RAPPORT DE PROJET

SQL3-ORACLE

module: BDA

Master1 IL - 24/25

Étude de cas :

Base de Données pour une Coopérative Agricole Intelligente

Binôme:

BOUAKKIZ Zineb
Groupe 3

BENBAREK Fatima
Groupe 2

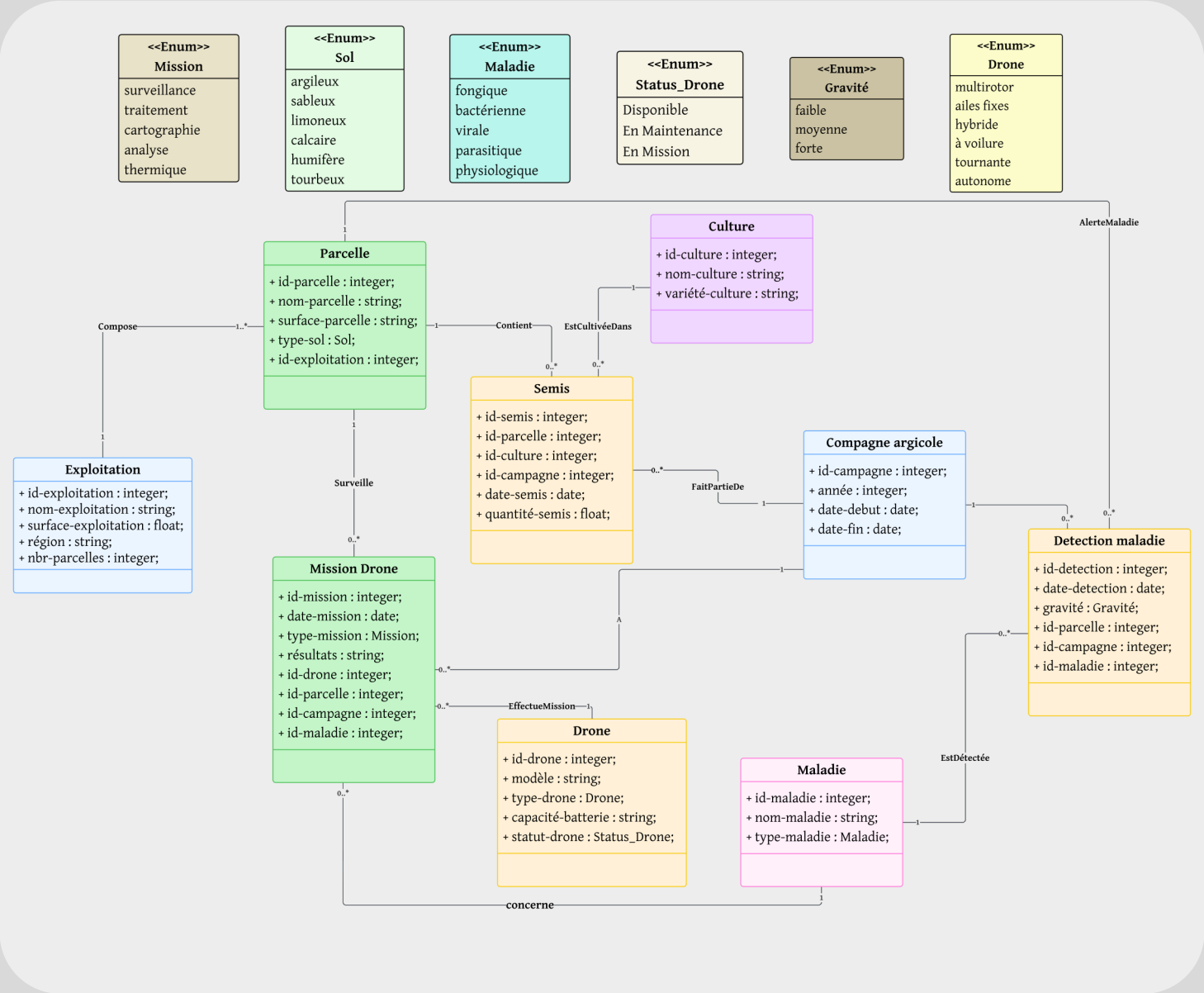
Objectifs:

L'objectif général de ce projet est de concevoir et implémenter une base de données orientée objet pour une coopérative agricole intelligente, en transformant un schéma relationnel en un modèle objet, en créant des tablespaces et un utilisateur, et en définissant des types, méthodes et tables pour gérer les données agricoles, tout en répondant à des requêtes spécifiques.

