

ACHEHBOUNE Mohammed Amine Rendu TD5

1. Activité 1 – Connexions & variables

1.1 Fenêtre 1 – Base 1 (PDBM1MIA)

```
C:\Users\aacbehboune\Downloads\instantclient-basic-windows.x64-23.26.0.0.0\instantclient_23_0>sqlplus /nolog
SQL*Plus: Release 21.0.0.0.0 - Production on Wed Nov 26 21:48:49 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

SQL> define REMOTEDB = PDBL3MIA.sub12051533510.vcnmimageuca.oraclevcn.com
SQL> define REMOTEDBNAME = PDBL3MIA
SQL> define CURRENTDB = PDBM1MIA
SQL>
SQL> define DRUSER = aachehboune1M2526
SQL> define DRUSERPASS = aachehboune1M252601
SQL>
SQL> define SCRIPTPATH = C:\Users\aacbehboune\Downloads\scripts
SQL>
SQL> connect &DRUSER@&CURRENTDB/&DRUSERPASS
Connected.
SQL> |
```

1.2 Fenêtre 2 – Base 2 (PDBL3MIA)

```
C:\Users\aacbehboune\Downloads\instantclient-basic-windows.x64-23.26.0.0.0\instantclient_23_0>sqlplus /nolog
SQL*Plus: Release 21.0.0.0.0 - Production on Wed Nov 26 21:50:48 2025
Version 21.3.0.0.0

Copyright (c) 1982, 2021, Oracle. All rights reserved.

SQL> define REMOTEDB = PDBM1MIA.sub12051533510.vcnmimageuca.oraclevcn.com
SQL> define REMOTEDBNAME = PDBM1MIA
SQL> define CURRENTDB = PDBL3MIA
SQL>
SQL> define DRUSER = aachehboune1M2526
SQL> define DRUSERPASS = aachehboune1M252601
SQL>
SQL> define SCRIPTPATH = C:\Users\aacbehboune\Downloads\scripts
SQL>
SQL> connect &DRUSER@&CURRENTDB/&DRUSERPASS
Connected.
SQL> |
```

2. Activité 2 – Chargement des données

2.1 Sur Base 1 (Fenêtre 1, PDBM1MIA)

```

SQL> connect &DRUSER@&CURRENTDB/&DRUSERPASS
Connected.
SQL> @&SCRIPTPATH\chap8_demoBld.sql

Session altered.

Session altered.

Building demonstration tables. Please wait.
Demonstration table build is complete.
SQL> select table_name
  2  from user_tables
  3  where table_name in ('EMP', 'DEPT', 'BONUS', 'SALGRADE');

TABLE_NAME
-----
BONUS
DEPT
EMP
SALGRADE

SQL> |

```

2.2 Sur Base 2 (Fenêtre 2, PDBL3MIA)

```

1 row created.

Table created.

1 row created.

1 row created.

1 row created.

Commit complete.

SQL> select table_name
  2  from user_tables
  3  where table_name in ('CLIENT', 'PRODUIT', 'COMMANDE');

TABLE_NAME
-----
CLIENT
COMMANDE
PRODUIT

SQL> |

```

2.3 Travail à faire (Q3) – Vérifier les imports

- Sur Base 1 : tables DEMO bien présentes.
- Sur Base 2 : tables CLIENT, PRODUIT, COMMANDE bien présentes.

Requêtes utilisées : celles ci-dessus (select table_name from user_tables ...).

3. Activité 3 – Database link

```
SQL> select * from all_db_links;

OWNER
-----
DB_LINK
-----
USERNAME
-----
HOST
-----
CREATED    HID SHA VAL INT
-----
PUBLIC
PDBL3MIA.SUB12051533510.VCNMIMAGEUCA.ORACLEVCN.COM

OWNER
-----
DB_LINK
-----
USERNAME
-----
HOST
-----
CREATED    HID SHA VAL INT
-----
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 129.151.234.184)(PORT = 1521))
```

