LABO 1 :

1.
```
select empnom
from gcuv.employe
where empsal > 85000
```

2.
```
Select empnom, empdpt
from gcuv.employe
where empnom LIKE '%ON%'
```

3.
```
select count(*)
from gcuv.employe
where empsexe = 'F'
```

4.
```
select count(*), empdpt
from gcuv.employe
group by empdpt
```

5.
```
select empdpt, AVG(empsal)
from gcuv.employe
group by empdpt
having AVG(empsal) > 85000
```

6.

7.
```
select dptadm, count(*)
from gcuv.departement
group by dptadm
having count(*) >= 2
```

8.
```
Select Distinct count(dptmgr)
from gcuv.departement
```

9.
```
select empnom, dptlib
from gcuv.employe
join gcuv.departement on empdpt=dptno
```

10.
```
select empnom
from gcuv.employe
join gcuv.departement on empno=dptmgr
```

11.

```
Select count(*), dptlib
from gcuv.departement
join gcuv.employe on empdpt = dptno
group by dptno, dptlib
```

12.

```
select dptlib, SUM(empsal)
from gcuv.departement
join gcuv.employe on empdpt= dptno
group by dptlib
```

13.

```
select dptmgr, count(*)
from gcuv.departement
join gcuv.employe on empdpt = dptno
group by dptmgr
```

14.

```
select dptlib
from gcuv.departement
join gcuv.employe on empno = dptmgr
where empsexe = 'F'
group by dptlib
having count(*) >= 2
```

15.

```
select mgr.empnom, count(e.empno)
from gcuv.departement
join gcuv.employe mgr on dptmgr = mgr.empno
join gcuv.employe e on dptno = e.empdpt
group by mgr.empnom
having count(e.empno) > 5
```

16.

```
select mgr.empnom, count(e.empno)
from gcuv.departement
join gcuv.employe mgr on dptmgr = mgr.empno
join gcuv.employe e on dptno = e.empdpt
group by mgr.empnom
having count(e.empno) > 5
```

17.

```
Select dt.dptlib, count(*)
from gcuv.departement dt
join gcuv.departement dp on dp.dptadm=dt.dptno
group by dt.dptlib
```

18.

```
select dptlib from gcuv.departement where dptadm is not null
```

```
SELECT directed.dptLib "Départements dirigés par un autre département"
from gcuv.departement directed
where directed.dptadm != directed.dptno
```

19.

LABO 2

1.

```
select empnom, empsal
from gcuv.employe
where empsal = (Select Max(empsal) from gcuv.employe)
```

2.

```
select empnom, empsal
from gcuv.employe
where empsal > (Select AVG(empsal) from gcuv.employe)
```

3.

```
select empnom, empsal
from gcuv.employe
where empsexe = 'F' and empsal > (Select AVG(empsal) from gcuv.employe where empsexe =
'M')
```

4.

```
select dptlib
from gcuv.departement
join gcuv.employe on dptno = empdpt
group by dptlib
having count(*) = (select max(count(empno)) from gcuv.employe group by empdpt)
```

5.

```
select dptlib
from gcuv.departement
join gcuv.employe on dptno = empdpt
group by dptlib
having sum(empsal) = (select max(sum(empsal)) from gcuv.employe group by empdpt)
```

6.

```
select empno
from gcuv.employe
where empno not in (Select dptmgr from gcuv.departement join gcuv.employe on dptmgr =
empno)
group by empno
```

7.

```
select count(distinct dptmgr)
from gcuv.departement
```

8.

```
select e.empno
from gcuv.employe e
where e.empsal > (select avg(empsal) from gcuv.employe eme where e.empdpt = eme.empdpt
group by e.empdpt)
```

9.
a.

```
Select empdpt
from gcuv.employe e
group by empdpt
having avg(empsal) > (select (1.1 * avg(empsal))
from gcuv.employe e2
where e2.empdpt != e.empdpt)
```

b.

```
SELECT dptNo, dptlib,dptadm
from gcuv.departement d1
join gcuv.employe on dptNo = empDpt
group by dptNo, dptlib,dptadm
having avg(empSal) >= 1.1* (SELECT avg(empSal)
from gcuv.departement d2
join gcuv.employe on dptNo = empDpt
where d1.dptNo != d2.dptNo);
```

10.

```
select dptlib
```

```
from gcuv.departement
join gcuv.employe on empdpt = dptno
group by dptlib, dptadm
having sum(empsal) > ( Select sum(empsal) from gcuv.employe e1 where dptadm =
e1.empdpt)
```

LABO 3 :

1.

a.

```
select distinct d1.dptlib , d1.dptadm
from gcuv.departement d1
left join gcuv.departement d2 on d1.dptno=d2.dptadm
where d1.dptadm is not null
```

b.

```
SELECT dp.dptlib administré, sup.dptlib supérieur
FROM gcuv.departement dp
LEFT JOIN gcuv.departement sup ON dp.dptadm = sup.dptno
ORDER BY supérieur
```

2.

a.

```
SELECT sup.dptlib supérieur, dp.dptlib administré
FROM gcuv.departement dp
right JOIN gcuv.departement sup ON sup.dptno = dp.dptadm
where dp.dptno is null
```

b.

```
Select chef.dptno, departement.dptno
from gcuv.departement chef
left join gcuv.departement departement on chef.dptno = departement.dptadm
where departement.dptno is null
```

2ème partie :

1.

```
select empnom
from gcuv.employe
join gcuv.departement on dptmgr=empno
where empdpt != dptno
```

2.

```
select avg(empsal)
from gcuv.employe
join gcuv.departement on empno = dptmgr
```

3.

```
select avg(empsal)
from gcuv.employe
join gcuv.departement on dptmgr != empno
```

4.

```
Select dptmgr
from gcuv.departement
group by dptmgr
having count(dptmgr) >=1;
```

5.

a.

```
select empno, empnom
from gcuv.employe
join gcuv.departement on dptmgr = empno
where empdpt != dptno
```

b.

```
Select empnom, empno
from gcuv.employe e
join gcuv.departement on dptmgr = e.empno
where e.empdpt != any (select dptno from gcuv.departement where dptmgr = e.empno);
```

6.

a.

```
select dptlib, count(*)
from gcuv.departement
join gcuv.employe on dptno = empdpt
where empnom like 'D%' OR empnom like 'M%'
group by dptlib
having count(*)>=1
```

b.

```
select distinct dptlib
from gcuv.departement
join gcuv.employe on dptno = empdpt
where empnom like 'D%' OR empnom like 'M%'
```

c.

```
select distinct dptlib
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like ('D%')
union
select distinct dptlib
```

```
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like 'M%'
```

7.

```
select distinct dptlib
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like 'D%'
intersect
select distinct dptlib
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like 'M%'
```

```
SELECT distinct dptLib
from gcuv.departement
join gcuv.employe e1 on dptNo = e1.empDpt
join gcuv.employe e2 on dptNo = e2.empDpt
where e1.empNom LIKE 'D%' AND e2.empNom LIKE 'M%'
```

8.

```
select distinct dptlib,empnom
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like 'D%'
minus
select distinct dptlib
from gcuv.departement
join gcuv.employe on empdpt = dptno
where empnom like 'M%'
```

9.

```
select dptlib, count(empsexe)
```

```
from gcuv.departement
left join gcuv.employe on empdpt = dptno
where empsexe = 'F' OR empnom is NULL
group by dptlib, empsexe
```

```
select dptlib, count(empsexe)
from gcuv.departement
left join gcuv.employe on empdpt = dptno
group by dptlib, empsexe
having empsexe NOT LIKE 'M'
```

Supplément :

Listez pour chaque département le nom de l'employé ayant le salaire minimum parmi tous ses employés dans son département.

```
select empdpt , empnom, empsal
from gcuv.employe
join gcuv.departement d on d.dptno = empdpt
group by empdpt , empnom , empsal
having empsal = (select min (empsal) from gcuv.employe e
 where d.dptno = e.empdpt )
```

LABO 4 :

1.

```
select nom, prenom
from ancien
where sexe = 2;
```

2.

```
select count(*)
from ancien
where adLoc = 'Uccle';
```

3.

```
select sexe, count(*)
from ancien
group by sexe;
```

4.

```
select nom, prenom
from ancien
join LocaliteBelge on cp = adCp
where region = 2 ;
```

5.

```
Select nom, prenom
from ancien
join Nationalite on adPays=iso2
where payUe = 0
order by nom ;
```