A- Modélisation orientée objet

1. **Transformation de schéma relationnel en un schéma Objet (diagramme de classes):**

- si ce n'est pas claire vous pouvez consulter le pdf, c'est plus lisible.



B- Création des TableSpaces et utilisateur

2. Création deux TableSpaces SQL3_TBS et SQL3_TempTBS:

Tablespace permanent et Tablespace temporaire

```
SQL> conn
Enter user-name: sys as sysdba
Enter password:
Connected.
SQL> CREATE TABLESPACE SQL3_TBS
  2    DATAFILE 'sql3_tbs01.dbf' SIZE 100M
  3    AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;

Tablespace created.

SQL> CREATE TEMPORARY TABLESPACE SQL3_TempTBS
  2    TEMPFILE 'sql3_temp_tbs01.dbf' SIZE 50M
  3    AUTOEXTEND ON NEXT 5M MAXSIZE UNLIMITED;

Tablespace created.
```

3. Création d'un utilisateur SQL3 en lui attribuant les deux tablespaces créés précédemment:

```
SQL> CREATE USER C##SQL3
  2    IDENTIFIED BY pwd
  3    DEFAULT TABLESPACE SQL3_TBS
  4    TEMPORARY TABLESPACE SQL3_TempTBS
  5    QUOTA UNLIMITED ON SQL3_TBS;

User created.
```

4. Attribution de tous les privilèges à cet utilisateur:

```
SQL> GRANT CONNECT, RESOURCE TO C##SQL3;

Grant succeeded.

SQL> GRANT DBA TO C##SQL3;

Grant succeeded.

SQL> GRANT UNLIMITED TABLESPACE TO C##SQL3;

Grant succeeded.
```

C- Langage de définition de données

5.définition de tous les types nécessaires, avec les associations qui existent:

on commance pas la définition des types objets incomplets:

```
SQL> CREATE TYPE TExploitation;
  2  /

Type created.

SQL> CREATE TYPE TParcelle;
  2  /

Type created.

SQL> CREATE TYPE TCulture;
  2  /

Type created.

SQL> CREATE TYPE TCampagneAgricole;
  2  /

Type created.

SQL> CREATE TYPE TSemis;
  2  /
```

```
SQL> CREATE TYPE TMaladie;
  2  /

Type created.

SQL> CREATE TYPE TDetectionMaladie;
  2  /

Type created.

SQL> CREATE TYPE TDrone;
  2  /

Type created.

SQL> CREATE TYPE TMissionDrone;
  2  /

Type created.
```

apres les types de collections de références (pour les associations 1-N et N-1):

Les REF facilitent la navigation “objet” (on suit la référence pour accéder aux attributs d’une autre table) tout en gardant les avantages du modèle relationnel (contrainte de clé étrangère, stockage normalisé)

```
SQL> CREATE TYPE TSetRefParcelle
  2  AS TABLE OF REF TParcelle;
  3  /

Type created.

SQL> CREATE TYPE TSetRefSemis
  2  AS TABLE OF REF TSemis;
  3  /

Type created.

SQL> CREATE TYPE TSetRefDetectionMa
  2  AS TABLE OF REF TDetectionMala
  3  /

Type created.

SQL> CREATE TYPE TSetRefMissionDron
  2  AS TABLE OF REF TMissionDrone;
  3  /

Type created.

SQL> CREATE TYPE TSetRefCampagne
  2  AS TABLE OF REF TCampagneAgric
  3  /

Type created.

SQL> CREATE TYPE TSetRefMaladie
  2  AS TABLE OF REF TMaladie;
  3  /

Type created.

SQL> CREATE TYPE TSetRefDrone
  2  AS TABLE OF REF TDrone;
  3  /

Type created.
```