4. Activité 4 – Consultations & mises à jour distantes

4.1 Consultations distantes

Fenêtre 1 (PDBM1MIA) :

```
SQL> desc &DRUSER..produit@&REMOTEDBNAME
Name          Null?    Type
-----
PID#          NOT NULL NUMBER(6)
PNOM          VARCHAR2(50)
PDESCRIPTION  VARCHAR2(100)
PPRIXUNIT    NUMBER(7,2)

SQL> desc &DRUSER..commande@&REMOTEDBNAME
Name          Null?    Type
-----
PCOMM#        NOT NULL NUMBER(6)
CDATE         DATE
PID#          NUMBER(6)
CID#          NUMBER(6)
PNBRE         NUMBER(4)
PPRIXUNIT    NUMBER(7,2)
EMPNO         NUMBER(4)

SQL> desc &DRUSER..client@&REMOTEDBNAME
Name          Null?    Type
-----
CID#          NOT NULL NUMBER(6)
CNOM          VARCHAR2(20)
CDNAISS       DATE
CADR          VARCHAR2(50)

SQL>
SQL> select * from &DRUSER..commande@&REMOTEDBNAME;
old   1: select * from &DRUSER..commande@&REMOTEDBNAME
new   1: select * from achehboune1M2526.commande@PDBL3MIA

PCOMM# CDATE      PID#      CID#      PNBRE     PPRIXUNIT    EMPNO
-----  -----      -----      -----      -----      -----
1 26-NOV-25    1000      1          4          2          7369
2 26-NOV-25    1000      1          10         2          7369
```

Substitution :

```
SQL> select * from achehboune1M2526.client@PDBL3MIA;
```

CID#	CNOM	CDNAISS
CADR		
1	Akim Washington	12-DEC-72
2	Erzulie Artibonite	12-DEC-42

```
SQL> |
```

4.2 Mises à jour distantes

Toujours dans Fenêtre 1 (PDBM1MIA) :

```
SQL> update &DRUSER..commande@&REMOTEDBNAME set empno = 7369;
old    1: update &DRUSER..commande@&REMOTEDBNAME set empno = 7369
new    1: update achehboune1M2526.commande@PDBL3MIA set empno = 7369

3 rows updated.

SQL> commit;

Commit complete.

SQL> insert into &DRUSER..commande@&REMOTEDBNAME
  2  (pcomm#, cdate, pid#, cid#, pnbre, pprixunit, empno)
  3 values (4, sysdate, 1000, 1, 9, 2, 7369);
old    1: insert into &DRUSER..commande@&REMOTEDBNAME
new    1: insert into achehboune1M2526.commande@PDBL3MIA

1 row created.

SQL> commit;

Commit complete.

SQL> |
```

Vérification depuis PDBL3MIA (Fenêtre 2) que la nouvelle commande est bien là :

```
SQL> select * from commande;
```

PCOMM#	CDATE	PID#	CID#	PNBRE	PPRIXUNIT	EMPNO
1	26-NOV-25	1000	1	4	2	7369
2	26-NOV-25	1000	1	10	2	7369
3	26-NOV-25	1000	1	9	2	7369
4	26-NOV-25	1000	1	9	2	7369

```
SQL> |
```

4.3 Travail à faire (Q4)

- Expliquez ce qu'il se passe lorsque vous exécutez une mise à jour distante.

4.1 Qui est responsable de la transaction ?

- Locale / site initiateur** : la base sur laquelle tu es connecté (ici PDBM1MIA).
- Globale** : le même site (PDBM1MIA) joue le rôle de **coordinateur global** de la transaction.

4.2 Qui pose les verrous nécessaires ?

- Les **verrous de lignes** sont physiquement posés sur la base qui contient les données : ici, PDBL3MIA, sur la table COMMANDE.
- Le **coordonnateur** (PDBM1MIA) gère la logique du commit, mais les verrous sont gérés par le moteur Oracle de la base où se trouve la table.

Requête possible pour observer les verrous sur la base distante (à lancer sur PDBL3MIA) :

```
select lo.object_id, o.object_name, l.session_id
from v$locked_object lo
join dba_objects o on o.object_id = lo.object_id
join v$lock l on l.sid = lo.session_id;
```

4.3 S'agit-il d'une transaction distribuée ?

- Oui.**
Dès qu'une transaction touche **au moins une base distante via un DBLINK**, c'est une **transaction distribuée**.