6.

```
Select promotion, count(*)
from diplome
group by promotion;
```

7.

```
select ent.nom, count(*)
from entreprie ent
join ancien anc on anc.entreprise = ent.id
group by ent.nom
order by count(*) desc ;
```

8.

a.

```
Select di.promotion
from diplome di
join ancien  an on di.ancien = an.id
where an.sexe = 2
group by di.promotion
Having count(*) >=3 ;
```

b.

```
Select di.promotion
from diplome di
join ancien  an on di.ancien = an.id
group by di.promotion, an.sexe
Having count(*) >=3 AND an.sexe = 2;
```

9.

a.

```
Select nom, secteur , sec.libelle
from entreprise
join secteur sec on secteur=sec.id
```

b.

```
 Select nom, secteur
from entreprise
```

10.

```
select ancien
from diplome
group by ancien
having count(*)=2
```

11.

```
Select di.promotion, an.nom, an.pays, ent.pays
from ancien an
join diplome di on di.ancien = an.id
join entreprise ent on an.entreprise = ent.id
where di.promotion >1987 AND an.adPays != ent.pays
```

12.

a.

```
select *
from ancien an
join diplome di on an.id = di.ancien
where
di.promotion = (Select promotion from diplome join ancien on ancien = id where nom = 'DEE' and
prenom = 'Jacques' )
and
```

di.section = (Select secteur from entreprise join ancien on entreprise = id where nom = 'DEE' and prenom = 'Jacques')

b.
select id from ancien join diplome on ancien = id group by id having section = (select section from diplome d where id = d.ancien and nom = 'DEE' and prenom = 'jacquees') and promotion = (select promotion from diplome d2 where d2.ancien = id and nom = 'DEE' and prenom = ' jacquees')

13.

```
select secteur, count(*) as nbrEmployés
from ancien an
join entreprise ent ON an.entreprise = ent.id
group by secteur
```

14.

```
select di.promotion, di.section , count(*)
from diplome di
join ancien an ON di.ancien = an.id
group by di.promotion, di.section
```

15.

a.

```
select titre, nom, prenom, adRue, adLoc, adCp, naLibelle
from  ancien
join diplome di ON di.ancien=id
join Nationalite na ON na.iso2 = adPays
group by titre, nom, prenom, adRue, adLoc, adCp, naLibelle, di.promotion
Having  di.promotion between 1985 AND 1995
```

b.

```
select titre, nom, prenom, adRue, adLoc, adCp, naLibelle
from  ancien
join diplome di ON di.ancien=id
join Nationalite na ON na.iso2 = adPays
where di.promotion between 1985 AND 1995
```

16.

a.

```
select di.promotion
from diplome di
group by di.promotion
having count(*) = (Select max(count(*))
```

```
                    from ancien an
                    join diplome d2 on an.id = d2.ancien
                    where an.sexe = 2  AND d2.section = 'G')
```

b.

```
select d1.promotion
from diplome d1
join ancien a1 on d1.ancien = a1.id
where a1.sexe = 2 and d1.section = 'G'
group by d1.promotion
having count(d1.ancien) = (select max(count(d.ancien))
                                from diplome d
                                where d.ancien = a1.id and a1.sexe = 2 and d.section = 'G'
                                 group by d.promotion)
```

LABO 5 :

1. Listez tous les départements qui ne sont pas administrer par un autre département.

1) Le nombre de département qui ne sont pas administrer par un autre département.

2. Listez les numéro et le nom des employés qui ont un manager avec comme prénom MAES.

2) Le nom et le n° des employés appartenant à un département dirigé par une personne dont le nom est MAES.

3. listez les libellés et les numéros de départements ayant le plus femmes comme employés.

3) Sélectionne le n° et le libéllé du département qui compte le plus de femme dans ses employés.

4) Sélectionne les numéro et les noms des employés qui ne sont pas managers.

5. listez les numéro et les noms des employés dont leur salaire est inférieur à n'importe employé de sexe féminin travaillant dans le même département.

Sélectionne les employés ayant un salaire inférieur au salaire minimum d’un employé de sexe féminin de leur propre département.

6. listez les libellés et les numéros des départements ayant un manager avec un salaire inférieur aux salaire des tous les managers

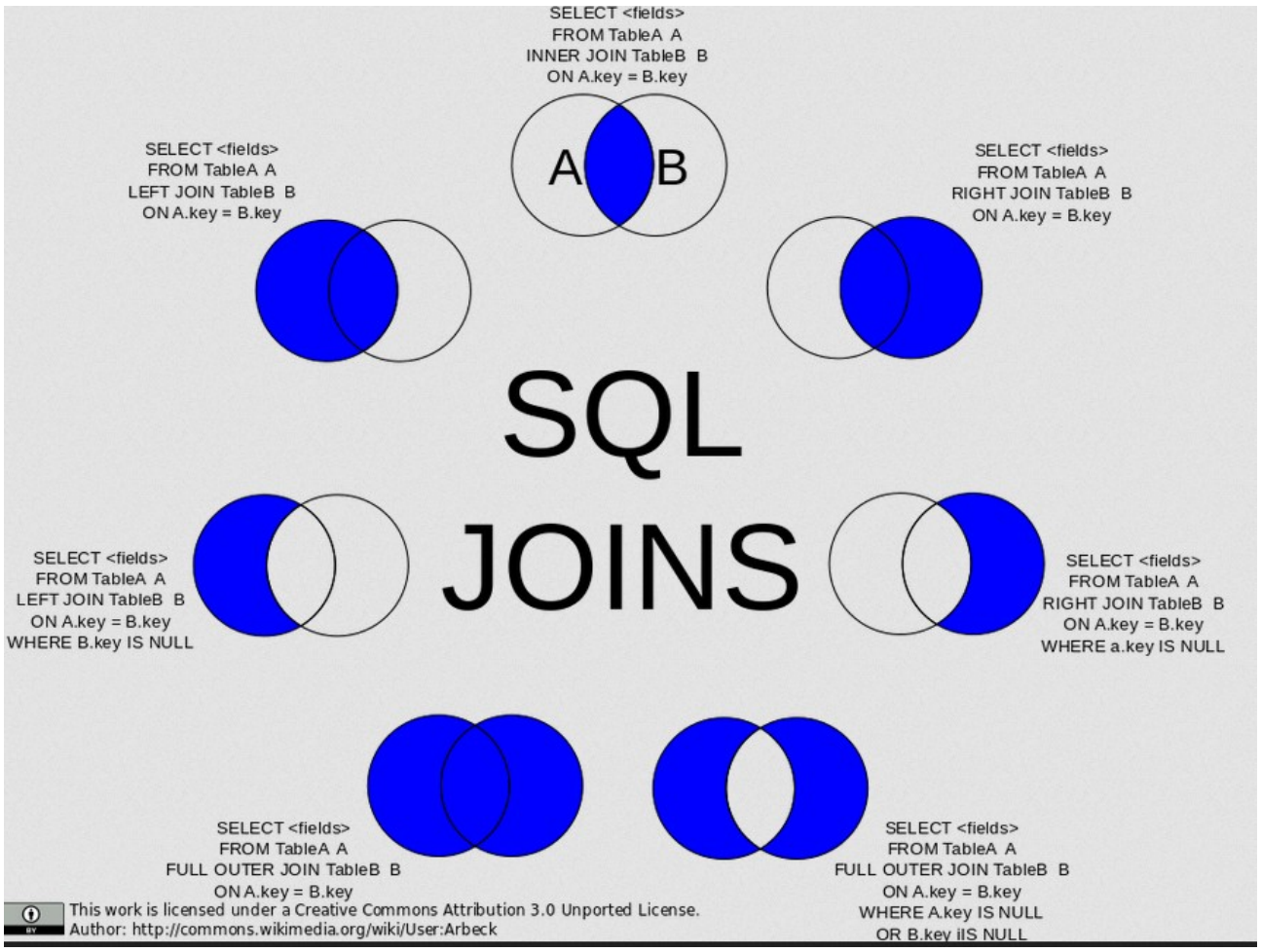
6) Selectionne le numéro et le libéllé des département des employe qui sont managers et dont le salaire est inférieur ou egale à tout les salaires des employé qui sont managers

7. listez les numéros de département ayant au moins 4 employés du même genre et d'un salaire supérieur à 110 000

7) Les départements ayant au moins 4 employés de meme sexe avec un salaire supérieur à 110000

8. Afficher les départements ayant un manager de sexe différent qu'un employé dans le même département mais étant un manager d'un autre département

b. On veut les employés de sexe différent qui sont managers qui travail dans le même département.



