ETL (Extraction, Transformation, Load)

Project Documentation (3rd Cut)

Javier Alejandro Vergara Zorrilla

Dayanna Vanessa Suarez Mazuera (2221224)

Juan Camilo Buitrago Gonzalez (2221314)

David Felipe Martínez (2205286)

Universidad Autónoma de Occidente

Group 4

May 24, 2024

This document has the purpose of illustrating the process which was done in order to develop the exercise, the elements, platforms, tools used during the same.

This cut, the challenge was developing a data streaming using Apache Kafka and the platform Airflow from the same developer, Apache Software Foundation. In the last cut, the team extract information to complement our dataframe using the Rawg Games API, unfortunately, during the development the owner went bankrupt due to economic reasons, making it impossible to perform further API queries, however, the dataset was saved in a csv file, and using this file, the exercise was possible to make.

With the API dataset, the team made a cleaning process and a merge between our dataframe and the API dataframe, with this new dataframe, the team developed a dimensional model, however there was problems with this dimensional model and was impossible to present during the last cut, in the present, the dimensional model was corrected and this will be streamed using Apache Kafka.

The following list indicate the merged dataframe characteristics and tools which were used during the develop process:

- **cleaned_merge_data**
    - **Number and name of the columns:** 6 columns
        - Title (object, 0 nulls)
        - Released (object, 0 nulls)
        - Rating (object, 0 nulls)
        - Rating_count (integer, 0 nulls)

        - Genres (object, 0 nulls)
        - Platforms (object, 0 nulls)
    - **Number of rows:** 14.707

- **Apache Kafka:** Apache Kafka is a distributed streaming platform that enables real-time publishing, subscribing, storing and processing of log streams.
- **Apache Airflow:** Apache Airflow is a workflow management platform for scheduling, monitoring, and managing complex workflows.
- **Docker:** Docker is a containerization platform that allows you to develop, ship and run applications in containers. A container is a standard software unit that packages the code and all its dependencies so that the application runs quickly and reliably in different environments.
- **Jupyter Notebook:** Jupyter Notebooks is an open-source web application that allows you to create and share documents containing live code, equations, visualizations and narrative text. It is widely used in data analysis, scientific research and education.

Libraries

- **asttokens**: Used to parse and manipulate Python abstract syntax trees (AST).
- **colorama**: Provides facilities for printing colored text to the console.
- **comm**: Library to handle communications efficiently between different processes or components.
- **contourpy**: Used to generate 2D and 3D contour plots.
- **cycler**: Provides tools to generate and manage color sequences and cyclic styles.

- **debugpy**: A remote debugger for Python.
- **decorator**: Allows you to define function decorators in Python.
- **executing**: Used to execute Python code with execution trace information.
- **fonttools**: A library for manipulating TrueType, OpenType, AFM and other font formats.
- **greenlet**: Implements lightweight microthreading for Python.
- **ipykernel**: Implements the IPython kernel for Jupyter.
- **ipython**: The enhanced interactive Python interpreter.
- **jedi**: Used for parsing and auto-completion of Python code.
- **jupyter_client**: Enables communication between client and server in Jupyter environments.
- **jupyter_core**: Jupyter core functionalities.
- **kiwisolver**: Solves optimization problems to solve mathematical equations.
- **matplotlib**: A library for creating static, interactive and animated visualizations in Python.
- **packaging**: Used to work with Python package distributions.
- **pandas**: Provides data structures and tools for data analysis in Python.
- **parso**: A Python syntax analyzer.
- **pillow**: Python image processing library.
- **platformdirs**: Helps find operating system-specific paths in Python.
- **prompt-toolkit**: Used to create interactive command line interfaces in Python.
- **psutil**: Provides an interface for querying operating system and process information in Python.
- **psycopg2-binary**: PostgreSQL database adapter for Python.

