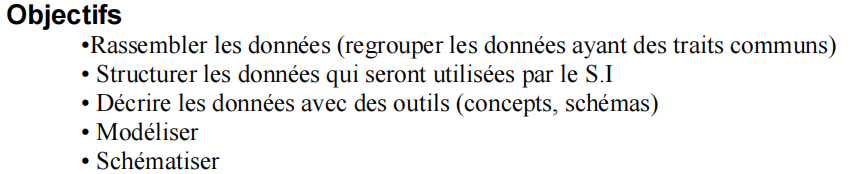
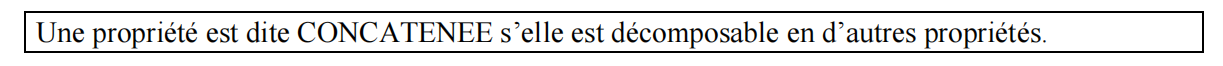
**Model conceptuel des données**







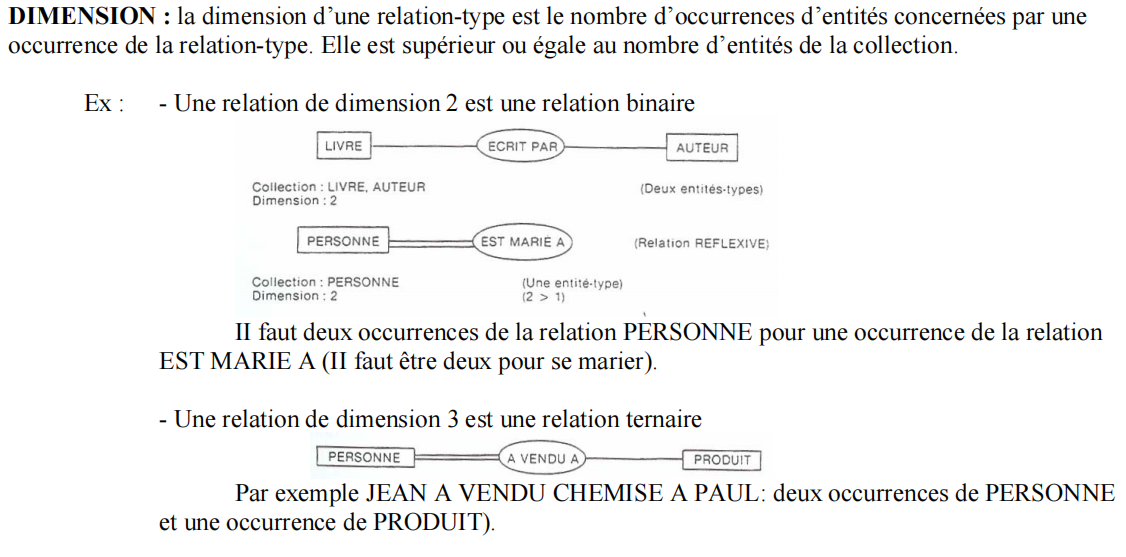


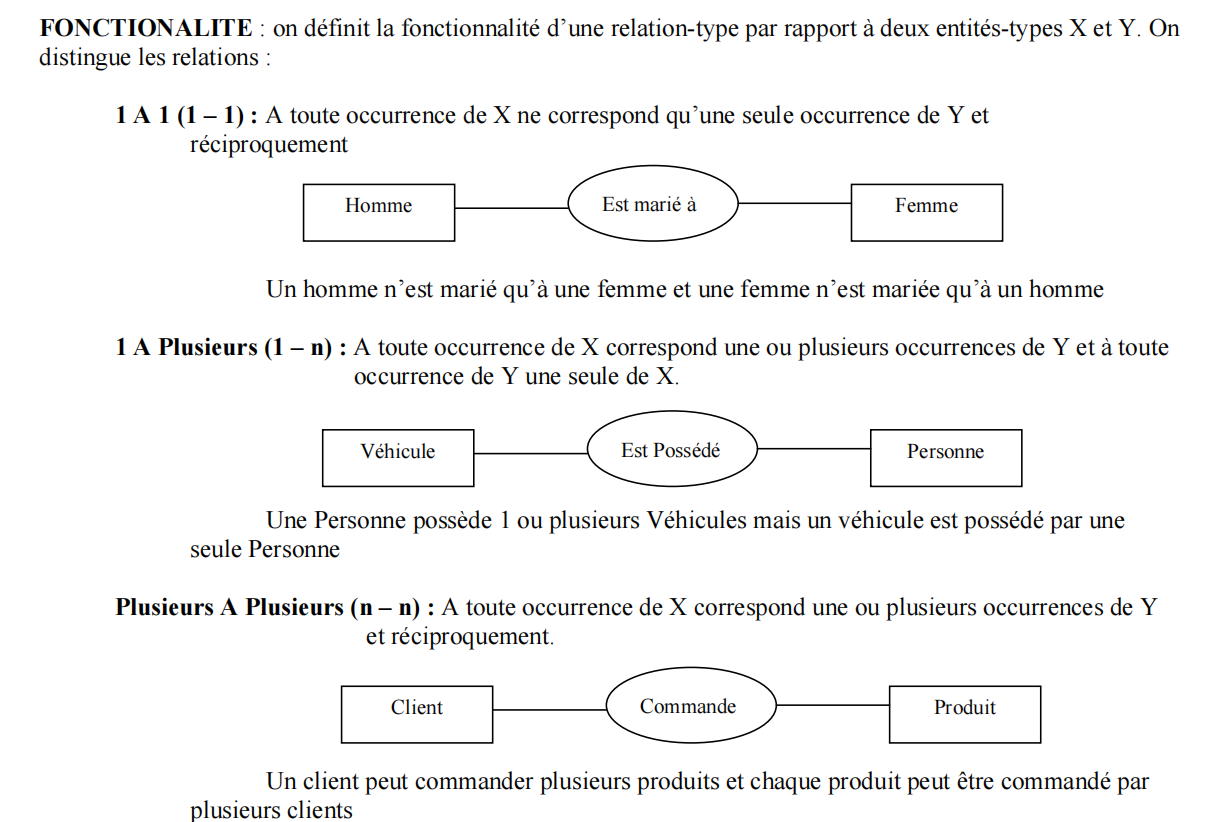
On distingue 3 sortes de propriétés :

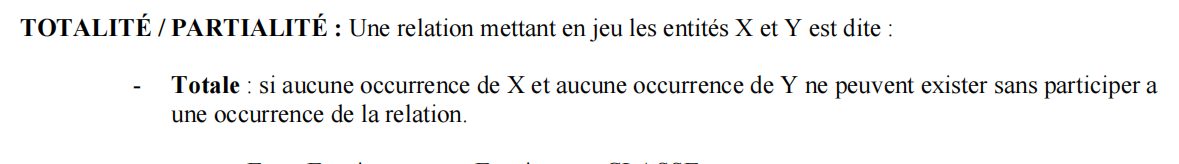
*Code:* á chaque objet en associé une et une seule valeur de code, ce qui permet la distinction entre les objets de même type (CIN, CNE…).

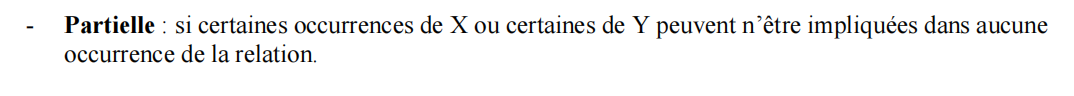
*Les libellés :* ce sont des chaines de caractères qui ne pourront pas participer au calculs, mais on peut les utiliser pour les tris ou bien les comparaison (Nom, Prénom, ...).

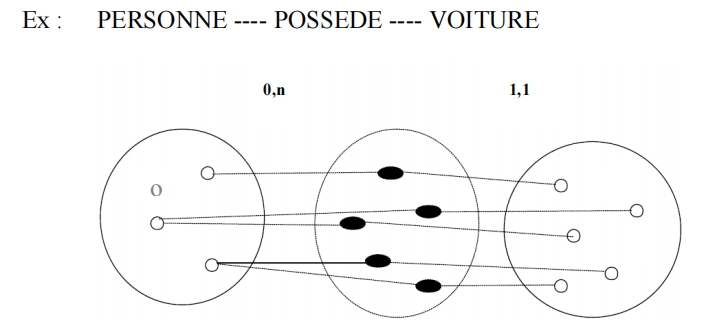
*Les montants :* ce sont des données numériques, qui pourront participer á des calculs (Prix, note …).