INTERRO BLANC : SELECT : (déjà fait) illimité

TD6 :

1. créer une table

```
create table Recette (
    rid int not null,
    lib varchar(30) not null,
    tPrep decimal (5,2) not NULL,
    nbPerso int default 4 not null);
```

a. créer une table avec contrainte sur ID et auto increment de l'ID

```
create table Recette (
```

```
    rid int GENERATED ALWAYS AS IDENTITY not null ,
    lib varchar(30) not null,
    tPrep decimal (5,2) not NULL,
    nbPerso int default 4 not null,
    constraint ridPK primary key(rid));
```

Supprimer une table

```
drop table matable;
```

1.1 Insérer des Tuples

- par tuple ;

```
insert into recette values (1,'Poulet tandori',30,5);

insert into recette(rid,lib,tPrep) values (1,'Poulet tandori',30);
```

-par plusieurs tuples ;

```
insert into maTable (tLib,tNb1, tNb2)
select 'truc',45.2,56.2 from dual union all
select 'truc',45.2,56.2 from dual;
```

2. Ajout de contraintes

Remarque : l'ajout d'une contrainte n'est accepté que si les données déjà présentes la respectent.

```
ALTER TABLE nomTable listes-de-clauses-de-modification
```

- ADD : ajout d'attribut, de contrainte
- MODIFY : modification d'attribut
- DROP : suppression d'attribut, de contrainte

```
ALTER TABLE Employe ADD empDateNaissance DATE;
ALTER TABLE Employe MODIFY empDateNaissance NOT NULL;
ALTER TABLE Employe DROP COLUMN empDateNaissance;
ALTER TABLE Employe ADD CONSTRAINT salPositifCheck
                                   CHECK (empSal > 0) ;
ALTER TABLE Employe DROP CONSTRAINT salPositifCheck ;
```

1. **clé primaire** pour ID :

```
alter table maTable add constraint idPK primary key (id);
```

2. condition : supérieur à 0 :

```
ALTER TABLE matable ADD CONSTRAINT CK_SupZero CHECK (tNb1>0);
```

3.condition : tNb2 doit être supérieur à tNb1 :

```
ALTER TABLE matable ADD CONSTRAINT CK_Nb2SupNb1 CHECK (tNb2>tNb1);
```

4. condition : deux tuples de ne peuvent pas avoir les mêmes valeurs pour la paire tNb1 et tNb2.

```
ALTER TABLE matable ADD CONSTRAINT UC_Nb1AndNb2 UNIQUE (tNb1,tNb2);
```

1. **clé étrangère** pour tLib :
Remarque : l'ajout d'une clé étrangère n'est accepté que si la valeur référence est du même type.

- Pour faire référence d'une colonne vers une autre colonne de la même table, il faut soit :
    1.

```
create table maTable (
    tid int not null ,
    tLib varchar(30) not null,
    tNb1 decimal (5,2) DEFAULT 12 not NULL,
    tNb2 decimal (8,2) not NULL,
    numero int not null,
    valuee int not null,
    constraint UC_test UNIQUE (valuee),
constraint PK_test primary key(tid),
 constraint FK_test FOREIGN key (valuee) REFERENCES maTable(tid)

    );
```

2. clé étrangère d'une autre table :

```
ALTER TABLE montable
ADD FOREIGN KEY (t2Ref) REFERENCES maTable(tid);
```

- Activé une contrainte (déjà existante) :

```
ALTER TABLE table_name
ENABLE CONSTRAINT primary_key_constraint_name;
```

-Désactivé une contrainte(déjà existante) :

```
ALTER TABLE table_name
DISABLE CONSTRAINT primary_key_constraint_name;
```

-Supprimer une contrainte

```
alter table maTable drop constraint FK_test;
```

```
- visualisé les contraintes existantes :
```

```
select * from user_constraints;
```

-Commentaire d'une colonne ou d'une table :

```
COMMENT ON TABLE nomTable IS 'Description de la table'
COMMENT ON COLUMN nomTable.nomColonne IS 'Description de la colonne'
```

6. ON DELETE CASCADE : permet la suppression des tuples lié par la clé étrangère

ex : supprimer recette numéro 1 supprimera tous les produits de cette recette.

```
alter table monTable
    add CONSTRAINT t2Refrences
    FOREIGN KEY(t2Ref) REFERENCES matable (tId) on delete cascade
```

Modification de comportement en cas de suppression d'un tuple reprenant une clé référencée

```
[CONSTRAINT nomContrainte]
    [FOREIGN KEY (les att. de la FK)]
    REFERENCES nomTableCible [(les att. Clé référencée)]
    [ ON DELETE {RESTRICT | SET NULL | CASCADE} ]
```

- **RESTRICT** est l'option par défaut qui correspond au refus de suppression
- **SET NULL** demande, en cas de suppression du tuple référencé, de remplacer la valeur de la FK par NULL
- **CASCADE** demande en cas de suppression du tuple référencé de supprimer les tuples qui y font référence

8. DEFERRABLE INITIALLY DEFERRED : permet de créer des contraintes qui ne seront actifs seulement à partir du prochain commit.

```
create table myTable ( city VARCHAR2(20 ) ) ;
insert into myTable values ('New-York');

ALTER table myTable
modify city constraint invalid_city_ck_nn NOT NULL
DEFERRABLE INITIALLY DEFERRED;
```

On pourra ajouter autant de null que l'on veut AVANT de commit.

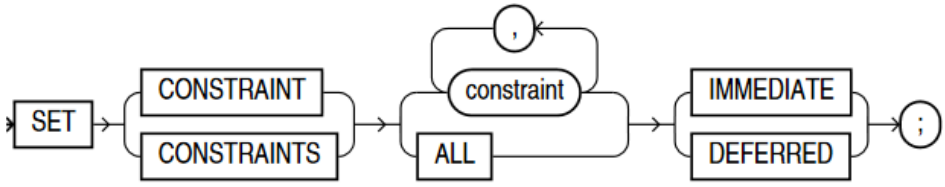
9. Scrpit : exemple deferrable avec commit (termine avec erreur logique)

```
create table mtable (
    mid int,
    lid varchar(50));

ALTER TABLE mtable modify lid varchar(50) not null
DEFERRABLE INITIALLY DEFERRED;

insert into mtable (mid,lid) values (1,null);

commit;
```

10. DEFERRABLE INITIALLY IMMEDIATE : permet de créer des contraintes qui ne seront actifs une fois exécuté.

```
ALTER TABLE foo
  ADD CONSTRAINT foo_bar_fk
  FOREIGN KEY (bar_id) REFERENCES bar (id)
  DEFERRABLE INITIALLY IMMEDIATE; -- the magic line
```

11. SET CONSTRAINTS ALL IMMEDIAT : permet de set toutes les contraintes existantes un fois exécuté.



3. Création du Schéma conceptuel

1 . créer scpit employe/Departement

```sql
create table employe(
    empNo char(3) not null constraint empnoPk primary key,
    empNom varchar(30) not null,
    empSexe char(11) not null,
    empsal numeric(18) not null,
    empDpt char(3) not null);



create table departement(
    dptno char(3) not null constraint dptPk primary key,
    dptLib varchar(30) not null,
    dptMgr char(3) not null,
    dptAdm char(3));


-- clé etranger
        alter table employe
        add constraint empDptFK FOREIGN KEY(empDpt) REFERENCES Departement (dptNo)

        -- clé etranger
        alter table departement
        add CONSTRAINT dptAdmFK FOREIGN KEY(dptAdm) REFERENCES Departement (dptNo)

        -- clé etranger
        alter table departement
        add CONSTRAINT dptmgrFK FOREIGN KEY(dptMgr) REFERENCES employe (empno);
```

