Exercice 0 : connexion à l'institut Galilée

1) Depuis chez vous, connectez-vous au serveur de l'institut Galilée pour y travailler dans un *terminal* et creez un nouveau répertoire nommé TP1. Deux méthodes sont possibles, soit via le service Guacamole en utilisant votre navigateur internet : Allez sur une machine linux indiquée en vert en cliquant sur le numéro de la machine, puis renseignez le login et mot de passe de votre compte étudiant, vous aurez alors besoin d'un *terminal*. Soit directement via ssh (Aide) : `login@sercalssh.ig-edu.univ-paris13.fr`

2) Depuis chez vous, récupérez le script `tp1-ecole-drop.sql` sur moodle et copiez le dans votre nouveau répertoire TP1. Par exemple utilisez `scp` ou tout autre solution qui vous est facile (NB : on peut bien sûr aussi utiliser directement Guacamole en mode graphique et travailler entièrement à distance sur une machine de Galilée, mais bon...). Voilà, vous êtes prêt pour `ORACLE` !

Exercice 1 : connexion à SQL*Plus

Depuis votre terminal, connectez-vous à `ORACLE`, modifiez votre mot de passe `ORACLE` puis déconnectez-vous. Pour cela utilisez les informations suivantes :

0) Dans votre terminal, tapez les premières lettres de la commande `sqlplus` puis essayez la complétion automatique (touche de tabulation). Si la commande est complétée alors vous pouvez passer directement au point 2).

1) Environnement utilisateur unix bash (.bashrc)

Les variables d'environnement nécessaires à la communication avec `ORACLE` se trouvent dans un fichier `.bashrc`. Donc si c'est la première fois que vous utilisez `ORACLE`, il vous faut :

- remplacer votre fichier `.bashrc` par `/export/home/users/COMMUN/.bashrc`
- décommenter la ligne concernant `ORACLE`, puis sauvegarder et exécuter la commande `source .bashrc`... mais peut-être qu'il vous faudra vous reconnecter.

2) Désormais vous n'aurez qu'à exécuter la commande unix `sqlplus login/password` pour vous connecter à `ORACLE` (vous pouvez aussi utiliser `sqlplus login` ou `sqlplus`).

- votre `login` est ... *attendez il faut faire ça tous ensemble*,
- votre `password` est le même que votre login. Vous devrez le changer.

3) Quelques commandes `SQL*PLUS`

`SQL*PLUS` est le langage qui permet de communiquer avec le **sgbd** `ORACLE`. Les commandes de `SQL*PLUS` servent à éditer ou executer les commandes de `SQL` standard implantées sous `ORACLE` ainsi que des commandes purement `SQL*PLUS`. Elles servent aussi au contrôle du formatage de l'affichage.

`SQL> quit`	quitter `SQL*PLUS`, idem avec `exit`
`SQL> password`	changement du mot de passe Oracle
`SQL> set pause on/off`	défilement page par page (via touche RETURN) ou continu
`SQL> host COMMANDE`	exécute la commande unix 'COMMANDE' (exemple : `host pwd`)
`SQL> desc table`	description de la structure de la table (exemple : `desc ALL_USERS`)

4) Remarques

- Le prompt 'SQL> ' indique que SQL*PLUS est prêt à recevoir une commande.
- ORACLE ne fait pas la distinction entre majuscule et minuscule dans les commandes de SQL*PLUS, mais cette distinction est faite au niveau du contenu des tables.
- Toute commande SQL se termine par le caractère ';'.

L'édition en ligne dans SQL*PLUS n'est pas aisée. Pour les modifications simples, elle se fait sur le tampon SQL contenant la dernière commande SQL éxecutée. L'édition peut se faire par copié/collé, à l'aide d'un éditeur externe à SQL*PLUS ou par l'execution d'un fichier contenant des commandes SQL*PLUS :

```
SQL> DEFINE_EDITOR = "nom_d_editeur"
SQL> edit                          lance l'éditeur externe précédement défini
SQL> @file                         exécute les commandes SQL du fichier file.sql
```

Le mot clef REM ou — en début de ligne sert à commenter une ligne dans un script .sql.