Définition des types objets complets:
les autres types sont dans le le fichier qui contient tous le script

```
SQL> CREATE OR REPLACE TYPE TParcelle AS OBJECT (
  2    idParcelle           NUMBER,
  3    nomParcelle          VARCHAR2(100),
  4    surfaceParcelle      FLOAT,
  5    typeSol              VARCHAR2(20),          -- {argileux, sableux, …}
  6    exploitation         REF TExploitation,      -- 1 Exploitation
  7    semis                TSetRefSemis,          -- 0..* Semis
  8    detections           TSetRefDetectionMaladie, -- 0..* Détections
  9    missions             TSetRefMissionDrone    -- 0..* Missions
 10  );
 11  /
```

5. Définition de methodes:

avant l'implementation de la methode on ajout la signature dans TExploitation

- tous les autres fonctions sont dans le fichier de script, on vas expliquer les points intéressants et compliques

```
SQL> ALTER TYPE TExploitation
  2    ADD MEMBER FUNCTION TotalSurface RETURN NUMBER
  3    CASCADE;

Type altered.

SQL> CREATE OR REPLACE TYPE BODY TExploitation AS
  2
  3    MEMBER FUNCTION TotalSurface RETURN NUMBER IS
  4      somme NUMBER;
  5    BEGIN
  6      /* On caste self.parcelles en TSetRefParcelle puis on
  7         récupère les REF via COLUMN_VALUE */
  8      SELECT NVL(SUM(DEREF(t.COLUMN_VALUE).surfaceParcelle), 0)
  9        INTO somme
 10        FROM TABLE(
 11               CAST(self.parcelles AS TSetRefParcelle)
 12             ) t;
 13
 14      RETURN somme;
 15    END TotalSurface;
 16
 17  END;
 18  /

Type body created.
```

- CAST(self.parcelles AS TSetRefParcelle) : on force Oracle à traiter self.parcelles comme une collection manipulable dans une requête SQL.
- TABLE(...) permet de "décomposer" la collection en lignes virtuelles.
- DEREF(t.COLUMN_VALUE) suit le REF pour accéder aux attributs de l'objet référencé (ici surfaceParcelle).

pour la 2 eme methode on utilise le cast:

- Double itération : on « joint » deux nested tables via deux appels à TABLE(CAST(...)).
- MULTISET(SELECT ...) reconstruit une collection à partir du résultat SQL.
- DISTINCT au sein du MULTISET évite les doublons de références.
- On finit par caster le résultat en type collection attendu (TSetRefCulture).

```
SQL> CREATE OR REPLACE TYPE BODY TExploitation AS
  2
  3    /* Méthode 1 : TotalSurface */
  4    MEMBER FUNCTION TotalSurface RETURN NUMBER IS
  5      somme NUMBER;
  6    BEGIN
  7      SELECT NVL(SUM(DEREF(t.COLUMN_VALUE).surfaceParcelle), 0)
  8        INTO somme
  9        FROM TABLE(
 10               CAST(self.parcelles AS TSetRefParcelle)
 11             ) t;
 12      RETURN somme;
 13    END TotalSurface;
 14
 15    /* Méthode 2 : CulturesParCampagne */
 16    MEMBER FUNCTION CulturesParCampagne (
 17      p_campagne IN REF TCampagneAgricole
 18    ) RETURN TSetRefCulture IS
 19
 20      l_cultures TSetRefCulture;
 21    BEGIN
 22      SELECT CAST(
 23               MULTISET(
 24                 SELECT DISTINCT DEREF(s.COLUMN_VALUE).culture
 25                   FROM   TABLE(CAST(self.parcelles AS TSetRefParcelle)) p,
 26                          TABLE(CAST(DEREF(p.COLUMN_VALUE).semis AS TSetRefSemis)) s
 27                  WHERE  DEREF(s.COLUMN_VALUE).campagne = p_campagne
 28               )
 29               AS TSetRefCulture
 30             )
 31        INTO l_cultures
 32        FROM dual;
 33      RETURN l_cultures;
 34    END CulturesParCampagne;
 35
 36  END;
 37  /

Type body created.
```