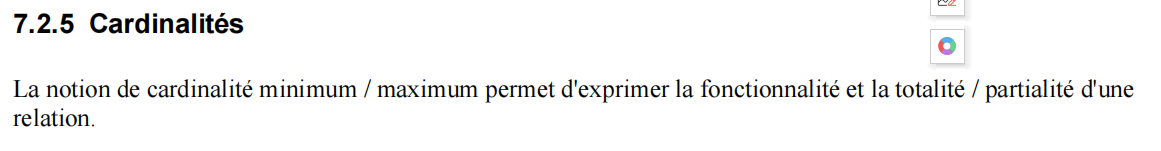


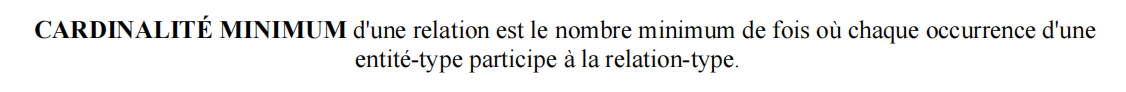


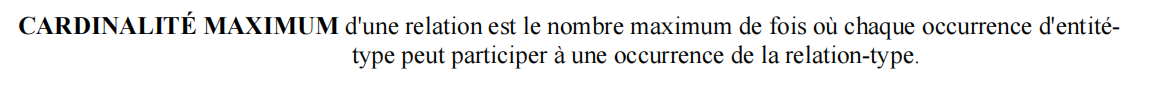


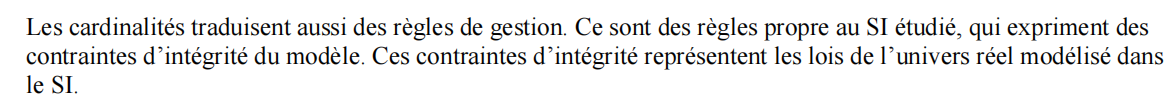




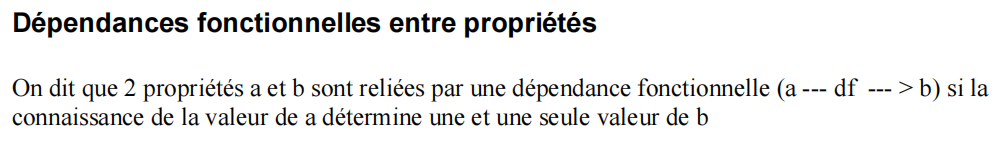






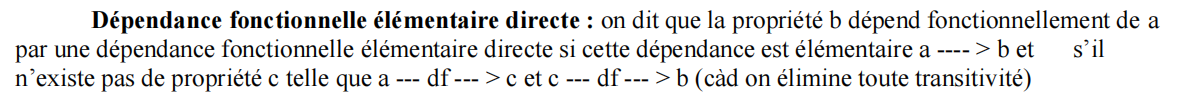


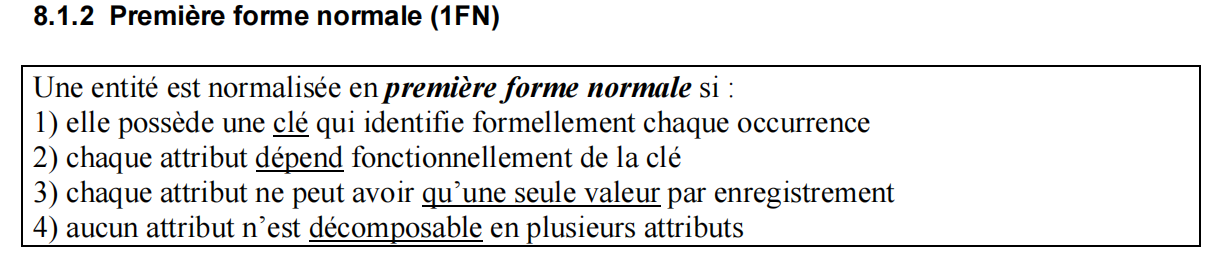


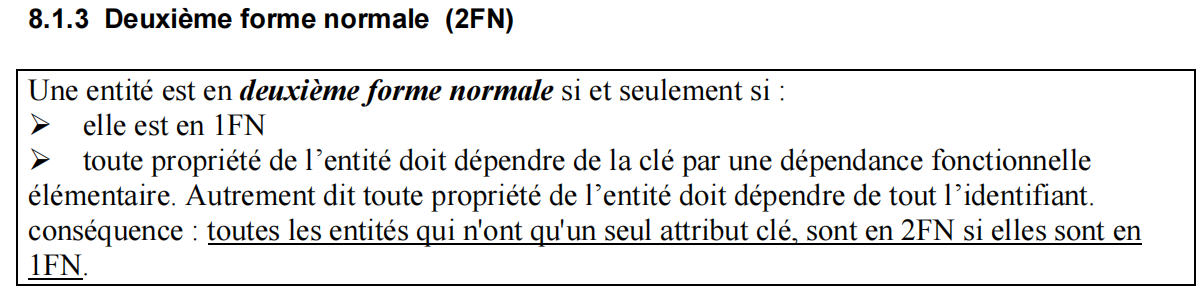


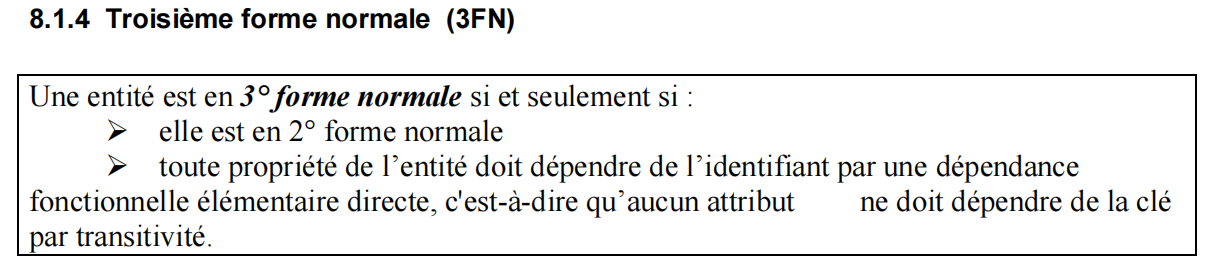
**Dépendances fonctionnelles élémentaire** : a – dfe -- > b, a ne doit être pas composable d’un élément qui ne détermine pas b.

 N’est pas une DFE.







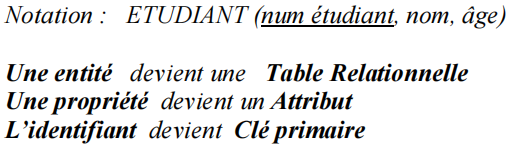


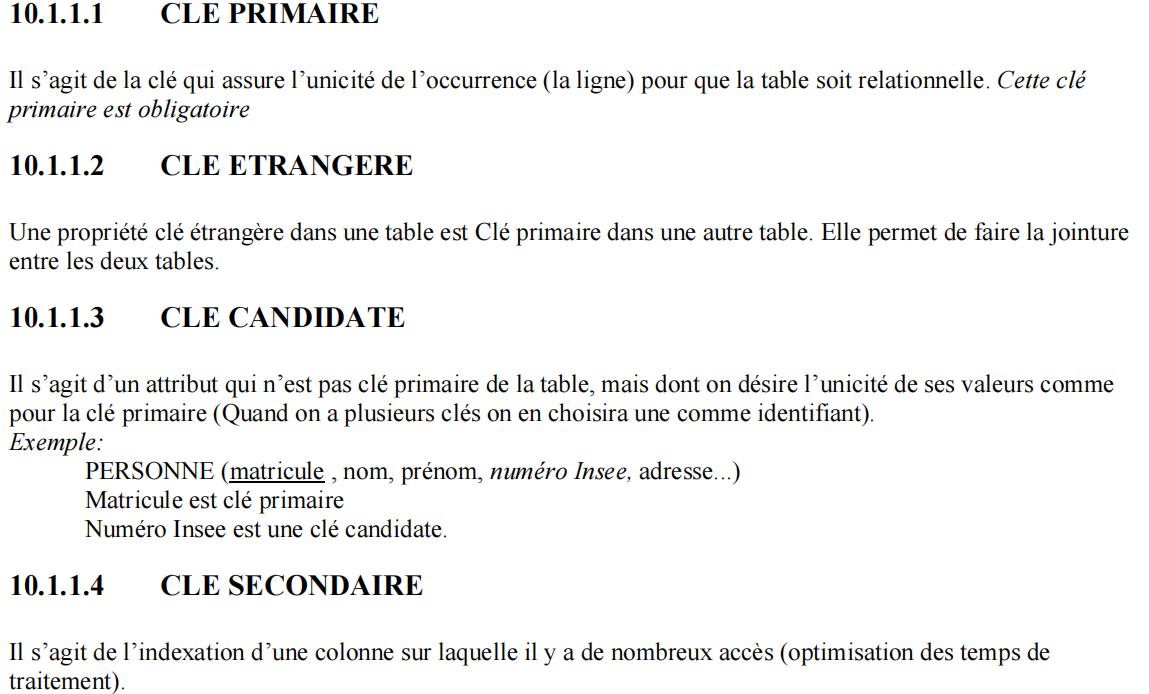
**Les contraintes d’intégrités:** un ensemble des règles permet de garantie la cohérence, et la maintenance de qualité d’une base de données.