L'usage principal que l'on fera est :

- éditer du code SQL chez vous (càd dans votre environnement préféré, p.ex. atom),
- le recopier sur votre compte à l'institut Galilée (p.ex. scp),
- exécuter ce script SQL dans votre terminal distant sous SQL*PLUS à l'aide de SQL> @file

Mettez cela en pratique en vous déconnectant d'ORACLE et en allant dans le répertoire TP1 où vous aviez copié le script tp1-ecole-drop.sql : reconnectez-vous d'ici et exécutez ce script. Comme vous avez des message d'erreurs (tables inexistantes), modifiez chez vous le script en remplaçant toue les drop table par drop table if exists, et recommencez. Vous venez de découvrir que la bonne pratique lorsque l'on veut créer des tables est de commencer... par les supprimer pour éviter les contraintes d'unicité de noms : de table, de colonnes, de contraintes, etc

Exercice 2 : manipulation de tables

A l'aide de la commande desc affichez la *description* des tables AIR1_33.ELEVES, AIR1_33.COURS, AIR1_33.PROFESSEURS, etc. Comment affichez le contenu des tables ?

Exercice 3 : script de création de tables et notion de contraintes

1) En vous servant des informations ci-dessous et de l'annexe, créez dans un fichier nommé creation.sql les tables suivantes :

```
EMP(Nom, Noss, Salaire, Departement)
DIR(Noss, Departement)
```

Un n-uplet de EMP désigne le nom, le numéro de sécurité sociale, le salaire d'un employé et le nom du département où il travail. Un n-uplet de DIR désigne le numéro de sécurité sociale d'un directeur et le nom du département qu'il dirige.

2) Vérifiez la description de ces tables, puis supprimez les. Vous prendrez la bonne habitude de commencer un fichier de création de tables... en les supprimants.

3) Recréez les tables EMP et DIR pour y ajouter plusieurs n-uplets à l'aide de la commande suivante que vous utiliserez dans un fichier nommé population.sql :

```
INSERT INTO nom_de_table VALUES n_uplet ;
```

4) Afficher le contenu de vos tables avec la commande :

```
SELECT * FROM nom_de_table ;
```

5) Testez les contraintes en essayant d'insérer deux fois la même valeur de clé, en insérant un employé de nom inconnu, en insérant un numéro `Noss` trop grand, un nom d'employé trop long.