- Lorsqu'on ajoute une méthode via ALTER TYPE … ADD MEMBER, il faut ensuite réécrire entièrement le TYPE BODY pour inclure la nouvelle méthode, même si on en conserve d'autres inchangées.
- Il faut veiller à la cohérence entre la spécification (ALTER TYPE) et l'implémentation (CREATE OR REPLACE TYPE BODY).

pour la fonction maladiesgraves

- On applique un filtre sur un attribut de l'objet référencé (gravite), en parcourant une nested table de références.

```
 23    /* Nouvelle méthode : maladies de gravité 'forte' */
 24    MEMBER FUNCTION MaladiesGraves RETURN TSetRefMaladie IS
 25      l_maladies TSetRefMaladie;
 26    BEGIN
 27      SELECT CAST(
 28               MULTISET(
 29                 SELECT DISTINCT DEREF(d.COLUMN_VALUE).maladie
 30                   FROM TABLE(CAST(self.detections AS TSetRefDetectionMaladie)) d
 31                  WHERE DEREF(d.COLUMN_VALUE).gravite = 'forte'
 32               )
 33               AS TSetRefMaladie
 34             )
 35        INTO l_maladies
 36        FROM dual;
 37      RETURN l_maladies;
 38    END MaladiesGraves;
 39
 40  END;
 41  /
```

7. Définition des tables nécessaires à la base de données

- Chaque attribut qui est une collection (ex. semis, detections, missions) est physiquement stocké dans une table séparée.
- Il faut gérer le mapping entre la ligne principale (Parcelles) et sa nested table (Parcelles_Semis), ce que fait Oracle automatiquement via un identifiant interne.

On obtient un modèle objet relationnel complet : tables “parent” + tables “enfant” pour les collections, sans écrire explicitement de jointures dans le DDL.

```
SQL> CREATE TABLE Exploitations OF TExploitation
  2  (
  3    CONSTRAINT pk_exploit PRIMARY KEY (idExploitation)
  4  )
  5    NESTED TABLE parcelles STORE AS Exploitations_Parcelles;

Table created.

SQL> CREATE TABLE Parcelles OF TParcelle
  2  (
  3    CONSTRAINT pk_parcelle PRIMARY KEY (idParcelle),
  4    CONSTRAINT fk_exploitation FOREIGN KEY (exploitation) REFERENCES Exploitations,
  5    CONSTRAINT check_type_sol CHECK (typeSol IN ('argileux', 'sableux', 'limoneux',
'calcaire', 'humifère', 'tourbeux'))
  6  )
  7    NESTED TABLE semis STORE AS Parcelles_Semis,
  8    NESTED TABLE detections STORE AS Parcelles_Detections,
  9    NESTED TABLE missions STORE AS Parcelles_Missions;

Table created.
```

D- Langage de manipulation de données

tous les insertions avec tous les details sont dans le fichier de script sql, mais on vas expliquer les points intéressants et compliqués

voici l’insertion dans la table exploitations:

```
SQL> INSERT INTO EXPLOITATIONS VALUES (
  2    TExploitation(
  3      1,                              -- idExploitation NUMBER
  4      'Ferme des Cèdres',             -- nomExploitation
  5      150,                            -- surfaceExploitation
  6      'Kabylie',                      -- region
  7      3,                              -- nbrParcelles
  8      TSetRefParcelle()               -- nested table initialement vide
  9    )
 10  );

1 row created.
```