- **pure-eval**: Allows you to safely evaluate Python expressions.
- **Pygments**: A generic syntax highlighter written in Python.
- **pyparsing**: Used for parsing and working with Python grammars.
- **python-dateutil:** Provides functionality for working with dates and times in Python.
- **python-dotenv**: Loads environment variables from .env files in Python.
- **kafka**-python: A library for interacting with Kafka in Python.
- **pytz**: Functionalities for working with time zones in Python.
- **pyzmq**: Provides bindings for ZeroMQ in Python.
- **seaborn**: A data visualization library based on Matplotlib in Python.
- **six**: Library to ensure compatibility between Python 2 and Python 3.
- **sqlalchemy**: Used to work with relational databases in Python.
- **tornado**: An asynchronous web and networking framework in Python.
- **traitlets**: Provides a configuration system for Python.
- **typing_extensions**: Python extensions for type annotations.
- **tzdata**: Provides timezone information in Python.
- **wcwidth**: Determines the width of fixed-width characters in Python.

notebooks/eda_004.ipynb

Resuming the last release, we performed a comprehensive process of loading, transforming and merging data from two different sources: a Metacritic dataset and several datasets obtained through an API. To do this, we used several key libraries, including pandas for data manipulation, re for regular expressions, json for JSON data manipulation, sqlalchemy for database connection, and os and sys for system management and path configuration. First, we established a connection to the PostgreSQL database using SQLAlchemy and credentials stored in a configuration file. Once the connection was established, we loaded the Metacritic data from a CSV file into a pandas DataFrame. In parallel, we loaded several CSV files containing the API data, which we concatenated into a single DataFrame. Then, we perform several transformations on the API data: we extract the names of classifications and genres from JSON structures using the json library, and normalize the platform names with the help of regular (re)expressions. We then select the relevant columns from the API data, adjust the column names and formats to ensure consistency, and apply a mapping to standardize the values of the classifications.

For the Metacritic data, we transformed the data in a similar way: we normalized the titles and release dates and standardized the column names. Once both data sources were prepared and in a consistent format, we proceeded to merge them using an outer join on key columns such as title, release date, genres, platforms, rating and rating count. During this process, we identified and reviewed rows with null values to understand the discrepancies and subsequently removed these rows to obtain a clean data set. The merged and cleaned data was saved in CSV files for later use. This detailed process not only ensures the integration of multiple data sources, but also the transformation and standardization necessary to maintain data quality and consistency.

dags/producer.py

First, the script loads the database configuration from a database.ini file using the load_config function, which uses the ConfigParser library. This configuration includes details such as the user, password, host and database name.

Next, the kafka_producer function is defined, which takes the database configuration, the name of the table from which the data will be extracted, and the name of the Kafka topic to which the data will be sent. The function creates an SQLAlchemy engine to establish a connection to the PostgreSQL database and executes an SQL query to select all the data from the specified table. This data is loaded into a DataFrame using pandas.

Next, a Kafka producer is set up using KafkaProducer. Each row in the DataFrame is converted to a JSON object and sent to the specified Kafka topic using the send method. The timestamp of each message sent is printed for tracking. In addition, a 2 second delay is included between sending each message.

Finally, if the file is executed directly (__name__ == "__main__"), the database configuration is loaded and the kafka_producer function is called to send the data from the merged_data table to the Kafka games_stream topic.

dags/consumer.py

First, the necessary libraries are imported, including json for handling JSON data, sys for system manipulation, requests for making HTTP requests, and pandas for structured data manipulation.

Next, you define the Power BI streaming dataset connection point URL, which will be used to send the consumed Kafka data to the Power BI streaming dataset.

A Kafka consumer (KafkaConsumer) is configured with the necessary options, such as topic name, startup servers, automatic clearing settings, group identifier, etc.

Then, the script enters an infinite loop where it continuously consumes messages from the Kafka topic. For each consumed message, the JSON payload is extracted and stored in a data_raw list. Then, a pandas DataFrame (data_df) is created from the raw data using a list of predefined headers. The DataFrame is converted to JSON and byte-encoded.

