Langage SQL

LCD

Commande	Explication
CLR SCR ;	Pour effacer l'écran du sql+
CONNECT User_name **/** pwd **AS SYSDBA ;**	Se connecter en tant que super utilisateur
CREATE USER User_name **IDENTIFIED BY** pwd**;**	Créer un utilisateur
DISC ;	Pour se déconnecter
ALTER USER User_name **Identified by** NW_pass_word**;**	Modifier le mot de passe
SELECT * FROM ALL_USERS;	Consulter les différents utilisateurs
SELECT table_name **FROM** user_tables **;**	Pour consulter l'ensemble des tableaux d'un utilisateur
SHOW user **;**	Pour savoir avec quel utilisateur vous êtes connecté
DESC table_name ;	Afficher la structure d'une table
DROP USER User_name **CASCADE;**	Supprimer un utilisateur (possède des objets)
ALTER USER User_name **ACCOUNT LOCK;** #Verrouillage **ALTER USER** User_name **ACCOUNT UNLOCK;** #Activation	Modification de statut
GRANT privilège (ou **ALL**) **ON** object_name **TO** user_name **[ou PUBLIC] [WITH GRANT OPTION]**;	Attribuer des droits à des utilisateurs sur vos objets. Les privilèges sont : **CONNECT, RESOURCE, DBA, SELECT, INSERT, UPDATE, DELETE, ALTER** ou (tout cela et donc **ALL**). **WITH GRANT OPTION**, permet à l'utilisateur à qui on vient d'attribuer des droits d'attribuer les mêmes droits à d'autres utilisateurs sur le même objet.
CREATE ROLE name_role;	Création d'un ROLE
GRANT <privilège> **ON** name_object **TO** name_role;	Attribuer des privilèges à un ROLE
GRANT role_name **TO** user_name**;**	Attribuer un ROLE à un utilisateur
DROP role_name**;**	Détruire un ROLE
REVOKE <privilège> (ou **ALL**) **ON** object_name **FROM** user_name [ou **PUBLIC**];	Permettre de **retirer des droits** à un utilisateur

LDD

Commande	Explication
CREATE TABLE table_name (column_1 data_type column_constraint, column_2 data_type column_constraint, ...);	Créer une table. Les contraintes peuvent être : **NOT NULL, DEFAULT, UNIQUE, CHECK, PRIMARY KEY, FOREIGN Key**
DROP TABLE persons;	Supprimer une table
ALTER TABLE table_name **ADD** column_name type constraint;	Pour ajouter une nouvelle colonne à une table
ALTER TABLE table_name **MODIFY** column_name type constraint;	Pour modifier les attributs d'une colonne
ALTER TABLE table_name **DROP COLUMN** column_name;	Pour supprimer une colonne existante d'une table
ALTER TABLE table_name **DROP** (column_1, column_2,...);	Pour supprimer plusieurs colonnes dans une table
ALTER TABLE table_name **RENAME COLUMN** column_name **TO** new_name;	Pour changer le nom d'une colonne dans une table
ALTER TABLE table_name **RENAME TO** new_table_name;	Pour donner un nouveau nom à une table
CONSTRAINT pk_table **PRIMARY KEY (**pk1, pk2..**)** **ADD CONSTRAINT** pk_Employees **PRIMARY KEY (**Id**);** **ALTER TABLE** table **DROP PRIMARY KEY ;** **ALTER TABLE** table **DROP CONSTRAINT** pk_Employees**;**	Pour créer une contrainte **PRIMARY KEY** sur les colonnes Pour définir une contrainte **PRIMARY KEY** sur la colonne 'Id' Pour supprimer les contraintes de clé primaire de la table
FOREIGN KEY (ID_EMP**) REFERENCES** Employes**(**Id**);**	pour spécifier une clé étrangère
DROP CONSTRAINT fk_Conges**;**	Pour supprimer une contrainte **FOREIGN KEY**