5. Activité 5 – Transactions distribuées

5.1 Jointure distribuée

Depuis Fenêtre 1 (PDBM1MIA) :

```

SQL> select c.pcomm#, c.pid#, e.empno, c.prixunit
  2  from &DRUSER..commande@&REMOTEDBNAME c,
  3      &DRUSER..produit@&REMOTEDBNAME p,
  4      emp e
  5 where c.pid# = p.pid#
  6   and c.empno = e.empno;
old  2: from &DRUSER..commande@&REMOTEDBNAME c,
new  2: fromachehboune1M2526.commande@PDBL3MIA c,
old  3:      &DRUSER..produit@&REMOTEDBNAME p,
new  3:     achehboune1M2526.produit@PDBL3MIA p,

```

PCOMM#	PID#	EMPNO	PPRIXUNIT
1	1000	7369	2
2	1000	7369	2
3	1000	7369	2
4	1000	7369	2

```
SQL> |
```

- c et p sont sur PDBL3MIA (distantes).
- emp est sur PDBM1MIA (locale).

Oracle exécute une requête distribuée.

5.2 Transaction distribuée (PL/SQL)

Toujours sur PDBM1MIA :

```

SQL> begin
  2      insert into &DRUSER..commande@&REMOTEDBNAME
  3          (pcomm#, cdate, pid#, cid#, pnbre, pprixunit, empno)
  4      values (6, sysdate, 1000, 1, 9, 2, 7369);
  5
  6      update emp set comm = nvl(comm, 0) + 2
  7      where empno = 7369;
  8  end;
  9 /
old  2:  insert into &DRUSER..commande@&REMOTEDBNAME
new  2:  insert intoachehboune1M2526.commande@PDBL3MIA

```

PL/SQL procedure successfully completed.

```
SQL> commit;
```

Commit complete.

5.3 Travail à faire (Q5)

5. Expliquez ce qu'il se passe lorsque vous exécutez une transaction distribuée.

5.1 Qui est responsable de la transaction ?

- **Locale / site initiateur** : PDBM1MIA (où tu es connecté).
- **Globale** : PDBM1MIA joue le rôle de **coordinateur global** de la transaction distribuée.

5.2 Qui pose quels verrous ?

- Sur PDBL3MIA : verrous sur la ligne insérée / modifiée de COMMANDE.
- Sur PDBM1MIA : verrous sur EMP (empno = 7369).
- Oracle utilise le **protocole de commit en deux phases (2PC)** pour garantir que les deux sites s'engagent ou s'annulent ensemble.

5.3 S'agit-il d'une transaction distribuée ?

- **Oui**, car la transaction modifie :
 - une table locale (EMP, PDBM1MIA)
 - une table distante (COMMANDE@PDBL3MIA)
 -

6. Activité 6 – Transparence via synonymes

On veut masquer la localisation :

Sur PDBM1MIA (Fenêtre 1) :

```
SQL> drop synonym commande;
drop synonym commande
*
ERROR at line 1:
ORA-01434: private synonym to be dropped does not exist

SQL>
SQL> create synonym commande
  2  for &DRUSER..commande@&REMOTEDBNAME;
old   2: for &DRUSER..commande@&REMOTEDBNAME
new   2: for achehboune1M2526.commande@PDBL3MIA

Synonym created.

SQL> -- => synonym commande for achehboune1M2526.commande@PDBL3MIA;
SQL> begin
  2  insert into commande
  3    (pcomm#, cdate, pid#, cid#, pnbre, pprixunit, empno)
  4  values (7, sysdate, 1000, 1, 9, 2, 7369);
  5
  6  update emp set comm = nvl(comm, 0) + 2
  7  where empno = 7369;
  8
  9  commit;
10 end;
11 /

PL/SQL procedure successfully completed.

SQL> |
```