- On n’insère pas ligne par ligne de colonnes primitives, mais un objet complet TExploitation.
- Les paramètres suivent strictement la signature du type (ordre et types), ce qui impose rigueur et cohérence.
- L’attribut parcelles est initialisé à une collection vide (TSetRefParcelle()), en préparation d’un remplissage ultérieur.

voici l'insertion dans la table parcelles:

```
SQL> INSERT INTO PARCELLES VALUES (
  2    TParcelle(
  3      1,                              -- idParcelle
  4      'Nord-1',                       -- nomParcelle
  5      50,                             -- surfaceParcelle
  6      'argileux',                     -- typeSol
  7      (SELECT REF(e)
  8         FROM Exploitations e
  9        WHERE e.idExploitation = 1),  -- réf. vers Ferme des Cèdres
 10      TSetRefSemis(),
 11      TSetRefDetectionMaladie(),
 12      TSetRefMissionDrone()
 13    )
 14  );

1 row created.
```

- Plutôt que de stocker une clé étrangère classique, on récupère un REF Oracle pointant sur la ligne d'objet.
- Cette sous-requête doit renvoyer exactement un résultat ; si elle renvoie zéro ou plusieurs lignes, l'insertion échoue.

Cancernant l'initialisation et peuplement des nested tables

- les collections nested tables ne sont pas remplies à l'insertion initiale : on découple création de l'objet et ajout des éléments.
- L'opérateur MULTISET UNION fusionne l'existant et le nouveau set, évitant d'écraser d'éventuels éléments précédemment ajoutés.
- CAST(MULTISET(...) AS TSetRefSemis) permet de convertir un résultat SQL (collection de REF) en type de collection attendu.

```
SQL> UPDATE Parcelles p SET semis = semis MULTISET UNION
  2    CAST( MULTISET (SELECT REF(s) FROM Semis s WHERE s.idSemis IN (1,2,3)) AS TSetRe
fSemis) WHERE p.idParcelle = 1;

1 row updated.
```

Population des nested tables dans plusieurs parents par exemple pour les missions:

```
SQL> UPDATE CampagnesAgricoles ca SET missions = missions MULTISET UNION
  2    CAST( MULTISET(SELECT REF(m) FROM MissionsDrone m WHERE m.idMission IN (1,2,3,4,5,6,7,8))AS TSetRefMissionDrone) WHERE ca.idCampagne = 3;

1 row updated.

SQL> UPDATE Drones dr SET missions = missions MULTISET UNION
  2    CAST( MULTISET(SELECT REF(m) FROM MissionsDrone m WHERE m.idMission IN (1))AS TSetRefMissionDrone) WHERE dr.idDrone = 1;

1 row updated.
```

- Chaque instance de TMissionDrone doit être liée à trois parents différents (drone, parcelle, campagne) ; on répète donc l'opération de MULTISET UNION sur chaque table mère concernée.
- La même REF de mission apparaît simultanément dans plusieurs nested tables, démontrant la flexibilité des collections d'objets.