LID

Commande	Explication
SELECT [liste des attributs] ou *** **FROM** [liste de table] **WHERE** [condition de recherche] **GROUP BY** [attributs de partitionnement] **HAVING** [condition de groupe] **ORDER BY** [liste de colonnes (ASC/DESC)]	Extraire des données d'une base ou calculer de nouvelles données à partir d'existantes. La condition de recherche peut être : **LIKE, BETWEEN, IN, IS NULL/ IS NOT NULL, EXISTS, EXISTS, ALL, ANY, SOME, AND OR NOT, (=, <> ou !=, <=, >=)** Pour le prédicat **LIKE :** (_) remplace n'importe quel caractère (un seul caractère). (%) remplace n'importe quelle suite de caractères.
SELECT aggregation function(column) **FROM** table_name;	Pour effectuer quelques statistiques de bases sur des tables. Les principales fonctions sont les suivantes : **COUNT(), MAX(), MIN(), SUM(), AV()**
SELECT column1 **AS** alias **FROM** table1 **AS** alias1, table2 **AS** alias2 **WHERE** alias1.attribut=alias2.attribut	Réaliser une jointure
CREATE INDEX nom_index **ON** nom_table (colonne1) ;	**Un index** est une structure de données utilisée pour améliorer la rapidité de récupération des lignes d'une table en fonction des valeurs d'une ou plusieurs colonnes. Il joue le rôle d'un orienteur qui guide le système de base de données pour trouver rapidement les enregistrements correspondant à certaines conditions.
SELECT * **FROM** table1 **WHERE** column_name = (**SELECT** column_name **FROM** table2 **LIMIT** 1) ;	Réaliser une SOUS-REQUÊTE Remarque : L'instruction **LIMIT**, vous spécifiez le nombre maximum de lignes à renvoyer dans le résultat de la requête.
SELECT nom_colonne1 **FROM** `table1` **WHERE EXISTS** (**SELECT** nom_colonne2 **FROM** `table2` **WHERE** nom_colonne3 = 10)	S'il y a au moins une ligne dans **table2** dont **nom_colonne3** contient la valeur **10**, alors la sous-requête retournera au moins un résultat. Dès lors, la **condition sera vérifiée** et la requête principale retournera les résultats de la colonne **nom_colonne1** de **table1**.

LMD

Commande	Explication
INSERT INTO nom_de_table (colonne1, colonne2, colonne3) **VALUES** (valeur1, valeur2, valeur3), (valeur4, valeur5, valeur6), (valeur7, valeur8, valeur9)**;**	Insérer une/plusieurs ligne.s en indiquant les informations pour chaque colonne existante (en respectant l'ordre) ou bien en spécifiant les colonnes à remplir.
UPDATE table_name **SET** column1 = value1, column2 = value2, column3 = value3, ... **WHERE** condition;	La clause WHERE sélectionne les tuples à modifier. La clause SET spécifie les attributs à modifier et leurs nouvelles valeurs. L'absence de clause WHERE signifie que les changements doivent être appliqués à toutes les lignes de la table cible.
DELETE FROM table_name **WHERE** condition;	La commande DELETE permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

LCT

Commande	Explication
INSERT INTO discounts (discount_name, amount, start_date, expired_date) **VALUES** ('Summer Promotion', 9.5, '2017-05-01', '2017-08-31'); **COMMIT**;	La commande **COMMIT** permet de valider les modifications apportées à une transaction et par conséquent de la rendre permanente.
DELETE FROM discounts; **ROLLBACK**;	La commande **ROLLBACK** permet d'annuler les modifications apportées à une transaction. **ROLLBACK** restaure l'état de la base de données au dernier point de validation.
SAVEPOINT pointer_name**;** **ROLLBACK TO** my_savepoint_1**;**	La commande **SAVEPOINT** permet de spécifier un point dans une transaction auquel vous pourrez revenir ultérieurement. Si on tape **COMMIT**, toutes les transactions sont validées et tous les SAVEPOINTS sont supprimés.

Programmation PL/SQL

Commande	Explication

DECLARE
Déclaration de variables, constantes et exceptions
BEGIN
Les instructions à exécuter (commandes exécutables, instructions SQL et PL/SQL, Possibilité de blocs fils (imbrication de blocs))
EXCEPTION
Traitement des exceptions (gestion des erreurs)
END;
	Syntaxe d'un bloc PL/SQL

BEGIN
DBMS_OUTPUT.put_line ('Hello World!');
END;
	Pour afficher le message "Hello World" sur l'écran.

DECLARE
nom **VARCHAR2(15);**
numero **NUMBER;**
date_jour **DATE;**
salaire **NUMBER(7,2);**
reponse **BOOLEAN :=true;**
PI **CONSTANT real:=3.14;**
Age **integer NOT NULL;**
BEGIN
. . .
END;
	Les types habituels correspondants aux types Oracle sont : INTEGER, NUMBER, VARCHAR2, etc.
	Il est important de noter qu'en PLSQL, on ne peut pas des déclarations multiples !
	L'initialisation d'une variable peut se faire par :
	• L'opérateur **:=** dans les sections DECLARE, BEGIN et EXCEPTION.
	• L'ordre **SELECT** … **INTO** …. dans la section **BEGIN**
	• Le traitement d'un curseur dans la section **BEGIN**