Vérification :

```
SQL> select * from commande;
```

PCOMM#	CDATE	PID#	CID#	PNBRE	PPRIXUNIT	EMPNO
1	26-NOV-25	1000	1	4	2	7369
2	26-NOV-25	1000	1	10	2	7369
3	26-NOV-25	1000	1	9	2	7369
4	26-NOV-25	1000	1	9	2	7369
6	26-NOV-25	1000	1	9	2	7369
7	26-NOV-25	1000	1	9	2	7369

6 rows selected.

```
SQL> select * from emp where empno = 7369;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
7369	SMITH	CLERK	7902	17-DEC-80	800	6
20						

6.1 Travail à faire (Q6)

1. Que font les requêtes ?

- drop synonym / create synonym : crée un alias logique commande qui pointe vers la table distante `achehboune1M2526.commande@PDBL3MIA`.
- Dans le bloc PL/SQL :
 - INSERT dans la table commande (en réalité `COMMANDE@PDBL3MIA` via le synonyme).
 - UPDATE sur EMP (table locale `PDBM1MIA`).
 - COMMIT : valide une transaction **distribuée**.

2. Pourquoi parle-t-on de transparence ?

- Parce que l'utilisateur écrit simplement `insert into commande ...` sans se soucier :
 - de savoir que ça part sur un DBLINK
 - ni sur quelle base physique est la table.

3. Quelle règle est implicitement respectée ?

- **Transparence de localisation** (principe de base des SGBD distribués).

- On peut aussi parler de **transparence d'accès** : même syntaxe que pour une table locale.

7. Activité 7 – Database link inverse

Déjà préparé par l'admin : depuis PDBL3MIA vers PDBM1MIA.

on peut tester dans PDBL3MIA :

```
SQL> select * from emp@PDBM1MIA
  2  where rownum <= 5;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
7369	SMITH	CLERK	7902	17-DEC-80	800	6
20						
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30						
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30						
EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
DEPTNO						
7566	JONES	MANAGER	7839	02-APR-81	2975	
20						
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30						

```
SQL> |
```

8. Activité 8 & 9 – Trigger distribué

8.1 Création du trigger (sur Base2)

Le trigger agit sur **COMMANDÉ**, qui est sur **PDBL3MIA**
 → il doit être créé sur **PDBL3MIA**.

```
SQL> connect &DRUSER/&DRUSERPASS@&CURRENTDB
Connected.
SQL> -- ici CURRENTDB = PDBL3MIA
SQL>
SQL> create or replace trigger update_employe_comm
  2  after delete or insert on commande
  3  for each row
  4  begin
  5    if inserting then
  6      update &DRUSER..emp@&REMOTEDBNAME e
  7      set e.comm = nvl(e.comm, 0) + 2
  8      where empno = :new.empno;
  9    elsif deleting then
10      update &DRUSER..emp@&REMOTEDBNAME e
11      set e.comm = nvl(e.comm, 0) - 2
12      where empno = :old.empno;
13    end if;
14  end;
15 /
old  6:      update &DRUSER..emp@&REMOTEDBNAME e
new  6:      updateachehboune1M2526.emp@PDBM1MIA e
old 10:      update &DRUSER..emp@&REMOTEDBNAME e
new 10:      updateachehboune1M2526.emp@PDBM1MIA e

Trigger created.

SQL> |
```

- Ici REMOTEDBNAME = PDBM1MIA, donc EMP@PDBM1MIA.

9.Test du trigger (Activité 9)

Toujours sur PDBL3MIA :

```
SQL> -- CURRENTDB = PDBL3MIA
SQL>
SQL> insert into commande
2   (pcomm#, cdate, pid#, cid#, pnbre, pprixunit, empno)
3 values (8, sysdate, 1000, 1, 9, 2, 7369);

1 row created.

SQL>
SQL> select * from commande where pcomm# = 8;

  PCOMM# CDATE        PID#      CID#    PNBRE  PPRIXUNIT    EMPNO
----- -----          -----      -----    -----  -----          -----
          8 26-NOV-25     1000       1         9        2        7369

SQL> select * from emp@PDBM1MIA where empno = 7369;

  EMPNO ENAME      JOB           MGR HIREDATE      SAL      COMM
----- -----      -----          ----- -----          -----  -----
DEPTNO
-----
  7369 SMITH      CLERK        7902 17-DEC-80      800        8
  20

SQL> |
```

9.1 Travail à faire (Q7)

1. Que fait le code proposé ?

- À chaque INSERT sur COMMANDE (Base2) :
 - le trigger augmente la COMM de l'employé empno = :new.empno sur la base distante (EMP@PDBM1MIA).
- À chaque DELETE sur COMMANDE :
 - le trigger diminue la COMM de 2 sur EMP@PDBM1MIA.

