C'est compris. Voici une approche de **Data Profiling pur et dur**. L'objectif ici n'est pas de toucher à la donnée, mais de **scanner** la table comme une radiographie pour révéler toutes les fractures (anomalies).

Ce Notebook est structuré pour **catégoriser les anomalies** sans interrompre l'exécution (nous utiliserons des fonctions de vérification qui retournent des indicateurs True/False ou des statuts, plutôt que des blocs try/except de correction).

Voici le code complet à copier dans votre JupyterLab.

Notebook : Audit et Détection d'Anomalies (Read-Only)

Python

```
# ==============================================================================
# 1. IMPORT DES LIBRARIES
# ==============================================================================
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

# Configuration pour afficher toutes les colonnes
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

# ==============================================================================
# 2. CHARGEMENT DES DONNÉES (RAW)
# ==============================================================================
# Configuration Oracle
DB_USER = 'VOTRE_USER'
DB_PASSWORD = 'VOTRE_PASSWORD'
DB_DSN = 'localhost:1521/XEPDB1'

# Connexion
engine = create_engine(f'oracle+oracledb://{DB_USER}:{DB_PASSWORD}@{DB_DSN}')
```

```python
# On charge TOUT pour l'audit
query = "SELECT * FROM ID_CODE_TIERS_CRM"
df = pd.read_sql(query, engine)

print(f"--- VOLUMÉTRIE ---")
print(f"Lignes chargées : {df.shape[0]}")
print(f"Colonnes : {df.shape[1]}")
print("-" * 30)

# ==============================================================================
# 3. ANOMALIES DE STRUCTURE & INTÉGRITÉ (Clé Primaire)
# ==============================================================================
print("\n[SCAN 1] ANALYSE DE LA CLÉ PRIMAIRE (ID_CODE_TIERS_CRM)")

# A. Doublons exacts (Ligne entièrement identique)
exact_dupes = df.duplicated().sum()
print(f"-> Lignes 100% identiques (Doublons stricts) : {exact_dupes}")

# B. Doublons d'ID (Schizophrénie : Même ID, données différentes)
# On compte combien de fois chaque ID apparaît
id_counts = df['ID_CODE_TIERS_CRM'].value_counts()
duplicate_ids = id_counts[id_counts > 1]
print(f"-> IDs utilisés plusieurs fois (Violation PK) : {len(duplicate_ids)}")

if len(duplicate_ids) > 0:
    print("   Exemple d'IDs problématiques :")
    print(duplicate_ids.head(5).to_string())

# ==============================================================================
# 4. ANOMALIES DE COMPLÉTUDE (VALEURS NULL)
# ==============================================================================
print("\n[SCAN 2] CARTE THERMIQUE DES MANQUANTS (MISSING VALUES)")

# Calcul du pourcentage de vide par colonne
missing_percent = df.isnull().mean() * 100
cols_critical_missing = missing_percent[missing_percent > 0].sort_values(ascending=False)

print("-> Top 10 des colonnes les plus vides (%) :")
print(cols_critical_missing.head(10))

# Visualisation (Heatmap)
plt.figure(figsize=(12, 6))
sns.heatmap(df.isnull(), cbar=False, yticklabels=False, cmap='viridis')
```

```
plt.title("Visualisation des données manquantes (Jaune = NULL)")
plt.show()

# ==============================================================================
# 5. ANOMALIES DE QUALITÉ DU CONTENU (VALIDITÉ MÉTIER)
# ==============================================================================
print("\n[SCAN 3] VALIDATION DU CONTENU (Phone, Email, ICE)")

# Fonction de détection (Sans correction, juste Flagging)
def audit_phone_number(num):
    if pd.isna(num) or num == '':
        return "MANQUANT"
    try:
        # On suppose MA par défaut, le parser nous dira si c'est valide
        parsed = phonenumbers.parse(str(num), "MA")
        if not phonenumbers.is_valid_number(parsed):
            return "INVALIDE (Format)"
        return "VALIDE"
    except:
        return "INVALIDE (Parsing Impossible)"

def audit_email(email):
    if pd.isna(email) or email == '':