nom_variable table.colonne**%TYPE ;** nom_variable2 nom_variable1**%TYPE;** nom_variable table**%ROWTYPE ;**	Types composites adaptés à la récupération des colonnes et lignes des tables SQL : %TYPE, %ROWTYPE, RECORD, TABLE. **%TYPE:** Variables de même type qu'une **colonne** dans la table. Variables de même type qu'une **variable** précédemment définie. **%ROWTYPE :** Variables reprenant la même structure qu'une **ligne** d'une table.
DECLARE nom_emp CHAR(15) ; Salaire emp.sal%TYPE ; commision emp.comm%TYPE ; nom_depart CHAR(15) ; **BEGIN** SELECT ename, sal, comm, dname INTO nom_emp, salaire, commision, nom_depart FROM emp, dept WHERE ename = 'Hammami' AND emp.deptno=dept.deptno ;	Ici dans cet exemple on a déclaré les variables nom_emp, salaire, commission et nom de départ. Apres on a affecté avec la close SELECT INTO les valeurs correspondantes. C'est-à-dire nom_emp va prendre la valeur de ename qui va être 'Hammami' et avec des conditions présentés par la close Where.

….. **END ;**	

DECLARE
TYPE EmployeeRecord **IS RECORD** (
EmployeeID NUMBER,
EmployeeName VARCHAR2(50),
Salary NUMBER
);
emp_info EmployeeRecord;
BEGIN
-- Utilisation du record
emp_info.EmployeeID := 101;
emp_info.EmployeeName := 'John Doe';
emp_info.Salary := 50000;
END;
	Type RECORD est un type local. Il est disponible seulement dans le bloc où il est déclaré.

IF condition 1 **THEN**
instructions 1;
ELSEIF condition 2 **THEN**
instructions 2;
ELSEIF ….
…
ELSE
instructions N;
END IF;
	Traitements conditionnels

DECLARE
nbre **NUMBER := 1 ;**
resultat **NUMBER;**
BEGIN
LOOP
INSERT INTO resultat **VALUES (**nbre**) ;**
nbre **:=** nbre **+ 1**
EXIT WHEN nbre > 10 **;**
END LOOP ;
END ;
	La boucle de base (LOOP)

DECLARE
fact **NUMBER := 1 ;**
resultat NUMBER;
BEGIN
FOR i **IN** 1..9
LOOP
fact **:=** fact ***** i **;**
END LOOP ;
INSERT INTO resultat **VALUES (**fact**);**
END ;
	La boucle FOR

DECLARE v1 **NUMBER := 1 ;** **BEGIN** **WHILE** v1<=10 **LOOP** **DBMS_OUTPUT.PUTLINE(**v1**);** v1=v1+1**;**	La boucle WHILE

END LOOP ; **END;**	

DECLARE
CURSOR dept_10 IS
Select ename,Sal FROM emp
WHERE deptno = 10
ORDER BY sal;
BEGIN
… ;
… ;
END;
	Déclaration d'un curseur explicite

OPEN nom_curseur**;**	Ouverture d'un curseur explicite
Fetch nom_curseur **INTO** liste_variables**;**	Traitement des lignes Après l'exécution du **SELECT**, les lignes ramenées sont traitées une par une. La valeur de chaque colonne du **SELECT** doit être stockée dans une variable réceptrice.
OPEN dept_10**;** **LOOP** **FETCH** dept_10 **INTO** nom**,** salaire**;** **IF** salaire >2500 **THEN** **INSERT INTO** resultat **VALUES (**nom**,salaire);** **END IF;** **EXIT WHEN** salaire = 5000**;** **END LOOP;**	Le Fetch ramène une seule ligne à la fois. Pour traiter n lignes, il faut prévoir une boucle
CLOSE nom_curseur;	Fermeture du curseur
SQL%FOUND (Curseur implicite) nom_curseur**%FOUND** **SQL%NOTFOUND** (Curseur implicite) nom_curseur**%NOTFOUND** **SQL%ISOPEN** (Curseur implicite, toujours égal à **FALSE**) nom_curseur**%ISOPEN** nom_curseur**%ROWCOUNT**	**%FOUND** Vrai si au moins une ligne a été traitée par la requête ou le dernier FETCH. **%NOTFOUND** Vrai si aucune ligne n'a été traitée par la requête ou le dernier FETCH. **%ISOPEN** Vrai si le curseur est ouvert (utile seulement pour les curseurs explicites). **%ROWCOUNT** Nombre de lignes traitées par le curseur.
nomrecord nomcurseur**%Rowtype;**	**%Rowtype** permet la déclaration implicite d'une structure dont les éléments sont d'un type identique aux colonnes ramenées par le curseur.
EXCEPTION **WHEN** NO_DATA_FOUND **THEN** dbms_outmut.putline('Aucun employé n''existe avec l''identifiant' || EmpID);	Les exceptions définies par le système
DECLARE **Nom_ano EXCEPTION;** **BEGIN** **...** **IF (condition_anomalie) THEN RAISE Nom_ano ;** **EXCEPTION** **WHEN (Nom_ano) THEN (traitement);**	Les exceptions déterminées par l'utilisateur Nommer l'anomalie (type exception) dans la partie DECLARE du bloc. Déterminer l'erreur et passer la main au traitement approprié par la commande **RAISE**. Effectuer le traitement défini dans la partie EXCEPTION du Bloc.