Exercice 4 : fonctions sur les chaines, dates, séquences et autres

La table `DUAL` d'`ORACLE` est une pseudo-table composée d'une ligne et d'une colonne. Elle permet d'effectuer des select sans utiliser de table particulière pour tester des fonctions (tests unitaires) ou récupérer des informations indépendantes des données mais liées à la base ou à Oracle (date système, séquence de valeurs, etc.). Testez les commandes suivantes : (si vous avez des problèmes avec le copier/coller depuis le pdf, le fichier source est sur mon compte `dual.sql`).

```
--Jour de naissance
SELECT SYSDATE FROM DUAL ;
SELECT TO_CHAR( TO_DATE( '22-01-2017', 'DD-MM-YYYY' ), 'DAY' ) JOURNAISSANCE
    FROM DUAL;

SELECT RPAD( 'Soleil', 17, 'bla' ) "RPAD exemple" FROM DUAL;
SELECT LPAD( 'AIR1', 15, '*.' ) "LPAD exemple" FROM DUAL;

--chaines
SELECT SUBSTR( 'ABCDEFGHIJ', 6, 3)  "SUBSTR exemple" FROM DUAL;
SELECT SUBSTR( 'ABCDEFGHIJ', -5, 4) "SUBSTR exemple" FROM DUAL;
SELECT TO_CHAR (SYSDATE, 'MM-DD-YYYY HH24:MI:SS' ) "Now" FROM DUAL;
SELECT LENGTH( 'WEB WAREHOUSE' ) "Nombre de caracteres" FROM DUAL;

--Arrondis
SELECT ROUND( 17.0958, 1 ) "ROUND exemple" FROM DUAL;
SELECT ROUND( 17.58,   2 ) "ROUND exemple" FROM DUAL;
SELECT TRUNC( 1958.0917, 1 ) "TRUNC exemple" FROM DUAL;
SELECT TRUNC( 1958.0917, 2 ) "TRUNC exemple" FROM DUAL;
SELECT ROUND( TO_DATE( '17-09-2016' ), 'YEAR' ) "New Year" FROM DUAL;

--Dates
SELECT TO_DATE( '17-SEPT.-2016', 'DD-MON-YYYY' ) FROM DUAL;
ALTER SESSION SET NLS_DATE_LANGUAGE = 'American' ;
SELECT ROUND(TO_DATE( '17-SEP-2016', 'DD-MON-YYYY' ), 'YEAR' ) "New Year" FROM DUAL;
ALTER SESSION SET NLS_DATE_LANGUAGE = 'American' ;
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MM-YYYY' ;
SELECT TO_DATE( '17-SEP-2016' , 'DD-MON-YYYY' ) FROM DUAL;
SELECT EXTRACT(YEAR FROM SYSDATE) FROM DUAL;
SELECT ADD_MONTHS(SYSDATE, 7 ) FROM DUAL;
SELECT TRUNC( MONTHS_BETWEEN( SYSDATE, TO_DATE( '29-MAY-2017' ) ) ) AS AGEBB
    FROM DUAL;
SELECT TO_NUMBER( TO_CHAR(SYSDATE, 'YYYY' ) ) FROM DUAL;
SELECT SYSTIMESTAMP FROM DUAL;

--Sequences
CREATE SEQUENCE MY_SEQ MINVALUE 1 ;
SELECT MY_SEQ.NEXTVAL FROM DUAL;
```

```
SELECT MY_SEQ.CURRVAL FROM DUAL;
DROP SEQUENCE MY_SEQ;
```

Exercice 5 : création de tables

Pour traiter de la vente par correspondance on considère la modélisation suivante (prenez un cas réel que vous connaissez) :

```
CLIENT(#client, nom, prenom, adresse, codePostal)
PRODUIT(#produit, designation, prixUnitaire)
FOURNISSEUR(#fournisseur, raisonSociale)
```

pour lesquels chaque numéro (noté #) identifie de façon unique un n-uplet. Ainsi que la table suivante reliant les informations contenues dans les précédentes :

```
COMMANDE_PRODUIT(#client, #produit, quantite, #fournisseur)
```

Utilisez des fichiers comme précédement pour :

1) Créez les trois premières tables en tenant compte du fait que chaque numéro est sur 3 chiffres (c.f. type de données `NUMBER`), que toute contrainte est identifiée à l'aide d'un nom et qu'un client commande des produits dans une quantité donnée. Ces tables sont-elles en troisième forme normale ?

2) Il est usuel de n'avoir dans la table `COMMANDE_PRODUIT` que les commandes en cours non traitées avec le moins de redondances possibles : si un client commande plusieurs fois le même produit, on met à jour le n-uplet correspondant en additionnant les quantités. Sachant que chaque produit est fourni par un unique fournisseur, la table `COMMANDE_PRODUIT` est-elle en troisième forme normale ? Si ce n'est pas le cas comment peut-on la transformer ?

3) On considère les tables ci dessous à la place de la précédente, est-il possible de les créer en ne précisant que des contraintes de clés étrangères ? Créez les tables sachant qu'un fournisseur peut fournir plusieurs produits.

```
COMMANDE(#client, #produit, quantite)
FOURNIT(#fournisseur, #produit)
```

4) Insérez au moins 4 n-uplets dans chaque table, puis définissez une vue sur ces deux tables afin d'accéder aux données comme à travers la table `COMMANDE_PRODUIT` de la question 2. Testez différentes commandes `SQL` sur cette vue, et consultez le dictionnaire de données `USER_tables` (par exemple sur les colonnes `owner` et `table_name`).