2. populer les tuples des tables employe/Departement

```
ALTER TABLE employe DISABLE CONSTRAINT empDptFK;
ALTER TABLE departement DISABLE CONSTRAINT dptAdmFK;
ALTER TABLE departement DISABLE CONSTRAINT dptmgrFK;

insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('020','DURANT','M',104000,'E21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('030','SMITH','M',118900,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('050','GOULD','M',100800,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('060','ALLEN','F',110200,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('070','DANOIS','M',140200,'D21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('090','VAN HE','F',100200,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('100','HIEL','M',110300,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('110','KOALA','M',95300,'A00');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('120','Mc CARTNEY','M',85300,'A00');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('130','ALBERT','M',92340,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('140','LENNON','M',64230,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('150','STARR','M',104235,'E21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('160','HARRISSON','M',84430,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('170','LISA','F',93456,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('180','MONA','F',93456,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('190','LUDI','M',72654,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('200','VAN BLAER','M',92654,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('220','DE COO','M',92654,'D11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('230','DANTE','M',50654,'D21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('240','MANTE','F',60654,'D21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('250','DATTA','M',100654,'D21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('260','MARKA','M',80654,'C01');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('270','PEREZ','F',110654,'D21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('280','MOZART','M',18654,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('290','BOURVIL','M',98654,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('300','DELON','M',139054,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('310','TOOR','M',69054,'E11');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('320','MAES','F',99054,'E21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('330','VAN RAES','F',56054,'E21');
insert into employe (empno, empnom, empsexe, empsal, empdpt) values ('340','GIELS','M',86054,'E21');

insert into departement (dptno, dptlib, dptmgr, dptadm) values ('A00','DEVELOPPEMENT','320','D21');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('B01','PRODUCTION','020','A00');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('C01','MAINTENANCE','030','A00');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('D11','SUPPORT','060','E11');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('D21','DIRECTION','070',null);
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('E01','MARKETING','050','E11');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('E11','VENTES','340','D21');
insert into departement (dptno, dptlib, dptmgr, dptadm) values ('E21','FORMATION','100','E11');
/*ALTER TABLE departement MODIFY (dptadm NOT NULL);
-- Ne fonctionne pas car le departement DIRECTION ne possède pas de dptadm */

ALTER TABLE employe ENABLE CONSTRAINT empDptFK;
ALTER TABLE departement ENABLE CONSTRAINT dptAdmFK;
ALTER TABLE departement ENABLE CONSTRAINT dptmgrFK;
--contraint additionnelles
/*alter table employe ADD CONSTRAINT CK_salaireBorne check (empsal between 50000 and 150000)
--Ne fonctionne pas car MOZART possède un salaire de 18500, largement inférieur au borne imposé */

alter table employe ADD CONSTRAINT CK_MaleFemal check (empsexe = 'M' OR empsexe = 'F')
alter table departement add constraint CK_dptnoDIFadm check (dptno != dptadm)
```

3. supprimer les tables employe/Departement

```
alter table employe drop constraint empDptFK;
alter table departement drop constraint dptAdmFK;
alter table departement drop constraint dptmgrFK;
alter table employe drop constraint empnoPk;
alter table departement drop constraint dptPk;

drop  table employe;
drop  table departement;
```

4. Schémas externes

1. Vue Manager

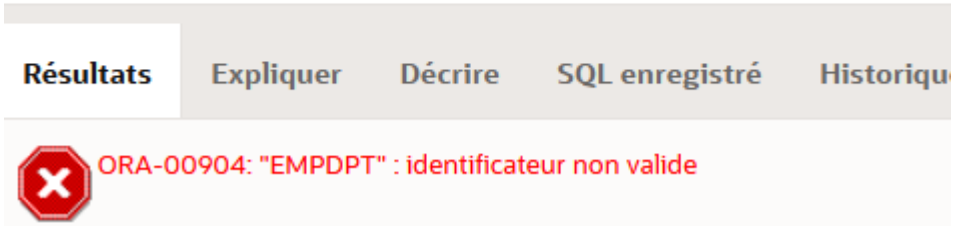
```
create view Managers(mgrNo, mgrNom, dptDirigéLib, nbEmpDirigés) AS
select mgr.empno, mgr.empnom, dmgr.dptlib,count(e.empno)
from departement dmgr
join employe mgr on dmgr.dptmgr = mgr.empno
join employe e on e.empdpt = dmgr.dptno
group by mgr.empno, mgr.empnom, dmgr.dptlib

drop view Managers
select * from Managers
```

2. Vue EmployeDirection

```
create view EmployeDirection(empno, empnom, empsal, empdpt) AS
select empno, empnom, empsal, empdpt
from employe
join departement on empdpt = dptno
where dptlib = 'DIRECTION'

select * from EmployeDirection
drop view EmployeDirection
```

3. Modification Impossible

```
10   create view masseSal(masse) as
11   select sum(empsal) from employe
12
13   select * from masseSal where empdpt = 'D21'
```



Modification possible à partir de la table initiale, ensuite vérification à partir de la vue (WITH CHECK OPTION)

5 . Gestion des privilèges

1.

```
GRANT SELECT ON EmployeDirection TO  labosql, samad
```

2+3 .

```
GRANT SELECT ON EmployeDirection TO  labosql, samad
REVOKE SELECT ON EmployeDirection FROM labosql, samad

GRANT SELECT,INSERT,UPDATE(empnom) ON EmployeDirection to labosql, samad
REVOKE UPDATE ON EmployeDirection FROM labosql, samad
```

L'utilisateur U0 propriétaire de Table1 exécute:

```
GRANT SELECT ON Table1 TO Public;  --1
GRANT INSERT, UPDATE(a1,a2) ON Table1 TO Ua;  --2
GRANT DELETE ON Table1 TO Ua, Ub;  --3
GRANT ALL ON Table1 TO Uc;  --4
GRANT INSERT Table1 TO Uz WITH GRANT OPTION;  --5
```

Après quoi l'utilisateur Uz exécute:

```
GRANT INSERT ON U0.Table1 TO Ub -- 6
GRANT ALL ON U0.Table1 TO Ud -- 7
-- La requête 7 ci-dessus ne donne que le privilège INSERT !
GRANT INSERT ON U0.Table1 TO Ue, Ua WITH GRANT OPTION -- 8
```

L'utilisateur U0 propriétaire de Table1 exécute:

```
REVOKE SELECT ON Table1 FROM Ua;  --1 (impossible !)
REVOKE DELETE ON Table1 FROM Public;  --2 (impossible !)
REVOKE UPDATE ON Table1 FROM Ua  --3
REVOKE DELETE Table1 FROM Uc;  --4
REVOKE INSERT Table1 FROM Uz --5
REVOKE ALL ON Table1 FROM Ub --6
--  Ub possède toujours le privilège par PUBLIC.
REVOKE SELECT Table1 FROM Uc --7
-- Uc possède toujours le privilège par PUBLIC.
```

```
REVOKE {liste de privilèges | ALL}
     ON nomObjet
     FROM {user | role | PUBLIC}
```

- Le privilège est enlevé immédiatement
- Pour pouvoir enlever un privilège à un utilisateur, il faut soit être propriétaire de l'objet soit avoir reçu ce privilège avec WITH GRANT OPTION
- On ne peut supprimer un privilège à un utilisateur si celui ci l'a reçu via PUBLIC. On ne peut supprimer via PUBLIC un privilège à un utilisateur l'ayant reçu personnellement.
- Si plusieurs utilisateurs ont donné un droit sur un objet, il faut que chacun retire ses droites pour que le bénéficiaire n'en dispose plus.
- Il y a cascade dans la révocation d'un privilège transmis avec WITH GRANT OPTION.

6 Synonymes

```
create view EmployeDirectionSamad as select * from samad.EmployeDirection

select * from EmployeDirectionSamad

-Le synonyme existe toujours même après que la vu ait été supprimer par le propriétaire.
Une fois l'accès revok, on ne peux plus y accéder.
```

7 Schéma interne

1. Créez un index sur l'attribut `EmpNom` de votre table `Employe`.

Lorsqu'une base de données possède un grand nombre d'enregistrements (exemple: plusieurs milliers ou plusieurs millions de lignes) un index permet de gagner un temps précieux pour la lecture de données.

Remarque : le SGBD créé automatiquement un index pour chaque clé primaire d'une table donnée

```
SELECT index_name, index_type, visibility, status
FROM all_indexes
WHERE  table_name in ('EMPLOYE','DEPARTEMENT')
```

Cette dernière requête `Permet de visualiser les index qui ont été créé automatiquement par Oracle.`

`-supprimer un index :` DROP INDEX `index_name;`

```
CREATE UNIQUE INDEX{nomIndex}ON{nomTable} ("colonne 1", "colonne
2"...);
```

Cette dernière requête permet de spécifier que des colonnes doivent être uniques.

Un index concaténé est un index qui est créé sur plusieurs colonnes.

```
CREATE INDEX last_first_name ON
member(last_name, first_name);
```

Il est possible de demander au SGBD d'afficher le plan d'exécution d'une requête donnée. Ce plan montre les différentes étapes prises pour l'exécution de la requête (avec, notamment, les éventuels index utilisés ainsi qu'une estimation du "coût" de chaque étape).