Les procédures et fonctions

Commande	Explication

CREATE OR REPLACE PROCEDURE nom_de_la_procedure (parametre1 MODE TYPE,…, parametre2 MODE TYPE) **IS**
Section déclarative optionnelle et sans utiliser le mot clé **DECLARE**
[déclaration variables locales]
BEGIN
Section exécutable obligatoire
[section exception]
END ;
	Pour une procédure, il y a trois MODES de passage de paramètre:
	IN : en entrée (par défaut) : ici on prend juste la valeur du paramètre.
	OUT : en sortie : ici le paramètre est initialement vide et on le remplie avec une valeur au cours de la procédure.
	INOUT : en entrée et sortie : ici on prend la valeur du paramètre et on la modifie après.

DECLARE
nb number;
BEGIN
add_dept(300,'IT',nb);
End;
	Appel de procédure

CREATE OR REPLACE FUNCTION nom_de_la_fonction (parametre1 TYPE,…, parametre2 TYPE) **RETURN** type_retour **IS** --Section déclarative optionnelle et sans utiliser le mot clé DECLARE [déclaration variables locales] **BEGIN** --Section exécutable obligatoire [section exception] **END ;**	Tous les paramètres d'une fonction sont en mode IN (dans ce cas, on n'est pas obligé d'écrire le mode).

DECLARE
v_employee_id NUMBER := 124;
v_result BOOLEAN;
BEGIN
-- Appel de la fonction stockée
v_result := fn_check_sal(v_employee_id);
END;
	Appel de fonction

DECLARE
PROCEDURE nom_procédure **(**parametre1 **MODE TYPE,…,** parametre2 **MODE TYPE) IS**
-–Section déclarative optionnelle et sans utiliser le mot clé DECLARE
[déclaration variables locales]
BEGIN
END nom_procédure **;**
BEGIN
-–Section exécutable obligatoire
END ;
	Les procédures non stockées (non autonomes)

DECLARE --déclaration des variables **FUNCTION** nom_fonction **(**parametre1 **TYPE,…,** parametre2 **TYPE)** **RETURN** type_retour **IS** -–Section déclarative optionnelle et sans utiliser le mot clé DECLARE [déclaration variables locales]	Les fonctions non stockées (non autonomes)

BEGIN
-–logique de la fonction
END nom_fonction **;**
BEGIN
-–Section exécutable obligatoire
END ;	

Déclencheurs (TRIGGERS)

Commande	Explication
CREATE OR REPLACE TRIGGER nom_trigger MOMENT —- BEFORE/AFTER EVENEMENT —- INSERT/UPDATE OF/DELETE ON nom_table [DECLARE -- Déclarations des variables locales (Facultatif) BEGIN -- Bloc de code PL/SQL END	2 moments: BEFORE AFTER 3 évènements: INSERT UPDATE OF DELETE 2 modes: Ordre Ligne (FOR EACH ROW)

CREATE OR REPLACE TRIGGER
after_insert_order_example
AFTER INSERT
ON employees
FOR EACH ROW
DECLARE
BEGIN
-- Actions à effectuer après chaque INSERT (par ligne)
DBMS DBMS_OUTPUT.PUT_LINE('Déclencheur AFTER INSERT par ligne exécuté pour l''employé avec ID ' || :NEW.employee_id);
END;
	Les variables spéciales :OLD et :NEW facilitent l'accès aux données avant et après une modification dans les déclencheurs PL/SQL.
	Elles sont particulièrement utiles pour comparer les valeurs et effectuer des actions basées sur les modifications.
	La nouvelle valeur appelée :new.colonne contient la nouvelle valeur de colonne après la modification.
	L'ancienne valeur est appelée :old.colonne contient l'ancienne valeur de colonne avant la modification.

RAISE_APPLICATION_ERROR(error_code, 'your_error_message');	**RAISE_APPLICATION_ERROR** est une procédure spéciale qui vous permet de générer une exception personnalisée avec un code d'erreur et un message d'erreur spécifiques. Cela peut être très utile pour identifier et gérer des situations exceptionnelles dans votre application.