Exercice 6 : contraintes

1) Créez les tables suivantes :

```
EMP (nom, num, fonction, n_sup, embauche, salaire, comm, n_dept)
DEPT (n_dept, nom, lieu)
GRAD (grade, salairemin, salairemax)
```

Sachant que :

- Un employé de numéro `num` a un `nom` occupe l'emploi `fonction`, a un responsable identifié par le numéro `n_sup`, une date d'embauche `embauche`, un `salaire`, une commission `comm` et un département de rattachement `n_dept`,

- Un Département de numéro `n_dept`, de nom `nom` est situé à `lieu`,
- A un grade donné correspond un salaire minimum et un salaire maximum.

2) Altération de tables

Lorsque certaines contraintes sont introduites dans la conception du schéma, le système leur donne un nom interne, vérifiez le en consultant la vue du dictionnaire de données `USER_constraints`. Ici le schéma initial ne comporte aucune contrainte et le concepteur constate qu'il a omis de déclarer des contraintes d'intégrité essentielles. Il s'agit donc de mettre en oeuvre ces contraintes en les créant et en les nommant (Consulter l'annexe à propos de `ALTER` et `ADD Constraint`).

Ajoutez sur la table `DEPT` la contrainte `dept_pk` définissant `n_dept` comme clé primaire. Ajoutez sur la table `EMP` les contraintes suivantes :

- `emp_pk` définissant `num` comme clé primaire,
- `nom_u` définissant `nom` comme nom unique,
- `responsable` définissant `n_sup` comme clé étrangère référençant `num`,
- `dept` définissant `n_dept` comme clé étrangère référençant `n_dept` de la table `DEPT`,
- `commission` définissant une contrainte telle que seuls les employés dont la fonction est commerciale aient une commission com non nulle (`NULL`).

3) Vérification

Vérifier que les contraintes ont été créées. Pour cela consulter la vue du dictionnaire de données `USER_constraints`. Vérifier que ces contraintes sont actives en essayant de les transgresser en réalisant des insertions ou des effacements.

4) Désactiver/Activer des contraintes

Dans certains cas, l'administrateur ou l'utilisateur qui a défini une contrainte peut souhaiter, pour effectuer des mises à jour qui la transgresse, désactiver momentanément cette contrainte.

Vous devez :

- Lister les noms de tables et de contraintes de l'utilisateur ;
- Consulter l'aide `DISABLE/ENABLE` ;
- Désactiver la contrainte `commission` ;
- Insérer des n-uplets dans `EMP` transgressant la contrainte commission ;
- Essayer de rétablir la contrainte. Conclusion ?
- Détecter les n-uplets qui provoquent l'erreur afin de pouvoir les supprimer et rétablir la contrainte.

Pour cela, créer une table que vous nommreez `REJETS` :

```
CREATE TABLE REJETS(
  ROW_ID ROWID,
  OWNER VARCHAR2(30),
  TABLE_NAME VARCHAR2(30),
  CONSTRAINT VARCHAR2(30)
);
```

Réactiver la contrainte `commission` en demandant la sauvegarde des n-uplets erronés dans la table `REJETS` par la commande :

```
alter table EMP enable constraint commission exceptions into REJETS;
```

Vérifier le contenu de la table `REJETS`, puis supprimer de la table `EMP` les n-uplets erronés. Rétablissez enfin la contrainte `commission`

5) Supprimer les contraintes de la table `EMP`.

Exercice 6 : Nettoyage et plus

Vos tables sont stockées dans la table de nom `TAB`. Affichez son contenu pour supprimer toutes vos tables.