Exemple :

1.

```
CREATE INDEX members_name_I
ON members(last_name,first_name);
```

2.

```
EXPLAIN PLAN FOR
SELECT *
FROM members
WHERE last_name LIKE 'A%'
    AND first_name LIKE 'M%';
```

3.

```
SELECT
    PLAN_TABLE_OUTPUT
FROM
    table (DBMS_XPLAN.DISPLAY());
```

- Exemple : on cherche par sexe 'M' l'employé qui la numéro '340' et qui travail dans le département D21(DIRECTION)

on peut estimer la recherche comme :

- 'M' ou 'F' = 2
- numéro d'employé = 30
- département de l'employé = 8

(1/2) * (1/30) * (1/8) = 1/480 ~ 0.20% of the rows (résultat précis)

- Exemple : on cherche les employés 'M' avec un salaire inférieur à 100 000.

on peut estimer la recherche comme :

- 'M' ou 'F' = 2
- salaire des employés = 30

(1/2) * (1/30) = 1/60 ~ 1.66% of the rows (résultat moins précis que précédent)

8 Consultation du catalogue

```
select * from user_tables -- Description of the user's own relational tables
select * from user_tab_columns -- Columns of user's tables, views and clusters
select * from USER_COL_COMMENTS -Comments on the tables and views owned by the u
ser
select * from USER_CONSTRAINTS -- Constraint definitions on user's own tables
select * from USER_CONS_COLUMNS
-- Information about accessible columns in constraint definitions
```

9 Mise en œuvre des transactions : Apex..

LABO 7 cours théorique : PL-SQL

possibilité de définir du code – au niveau du serveur – qui sera déclenché automatiquement à la survenance de certains événements (ce sont les déclencheurs ou triggers en anglais).

PL/SQL pour Procedural Language / SQL est un langage **propriétaire d'Oracle.** Il permet d'écrire:

- des bloc de code '**anonymes**' (a usage unique, qui ne seront pas stockés dans la DB),
- des **procédures** (bloc de code stocké dans la DB, ayant un nom, qui ne renvoie aucune valeur),
- des **fonctions** (bloc de code stocké dans la DB, ayant un nom, renvoie une valeur),
- des **déclencheurs** ou **triggers** (bloc de code stocké dans la DB, ayant un nom, exécutés lorsqu'une certaine condition est satisfaite).

Quelques règles de base en PL/SQL:

- ; est le délimiteur d'instructions
- := L'assignation (ex : nombre := 25 ;)
- = La comparaison
- /* */ délimite les commentaires
- Le langage est insensible à la casse,
- Toute variable doit être déclarée.

- Quelques types :

```
CHAR(n),
VARCHAR2(n) (VARCHAR(n),...), ex.: nom VARCHAR(100) ;
DATE,
NUMBER(p,s) (NUMERIC, FLOAT, INTEGER,...),
            ex.: salaire INTEGER ;
```

..

- Référence au type d'une colonne ou d'une ligne d'une table :

```
numEmploye      employe.empNo%TYPE ;
ligneEmploye    employe%ROWTYPE ;
```

- Initialisation d'une variable à la déclaration :

```
salutations VARCHAR(1000) := 'Veuillez agréer ' ;
```

```
IF condition1 THEN
    sequence_of_statements1;
ELSIF condition2 THEN
    sequence_of_statements2;
ELSE
    sequence_of_statements3;
END IF;

WHILE condition LOOP
    sequence_of_statements;
END LOOP;

FOR counter IN [REVERSE] lower_bound..higher_bound LOOP
    sequence_of_statements;
END LOOP;
```

Exemple :

```
DECLARE
    minimums REAL;
    prime CONSTANT REAL :=50;
BEGIN
    select min(empsal) --into minimums
    from employe
    where empdpt = 'A00';

    UPDATE employe SET empsal = empsal + prime
        where empsal = minimums;
END;
```

Comment exploiter le résultat d'une requête SELECT comportant plus d'une ligne ? -> curseur

Un curseur est simplement un pointeur pointant vers le résultat d'une requête. Il existe des curseurs explicites ou implicites.

exemple 1 version .a :

```
DECLARE
    ligneEmploye employe%ROWTYPE;
    CURSOR lesEmployes IS SELECT * FROM employe;
    n number(10);
    t_ben LONG;
BEGIN
    SELECT count(*) INTO n FROM user_tables WHERE table_name = 'BENEVOLE';
    if(n<=0)
    THEN
        t_ben:=
        'create table Benevole(
         empNo char(3) not null,
         empNom varchar(30) not null,
         empSexe char(11) not null,
         empsal numeric(18) not null,
         empDpt char(3) not null)';
    execute immediate t_ben;
    end if;
    OPEN lesEmployes;
    FETCH lesEmployes into ligneEmploye;
    while lesEmployes%FOUND LOOP

        IF ligneEmploye.empsal <= 20000 THEN
            INSERT INTO Benevole VALUES ligneEmploye;
        END IF;
        FETCH lesEmployes INTO ligneEmploye;
    END LOOP;
    CLOSE lesEmployes;
END;
```

Exemple 2 version .b :

```
DECLARE
    ligneEmploye employe%ROWTYPE;
    CURSOR lesEmployes IS SELECT * FROM employe;
BEGIN
    for ligneEmploye in lesEmployes LOOP
        if ligneEmploye.empsal <= 20000 then
            insert into BENEVOLE values ligneEmploye;
        end if;
    end loop;
END;
```

```
DECLARE
    prime constant real :=50;
    ligneEmploye employe%ROWTYPE;
    CURSOR lesEmployes IS SELECT * FROM employe;
BEGIN
    Update employe SET empsal=empsal+prime where empdpt = 'D21';
    if sql%NOTFOUND then
        dbms_output.put_line('Pas d''employé trouvé');
    elsif sql%FOUND then
        dbms_output.put_line(sql%rowcount || ' Salaire modifiés');
    end if;
END;
```

- **sql%FOUND** (vaut TRUE si la requête associée au curseur a affecté au moins une ligne),
- **sql%NOTFOUND**
- **sql%ROWCOUNT** (le nombre de lignes affectées par la requête).

- Pour le traçage du code, nous pouvons utiliser les commandes:
  - SET SERVEROUTPUT ON ; – activer l'affichage pour la session
  - DBMS_OUTPUT.PUT(...); – permet d'ajouter des informations dans la ligne en cours du tampon
  - DBMS_OUTPUT.PUT_LINE(...); – permet de générer une ligne entière dans le tampon. Un caractère fin de ligne est automatiquement ajouté en fin de ligne
  - DBMS_OUTPUT.NEW_LINE(...); – permet d'ajouter au tampon un caractère fin de ligne

Oracle met à la disposition des utilisateur une table DUAL. Celle-ci peut être utilisée pour exécuter des requêtes ne nécessitant en réalité aucune table:

```
--  Renvoie le résultat de la multiplication 5322*1233
SELECT 5322*1233 FROM dual;
-- Renvoie la date courrante
SELECT sysdate  FROM dual;
-- Renvoie la date + l'heure courante
SELECT SYSTIMESTAMP FROM dual;
-- ...
SELECT UPPER('Salut') FROM dual;
```

Pour chaque bloc PL/SQL, nous pouvons prévoir une gestion d'exception.
La liste des exceptions prédéfinies est accessible dans le guide de référence de PL/SQL à partir de 7-5.

```
1  DECLARE
2    nom VARCHAR2(30);
3  BEGIN
4    SELECT empNom INTO nom
5        FROM employe
6        WHERE empNo = '932';
7  EXCEPTION
8    WHEN NO_DATA_FOUND THEN
9        RAISE_APPLICATION_ERROR(-20142,'Mon erreur à moi');
10 END;
/
Erreur
ORA-20142: Mon erreur à moi
ORA-06512: à ligne 9
```

Une telle 'User Exception' doit reprendre un code compris entre -20000 et -20999 et un libellé de maximum 2048 caractères.

```
declare
    nom varchar2(30);
begin
    select empnom into nom from employe where empno = '932';

EXCEPTION
    when no_data_found then raise_application_error(-20142, 'Mon erreur à moi car pas d''em-
ployé 932');
end;
```

Les procédures et fonctions stockées ?

