```python
import os
import time
import pandas as pd
import requests
from sqlalchemy import create_engine, text
from dotenv import load_dotenv

# Load environment variables
load_dotenv()
API_KEY = os.getenv('OMDB_API_KEY')
if not API_KEY:
    raise ValueError("OMDB_API_KEY not found in .env file")

# Database setup (SQLite)
engine = create_engine('sqlite:///movie_pipeline.db')

def extract_csv_data():
    """Extract data from CSV files."""
    movies_df = pd.read_csv('movies.csv')
    ratings_df = pd.read_csv('ratings.csv')
    return movies_df, ratings_df

def fetch_omdb_data(title):
    """Fetch additional data from OMDb API."""
    url = f"http://www.omdbapi.com/?t={title}&apikey={API_KEY}"
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        if data.get('Response') == 'True':
            return {
                'director': data.get('Director', 'Unknown'),
                'plot': data.get('Plot', 'Unknown'),
                'box_office': data.get('BoxOffice', 'Unknown')
            }
    return None  # Movie not found

def transform_data(movies_df, ratings_df):
    """Transform and enrich data."""
    # Clean movies data
    movies_df['year'] = pd.to_numeric(movies_df['title'].str.extract(r'\((\d{4})\)', expand=False), errors='coerce')
    movies_df['title'] = movies_df['title'].str.replace(r'\s*\(\d{4}\)', '', regex=True)
    movies_df['genres'] = movies_df['genres'].str.replace('|', ', ')

    # Enrich with API data
    enriched_movies = []
    for _, row in movies_df.iterrows():
        api_data = fetch_omdb_data(row['title'])
        if api_data:
            row = row.copy()
            row.update(api_data)
        else:
            row['director'] = 'Unknown'
            row['plot'] = 'Unknown'
            row['box_office'] = 'Unknown'

        # Feature engineering: Add decade
        if pd.notna(row['year']):
            row['decade'] = f"{(row['year'] // 10) * 10}s"
        else:
            row['decade'] = 'Unknown'

        enriched_movies.append(row)
        time.sleep(1)  # Rate limit handling

    enriched_movies_df = pd.DataFrame(enriched_movies)

    # Clean ratings data
    ratings_df['timestamp'] = pd.to_datetime(ratings_df['timestamp'], unit='s')

    return enriched_movies_df, ratings_df

def load_data(movies_df, ratings_df):
    """Load data into the database idempotently."""
    with engine.connect() as conn:
        # Load movies (check for existence)
        for _, row in movies_df.iterrows():
            query = text("""
                INSERT OR IGNORE INTO movies (movieId, title, year, genres, director, plot, box_office, decade)
                VALUES (:movieId, :title, :year, :genres, :director, :plot, :box_office, :decade)
            """)
            conn.execute(query, row.to_dict())

        # Load ratings (check for existence via primary key)
        for _, row in ratings_df.iterrows():
            query = text("""
                INSERT OR IGNORE INTO ratings (userId, movieId, rating, timestamp)
                VALUES (:userId, :movieId, :rating, :timestamp)
            """)
            conn.execute(query, {
                'userId': row['userId'],
                'movieId': row['movieId'],
                'rating': row['rating'],
                'timestamp': row['timestamp'].timestamp()  # Convert to int
            })
        conn.commit()

if __name__ == "__main__":
    print("Starting ETL pipeline...")
    movies_df, ratings_df = extract_csv_data()
    movies_df, ratings_df = transform_data(movies_df, ratings_df)
    load_data(movies_df, ratings_df)
    print("ETL pipeline completed!")
```