`ORACLE` gère les informations systèmes concernant les utilisateurs et leurs bases sous la forme relationnelle. Donc, l'ensemble de ces informations se trouvent dans des tables, appelées le **dictionnaire des données**. Certaines de ces tables sont accessibles en lecture à tout utilisateur via les commandes `SQL`, d'autres sont réservées à l'administrateur ou au sytème.
La liste de toutes ces tables se trouve dans `DICT` (=dictionnaire), ainsi que dans la table `ALL_CATALOG`.
Appliquez des commandes `SQL` sur certaines de ces tables pour voir leur contenu. (Attention: le contenu de certaines de ces tables tiennent sur plusieurs pages)
Ex: `ALL_TABLES, ALL_USERS, USER_TABLES, USER_TAB_COLUMNS, USER_CONSTRAINTS, USER_SYS_PRIVS, ...`

Annexes

1) Quelques types de données

```
NUMBER [(l [,d])]
CHAR(l)
VARCHAR2(l)
DATE
```

`l` est la longueur totale, `d` le nombre de décimales

2) Création de tables et contraintes

```
CREATE TABLE nom_de_table ( { nom_de_colonne type_de_donnees [ DEFAULT expression
] [ contrainte_de_colonne [ ... ] ] | contrainte_de_table } [, ... ] ) ;

contrainte_de_colonne ::= [ CONSTRAINT nom_de_contrainte ] {
NOT NULL | NULL | UNIQUE | PRIMARY KEY | CHECK (condition) |
REFERENCES table_de_reference [ (colonne_de_reference) ] [ ON DELETE action | ON UPDATE
action ]
} [ DEFERRABLE | NOT DEFERRABLE ]

contrainte_de_table ::= [ CONSTRAINT nom_de_contrainte ]{
UNIQUE (liste_de_colonne) | PRIMARY KEY (liste_de_colonne) | CHECK (condition) |
FOREIGN KEY (liste_de_col) REFERENCES table_de_ref [ (liste_de_col_de_ref) ]
[ ON DELETE action ]
} [ DEFERRABLE | NOT DEFERRABLE ]
```

`CREATE TABLE` créera une nouvelle table initialement vide dans la base de données courante. La table sera la propriété par l'utilisateur lançant la commande.
Si un nom de schéma est donné (par exemple, `CREATE TABLE monschema.matable ...`), alors la table est créée dans le schéma spécifié. Sinon, il est créé dans le schéma actuel. Le nom de la table doit être distinct des noms des autres tables, séquences, index ou vues dans le même schéma.
`CREATE TABLE` crée aussi automatiquement un type de données qui représente le type composé correspondant à une ligne de la table. Du coup, les tables ne peuvent pas avoir le même nom que tout type de données du même schéma.
Les clauses de contrainte optionnelle spécifient les contraintes que les nouvelles lignes ou les lignes mises à jour doivent satisfaire pour qu'une opération d'insertion ou de mise à jour réussisse. Une contrainte est un objet `SQL` qui aide à définir l'ensemble de valeurs valides de plusieurs façons.
Il existe deux façons de définir des contraintes : les contraintes de table et celles des colonnes. Une contrainte de colonne est définie pour faire partie d'une définition de la colonne. Une définition de la contrainte des tables n'est pas liée à une colonne particulière. Chaque contrainte de colonne peut aussi être écrite comme une contrainte de table. La contrainte de clé primaire spécifie qu'une ou plusieurs colonnes d'une table pourraient contenir seulement des valeurs uniques, non null. Techniquement, `PRIMARY KEY` est simplement une combinaison de `UNIQUE` et `NOT NULL`, mais identifier un ensemble de colonnes comme clé primaire fournit aussi des métadonnées sur le concept du schéma, car une clé primaire implique que d'autres tables pourraient se lier à cet ensemble de colonnes comme un unique identifiant pour les lignes. Une seule clé primaire peut être spécifiée pour une table, qu'il s'agisse d'une contrainte de colonne ou de table.