Finally, a POST request is made to the Power BI API with the JSON data. The response of the request is printed to verify that the data has been sent correctly.

originally, the information was streamed as part of an airflow task, however, there was a bug with the services that forced the creation of the consumer and producer external to the airflow workflow.

dags/etl_decorators.py

The dag_decorators.py file contains a set of decorators that define an ETL (Extract, Transform, Load) workflow using Airflow. These decorators allow you to define individual tasks and how they are connected together to form a DAG (Directed Acyclic Graph).

At the beginning, the necessary libraries are imported, including datetime for date and time handling, as well as the Airflow specific decorations (dag and task). The etl module, which contains the functions to perform data extraction, transformation and loading, is also imported.

You define the default arguments for the DAG, which include the owner, the email settings, the number of retry attempts, etc.

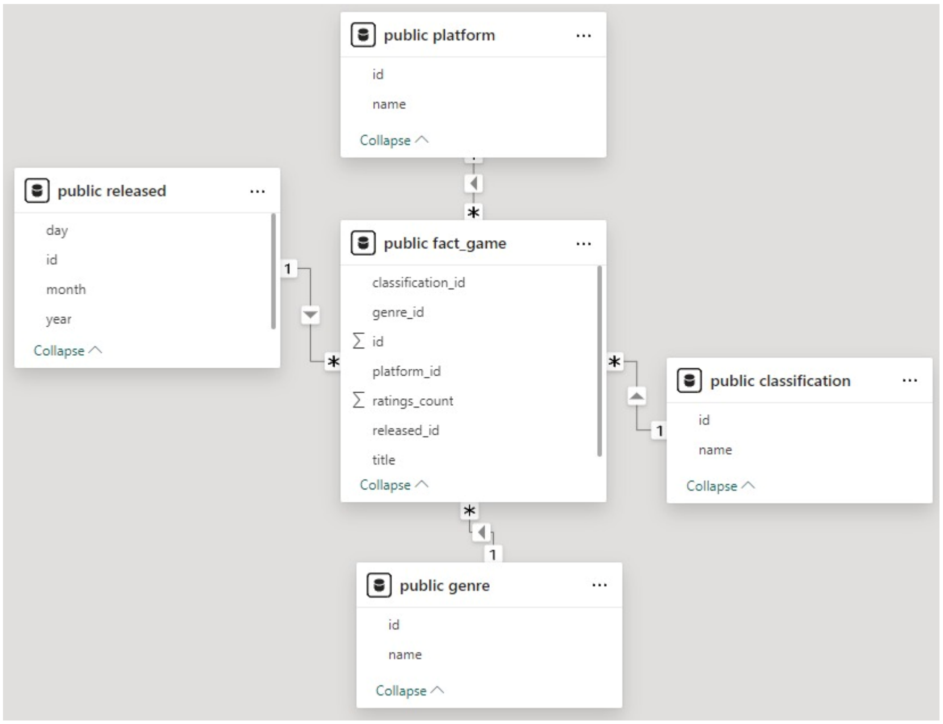
Then the DAG itself is defined using the dag decorator. You specify the default arguments, the description of the DAG, the scheduling interval, and the tags.

Within the DAG, several tasks are defined using the task decorator. Each task represents a stage of the ETL workflow. The tasks include extracting data from Metacritic, extracting data from the API, transforming data from Metacritic, transforming data from the API, merging data and loading data into PostgreSQL, and sending data to Kafka.