Sans procédures stockées, le client se verra à faire des requêtes SELECT et UPDATE.
Avec procédures stockées, le client se verra à faire appel à ces procédures stockée sur le serveur !

```
CREATE [OR REPLACE] FUNCTION nomFct (liste paramètres)
    RETURN dataType IS
    -- Code de la fonction ...

CREATE [OR REPLACE] PROCEDURE nomProc (liste paramètres) IS
    -- Code de la procédure ...
```

- Liste paramètres est une liste d'éléments

  `nomParametre {IN | OUT | IN OUT} dataType`

- Les paramètres IN ne sont pas modifiables
- Par défaut, les paramètres sont considérés comme IN.

La déclaration des variables éventuelles se fait juste après le mot clé IS (il n'y a plus de section DECLARE):

```
CREATE [OR REPLACE] PROCEDURE nomProc (liste paramètres) IS
   -- Déclaration de variables ...
BEGIN
   -- Instructions ...
EXCEPTION
   -- Le traitement des exceptions ....
END;
```

Exemples d'entête:

```
CREATE OR REPLACE FUNCTION isbnValide (isbn VARCHAR)
        RETURN NUMERIC IS ... ;

CREATE OR REPLACE FUNCTION anneePremDiplome (anc NUMERIC)
        RETURN NUMERIC IS ... ;

CREATE OR REPLACE PROCEDURE emprunte
     (lecteur NUMERIC,
      livre NUMERIC,
      resultat OUT NUMERIC) IS ... ;
```

Exemple fonction 1 :

```
create or replace function libSexe (sexe CHAR)
    return varchar AS
        BEGIN
        IF sexe = 'M' THEN
            RETURN 'Masculin';
        ELSE
            IF sexe = 'F' THEN
                return 'Féminin';
            ELSE
                return '**********';
            end if;
        END IF;
    END;

--Appel de libSexe
   1. Select libSexe('G') FROM dual;
   2.select empnom, libSexe(empsexe) from employe;
   3. CREATE VIEW mesEmployés AS
      select empno, empnom, libSexe(empsexe) AS sexeLibelle from employe;
```

Exemple fonction 2 :

```
create or replace function nbEmp(idDptNum departement.dptno%TYPE)
    return NUMBER AS
    nb INTEGER;
    BEGIN
        --compte le nombre d'employé par departement
        SELECT COUNT(*) INTO nb FROM employe
            WHERE empdpt = idDptNum;
        return nb;
    END;

--affiche par employés le nombre de collègue qu'il possède (0.02 seconde)
SELECT empno, empnom, nbemp(empdpt)-1 as nbcollegues from employe;

--affiche par employés le nombre de collègue qu'il possède (0.00 seconde) <- Plus performante
SELECT il.empno, il.empnom, count(*)-1
from employe il
    join employe ils on il.empdpt = ils.empdpt
group by il.empno, il.empnom
```

Une **procédure** stockée sera utilisée seule alors qu'une **fonction** stockée sera utilisée à l'intérieur d'une requête **SQL**

Exemple Procedure 1 :

Update departement set dptadm = 'E11' where dptno = 'D21'

```
create procedure changeAdm (nouveauPere departement.dptno%TYPE,
                            nouveauFils departement.dptno%TYPE) AS
                            pere departement.dptno%TYPE;

    BEGIN
-- '<>' -> '!='
        pere := nouveauPere;
        --Tant que le père dif du fils, alors set le dptadm du nouveau fils dans le père
        while pere is not null and pere <> nouveauFils LOOP
            Select dptadm into pere from departement where dptno = pere;
        END LOOP;
        -- si le pere n'existe pas, alors direct set le dptadm du nouveau fils au pere
        IF(pere is NULL) THEN
            UPDATE departement SET dptadm = nouveauPere where dptno = nouveauFils;
        ELSE
            RAISE_APPLICATION_ERROR(-20001, 'Un cylce!');
        END IF;
        --Exception : Si pas de data , error
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR(-20154,'Arbre incorrect ou département enexistant!');
        END;
```

A priori, aucun COMMIT ou ROLLBACK dans une procédure ou une fonction stockée.

Pourquoi ?

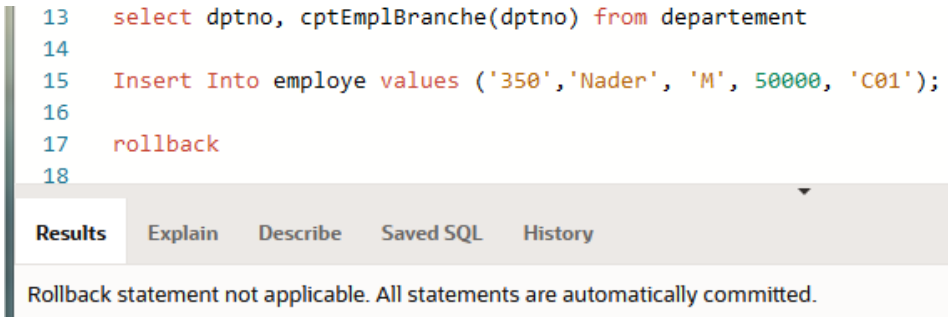
C'est après avoir appelé la procédure que l'on peut effectuer un ROLLBACK

Oracle adds an implicit commit before and after the DDL.

ROLLBACK ?

Lorsque vous faites un **rollback**, votre session va remonter toute la chaine des enregistrements d'*undo* qu'elle a généré, et se servir de chacun pour **faire un retour arrière sur la modification précédente SEULEMENT SI PAS COMMIT !**

Elle va aussi flagger l'enregistrement *undo* comme 'appliqué par l'utilisateur'.




```
create or replace function ListeEmp(dpt departement.dptno%TYPE)
return varchar AS  nom employe.empnom%TYPE;
    chaine varchar(2000);

    CURSOR mesEmployes IS SELECT EMPNOM from employe where empdpt= dpt order by empnom;
    BEGIN
        chaine := '';
        OPEN mesEmployes;
        FETCH mesEmployes INTO nom;
        --Tant qu'on trouve des employes du département en paramètre, alors
        WHILE mesEmployes%FOUND LOOP

            chaine:=chaine||', '||nom; -- || = concaténation
            FETCH mesEmployes INTO nom; --next employe
        END LOOP;
        CLOSE mesEmployes;
        --Supprimer la virgule du deuxième caractère
        IF lengTH (chaine) > 0 THEN
            chaine:= SUBSTR(chaine,2);
        END IF;
        return chaine;
    END;

--1
SELECT dptno, ListeEmp(dptno) from departement
--2
```

```
CREATE VIEW listEmployeeParDepartement AS SELECT dptno, ListeEmp(dptno) AS listeemployeses
                                        from departement;

select  * from listEmployeeParDepartement

DROP VIEW listEmployeeParDepartement
```

-Visualiser les procédures créer par user :

```
select * from user_procedures
```

-Visualiser le code d'une procédure donner (ligne/ligne) :

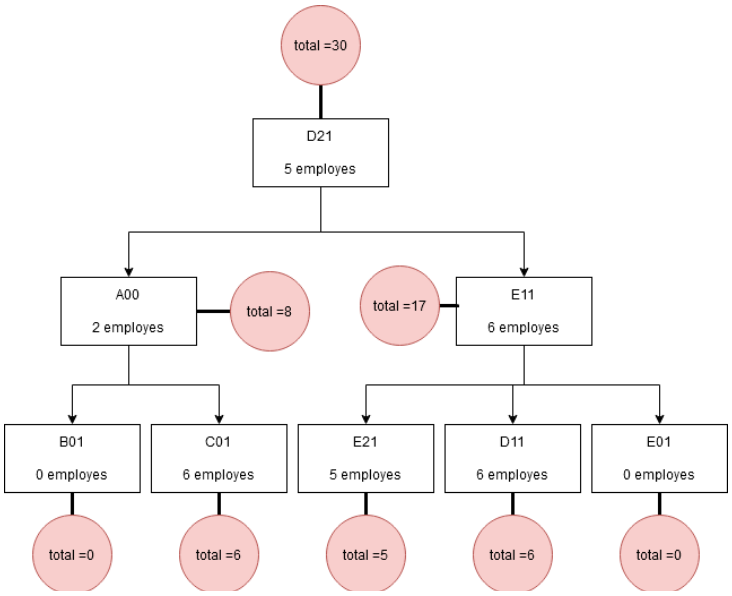
```
select * from user_source where name = 'MODADM'
```

-Visualiser les entêtes générales d'une fonction/procedure/table :

```
describe nom_table || nom_procedure  || nom_fonction
```

Une **procédure** ou fonction est **récursive** si son corps comporte un ou plusieurs appels à elle-même.