Lorsque les donnés des colonnes référencées sont modifiées, certaines actions sont réalisées sur les données dans les colonnes de cette table. Les clauses `ON DELETE` et `ON UPDATE` spécifient l'action à réaliser lorsqu'une ligne référencée de la table référencée est en cours de suppression/modification. Les actions suivantes sont possibles : `NO ACTION, RESTRICT` (jamais déferrée), `CASCADE , SET NULL, SET DEFAULT`. Remarque : `NOT DEFERRABLE` = vérifiée immédiatement après chaque commande.

3) Suppression de tables

```
DROP TABLE nom_de_table [, ...]  [ CASCADE | RESTRICT ] ;
```

`DROP TABLE` supprime des tables de la base de données. Seul son propriétaire peut détruire une table. Pour vider une table de lignes, sans détruire la table, utilisez `DELETE`.
`DROP TABLE` supprime tout index, règle, déclencheur et contrainte existant sur la table cible. Néanmoins, pour supprimer une table qui est référencé par une contrainte de clé étrangère d'une autre table, `CASCADE` doit être spécifié (`CASCADE` supprimera la contrainte de clé étrangère, pas l'autre table elle-même) .

4) Altération de tables

```
ALTER TABLE nom_de_table une_alteration ;
une_alteration ::=
    ADD nom_de_colonne domaine [DEFAULT expression] [ contrainte_de_colonne ] |
    ADD contrainte_de_table |
    DROP COLUMN nom_de_colonne [ RESTRICT | CASCADE ] |
    MODIFY nom_de_colonne [domaine] [DEFAULT expression] [contrainte_de_colonne] |
    RENAME COLUMN nom_de_colonne TO nouvelle_colonne |
    RENAME TO nouveau_nom |
    { DROP | ENABLE | DISABLE } CONSTRAINT nom_de_contrainte [ RESTRICT | CASCADE ]
```

`ALTER TABLE` change la définition d'une table existante. Il y a plusieurs variantes:

- `ADD nom_de_colonne domaine` et `ADD (liste_de_colonnes)` ajoutent une nouvelle colonne à la table en utilisant la même syntaxe que `CREATE TABLE`. Une nouvelle colonne ne peut avoir la contrainte `NOT NULL` que si la table est vide.
- `ADD contrainte_de_table` ajoute une nouvelle contrainte à une table en utilisant la même syntaxe que `CREATE TABLE`.
- `DROP COLUMN nom_de_colonne` supprime une colonne d'une table. Les indexes et les contraintes de table référençant cette colonne sont automatiquement supprimés. Il faut utiliser l'option `CASCADE` si certains objets hors de la table dépendent de cette colonne, comme par exemple des références de clés étrangères ou des vues.
- `DEFAULT expression` ne modifient pas les lignes déja présentes dans la table. Les valeurs par défaut ne s'appliquent qu'aux prochaines commandes `INSERT`. Des valeurs par défaut peuvent aussi être créées pour les vues.
- `RENAME TO` change le nom d'une table (ou d'un index, d'une séquence, d'une vue) ou le nom d'une colonne de la table. Elle est sans effet sur les données stockées.
- `DROP CONSTRAINT` supprime des contraintes d'une table. Actuellement, les contraintes n'ont pas nécessairement un nom unique, si bien qu'il peut y avoir plusieurs contraintes qui ont le nom donné en paramètre. Toutes ces contraintes sont supprimées. On peut aussi les activer ou désactiver.

`ALTER TABLE` effectue une copie temporaire de la table originale. Les modifications sont faites sur cette copie, puis l'original est effacée, et enfin la copie est renommée pour remplacer l'originale. Cette méthode permet de rediriger toutes les commandes automatiquement vers la nouvelle table sans pertes. Durant l'exécution de `ALTER TABLE`, la table originale est lisible par d'autres clients. Les modifications et insertions sont reportées jusqu'à ce que la nouvelle table soit prête. C'est un méchanisme transactionnnel.