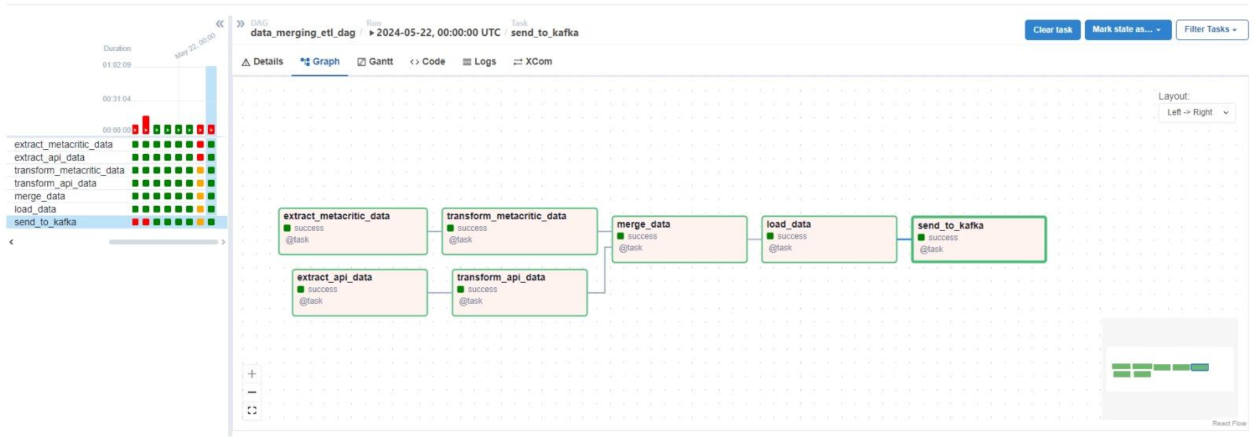
In the root, we also find key files, the docker-compose.yml file contains the docker configuration for the containers kafka and airflow, we also have the Dockerfile file, this file contains the specifications to create the images of the containers of the respective services, this configuration includes version of the images, libraries, etc. These files provide the system with the necessary information to be able to execute the services correctly, without them, it would be very difficult to execute the whole script satisfactorily.