E- Langage d'interrogation de données

9. Lister les exploitations et leurs parcelles:

```
SQL> SELECT
  2    e.idExploitation AS "ID Exploitation",
  3    e.nomExploitation AS "Nom Exploitation",
  4    DEREF(p.COLUMN_VALUE).idParcelle AS "ID Parcelle",
  5    DEREF(p.COLUMN_VALUE).nomParcelle AS "Nom Parcelle"
  6  FROM
  7    Exploitations e,
  8    TABLE(e.parcelles) p;

ID Exploitation Nom Exploitation                         ID Parcelle Nom Parcelle
--------------- ---------------------------------------- ----------- ------------------------------
              1 Ferme des Cèdres                                   1 Nord-1
                                                                   2 Sud-2
                                                                   3 Est-3

              2 Oasis Verte                                        4 Ouest-1
                                                                   5 Central

ID Exploitation Nom Exploitation                         ID Parcelle Nom Parcelle
--------------- ---------------------------------------- ----------- ------------------------------
              3 Domaine El-Kheir                                   6 Verger-1
                                                                   7 Verger-2
                                                                   8 Plaine
                                                                   9 Hauteur

              4 Agro-Sud                                          10 Test-1
                                                                  11 Oasis

ID Exploitation Nom Exploitation                         ID Parcelle Nom Parcelle
--------------- ---------------------------------------- ----------- ------------------------------
              5 Bio Champs                                        12 Bio-Est
                                                                  13 Bio-Ouest

              6 Ferme Saharienne                                  14 Sahara-1
                                                                  15 Sahara-2
                                                                  16 Palmeraie

ID Exploitation Nom Exploitation                         ID Parcelle Nom Parcelle
--------------- ---------------------------------------- ----------- ------------------------------
              7 Jardins du Tell                                   17 Montagne

              8 Coopérative El-Feth                               18 El-Feth-1
                                                                  19 El-Feth-2
                                                                  20 El-Feth-3

ID Exploitation Nom Exploitation                         ID Parcelle Nom Parcelle
--------------- ---------------------------------------- ----------- ------------------------------
              9 Les Vergers                                       21 Vergers-Nord
                                                                  22 Vergers-Sud

             10 TerraNova                                         23 Terra-1
                                                                  24 Terra-2
```

- TABLE(e.parcelles) : transforme la nested table d'Exploitations en un « jeu de lignes » que l'on peut joindre.
- DEREF(p.COLUMN_VALUE) : chaque ligne de la nested table est un REF; DEREF suit ce pointeur pour accéder aux attributs de TParcelle.

```
SQL> SELECT
  2    d.parcelle.idParcelle        AS "ID Parcelle",
  3    d.campagne.annee             AS "Année Campagne",
  4    COUNT(d.idDetection)         AS "Nombre Maladies",
  5    (COUNT(d.idDetection) * 100.0 / (
  6      SELECT COUNT(*)
  7      FROM DetectionsMaladie dm
  8      WHERE dm.campagne.idCampagne = d.campagne.idCampagne
  9    ))                           AS "Taux (%)"
 10  FROM
 11    DetectionsMaladie d
 12  GROUP BY
 13    d.parcelle.idParcelle,
 14    d.campagne.annee,
 15    d.campagne.idCampagne;

ID Parcelle Année Campagne Nombre Maladies   Taux (%)
----------- -------------- --------------- ----------
         16           2024               1 14.2857143
          5           2024               1 14.2857143
          1           2024               1 14.2857143
          2           2024               1 14.2857143
         10           2024               1 14.2857143
         14           2024               1 14.2857143
         15           2024               1 14.2857143
SQL>
```

10. Calculer le taux de maladies pour chaque parcelle et campagne:

- Agrégation multi-niveau : on regroupe d'abord par parcelle ET par campagne (clé primaire jointe dans le GROUP BY).
- Sous-requête corrélée dans la projection : pour chaque groupe, on calcule le total de détections de la même campagne pour le pourcentage.

11. Lister les missions de drones de traitement:

- Deux DEREF successifs : suivre d'abord la référence vers TDrone, puis vers TParcelle.
- Pas de jointure explicite : tout se fait via le modèle objet.

```
SQL> SELECT
  2    m.idMission AS "ID Mission",
  3    m.dateMission AS "Date Mission",
  4    DEREF(m.drone).modele AS "Modèle Drone",
  5    DEREF(m.parcelle).nomParcelle AS "Parcelle"
  6  FROM
  7    MissionsDrone m
  8  WHERE
  9    m.typeMission = 'traitement';

ID Mission Date Miss
---------- ---------
Modèle Drone
--------------------------------------------------------------------------------
Parcelle
--------------------------------------------------------------------------------
         2 07-APR-24
XAG P100
Nord-1

ID Mission Date Miss
---------- ---------
Modèle Drone
--------------------------------------------------------------------------------
Parcelle
--------------------------------------------------------------------------------
         6 11-APR-24
XAG P100
Sahara-2
SQL>
```