2. Quelle base doit enregistrer ce trigger ? Pourquoi ?

- Le trigger doit être créé sur LA BASE QUI CONTIENT LA TABLE SURVEILLÉE :
 - ici COMMANDE est sur PDBL3MIA → donc le trigger est sur PDBL3MIA.
- C'est toujours comme ça en Oracle : un trigger appartient au schéma où se trouve la table.

3. Qu'est-ce qui change du point de vue :

- Transactions :

- Chaque insertion/suppression dans COMMANDE devient automatiquement une transaction distribuée, car elle inclut aussi une mise à jour sur EMP@PDBM1MIA via DBLINK.

- **Verrous :**

- COMMANDE (sur PDBL3MIA) est verrouillée localement.
- EMP (sur PDBM1MIA) est verrouillée à distance via DBLINK.
- On peut voir ça (si priviléges) via v\$locked_object sur chaque base.

Activité 10 – Simulation de pannes

10.1 Modèle de transaction distribuée avec panne simulée

Sur la base coordonnatrice (par ex. PDBM1MIA) :

```
SQL> update emp set sal = sal * 1.1 where empno = 7369;
1 row updated.

SQL>
SQL> update &DRUSER..produit@&REMOTEDBNAME
  2 set pdescription = pdescription || 'BravoBravo'
  3 where pid# = 1000;
old   1: update &DRUSER..produit@&REMOTEDBNAME
new   1: update achehboune1M2526.produit@PDBL3MIA

1 row updated.

SQL>
SQL> commit comment 'ORA-2PC-CRASH-TEST-6';
commit comment 'ORA-2PC-CRASH-TEST-6'
*
ERROR at line 1:
ORA-02054: transaction 1.29.9095 in-doubt
ORA-02053: transaction 3.10.14329 committed, some remote DBs may be in-doubt
ORA-02059: ORA-2PC-CRASH-TEST-6 in commit comment
ORA-02063: preceding 2 lines from PDBL3MIA

SQL> select * from dba_2pc_pending;

LOCAL_TRAN_ID
-----
GLOBAL_TRAN_ID
-----
STATE      MIX_A
-----
TRAN_COMMENT
-----
FAIL_TIME FORCE_TIM RETRY_TIM
-----
OS_USER
```

LOCAL_TRAN_ID

GLOBAL_TRAN_ID

STATE MIX A

TRAN_COMMENT

FAIL_TIME FORCE_TIM RETRY_TIM

OS_USER

OS_TERMINAL

HOST

DB_USER

COMMIT#

ORA-2PC-CRASH-TEST-9

LOCAL_TRAN_ID

GLOBAL_TRAN_ID

STATE MIX A

TRAN_COMMENT

FAIL_TIME FORCE_TIM RETRY_TIM

OS_USER

OS_TERMINAL

HOST

DB_USER

COMMIT#

25-NOV-25 26-NOV-25

Pour chaque N de 1 à 10, l'idée est :

1. Démarrer une transaction distribuée (UPDATE local + UPDATE distant).
2. Faire commit comment 'ORA-2PC-CRASH-TEST-N';
3. Observer :
4. select * from dba_2pc_pending;
5. select * from dba_2pc_neighbors;

Interprétation générale :

- Certains N simulent une panne :
 - avant la phase *prepare* → aucune trace durable, transaction annulée.
 - après *prepare* mais avant le commit global → transaction en état **in doubt** dans DBA_2PC_PENDING.
 - après le commit sur le site de référence mais panne sur un participant → recouvrement nécessaire via le **distributed recovery**.