Evidences

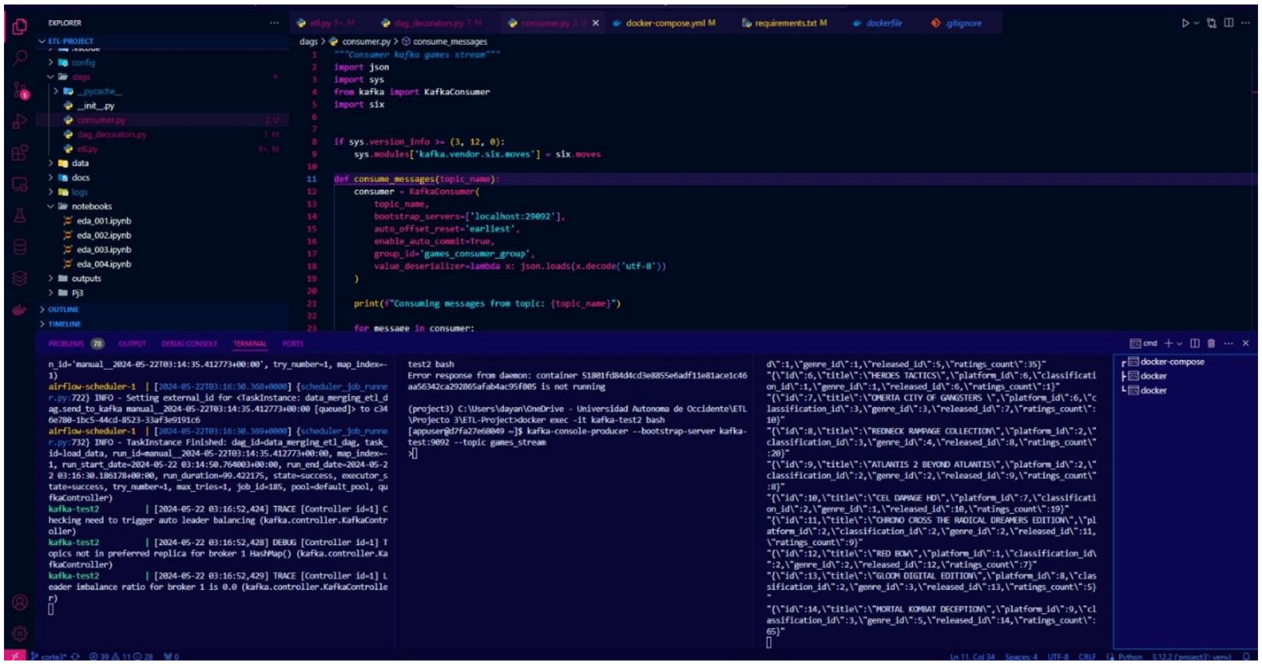
Dimensional Model



Workflow running successfully



Data Streaming



```python
"""Consumer kafka games stream"""
import json
import sys
from kafka import KafkaConsumer
import six


if sys.version_info >= (3, 12, 0):
    sys.modules['kafka.vendor.six.moves'] = six.moves

def consume_messages(topic_name):
    consumer = KafkaConsumer(
        topic_name,
        bootstrap_servers=['localhost:29092'],
        auto_offset_reset='earliest',
        enable_auto_commit=True,
        group_id='games_consumer_group',
        value_deserializer=lambda x: json.loads(x.decode('utf-8'))
    )

    print(f"Consuming messages from topic: {topic_name}")

    for message in consumer:
```