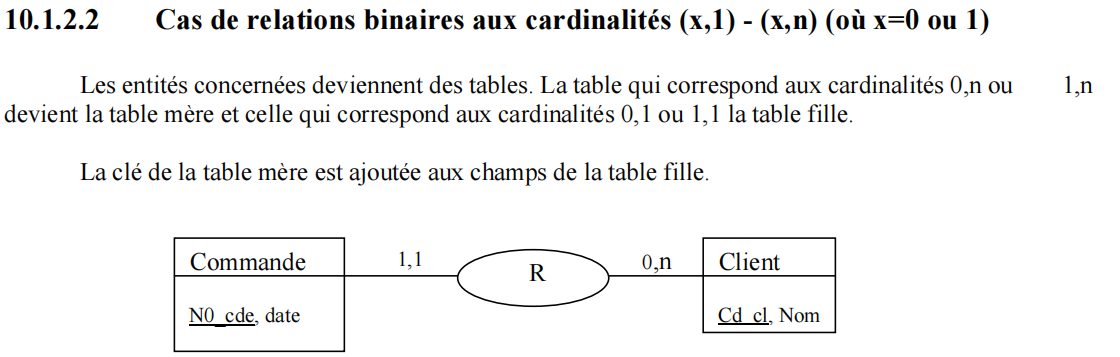
1. **Contraintes de données :** ce sont des contraintes qui s’applique sur le type de données d’une certaine colonne, par exemple une colonne définit par varchar, ne peut pas contenir un autre type de données diffèrent de varchar.
2. **Contrainte d’intégrité de l’entité (table) :** est une contrainte qui indique que la valeur d’une clé primaire d’une table ne peut pas être ni null, ni doublé.
3. **Contraintes d’intégrité référentielle :** s’applique sur une clé étrangère qui est une clé primaire dans une autre table et qui indique que ses valeurs peuvent être null, ou bien des valeurs qui existe déjà dans la table de référence.
4. **Contraintes d’intégrité fonctionnel :** est une dépendance forte entre 2 entités, dans laquelle la connaissance d’une valeur d’entité fille nous détermine une et une seul valeur entité mère, ce qui se traduit dans le modèle physique de données, par une clé étrangère dans la table fille qui référence la clé primaire de la table mère (côté fille ManyToOne, côté mère).
5. **Contrainte d’intégrité multiple :** la connaissance d’une valeur d’entité 1 détermine une ou plusieurs valeur d’entité 2, la relation se traduit par une nouvelle table a comme clé primaire la concaténation des clés primaire des 2 autres tables. (ManyToMany dans les 2 tables).

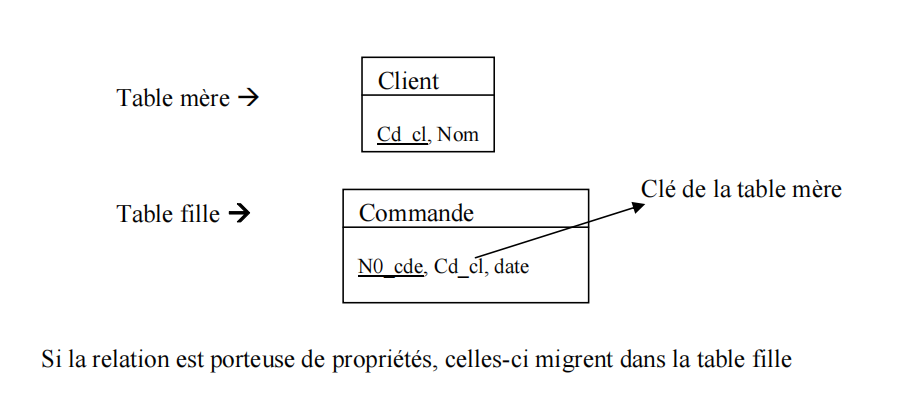
**Model logique des données**

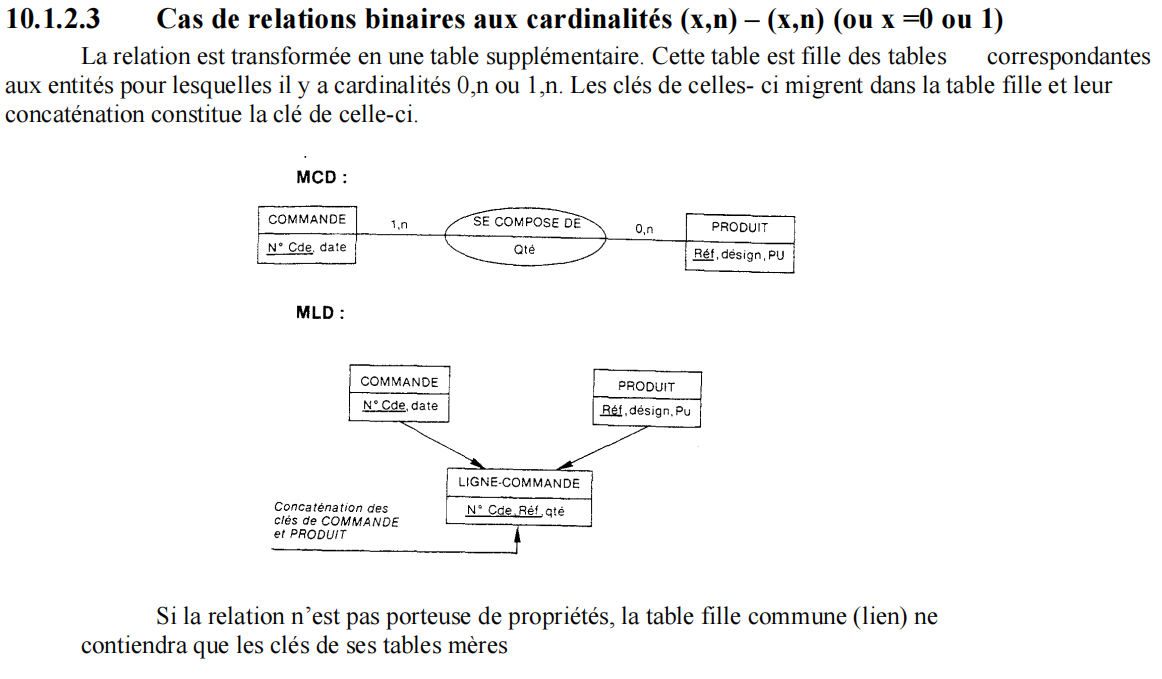
Les données seront représentées sous forme de tables appelées **tables relationnelles.**











**Model physique des données**

C'est le model par lequel on implémente notre base de données, il se fait à l'aide d'un langage s'appelle SQL, Ce langage se constitue des 3 bases principales :

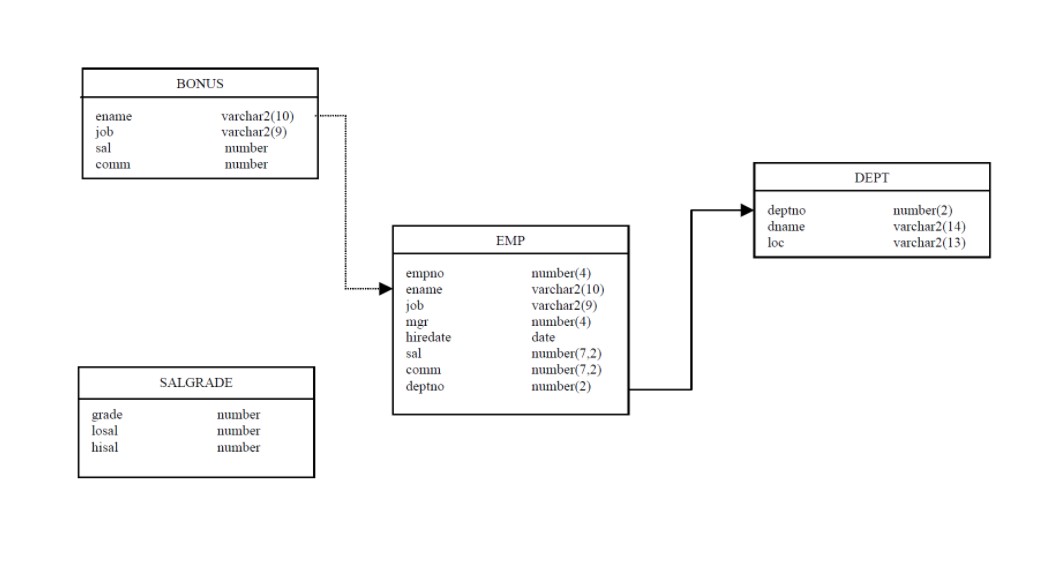
🡪 DML : Data Manipulation Language --> insertion des données dans les tables + mis-a-jour des données dans les tables + suppression des données. (Généralement ce fait sur les lignes, mais pas sur les colonnes).

