C'est noté. Le fait que la table soit dans un schéma spécifique (crm.tiers) est une information importante pour la connexion et les droits d'accès.

Voici le code **Jupyter complet et mis à jour** pour cibler spécifiquement la table **crm.tiers**.

J'ai ajusté la requête de chargement et les commentaires pour refléter ce changement. Copiez ce bloc dans une nouvelle cellule de votre JupyterLab.

Python

```python
# ==============================================================================
# 1. IMPORT DES LIBRARIES (Pas de changement)
# ==============================================================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine
import oracledb
import phonenumbers
import re
from ydata_profiling import ProfileReport

# Options d'affichage
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

# ==============================================================================
# 2. CONNEXION ET CHARGEMENT (CORRIGÉ POUR crm.tiers)
# ==============================================================================
print("--- INITIALISATION ---")

# Vos identifiants Oracle
DB_USER = 'VOTRE_USER'
DB_PASSWORD = 'VOTRE_PASSWORD'
DB_DSN = 'localhost:1521/XEPDB1' # À adapter selon votre TNS

# Création du moteur
try:
    engine = create_engine(f'oracle+oracledb://{DB_USER}:{DB_PASSWORD}@{DB_DSN}')
    print("Connexion moteur SQLAlchemy : OK")
```

```
except Exception as e:
    print(f"Erreur de connexion : {e}")

# REQUÊTE MISE À JOUR : On cible crm.tiers
# On utilise les guillemets pour éviter les erreurs si la table est sensible à la casse
query = "SELECT * FROM crm.tiers"

print(f"Chargement des données depuis {query} ...")
try:
    df = pd.read_sql(query, engine)
    print(f"Succès ! {df.shape[0]} lignes chargées, {df.shape[1]} colonnes.")
except Exception as e:
    print(f"ERREUR SQL : Impossible de lire crm.tiers. Vérifiez les droits d'accès du user {DB_USER} sur le schéma CRM.")
    print(f"Détail : {e}")

# ==============================================================================
# 3. PROFILING : INCOHÉRENCES CLÉ PRIMAIRE (ID_CODE_TIERS_CRM)
# ==============================================================================
if 'df' in locals():
    print("\n[ANALYSE 1] INTÉGRITÉ DE LA CLÉ PRIMAIRE")

    # Vérifier si ID_CODE_TIERS_CRM est bien unique
    # Note: Adaptez le nom de la colonne si elle diffère dans crm.tiers
    pk_col = 'ID_CODE_TIERS_CRM'

    if pk_col in df.columns:
        n_dupes = df.duplicated(subset=[pk_col], keep=False).sum()
        print(f"-> Doublons de clé primaire ({pk_col}) : {n_dupes} lignes affectées")

        if n_dupes > 0:
            print("   Exemple de doublons (Même ID, contenu potentiellement différent) :")
            print(df[df.duplicated(subset=[pk_col], keep=False)].sort_values(pk_col).head(6))
    else:
        print(f"ATTENTION : La colonne {pk_col} n'existe pas dans crm.tiers. Vérifiez le nom des colonnes avec df.columns")

# ==============================================================================
# 4. PROFILING : VALIDITÉ DES TÉLÉPHONES (CRD_TELEPHONE)
# ==============================================================================
    print("\n[ANALYSE 2] VALIDITÉ TÉLÉPHONE")

    def check_phone(num):
        if pd.isna(num) or str(num).strip() == '': return "VIDE"
```

```
        try:
            # On force la région MA (Maroc) par défaut pour l'analyse
            parsed = phonenumbers.parse(str(num), "MA")
            if phonenumbers.is_valid_number(parsed):
                return "VALIDE"
            else:
                return "INVALIDE (Format)"
        except:
            return "INVALIDE (Erreur Parsing)"

    if 'CRD_TELEPHONE' in df.columns:
        df['AUDIT_TEL'] = df['CRD_TELEPHONE'].apply(check_phone)
        print(df['AUDIT_TEL'].value_counts())

        # Visualisation rapide
        plt.figure(figsize=(6,4))
        sns.countplot(x='AUDIT_TEL', data=df, palette='viridis')
        plt.title("Qualité des Numéros de Téléphone")
        plt.show()

# ==============================================================================
# 5. PROFILING : DÉTECTION DOUBLONS CACHÉS (Même ICE/Email)
# ==============================================================================
    print("\n[ANALYSE 3] DOUBLONS FONCTIONNELS (Même client, ID différent)")

    # Doublons ICE
    if 'ID_ICE_CIN' in df.columns:
        dupe_ice = df[df.duplicated('ID_ICE_CIN', keep=False) & df['ID_ICE_CIN'].notnull()]
        print(f"-> Clients avec le même ICE mais plusieurs lignes : {len(dupe_ice)}")

    # Doublons EMAIL
    if 'CRD_EMAIL' in df.columns:
        dupe_email = df[df.duplicated('CRD_EMAIL', keep=False) & df['CRD_EMAIL'].notnull()]
        print(f"-> Clients avec le même Email mais plusieurs lignes : {len(dupe_email)}")

# ==============================================================================
# 6. RAPPORT HTML FINAL
# ==============================================================================
    print("\n[FINAL] GÉNÉRATION DU RAPPORT HTML...")
    # On retire la colonne d'audit temporaire pour le rapport final
    df_report = df.drop(columns=['AUDIT_TEL'], errors='ignore')

    profile = ProfileReport(df_report, title="Audit Qualité - crm.tiers", explorative=True)
```

```
profile.to_file("Audit_CRM_Tiers.html")
print("Fichier 'Audit_CRM_Tiers.html' généré avec succès.")
```

Point d'attention sur crm.tiers :

Si vous obtenez une erreur **"Table or view does not exist"** avec ce code, cela signifie généralement que l'utilisateur connecté (DB_USER) n'a pas les droits SELECT sur le schéma crm.

Dans ce cas, vous devrez exécuter ceci en SQL (en tant qu'admin) :

SQL

```
GRANT SELECT ON crm.tiers TO VOTRE_USER;
```