- Comment écrire une fonction retournant le nombre d'employés d'une branche de l'arbre des départements dont on donne le numéro de département de la racine ?



Exemple 1 :

```
CREATE OR REPLACE FUNCTION cptEmplBranche(dpt departement.dptno%TYPE)
RETURN INTEGER AS nb INTEGER;

dptCourant departement.dptno%type;

cursor lesDepartements_correspondant_Au_ADM IS SELECT dptNo from departement where dptadm = dp
t;

BEGIN
    select count(*) into nb from employe where empdpt=dpt;

    OPEN lesDepartements_correspondant_Au_ADM;
    --récupère le département de l'employe
    FETCH lesDepartements_correspondant_Au_ADM into dptCourant;

    WHILE lesDepartements_correspondant_Au_ADM%FOUND LOOP

        nb:=nb+cptEmplBranche(dptCourant);
        FETCH lesDepartements_correspondant_Au_ADM into dptCourant;
    END LOOP;
    CLOSE lesDepartements_correspondant_Au_ADM;
    return nb;
END;

select dptno, cptEmplBranche(dptno) from departe-
ment;
```

```
1.  select dptno, cptEmplBranche(dptno) from departement where dtpno = 'E11'

dptCourant = Null;
nb = 6 ;
dptCourant = E21;
Liste vide ? non (FOUND)

nb = 6 + cptEmplBranche(E21); ->

                2.  select dptno, cptEmplBranche(dptno) from departement where dtpno = 'E21'
                dptCourant = null;
                nb = 5 ;
                Liste vide ? Oui (NOT_FOUND)
                return 5;

nb = 6+5;
dptCourant++; (fetch)  => dptCourant = 'D11
Liste vide ? non (FOUND)

nb = 6+5+ cptEmplBranche(D11); ->

                3.  select dptno, cptEmplBranche(dptno) from departement where dtpno = 'D11'
                dptCourant= null;
                nb=6;
                Liste vide ? Oui (NOT_FOUND)
                return 6;

nb=6+5+6;
dptCourant++; (fetch) => dptCourant = 'E01'
liste vide ? non (FOUND)

nb=6+5+6+cptEmplBranche(E01); ->

                4.  select dptno, cptEmplBranche(dptno) from departement where dtpno = 'E01'
                dptCourant= null;
                nb=0;
                Liste vide ? Oui (NOT_FOUND)
                return 0;

nb = 6+5+6+0
dptCourant++; (fetch) => dptCourant = NULL
Liste vide ? Oui (NOT_FOUND)
```

- **SGBD Actif** (Active DBMS) ?

SGBD capable de réagir à des événements afin de :

- contrôler l'intégrité
- gérer des redondances
- autoriser ou interdire des accès
- alerter des utilisateurs

et plus généralement gérer le comportement réactif des applications.

Un SGBD actif fournit la notion de **déclencheur (trigger)** :

Un déclencheur (trigger) est simplement un **bloc de code** gouverné par un triplet ECA:

1.dès qu'un événement (E) survient,
2. si la condition (C) est satisfaite,
3. exécuter l'action (A).

Il est aussi possible d'écrire un déclencheur pour une vue.

Oracle n'accepte les déclencheurs INSTEAD OF que pour les vues (et pas pour les tables).

```
CREATE OR REPLACE TRIGGER [ schéma. ]nomTrigger
    { BEFORE | AFTER | INSTEAD OF }
    DELETE | INSERT | UPDATE [OF column [, column]...]
        [ OR DELETE | INSERT | UPDATE [ OF column [ ,column ]... ]... ]
    ON [ schéma. ]nomTable
    [ REFERENCING OLD [ AS ] Old NEW [ AS ] New ]
    [ FOR EACH ROW ]
    [ WHEN (Condition) ]
Bloc PL/SQL
```

- BEFORE : le bloc PL/SQL est exécuté AVANT la vérification des contraintes de tables et la mise à jour des données dans la table
- AFTER : le bloc PL/SQL est exécuté APRES la mise à jour des données dans la table
- INSTEAD OF : le bloc PL/SQL remplace le traitement standard associé à l'instruction qui a déclenché le trigger
- DELETE/INSERT/UPDATE [OF col,...] : Instruction associée au déclenchement du trigger
- FOR EACH ROW : le trigger s'exécute pour chaque ligne traitée par l'instruction associée
- WHEN : la condition donnée doit être vérifiée pour que le code s'exécute

- Si le mot clé **FOR EACH ROW** n'est pas précisé le trigger ne s'exécute qu'une seule fois, quel que soit le nombre de lignes modifiées par la requête INSERT/UPDATE/DELETE.

Exemple 1 :

Nous souhaiterions, lorsque le solde du compte d'un client est mis à jour(c'est l'Evénement), si ce solde est négatif (c'est la Condition), supprimer les commandes relatives à ce client (c'est l'Action).

```
create or replace trigger deleteCom_If_Sold_Null

--Commandes du client supprimées si compte client vide ou inférieur.

AFTER UPDATE OF compte ON cliente

FOR each row
When (new.compte <=0)
BEGIN
    DELETE FROM commande WHERE nclient=new.ncli;
END;
```

Trois conditions sont disponibles dans le bloc PL/SQL de l'action d'un trigger :

- UPDATING
- DELETING
- INSERTING

respectivement vraies si l'événement déclencheur est un update, delete ou insert.

Attention : Une erreur dans un trigger met en erreur l'événement qui le déclenche.

```
CREATE OR REPLACE TRIGGER exemple
BEFORE UPDATE OF maTable OR INSERT ON maTable
REFERENCING OLD AS ancien NEW AS nouveau FOR EACH ROW
BEGIN
    -- ancien est synonyme de OLD
    -- nouveau est synonyme de NEW
    IF INSERTING THEN
        -- Si le trigger a été déclenché par un insert ...
    END IF;
    IF UPDATING THEN
       -- Si le trigger a été déclenché par un update ...
    END IF;
END;
```

Gérer des redondances souhaitées pour raison de performance, exemple:

- Colonne permettant le tri alphabétique et gestion du nom condensé.
- Nombre d'employés repris dans chaque département.

Gérer des CI non gérables déclarativement, exemple:

- Un salaire ne peut qu'augmenter.
- Une clé primaire ne peut pas être modifiée.
- Un employé ne peut pas gagner plus que son manager.

EXEMPLE SUPPLEMENTAIRE :

Un salaire ne peut pas diminuer:

```
CREATE OR REPLACE TRIGGER modifierSalaire1
BEFORE UPDATE OF empSal ON employe
FOR EACH ROW
BEGIN
    IF :new.empSal < :old.empSal THEN
        :new.empSal := :old.empsal
    END IF;
END;
```

Gestion du nom condensé:

```
CREATE OR REPLACE TRIGGER condenserNom
BEFORE UPDATE OF empDpt OR INSERT
ON employe
REFERENCING OLD AS ancien NEW AS nouveau FOR EACH ROW
BEGIN
  :nouveau.empNomCd :=
  condenser(:nouveau.empNom, :nouveau.empDpt);
END;
```

(Ici, "condenser" désigne une fonction PL/SQL effectuant la condensation selon une procédure définie . . .)

Gestion du nombre d'employé:

```
CREATE OR REPLACE TRIGGER nombreEmploye
AFTER DELETE OR INSERT OR UPDATE OF empDpt ON employe
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        modNbEmp(:new.empNom,1);
    END IF;
    IF DELETING THEN
        modNbEmp(:old.empNom,-1);
    END IF;
    IF UPDATING THEN
        modNbEmp(:new.empNom,1);
        modNbEmp(:old.empNom,-1);
    END IF;
END;
```

Audit pour contrôler les évolutions salariales des employés:

```
CREATE OR REPLACE TRIGGER auditModSalaire
BEFORE INSERT OR UPDATE OF empSal ON employe
FOR EACH ROW
BEGIN
    INSERT INTO myAudit(auDate, auUser, auTerm, auAnc, auNv)
        VALUES(sysdate, user, userenv('terminal'),
                    :old.empSal, :new.empSal) ;
END;
```

(Supplément ... cadeau pour samad)

Utilisez l'instruction CREATE SEQUENCE pour créer une séquence, qui est un objet de base de données à partir duquel plusieurs utilisateurs peuvent générer des entiers uniques. Vous pouvez utiliser des séquences pour générer automatiquement des valeurs de clé primaire.