🡪 DDL : Data Defintion language --> se fait sur les tables, création des tables, ajoute, modification, Et suppression, des colonnes,

🡪DCL : Data Control Language --> on l’utilise pour donner ou retirer des privilèges aux diffèrent utilisateurs, ou bien crée des utilisateurs, et leur définir un mot de passe …,et le GREANT AND REVOKE.

🡪Il y'a aussi l'analyse des données qui se fait par SELECT, et le contrôle des transactions avec (COMMIT, ROLLBACK, SAVEPOINT).

🡪Les tables dans la figure ci-dessous sont les tables avec lesquelles on va travailler le long de ce cours



Sélection des données

-SELECT \* FROM TAB; 🡪 pour afficher les tables associer à l’utilisateur connecte.

-SELECT \* FROM dept; 🡪permet de sélectionner tous les enregistrement de la table dept.

-SELECT deptno, loc FROM dept; 🡪permet de préciser les colonne qu'on veut afficher, ici on va afficher juste les colonnes deptno, et loc.

-SELECT ename, sal, sal+300 FROM emp; 🡪La colonnes sal+ 300 et virtuelle, opération sur la colonnes sal.

-SELECT ename, sal, 12\*sal "salaire annuel" FROM emp;🡪plus que l’affichage du nom, et le salaire, on affiche aussi une colonne virtuelle(n’existe pas vraiment dans la table) qui porte le nom ‘’salaire annuel’’ on multipliant le salaire par 12.

-SELECT ename||'-'||job "Employes" FROM emp ;  🡪permettre de concaténer ename et job sous une seule colonne virtuelles ‘’Employés’’.

-SELECT DISTINCT deptno FROM emp;🡪 nous permettre de sélectionner les différentes valeurs de la colonne deptno.

Restriction et tris des données

WHERE

On l’utilise souvent pour filtrer(condition) une requête sur une ensemble bien définit des données.

-SELECT ename, job, deptno FROM emp WHERE deptno = 10;🡪permet de sélectionner juste les lignes où on a deptno = 10;

-SELECT ename, job, deptno FROM emp WHERE ename = 'JAMES'; 🡪permet de sélectionner les lignes où on a ename = 'JAMES'.

-SELECT ename, job, deptno FROM emp WHERE hiredate = '03-DEC-81'; 🡪permet de sélectionner les gens engager en 03-DEC-81.

-SELECT ename, sal, comm FROM emp WHERE sal < comm; 🡪les employées ayant le salaire inférieure de leur commission.

ALL

-SELECT ename, sal FROM emp WHERE sal >= all ( 1000, 3000);🡪Permet de sélectionner les salaires qui sont supérieur a toutes la liste(ca veut dire 1000, 3000, dans ce cas) ALL nécessite l'utilisation des opérateurs.

ANY

-SELECT ename, sal FROM emp WHERE sal >= ANY ( 1000, 3000); 🡪Permet de sélectionner les salaires qui sont supérieur á au moyen une seul valeur de la liste, nécessite l'utilisation des opérateurs.

IN

-SELECT ename, deptno FROM emp WHERE deptno IN (10,20);🡪Permet de afficher les employés qui appartient aux départements 10 et 20.

-SELECT ename , mgr FROM emp WHERE mgr IN (7902, 7566, 7788);🡪Les employés qui travaille pour les managers 7902, 7566, 7788.

BETWEEN

-SELECT \* FROM dept WHERE deptno BETWEEN 20 AND 40;🡪Permet d’afficher toutes les colonnes de la table emp où le salaire entre 20 et 40

-SELECT ename, sal FROM emp WHERE sal BETWEEN 1000 AND 1500;🡪Affiche Les employés qui touchent un salaire entre 1000 et 1500.

LIKE

On utilise LIKE souvent pour chercher des chaines bien spécifier, % équivalente á n'importe quelle chaine, \_ équivalente á n'importe quel caractère.

-SELECT ename, job FROM emp WHERE ename LIKE 'S%';🡪Affiche les employés dont leur nom commence par S.

-SELECT ename, hiredate FROM emp WHERE hiredate LIKE '%81';🡪Affiche les employés engager en 1981;

IS NULL

IS NULL permet de sélectionner les champs qui peuvent être null ;

-SELECT ename FROM emp WHERE mgr IS NULL;🡪Permet de sélectionner les employés qui n’ont pas de manager.

-SELECT ename FROM emp WHERE mgr IS NOT NULL;🡪Permet de sélectionner les employés ayant un manager.

AND, OR, NOT

-SELECT ename, job, sal FROM emp WHERE job LIKE 'CLERK' AND sal > 1000;🡪Permet de sélectionner les employés ayant le job clerk et un salaire supérieur á 1000.

-SELECT ename, job, sal FROM emp WHERE job = 'CLERK' OR sal > 1000; 🡪Permet de sélectionner les employés ayant le job clerk ou un salaire supérieur á 1000.

-SELECT ename, job FROM emp WHERE job NOT ('MANAGER','CLERK'); 🡪Permet de sélectionner les employés dont leur job ni manger, ni clerk.

-SELECT ename, job, sal FROM emp WHERE job = 'SALESMAN' OR (job = 'PRESIDENT' AND sal > 1500); 🡪Permet de sélectionner les salesman où les président ayant un salaire sup a 1500.

-SELECT ename, job, sal FROM emp WHERE (job = 'SALESMAN' OR job = 'PRESIDENT') AND sal > 1500; 🡪Permet la sélectionner des salesmans ou les présidents, et ayant un salaire supérieure 1500.

ORDER BY

-SELECT ename, hiredate FROM emp ORDER BY hiredate; 🡪Permet de sélectionner les employés par ordre croissant de la date d'engagement.

-SELECT ename, hiredate FROM emp ORDER BY hiredate DESC; 🡪Par ordre décroissant.

-SELECT ename, 12\*sal "salaire annuelle" FROM emp ORDER BY 'salaire annuelle'; 🡪Permet de trier par ordre croissant du salaire annuelle des employés.

SELECT ename, 12\*sal "salAnn" FROM emp ORDER BY 2;

Pour appliquer le tri sur une colonne virtuelle, on est obligé d'utiliser un alias ‘salaire annuelle’ ou bien numéro de la colonne de la clause SELECT directement.

SELECT ename, deptno, sal FROM emp ORDER BY deptno, sal;🡪En tri d’abord par-rapport au département, si on se trouve avec une égalité, on trie alors par salaire.

DISTINCT

FONCTIONS SINGLE ROW

Une fonction single row retourne une valeur pour chaque enregistrement.

LOWER, UPPER, INNITCAP

-SELECT LOWER(ename) "nom employe" FROM emp;🡪Afficher tous les noms en minuscule.

-SELECT INITCAP(ename), UPPER(job) FROM emp;🡪Mettre en majuscule le job, et le premier lettre du nom.

CONCAT

-SELECT CONCAT(empno, ename) FROM emp;🡪Concaténer le numéro de l'employé avec son nom.

SUBSTR

-SELECT CONCAT(empno, SUBSTR(ename,1,3)) FROM emp;🡪Concaténer le No.emp avec 3 caractères de son nom.

LENGTH

-SELECT ename||' sa longueur = '||LENGTH(ename) "le nom et sa longueur" FROM emp;🡪Affiche le nom et sa longueur.

INSTR

Cherche si une chaine existe dans une autre, si oui on retourne sa position.

-SELECT ename, INSTR (ename, 'LL') FROM emp;🡪En cherche la position de 'LL', dans tous les noms de la table emp.

LPAD

Retourne une chaine, on la complétant à gauche par un symbole donné.

-SELECT LPAD(ename, 10,'-') FROM EMP;🡪Résultat: -----SMITH;

RPAD

Retourne une chaine, on la complétant à droite par un symbole donné.

-SELECT RPAD(ename, 10,'-') FROM emp;🡪Résultat: SMITH-----;

SYSDATE

-SELECT SYSDATE FROM DUAL;🡪Retourne la date courante de système, se trouve dans la table système DUAL

MONTH\_BETWEEN

Retourne le nombre des mois entre 2 dates.

SELECT MONTHS\_BETWEEN('01-SEP-95', '11-JAN-94') FROM dual;🡪Résultat: 19.6774194,

ROUND

Arrondie un nombre décimal, retourne une valeur décimale.

-SELECT ROUND((SELECT MONTHS\_BETWEEN('01-SEP-95', '11-JAN-94') FROM DUAL), 2) FROM DUAL; 🡪Résultat: 19.68.

On pout aussi arrondie aussi une date, par-rapport a un mois ou par-rapport á une année.