        return "MANQUANT"
    # Regex simple pour audit
    if not re.match(r"[^@]+@[^@]+\.[^@]+", str(email)):
        return "INVALIDE (Syntaxe)"
    return "VALIDE"

def audit_ice(ice):
    if pd.isna(ice): return "MANQUANT"
    ice_str = str(ice).replace('.0', '') # Nettoyage float vers string
    if len(ice_str) != 15:
        return f"ANOMALIE LONGUEUR ({len(ice_str)} chiffres)"
    if not ice_str.isdigit():
        return "ANOMALIE CONTENU (Non-numérique)"
    return "VALIDE"

# Application des audits (Création de colonnes de métadonnées temporaires)
print("... Scan des téléphones en cours ...")
df['AUDIT_TEL_STATUS'] = df['CRD_TELEPHONE'].apply(audit_phone_number)

print("... Scan des emails en cours ...")
df['AUDIT_EMAIL_STATUS'] = df['CRD_EMAIL'].apply(audit_email)
```

```
print("... Scan des ICE en cours ...")
df['AUDIT_ICE_STATUS'] = df['ID_ICE_CIN'].apply(audit_ice)

# RÉSULTATS DU SCAN CONTENU
print("\n--- RÉSULTATS AUDIT TÉLÉPHONE ---")
print(df['AUDIT_TEL_STATUS'].value_counts())

print("\n--- RÉSULTATS AUDIT EMAIL ---")
print(df['AUDIT_EMAIL_STATUS'].value_counts())

print("\n--- RÉSULTATS AUDIT ICE (Identifiant Entrep.) ---")
print(df['AUDIT_ICE_STATUS'].value_counts())

# ==============================================================================
# 6. ANOMALIES DE COHÉRENCE (DOUBLONS FONCTIONNELS)
# ==============================================================================
print("\n[SCAN 4] DÉTECTION DES DOUBLONS CACHÉS (Même ICE/Email, ID Différent)")

# Doublons par ICE
dupes_ice = df[df.duplicated(subset=['ID_ICE_CIN'], keep=False) & df['ID_ICE_CIN'].notnull()]
print(f"-> Nombre de lignes partageant le même ICE avec une autre ligne : {len(dupes_ice)}")

# Doublons par Email
dupes_email = df[df.duplicated(subset=['CRD_EMAIL'], keep=False) & df['CRD_EMAIL'].notnull()]
print(f"-> Nombre de lignes partageant le même Email avec une autre ligne : {len(dupes_email)}")

# ==============================================================================
# 7. ANOMALIES DE FORMAT (COLONNES BOOLÉENNES/TEXTE)
# ==============================================================================
print("\n[SCAN 5] VALEURS NON-STANDARDISÉES")

# Vérification des colonnes qui semblent être des booléens ou codes courts
cols_to_check = ['SYNERGIE', 'REVENDEUR', 'COMMUN', 'PAYS']

for col in cols_to_check:
    if col in df.columns:
        print(f"\nDistribution des valeurs pour '{col}':")
        # On affiche les valeurs uniques pour voir si on a "Oui", "OUI", "Yes", "1", etc.
        print(df[col].value_counts(dropna=False).head(10))

# ==============================================================================
# 8. GÉNÉRATION DU RAPPORT COMPLET (YDATA PROFILING)
# ==============================================================================
```

```
print("\n[FINAL] GÉNÉRATION DU RAPPORT HTML AUTOMATISÉ...")

# On exclut les colonnes d'audit qu'on vient de créer pour garder le rapport propre sur la donnée source
cols_exclude = ['AUDIT_TEL_STATUS', 'AUDIT_EMAIL_STATUS', 'AUDIT_ICE_STATUS']
df_source = df.drop(columns=cols_exclude, errors='ignore')

profile = ProfileReport(
    df_source,
    title="Rapport d'Anomalies - ID_CODE_TIERS_CRM",
    minimal=False, # Mode complet
    explorative=True
)

profile.to_file("Audit_Anomalies_CRM.html")
print("Rapport généré : Audit_Anomalies_CRM.html")
```

Ce que ce script va révéler :

1. **Le diagnostic Vital (PK) :** Vous saurez exactement combien d'IDs sont corrompus.
2. **La "Santé" des contacts :** Le value_counts() sur AUDIT_TEL_STATUS vous dira directement : "Vous avez 30% de numéros invalides et 10% manquants".
3. **La "Saleté" des champs textes :** La section 7 montrera si vous avez des mélanges du type "FRANCE", "Fr", "Maroc", "MAROC" dans la colonne Pays.
4. **Les "Doublons Cachés" :** Vous verrez combien de clients ont été créés plusieurs fois (même ICE) mais avec des IDs différents.

C'est l'étape obligatoire avant d'écrire la moindre ligne de code de correction.