12. Historique des cultures semées par parcelle et exploitation

- Mix de nested table (TABLE(p.semis)) et jointure classique sur REF (p.exploitation.idExploitation = e.idExploitation).
- Chaînage de deref : DEREF(...).culture.nomCulture pour descendre jusqu'à l'attribut de l'objet TCulture.

```
SQL> SELECT
  2    p.idParcelle AS "ID Parcelle",
  3    e.nomExploitation AS "Exploitation",
  4    DEREF(s.COLUMN_VALUE).culture.nomCulture AS "Culture",
  5    DEREF(s.COLUMN_VALUE).dateSemis AS "Date Semis"
  6  FROM
  7    Parcelles p,
  8    TABLE(p.semis) s,
  9    Exploitations e
 10  WHERE
 11    p.exploitation.idExploitation = e.idExploitation;

ID Parcelle Exploitation
----------- ----------------------------------------------------------------------------------------------------
Culture                                                                                          Date Semi
-------------------------------------------------------------------------------------------------- ---------
          1 Ferme des Cèdres
Blé                                                                                              15-FEB-23
          2 Ferme des Cèdres
Orge                                                                                             18-FEB-23
          4 Oasis Verte

ID Parcelle Exploitation
----------- ----------------------------------------------------------------------------------------------------
Culture                                                                                          Date Semi
-------------------------------------------------------------------------------------------------- ---------
Tomate                                                                                           05-MAR-23
          6 Domaine El-Kheir
Olive                                                                                            25-FEB-23
          5 Oasis Verte
Pomme de terre                                                                                   10-FEB-24

ID Parcelle Exploitation
----------- ----------------------------------------------------------------------------------------------------
Culture                                                                                          Date Semi
-------------------------------------------------------------------------------------------------- ---------
```

13. Drones disponibles pour une mission de surveillance

- Filtre principal sur un attribut primitif (d.statutDrone).
- Sous-requête retourne des REF dé-référencés (pour extraire idDrone) et compare au champ primitif d.idDrone.

```
SQL> SELECT
  2    d.idDrone AS "ID Drone",
  3    d.modele AS "Modèle",
  4    d.typeDrone AS "Type"
  5  FROM
  6    Drones d
  7  WHERE
  8    d.statutDrone = 'Disponible'
  9    AND d.idDrone IN (
 10      SELECT DEREF(m.drone).idDrone
 11      FROM MissionsDrone m
 12      WHERE m.typeMission = 'surveillance'
 13    );

  ID Drone Modèle
---------- --------------------------------------------------
Type
--------------------
         1 DJI Agras T30
multirotor
         5 Matrice 300 RTK
multirotor
SQL>
```

14. Année avec le plus de maladies détectées

- Jointure implicite entre DetectionsMaladie et CampagnesAgricoles via la référence objet.
- Tri descendant et limitation (FETCH FIRST 1 ROW ONLY) pour ne récupérer que l'année la plus impactée.

```
SQL> SELECT
  2    c.annee AS "Année",
  3    COUNT(d.idDetection) AS "Nombre Maladies"
  4  FROM
  5    DetectionsMaladie d,
  6    CampagnesAgricoles c
  7  WHERE
  8    d.campagne.idCampagne = c.idCampagne
  9  GROUP BY
 10    c.annee
 11  ORDER BY
 12    COUNT(d.idDetection) DESC
 13  FETCH FIRST 1 ROW ONLY;

     Année Nombre Maladies
---------- ---------------
      2024               7
SQL>
```

Conclution:

En somme, ce projet illustre pleinement la puissance de l'object-relational sous Oracle :

1. une modélisation riche avec types objets, collections et références croisées,
2. des méthodes PL/SQL sophistiquées (CAST, TABLE, DEREF, MULTISET) encapsulant logique métier et agrégations,
3. un DML orienté objet pour peupler finement les nested tables et garantir l'intégrité des graphes d'objets,
4. un SQL de requête capable de naviguer naturellement dans ces structures pour produire analyses et rapports.

Cette approche démontre à la fois la flexibilité, la cohérence et la maintenabilité d'une base de données orientée-objet complexe.