ROUND('25-JUL-95','MONTH') 🡪 '01-AUG-95'

ROUND('25-JUL-95','YEAR') 🡪 '01-JAN-96'

TRUNC

Pour tronquer un nombre décimal.

-SELECT TRUNC((SELECT MONTHs\_BETWEEN('01-SEP-95', '11-JAN-94') FROM dual) , 2) FROM DUAL;🡪Résultat : 19.67

On peut aussi tronquer une date, on l’appliquant par-rapport á un mois ou par-rapport á une année.

TRUNC('25-JUL-95','MONTH') 🡪'01-JUL-95'.

TRUNC('25-JUL-95','YEAR') 🡪'01-JAN-95'.

MOD

Pour calculer le reste de la division.

-SELECT ename, MOD(sal, 400) FROM emp;🡪Pour afficher le reste de la division du salaire sur 400, pour tous les employés.

ADD\_MONTHS

Ajouter un nombre de mois á une date.

-SELECT ename, hiredate ,ADD\_MONTHs(hiredate, 2) FROM emp; 🡪Pour ajouter 2 mois sur la date d’engagement.

NEXT\_DAY, LAST\_DAY

-SELECT NEXT\_DAY(SYSDATE, 'SUNDAY') FROM DUAL;🡪Retourne la date du prochain dimanche.

-SELECT LAST\_DAY(SYSDATE) FROM dual; 🡪Retourne la date de dernier jour du mois.

TO\_CHAR, TO\_NUMBER, TO\_DATE

TO\_CHAR(458) 🡪'458', TO\_CHAR(DATE) 🡪 'DATE'.

TO\_NUMBER('424') 🡪 424 (il faut que la chaine contient que des nombres).

TO\_DATE('25-JUL-94') ==> '25-JUL-94' (Converti une chaine vers une date)

Pour la fonction TO\_CHAR(date,'fmt') on peut préciser le format avec laquelle on veut afficher une date.

YYYY Année en 4 chiffres, YEAR Année en toute lettres, MM Mois en 2 chiffres, MONTH Mois en toutes lettres, DAY Jour de la semaine en lettre

NVL

Pour convertir un champ NULL en une valeur par défaut, par exemple :

UPDATE employee SET commission = NVL(commission, 100); 🡪Mettre 100 dans la commission pour tous les employés qui n’ont pas de commission.

Requêtes sur tables multiples

EQUI-JOINTURE

Deptno est une clé étrangère dans la table emp, pointe sur dept(deptno) sert á faire la liaison entre les 2 tables(emp,detp), pour par exemple afficher l’employée(ename existe dans emp), et la localisation(loc existe dans dept) où il travaille, il faut faire une jointure entre la table emp et la table dept, avec la clause WHERE suivante <<WHERE emp.deptno = dept.deptno >>

-SELECT e.empno, e.ename, d.loc FROM emp e, dept d WHERE e.deptno = d.deptno;

PRODUIT CARTESIEN

C'est une jointure dans laquelle on ne précise pas la clause WHERE, si on a par exemple N lignes dans la table a, et M lignes dans la table b, on aura NxM lignes á la sortie d’une requête de produit cartésien.

OUTER JOIN

Affiche tous les champs qui satisfait la condition, et ceux que ne satisfait pas la condition.

SELECT e.ename, e.deptno ,d.loc FROM emp e, dept d WHERE e.deptno(+) = d.deptno; 🡪Permet d'afficher même les départements qui n'emploie personne, à l’aide de l’ajoute de l’opérateur (+) á la cote inverse .

SELF JOIN

Permet de joindre les données d'une même table, si bien sûr elle contient une clé étrangère(mgr) sur sa clés-primaire(empno)

SELECT e.ename "employe" , m.ename "manager" FROM emp e, emp m WHERE e.mgr = m.empno;🡪Affiche à chaque employé, le nom de son manager.

Fonctions de groupe

La différence avec single row, les fonctions de groupe calcule un résultat à partir de plusieurs enregistrements ;

AVG, MAX, MIN, SUM

-SELECT AVG(sal) "la moyenne", MIN(sal) "salaire minimum", MAX(sal) "salaire maximum" ,SUM(sal) "la somme de salaire", COUNT(\*) "le nombre des employes" FROM emp WHERE job LIKE 'SALE%';🡪Permet d’afficher la moyenne, le min, le max, la somme de salaire pour les employés ayant le job SALES.

GROUP BY

Permet de regrouper des résultats on se basant sur les valeurs d'une certaine colonne. Par exemple afficher la somme des salaires par job

-SELECT job ,SUM(sal) FROM emp GROUP BY job;

On ne peut pas utiliser un group by avec where, ou avec des champs incompatibles(Il faut spécifier juste les champs concerner par le group by).

Regroupement sur plusieurs colonnes, par exemple pour chaque département on veut savoir la somme des salaires des différents jobs qu’il contient.

On peut dire aussi on applique le regroupement sur le job dans chaque département.

-SELECT deptno, job, SUM(sal) FROM emp GROUP BY deptno, job;

HAVING

Pour appliquer un filtrage sur le résultat de groupe by.

SELECT job, SUM(sal) FROM emp GROUP BY job HAVING SUM(sal) > 5000; 🡪Affichez les jobs dont la somme des salaires supérieur á 5000 ;

LES SOUS REQUETES

On utilise les sous-requête souvent lorsqu’on a besoin d’un résultat qui se base sur une autre requête, par exemple, quels sont les employés ayant un salaire supérieur au salaire de JONES (salaire de jones on ne le connait pas d’avance).

-SELECT ename, sal FROM emp WHERE SAL > (SELECT sal FROM emp WHERE ename = 'JONES');

SOUS-REQUETE (SINGLE ROW)

Retourne un seul enregistrement. 🡪 Pour recevoir son résultat, on Utilise des opérateurs single row (=, ! =, > …), Par exemple la requête (SELETC sal FROM emp WHERE ename = ‘JONES’) est une requête single row, car il renvoie un seul salaire, et c’est le salaire de JONES.

SOUS-REQUETE (MULTI ROW)

Possible de retourner plusieurs enregistrement (une liste). ==> Pour recevoir ses résultats, on utilise les opérateurs multi row (in, all, any).

Par exemple la requête SELECT sal FROM emp WHERE job = ‘CLERK’ ; retourne tous les salaires (ça veut dire une liste des salaires) de job clerk.

Par exemple afficher les employés ayant un job diffèrent de CLERK et un salaire inférieur á au moins á un des employés de job CLERK

-SELECT ename FROM emp WHERE JOB NOT LIKE 'CLERK' AND ( SAL < ANY (SELECT sal FROM emp WHERE JOB = 'CLERK'));

Afficher les employés avec un job différent de 'CLERK' ayant un salaire inférieure á toutes les moyennes salariales par département ;

(SELECT AVG(sal) FROM emp GROUP BY deptno) ==> retourne la moyenne salariales par départements, il suffit juste de l’utiliser dans la requête principale.

SELECT ename, job FROM emp WHERE job != 'CLERK' AND sal < ALL(SELECT AVG(sal) FROM emp GROUP BY deptno);

TABLE VIRTUELLE

Une table virtuelle est une table qui n’existe pas dans la réalité, mais c’est une sous-requête qui va jouer le rôle d’une table pour des besoins assez précis.

Comment afficher une liste des employés reprenant leur nom, salaire, et salaire moyen par leur département tel que ce salaire est supérieur au salaire de leurs départements.

On va créer une table virtuelle qui va contenir deptno (pour faire la jointure avec la table emp) et AVG (sal) la moyenne par départements avec la sous requête suivant (SELECT deptno, AVG(sal) avgsal FROM emp GROUP BY deptno). avgsal c'est le nom qu’on va utiliser pour afficher la moyenne par départements dans la requête principale.

-SELECT e.ename, e.sal, m.avgsal FROM emp e, (SELECT deptno, AVG(sal) avgsal FROM emp GROUP BY deptno) m WHERE e.deptno = m.deptno AND e.sal >m.avgsal;

VARIABLES

Les variables peut remplacer:(condition where, order by, colonne, table, un select).

Example :

-DEFINE deptname = SALES;

-SELECT \* FROM dept WHERE dname = &deptname;🡪deptname sera remplacer par sales ;

UNDEFINE nom\_variable ==> pour retirer une variable;

DML (Data Manipulation Language)

INSERTION DES DONNÉES

***LIGNE PAR LIGNE :***

-INSERT INTO dept(deptno, dname) VALUES(70,'FINANCE'); 🡪Localisation pas indiquer, par conséquence loc va prendre comme valeur, la valeur null;

-INSERT INTO dept VALUES (80,’NETWORKING’,BERLIN); 🡪Cette méthode exigé la spécification de tous les différentes valeurs des colonnes, lors de l’ajout.