Lorsqu'un numéro de séquence est généré, la séquence est incrémentée, indépendamment de la validation ou de l'annulation de la transaction. Si deux utilisateurs incrémentent simultanément la même séquence, les numéros de séquence que chaque utilisateur acquiert peuvent présenter des espaces, car les numéros de séquence sont générés par l'autre utilisateur. Un utilisateur ne peut jamais acquérir le numéro de séquence généré par un autre utilisateur. Une fois qu'une valeur de séquence est générée par un utilisateur, cet utilisateur peut continuer à accéder à cette valeur, que la séquence soit incrémentée ou non par un autre utilisateur.

Gérer des numérotations automatiques:

SQL3 a introduit la notion de séquence.

```
CREATE SEQUENCE nomSequence
     START WITH v1
     INCREMENT BY v2
     MINVALUE vMin
     MAXVALUE vmax
     { CYCLE | NOCYCLE } { CACHE | NOCACHE }

CREATE SEQUENCE maseq INCREMENT BY 1;

SELECT maseq.nextval FROM DUAL;
SELECT maseq.currval FROM DUAL;
```

LABO 7 :

1. Rédaction et test de fonctions

1.

```
CREATE OR REPLACE FUNCTION libSexe(sexe CHAR)
RETURN VARCHAR AS
BEGIN
    IF sexe ='M' THEN
        RETURN 'Masculin';
    ELSE IF sexe = 'F' THEN
        RETURN 'Féminin';
    ELSE
        RAISE_APPLICATION_ERROR(-20001,'Pas sexe!');
    END IF;
END IF;
END;
     Select libSexe('G') from dual;
```

2.

```
create oR replace function MasSal(dpt departement.dptno%type)
        return numeric
        as samad numeric ;
        begin
        select sum(empsal) into samad from employe where empdpt= dpt;
        return samad;
        end;

select dptno, MasseSalC(dptno) from departement where dptno = 'D21'
```

3.

```
create or replace function Fcondense(chaine varchar)
    return varchar as
    begin
        return regexp_replace(translate(upper(chaine), 'ÁÀÃÂÉÈÊÍÌÎÕÓÒÔÚÙÛ', 'AAAAEEEII-
IOOOOUUU'),
         '[^A-Z]');
    end;

--test
select Fcondense('sûmaâêd05') from dual;
--add coluwn
    ALTER TABLE employe
ADD empnomCD varchar(200);

update employe set empnomCD = Fcondense('sûmaâêd05');

select * from employe;
```

4.

```
create or replace function FnumNiv(dpt departement.dptNo%type)
return numeric AS nb numeric;
dpCourant departement.dptadm%type;

BEGIN
nb:=0;
select dptadm into dpCourant from departement where dpt=dptno;
while dpCourant is not null LOOP
nb:=nb+1;
select dptadm into dpCourant from departement where dpCourant=dptno;
end LOOP;
return nb;
END;

select FnumNiv(dptno) from departement where dptno = 'C01';
```

2. Création de vues dont la définition utilise des fonctions

5.

```
create or replace function ListeEmp(dpt departement.dptno%TYPE)
return varchar AS  nom employe.empnom%TYPE;
    chaine varchar(255);

    CURSOR mesEmployes IS SELECT EMPNOM from employe where empdpt= dpt order by
empnom;
    BEGIN
        chaine := '';
        OPEN mesEmployes;
        FETCH mesEmployes INTO nom;
        --Tant qu'on trouve des employes du département en paramètre, alors
        WHILE mesEmployes%FOUND LOOP

            chaine:=chaine||', '||nom; -- || = concaténation
            FETCH mesEmployes INTO nom; --next employe
        END LOOP;
        CLOSE mesEmployes;
        --Supprimer la virgule du deuxième caractère
        IF lengTH (chaine) > 0 THEN
            chaine:= SUBSTR(chaine,2);
        END IF;
        return chaine;
    END;

--1
CREATE VIEW VDptDetail AS SELECT dptno,dptlib, ListeEmp(dptno) AS listeemploye-
ses
                        from departement;
--2
select * from VDptDetail;
```

6.

```
CREATE OR REPLACE FUNCTION DptNbDir(dpt departement.dptno%type)
RETURN INTEGER AS nb INTEGER :=0;

dptCourant departement.dptno%type;

cursor lesDepartements_enfants IS SELECT dptNo from departement where dp-
tadm = dpt;

BEGIN

    OPEN lesDepartements_enfants;
    --récupère le département de l'employe
    FETCH lesDepartements_enfants into dptCourant;

    WHILE lesDepartements_enfants%FOUND LOOP
        nb:=nb+1;
        nb:=nb+DptNbDir(dptCourant);
        FETCH lesDepartements_enfants into dptCourant;
    END LOOP;
    CLOSE lesDepartements_enfants;
    return nb;
END;


select DptNbDir(dptno) from departement where dptno = 'A00'
```

Demonstration explicative 1 :

```
1.cursor lesDepartements_enfants IS SELECT dptNo from departement where dptadm = D21;


Liste_lesDepartements_enfants = {A00,E11};

dptCourant = A00;

Liste_lesDepartements_enfants vide ? non (FOUND)

nb= nb + 1;
```

```
nb= 1 + DptNbDir('A00');  -> 2.cursor lesDepartements_enfants IS SELECT dptNo from departement
where dptadm = A00;

            nb=0;
            Liste_lesDepartements_enfants = {B01,C01};
            dptCourant = B01;
            Liste_lesDepartements_enfants ? non (FOUND)
            nb=nb +1;
            nb= 1 + DptNbDir(B01); -> 3.cursor lesDepartements_enfants IS SELECT dptNo from de
partement where dptadm = B01;

                           nb=0;
                           Liste_lesDepartements_enfants = {};

                           dptCourant = NULL;
                           Liste_lesDepartements_enfants ? OUI (NOT_FOUND)
                           nb=0;


            nb=1+0;
            dptCourant = C01;
            Liste_lesDepartements_enfants ? non (FOUND)
            nb=1+0+1;
            nb=1+0+1+DptNbDir(C01); -> 4.cursor lesDepartements_enfants IS SELECT dptNo from d
epartement where dptadm = C01;
                           nb=0;
                           Liste_lesDepartements_enfants = {};

                           dptCourant = NULL;
                           Liste_lesDepartements_enfants ? OUI (NOT_FOUND)

            nb=0;
            nb=1+0+1+0;
            dptCourant = NULL;
            Liste_lesDepartements_enfants ? OUI (NOT_FOUND)
             nb=2
```

3. Rédaction de procédures stockées

7.

```
create or replace procedure PtsfGroupe(Dpt1 departement.dptno%TYPE, Dpt2 departement.dptno
%TYPE)
 IS EmployeRemake employe%ROWTYPE;
cursor liste is select * from employe where empdpt=Dpt1;
BEGIN
OPEN liste;
FETCH liste into EmployeRemake;
while liste%FOUND LOOP
    UPDATE employe set empdpt=Dpt2 where empdpt=dpt1;
    FETCH liste into EmployeRemake;
END LOOP;
CLOSE liste;
END;

select empdpt,count(*) from employe group by empdpt;

BEGIN
PtsfGroupe('A00','E01');
END;

select * from employe where empdpt = 'E01'
```

8.

```
create OR REPLACE procedure changeAdm (nouveauFils departement.dptno%TYPE,
                          nouveauPere departement.dptno%TYPE) AS
                          pere departement.dptno%TYPE;

    BEGIN
-- '<>' -> '!='
        pere := nouveauPere;
        --Tant que le père dif du fils, alors set le dptadm du nouveau fils dans le père
        while pere is not null and pere <> nouveauFils LOOP
            Select dptadm into pere from departement where dptno = pere;
        END LOOP;
        -- si le pere n'existe pas, alors direct set le dptadm du nouveau fils au pere
        IF(pere is NULL) THEN
            UPDATE departement SET dptadm = nouveauPere where dptno = nouveauFils;
        ELSE
            RAISE_APPLICATION_ERROR(-20001, 'Un cylce!');
        END IF;
        --Exception : Si pas de data , error
        EXCEPTION
            WHEN NO_DATA_FOUND THEN
                RAISE_APPLICATION_ERROR(-20154,'Arbre incorrect ou département enexistant!');
        END;

Select * from departement

BEGIN
changeAdm('D21','E01');
END;

update departement set dptadm='E11' where dptno='E01'
```

9.

```
GRANT EXECUTE ON changeAdm to labosql
```