-06-2004','Salle192L','INF');

INSERT INTO Epreuve VALUES(31030,'02-06-2004','Salle05R','ECO');

INSERT INTO Notation VALUES (110,11031,10);

INSERT INTO Notation VALUES (110,11032,11.5);

INSERT INTO Notation VALUES (110,21031,8.5);

INSERT INTO Notation VALUES (110,21032, NULL);

INSERT INTO Notation VALUES (110,31030,13);

INSERT INTO Notation VALUES (222,11031,9);

INSERT INTO Notation VALUES (222,11032,14);

INSERT INTO Notation VALUES (222,21031,12);

INSERT INTO Notation VALUES (222,21032,16);

INSERT INTO Notation VALUES (222,31030,20);

INSERT INTO Notation VALUES (300,11031,14);

INSERT INTO Notation VALUES (300,11032,20);

INSERT INTO Notation VALUES (300,21031,20);

INSERT INTO Notation VALUES (300,21032,13.5);

INSERT INTO Notation VALUES (300,31030,16);

INSERT INTO Notation VALUES (421,11031,5.5);

INSERT INTO Notation VALUES (421,11032,17);

INSERT INTO Notation VALUES (421,21031,1.5);

INSERT INTO Notation VALUES (421,21032, NULL);

INSERT INTO Notation VALUES (421,31030,10);

INSERT INTO Notation VALUES (575,11031,13);

INSERT INTO Notation VALUES (575,11032,9);

INSERT INTO Notation VALUES (575,21031,12.5);

INSERT INTO Notation VALUES (575,21032,14);

INSERT INTO Notation VALUES (575,31030,7);

INSERT INTO Notation VALUES (667,11031,16);

INSERT INTO Notation VALUES (667,11032,20);

INSERT INTO Notation VALUES (667,21031,8.5);

INSERT INTO Notation VALUES (667,21032,9.5);

**II - Manipulation de Base SQL\*PLUS :**

*Aide en ligne*

**help <nom-commande>** : aide sur la commande citée (ex. : help exit)

*Dialogue avec le système hôte*

**host <commande>** ou **! <commande>** : lance une commande du système d'exploitation.

*Travail sous sqlplus*

**Sous l’interface SQLPLUS directement :** la plupart des commandes sql peuvent être tapées sur une ou plusieurs lignes et doivent être terminées par ; SQL ne fait pas la différence entre ses commandes minuscules et majuscules. Il est toutefois recommandé de mettre les mots SQL en majuscules, d'aller à la ligne et d'indenter les lignes de commande.

**A partir d'un script :** On écrit une suite de commandes (terminées par un ;) à l'aide d'un éditeur de texte (vi, emacs, xemacs, ...), bien qu'il y ait un buffer SQL et l'on sauvegarde le fichier dans le répertoire de travail, le nom du fichier devant avoir pour extension **.sql.** On exécute les commandes soit en faisant du "couper/coller" de l'éditeur vers SQL, soit en dialoguant avec le système hôte de la manière suivante :

**start** <*nomprog*> ou **@** <*nomprog*> : charge le fichier et lance les commandes contenues dans le fichier *nomprog.sql* (inutile de spécifier l’extension .sql)

*Memento SQL\*PLUS*

**save filename.sql** sauve le contenu du buffer dans un fichier de nom filename.sql

**get filename.sql** charge le buffer avec le contenu du fichier de nom filename.sql

**start filename** charge le buffer et lance l'exécution du fichier script sql

**run**  lance l'exécution du contenu du buffer

**spool filename.txt** copie la sortie écran sur le fichier filename.txt

**spool off** suspend l'opération précédente

**help commande** pour obtenir de l'aide sur la commande donnée en argument

*Variables d'environnement*

**set long 1024** pour voir la totalité des définitions de vues

Dans SQLPLUS, pour paramétrer de façon correcte la fenêtre d’affichage, après chaque connexion, tapez :

**set pause on** ne visualise la sortie qu'après un 2ième Return - bloc par bloc

**set pagesize 40** (40 lignes par page)

**set linesize 100** (100 colonnes par pages)

*Schéma d'information ( ou "méta-base")*

**desc[ribe] tablename** donne le schéma de la relation tablename

**all\_catalog** relation donnant toutes les tables accessibles

**user\_catalog** relation donnant les seules tables du USER

**cat**  synonyme de la précédente

**user-tables** relation donnant toutes les tables du USER

**tab** synonyme de la précédente

**all\_objects** relation donnant tous les objets accessibles

**user\_objects** relation donnant les seuls objets du USER

**obj**  synonyme de la précédente ..

**user\_sys\_privs** relation donnant les privilèges système du USER

**user\_tab\_privs** relation donnant les privilèges sur les objets accessibles

**Quelques facilités : Edition d’ordres avec SQLPLUS (**directives uniquement disponible sous ORACLE) :

SQL> **save fic** permet de sauver l’ordre courant dans le fichier fic.sql

SQL> **get fic** permet de récupérer le contenu de fic.sql pour alimenter le buffer de sqlplus.

SQL> **start fic** lance l’exécution du contenu du buffer.

SQL> **l** permet de lister le contenu du buffer.

SQL> **ed** permet d’appeler l’éditeur du système d’exploitation (vi ou nedit sous UNIX).

SQL> **!** est le caractère d’échappement qui permet de lancer une commande du système d’exploitation sans sortir de sqlplus.

SQL> **exit** permet de sortir de sqlplus.

SQL> **def variable=emp** permet de définir une variable ;

SQL> select \* from &variable ;

SQL> **undef** permet d’annuler la définition de variable : undef variable.

SQL> **show all** permet de lister les variables de sqlplus.

SQL> **set** permet de positionner des variables :

SQL> set linesize 80 ;

SQL> set pagesize 24 ;

SQL> set pause on ;

SQL> set pause “ PAGE SUIVANTE ” permet d’arrêter le défilement entre chaque page.

SQL> **spool fichier** permet de copier la sortie écran dans un fichier fichier.lst. L’opération est terminée par

**spool off** pour permettre le vidage du buffer de sortie.

1. On considérera ici seulement les mises à jour de l’instance. [↑](#footnote-ref-1)
2. set autocommit on/off, show autocommit ; positionnement par défaut à la connexion [↑](#footnote-ref-2)
3. l’état de la base en résultant [↑](#footnote-ref-3)
4. Un exercice apparemment un peu scolaire, mais incontournable pour bien utiliser ensuite les outils dans ses propres programmes. [↑](#footnote-ref-4)