***INSERTION DE PLUSIEURS LIGNE Á LA FOIS :***

-INSERT ALL

-INSERT INTO dept VALUES(70,’FINANCE’,NULL) ;

-INTO dept VALUES(80,’NETWORKING’,NULL) ;

-SELECT \* FROM dual ;

***INSERTION Á PARTIR D’UNE AUTRE TABLE :***

-INSERT INTO manager(id, name, salary, hirdate) SELECT empno, ename, sal, hiredate FROM emp WHERE job = 'MANAGER';

MIS-Á-JOUR DES DONNÉES

-UPDATE emp SET deptno = 20 WHERE empno = 7782;// 🡪Pour modifier le département de l'employer ayant le id 7782. Si on n’a pas indiqué la clause where, le mis-a-jour sera appliqué sur tous les deptno de la table.

-UPDATE emp SET deptno = (SELECT deptno FROM EMP WHERE empno = 7788) WHERE job = (SELECT job FROM emp WHERE empno = 7788) ;

Mettre les employés ayant le même job que l’employé 7788, dans le même département que lui, á l’aide d’une sous-requête.

SUPPRESSION DES DONÉES

-DELETE FROM dept WHERE dname = 'DEVELOPMENT'; 🡪Supprime les enregistrement appartient au département développement. Sans la clause WHERE la tables sera vider.

-DELETE FROM emp WHERE deptno = (SELECT deptno FROM dept WHERE dname = 'SALES'); 🡪supprime les enregistrements des employés qui appartient au département SALES, cette suppression sera faite obligatoirement par une sous-requête, car dname se trouve dans une autre table (la table dept)

GESTION DES TABLES

***CREATION EXPLICITE :***

Création de la table département.

CREATE TABLE dept(

deptno NUMBER(2) NOT NULL,

dname VARCHAR2 (14) ,

loc VARCHAR2(13)

) ;

[Citez votre source ici.]

***Á L’AIDE D’UNE SOUS-REQUETE :***

-CREATE TABLE tableName [(col1, col2,...)] AS SUBQUERY ;

Exemple :

-CREATE TABLE employee (id, name, salary) AS (SELECT empno, ename, sal FROM emp); 🡪 la table employée aura 3 colonnes, id, name, et salary basée sur les colonnes empno, ename, et sal de la table emp respectivement.

-CREATE TABLE employee AS (SELECT empno id, ename name, sal salary FROM emp); 🡪même fonctionnement de la requête précèdente, mais avec une syntaxe différente.

***ALTER TABLE :***

🡪L’ajout d’une colonne

-ALTER TABLE dept ADD loc VARCHAR2(50); 🡪Si on suppose par exemple que la table dept n’a pas de colonne de la localisation.

🡪Modification de la description d'une colonne

-ALTER TABLE dept MODIFY loc VARCHAR (100); 🡪Localisation peut maintenant contenir 100 caractères .

🡪Changement du nom d'une colonne

-ALTERT TABLE dept RENAME COLUMN loc to localisation ;

🡪Changement du nom d’une table

-ALTER TABLE dept RENAME TO departement;

-RENAME dept to departement ;

🡪Supprimer une colonne ;

-ALTER TABLE dept drop loc;

🡪Suppression d'une table :

-DROP TABLE dept;

LES CONTRAINTES

On les ajoute lors de la création des tables, ou bien après (avec ALTER TABLE ADD CONSTRAINT).

***NOT NULL :*** Les champs ayant NOT NULL á la création doivent tjr avoir une valeur, par exemple (ename VARCHAR (30) NOT NULL 🡪 ça veut dire, on doit toujours avoir un nom).

***UNIQUE KEY :*** interdit d'avoir 2 valeurs semblables dans la même colonne.

-CONSTRAINT nom\_colonne\_UK UNIQUE KEY (colonne1[, colonne 2...]). 🡪Si on le fait sur plusieurs colonnes, en interdit d'avoir 2 combinaisons colonne1colonne2 semblables.

***PRIMARY KEY :*** permet de définir la clés primaire(identifiant) d’une table, les clés primaires sont á la fois unique et á la fois not null.

-CONSTRAINT nom\_colonne\_PK PRIMARY KEY (colonne1[,colonne2]), 🡪On peut définir une clés primaire sur plusieurs colonnes.

***FOREIGN KEY :*** il s'agit d'une clé étrangère (qui est une clé primaire dans une autre table), on l'ajoute pour représenter une relation entre 2 tables.

-CONSTRAINT nom\_colonne\_FK FOREIGN KEY (colonne?) REFERENCES table? (colonne?); 🡪Il faut que la colonne qui reçoit la clé étrangère soit du même type que la clé primaire de la 2-ème table, dans notre table emp, deptno est une clé étrangère de la clé dept(deptno)

-CONSTRAINT deptno\_FK FOREIGNE KEY (deptno) REFERENCES dept(deptno);

***CHECK :*** permet d'ajouter des contraintes de vérification sur une colonne (règles de gestion).

-CONSTRAINT note\_check CHECK (note BETWEEN 0 AND 20);🡪Pour interdire d'avoir des valeurs inférieure á 0 ou des valeurs supérieur á 20 dans la colonne note.

On peut ajouter certaines contraintes après la création de la table, á l’aide de ALTER TABLE tableNom ADD CONSTRAINT, par exemple pour spécifier á chaque employés son manager, on ajoute une clé étrangère mgr (manger) qui va pointer sure empno (employé).

-ALTER TABLE emp ADD CONSTRAINT mgr\_fk FOREIGN KEY (mgr) REFERENCES emp(empno);

🡪Supprimer une contrainte :

-ALTER TABLE emp DROP CONSTRAINT mgr\_fk;

-ALTER TABLE deptno DROP PRIMARY KEY deptno\_PK CASCADE ;

🡪Désactiver une contrainte.

-ALTER TABLE emp DISABLE CONSTRAINT emp\_empno CASCADE; 🡪CASCADE pour désactiver aussi les colonnes qui ont une référence sur la clé primaire.

🡪Activer une contrainte.

-ALTER TABLE emp ENABLE CONSTRAINT emp\_empno;

Vue

On les utiliser pour représenter des données provenant d'une ou plusieurs tables, restreindre l'accès aux données (afficher des colonnes précises, Cacher la complexité d'une requête(table)).

SYNTAXE :

-CREATE [OR REPLACE] VIEW [FOCE|NOFORCE] viewname [(alias1, alias2...)] AS subquery [WITH CHECK OPTION [CONSTRAINT constraint1] [WITH READ ONLY] // 🡪Pour modifier une vue déjà existante, on précise l’option OR REPLACE.

-CREATE VIEW employee\_dept10 AS SELECT empno employee\_id, ename nom, job emploie FROM emp WHERE deptno = 10; 🡪Employee\_dept10 sera une nouvelle vue portera les colonnes employee\_id, nom et emploie, et contient seulement les enregistrements des employées appartenant au département 10.

Crée une vue deptSum contient le nom de département, salaire minimum, salaire maximum, et salaire moyenne groupé par département.

-CREATE VIEW deptSum (nom, Salaire\_minimum, Salaire\_maximum, Salaire\_moyenne) AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal) FROM emp e, dept d WHERE d.deptno = e.deptno GROUP BY d.dname;

-CREATE VIEW empvu20 AS SELECT \* FROM emp WHERE deptno = 20 WITH CHECK OPTION CONSTRAINT empvu20\_ck;

Le CHECK OPTION va interdire de changer le numéro de département vers un nombre diffèrent que 20

-drop view empvu20 ; 🡪Pour supprimer une vue.

AUTRES OBJETS SQL

LA SEQUENCE

On les utilise pour générer des valeurs bien définit (utilisable par exemple dans les clés primaires).

-CREATE SEQUENCE seq\_deptno INCREMENT BY 1 START WITH 91 MAXVALUE 100 NOCYLE NOCACHE;

CURRVAL : retourne la valeur courante de la séquence.

-SELECT seq\_dept\_deptno.currval FROM dual;

NEXTVAL : retourne la valeur suivante de la séquence.

🡪Générer une valeur pour une clé primaire(deptno).

-INSERT INTO dept (deptno, dname, loc) VALUES(seq\_deptno.NEXTVAL,'MARKETING','SAN-DIEGO');

🡪Modifier une séquence :

-ALTER SEQUENCE seq\_deptno INCREMENT BY 10 MAXVALUE 100 NOCACHE NOCYCLE;

🡪Supprimer une séquence :

-DROP SEQUENCE seq\_deptno;

LES INDEXS

On utilise les indexes pour accélérer les requêtes de recherche, on se basant sur le tri et le recherche dichotomique, afin de trouver plus rapidement le résultat voulu.

-CREATE INDEX ename\_idx ON emp(ename); 🡪Pour crée un index sur le nom de l’employé dans la table emp, facilite les recherches par noms.

-DROP INDEX ename\_idx ;🡪Pour supprimer un index.

Sécurité d'accès

Oracle gère les accès sous l'utilisation des profile (Rôle), DBA par exemple, c'est le rôle avec tous les privilèges(autorisations) possibles.

🡪Cree un utilisateur :

-CREATE USER username IDENTIFIED BY password;

🡪Un DBA peut accorder des privilèges á d'autres utilisateurs :

-GRANT privilege\_name [, privilage\_name2,...] TO user1 [, user2 ...]

🡪Des exemples des privilèges possibles : (CONNECT, CREATE SESSION, CREATE TABLE, SELECT-INSERT-DELETE-UPDATE);

🡪Regrouper plusieurs privilèges sous le même rôle.

-CREATE ROLE manager. 🡪Création d’un rôle manager pour le donner á certains utilisateurs.

-GRANT create table, create view TO manager; 🡪Pour Accorder les privilèges de crée table, et crée vue au rôle manager.

-GRANT manager TO nouveau;//🡪Permet de donner les autorisations du rôle manager á l'utilisateur nouveau.

🡪Changez le mot de passe pour un utilisateur donné.

-ALTER USER nouveau IDENTIFIED BY lion;

GRANT

🡪Un utilisateur peut donner ses privilèges pour les autres utilisateurs.

-GRANT select ON table TO user;

-GREANT update ON (userid, date\_val) ON table to user;

🡪Autoriser aussi les autres de donnes ses propre privilèges (GRANT OPTION):

-GRANT select, insert ON table TO user WITH GRANT OPTION;

-GRANT select,insert ON table TO PUBLIC🡪PUBLIC regroupe tous type d'utilisateurs.

REVOKE

Pour annuler les privilèges données (aussi les privilèges donnés par le Grant option).

-REVOKE select, insert ON TABLE FROM nouveau;

Des requête á savoir.

-SELECT table\_name FROM user\_tables; 🡪Afficher les tables de l'utilisateur courant.

-SELECT view\_name FROM user\_views; 🡪Afficher les vues de l'utilisateur courant.

-SELECT constraint\_name FROM user\_constraints; 🡪Afficher les contraintes de l'utilisateur courant.

- ALTER TABLE employee RENAME column ename to last\_name;🡪Pour renommer une colonne

-CLEAR SCREEN; 🡪Pour nettoyer la console.

-SET LINE 400 ; 🡪Permettre d’afficher proprement les données d’une table

**PSQL**

-SELECT username from dba\_users; 🡪Afficher tous les utilisateurs d’une instance oracle.

-SELECT object\_name from all\_objects 🡪 Afficher tous les objet accessible par l’utilisateur courant.

User\_xxx 🡪 pour afficher les xxx de l’utilisateur.

all\_xxx 🡪 pour afficher les xxx accessible par l’utilisateur courant.

Dba\_xxx 🡪 pour afficher les xxx de toute l’instance oracle.

*La liste de toutes les fonctions crée dans une instance donnée*

-SELECT object\_name FROM dba\_objects WHERE object\_type = ‘FUNCTION’ ;

-SELECT name FROM dba\_source WHERE type = ‘FUNCTION’ ;

*La liste des vues crée par un utilisateur donné*

-SELECT object\_name FROM user\_objects WHERE object\_type = ‘VIEW’ ;

-SELECT view\_name FROM user\_views ;

*L’utilité de dictionnaire oracle*

Le dictionnaire de données Oracle représente le cœur de la base de données. Il s'agit d'un ensemble de tables systèmes contenant des informations relatives à la structure de la base de données :

* Utilisateurs de la base (ainsi que leurs privilèges et leur rôle)
* Noms et caractéristiques des objets contenus dans la base (tables, vues, index, clusters, triggers, packages, ...)
* Contraintes d'intégrité
* Ressources physiques allouées à la base
* ...

Le dictionnaire est créé au moment de la création de la base et est mis à jour.

Il appartient à l'utilisateur *SYS*, mais l'utilisateur *SYSTEM*, c'est-à-dire l'administrateur de la base, possède des droits de lecture sur des vues du dictionnaire. Enfin le dictionnaire de données est conservé dans le tablespace *SYSTEM*.

Dictionnary[ table\_name, comments]

Dict\_columns[ table\_name, column\_name, comments]

*Peut-on se connecter au compte sys sans fournit le mot de passe ?*

Oui, parce que lors d’installation d’oracle le système crée un groupe ORA-DBA auquel le propriétaire de la machine appartient.

-CONNECT sys AS sysdba 🡪 pour se connecter sans fournir un mot de passe.

*L’intérêt de la vue de dictionary dba\_sys\_privs*

Permet de visualiser les privilèges associée á un user ou un rôle donnée en utilisant la requête :

-SELECT privilege FROM dba\_sys\_privs WHERE grantee = ‘XXX’ ;

Grantee 🡺 qui a reçu les droit sur un objet, Granter 🡺 qui a accorder les droit sur un objet, Owner 🡺 le propriétaire de l’objet.

*Un privilège :*

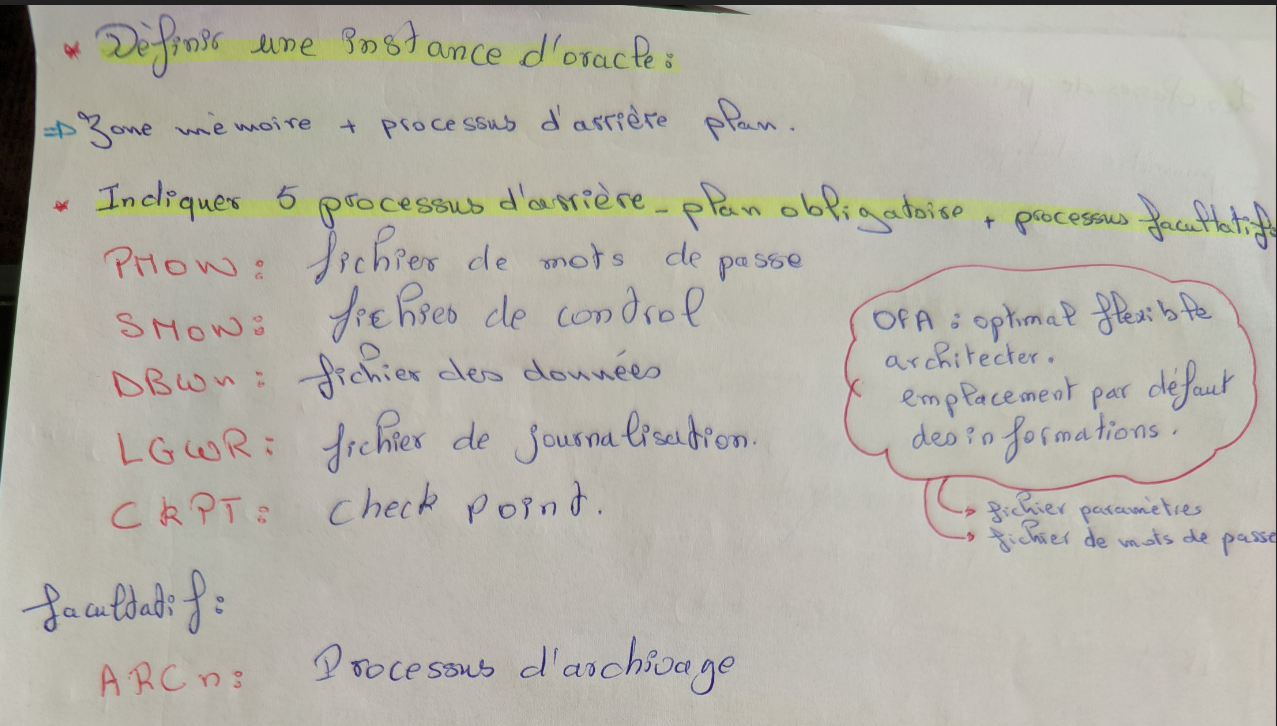
C’est le droit d’exécuter certaines commandes SQL ou le droit d’accéder á une ressource, il y’a 2 types de privilège :

Privilège système : connect, ressource, dba ….

Privilège objet : create, insert, …

*Un rôle :*

Permet de regrouper plusieurs privilèges dans une seule structure pour pouvoir les accorder toute á la fois á un utilisateur ou bien á un rôle.



*Trigger :*

Est un bloc PL/SQL associé á une table permettant de déclencher une action avant ou après un évènement d’insert, update, ou bien delete, on peut les utilise pour exiger des contraintes dynamique (salaire d’un employé ne peut pas diminuer, âge ne pas diminuer …)

Dans un trigger pour déclarer des variables on utilise DECLARE, ainsi qu’il faut obligatoirement commencer par FOR EACH ROW, par contre dans une fonction ou procédure on n’utilise ni FOR EACH ROW ni DECLARE, mais la déclaration se fait directement entre le mot clé IS et le BEGIN.

Dans une fonction après la déclaration des paramètres on indique le type de retour avec RETURN typeRetour, et cela et la seule différence avec et les procédures.

SYNTAXE:

CREATE OR REPLACE TRIGGER nom\_de\_trigger ( BEFORE | AFTER ) (INSERT | [or] UPDATE | [or] DELETE) ON une\_table

FOR EACH ROW

[DECLARE

Var1 var1%type ; Var2 var2%type ; …]

BEGIN

/\* LES INSTRUCTIONS DE TRIGGER \*/

END ;

/

DES EXEMPLES :

1-SALAIRE D’UN EMPLOYÉ NE PEUT PAS DIMINUER :

Create or Replace Trigger raise\_sal Before update On emp

FOR EACH ROW

DECLARE

    exception\_sal EXCEPTION ;

BEGIN

    IF ( :old.sal > :new.sal) then raise exception\_sal;

    END IF;

EXCEPTION

    WHEN exception\_sal THEN

    raise\_application\_error(-20100,'le nouveau salaire ne doit pas etre moin que l''ancient');

END;

/

2-Incrementation automatique de deptno avant l’insertion

CREATE OR REPLACE  TRIGGER ADDEMP BEFORE INSERT ON dept

FOR EACH ROW

DECLARE

MAXI NUMBER;

BEGIN

    SELECT MAX(DEPTNO) INTO MAXI FROM dept;

    IF (MAXI > 0) THEN

        :NEW.deptno := MAXI+1;

    ELSE :NEW.deptno := 1;

    END IF;

END;

/

3-Interdire les mis-á-jour et toute opérations les lundis

CREATE OR REPLACE TRIGGER banned\_update\_monday

BEFORE INSERT OR UPDATE OR DELETE ON dept

FOR EACH ROW

BEGIN

    IF (TO\_CHAR(SYSDATE, 'D') = 2) THEN

        raise\_application\_error(-20100,'Toute operation n''est pas autorise le lundi');

    END IF;

END;

/

Si on a un trigger qui se déclenche sur update, delete, ou insert on peut spécifier le traitement selon ces opérations á l’aide de ce bout de code.

IF UPDATING THEN /\* INSTRUCTIONS \*/ END IF ;

IF INSERTING THEN /\* INSTRUCTIONS \*/ END IF ;

IF DELETING THEN /\* INSTRUCTIONS \*/ END IF ;

*CURSOR :*

Les CURSORS sont des boucles qu’on utilise pour faire des itérations sur le résultat d’une requête SELECT ligne par ligne.

SYNTAXE :

DECLARE

CURSOR nom\_cursor IS SELECT … FROM …;

BEGIN

OPEN nom\_cursor ;

LOOP

FETCH nom\_cursor INTO var1 [, var2, …]

EXIT WHEN nom\_cursor%NOTFOUND ;

/\*INSTRUCTION DE CURSOR \* /

END LOOP ;

CLOSE nom\_cursor ;

END ;

/

EXEMPLE :

1-Cursor qui boucle et affiche la table dept

DECLARE

CURSOR cur IS SELECT \* FROM dept;

v\_ligne cur%ROWTYPE;

BEGIN

    OPEN cur;

    LOOP

        FETCH cur INTO v\_ligne; EXIT WHEN cur%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('numéro de département '||v\_ligne.deptno||', nom de département '

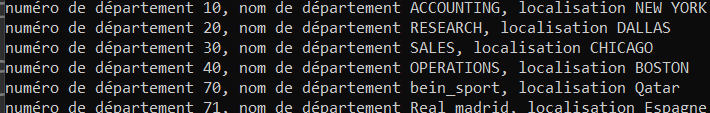
        ||v\_ligne.dname||', localisation '||v\_ligne.loc);

    END LOOP;

    CLOSE cur;

END;

/



2-Cursor qui affiche le nom, job et salaire des employés de département 10 de king ;

DECLARE

CURSOR cur IS SELECT ename, job, sal FROM emp WHERE deptno = 10;

nom emp.ename%TYPE;

travaille emp.job%TYPE;

salaire emp.sal%TYPE;

BEGIN

    OPEN cur;

    LOOP

        FETCH cur INTO nom, travaille, salaire ; EXIT WHEN cur%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('nom de l''employe '||nom||', son travaille '

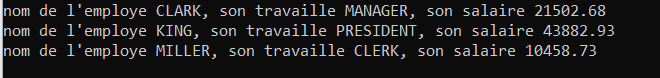
        ||travaille||', son salaire '||salaire);

    END LOOP;

    CLOSE cur;

END;

/



*PROCEDURE :*

SYNTAX:

CREATE OR REPLACE PROCEDURE nom\_procedure([var1 (IN | OUT) typ1Var1, …]) IS

[var typevar; cursor cur …]

BEGIN

/\* INSTRUCTIONS DE PRCEDURE \*/

END;

/

EXAMPLE:

1-Procédure qui crée un utilisateur en lui donnant les rôle connect et resouce.

CREATE OR REPLACE PROCEDURE ADD\_USER(v\_user IN VARCHAR2,v\_password IN VARCHAR2) IS

BEGIN

    EXECUTE IMMEDIATE ('CREATE USER '||v\_user||' IDENTIFIED BY '||v\_password);

    EXECUTE IMMEDIATE 'GRANT connect, resource to '||v\_user;

END;

Execute immedaite utiliser dans les procédures et fonctions lorsqu’on veut exécuter une requête de DDL (CREATE, DROP …), OU DCL (GRANT, REVOKE …).

Pour exécuter cette procédure 🡪 execute ADD\_USER(‘USER1’,’1234’) ;

2-Procedure qui affiche nom, profession, et département de tous les employés.

CREATE OR REPLACE PROCEDURE Q3 IS

        v\_ename emp.ename%TYPE;

        v\_job emp.job%TYPE;

        v\_deptno dept.deptno%TYPE;

        cursor empCur is SELECT ename, job, deptno FROM emp;

    BEGIN

            OPEN empCur;

            LOOP

            FETCH empCur INTO v\_ename, v\_job, v\_deptno;

            EXIT  WHEN empCur%NOTFOUND;

            DBMS\_OUTPUT.PUT\_LINE('l''employe '||v\_ename||' a la profession '||v\_job||' dans le departement '||v\_deptno);

            END LOOP;

            CLOSE empCur;

    END;

/

3-Procedure qui augmente les salaires des employés en fonction de leur grade (1,2 avec 10%, 3 avec 15%, 4,5 avec 20%)

CREATE OR REPLACE PROCEDURE Q6 IS

v\_empno emp.empno%TYPE;

v\_grade salgrade.grade%TYPE;

CURSOR cur IS SELECT empno, grade FROM emp, salgrade WHERE sal BETWEEN losal AND hisal;

BEGIN

    OPEN cur;

    LOOP

    FETCH cur INTO v\_empno, v\_grade;

    EXIT WHEN cur%NOTFOUND;

    IF (v\_grade IN (1,2)) THEN UPDATE emp SET sal = sal\*1.1 WHERE empno = v\_empno;

    ELSIF (v\_grade = 3) THEN UPDATE emp SET sal = sal\*1.15 WHERE empno = v\_empno;

    ELSE UPDATE emp SET sal = sal\*1.2 WHERE empno = v\_empno;

    END IF;

    END LOOP;

END;

/

*Function :*

CREATE OR REPLACE FUNCTION nom\_FUNCTION([var1 (IN | OUT) typ1Var1, …]) RETURN type\_de\_retour IS

[var typevar; cursor cur …]

BEGIN

/\* INSTRUCTIONS DE PRCEDURE \*/

Return une\_variable;

END;

/

Example:

1-Fonction qui retourne le salaire maximum dans un département

CREATE OR REPLACE FUNCTION salMaxInDept(v\_deptno IN number) return number is

v\_max number(5);

BEGIN

    SELECT MAX(sal) INTO v\_max FROM emp WHERE deptno = v\_deptno;

    return v\_max;

END;

/

Pour exécuter cette fonction 🡪 SELECT salMaxInDept(10) from dual ;

Dual est une table virtuelle utiliser pour compléter les requêtes qui n’appartient á aucune table.

2-Fonction qui retourne le nom de manager d’un employé, s’il n’a pas un manager alors il faut afficher aucun.

CREATE OR REPLACE FUNCTION Q5(v\_empno IN emp.empno%TYPE) return emp.ename%TYPE IS

v\_ename emp.ename%TYPE;

v\_mgr emp.mgr%TYPE;

BEGIN

    SELECT mgr INTO v\_mgr FROM emp WHERE empno = v\_empno;

    if v\_mgr IS NOT NULL THEN

    SELECT ename INTO v\_ename FROM emp WHERE empno = v\_mgr;

    ELSE

    v\_ename := 'Aucun';

    END IF;

    return v\_ename;

END;

/