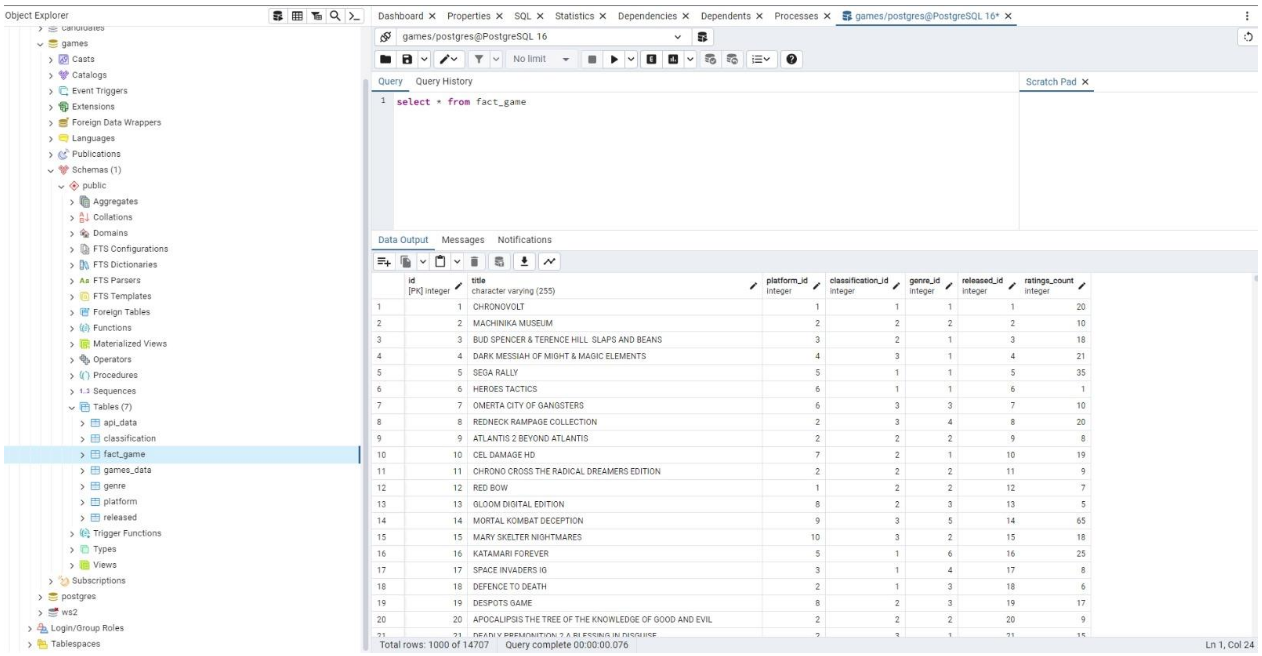
```
n_id='manual__2024-05-22T03:14:35.412773+00:00', try_number=1, map_index=-
1)
airflow-scheduler-1  | [2024-05-22T03:16:30.368+0000] {scheduler_job_runne
r.py:722} INFO - Setting external_id for <TaskInstance: data_merging_etl_d
ag.send_to_kafka manual__2024-05-22T03:14:35.412773+00:00 [queued]> to c34
6e780-1bc5-44cd-8523-33af3e9191c6
airflow-scheduler-1  | [2024-05-22T03:16:30.369+0000] {scheduler_job_runne
r.py:732} INFO - TaskInstance Finished: dag_id=data_merging_etl_dag, task_
id=load_data, run_id=manual__2024-05-22T03:14:35.412773+00:00, map_index=-
1, run_start_date=2024-05-22 03:14:50.764003+00:00, run_end_date=2024-05-2
2 03:16:30.186178+00:00, run_duration=99.422175, state=success, executor_s
tate=success, try_number=1, max_tries=1, job_id=185, pool=default_pool, qu
fkaController)
kafka-test2          | [2024-05-22 03:16:52,424] TRACE [Controller id=1] C
hecking need to trigger auto leader balancing (kafka.controller.KafkaContr
oller)
kafka-test2          | [2024-05-22 03:16:52,428] DEBUG [Controller id=1] T
opics not in preferred replica for broker 1 HashMap() (kafka.controller.Ka
fkaController)
kafka-test2          | [2024-05-22 03:16:52,429] TRACE [Controller id=1] L
eader imbalance ratio for broker 1 is 0.0 (kafka.controller.KafkaControlle
r)
```

```
test2 bash
Error response from daemon: container 51801fd84d4cd3e8855e6adf11e81ace1c46
aa56342ca292865afab4ac95f805 is not running

(project3) C:\Users\dayan\OneDrive - Universidad Autonoma de Occidente\ETL
\Projecto 3\ETL-Project>docker exec -it kafka-test2 bash
[appuser@d7fa27e68049 ~]$ kafka-console-producer --bootstrap-server kafka-
test:9092 --topic games_stream
>
```

```
d\":1,\"genre_id\":1,\"released_id\":5,\"ratings_count\":35}"
"{\"id\":6,\"title\":\"HEROES TACTICS\",\"platform_id\":6,\"classificati
on_id\":1,\"genre_id\":1,\"released_id\":6,\"ratings_count\":1}"
"{\"id\":7,\"title\":\"OMERTA CITY OF GANGSTERS \",\"platform_id\":6,\"c
lassification_id\":3,\"genre_id\":3,\"released_id\":7,\"ratings_count\":
10}"
"{\"id\":8,\"title\":\"REDNECK RAMPAGE COLLECTION\",\"platform_id\":2,\"
classification_id\":3,\"genre_id\":4,\"released_id\":8,\"ratings_count\"
:20}"
"{\"id\":9,\"title\":\"ATLANTIS 2 BEYOND ATLANTIS\",\"platform_id\":2,\"
classification_id\":2,\"genre_id\":2,\"released_id\":9,\"ratings_count\"
:8}"
"{\"id\":10,\"title\":\"CEL DAMAGE HD\",\"platform_id\":7,\"classificati
on_id\":2,\"genre_id\":1,\"released_id\":10,\"ratings_count\":19}"
"{\"id\":11,\"title\":\"CHRONO CROSS THE RADICAL DREAMERS EDITION\",\"pl
atform_id\":2,\"classification_id\":2,\"genre_id\":2,\"released_id\":11,
\"ratings_count\":9}"
"{\"id\":12,\"title\":\"RED BOW\",\"platform_id\":1,\"classification_id\
":2,\"genre_id\":2,\"released_id\":12,\"ratings_count\":7}"
"{\"id\":13,\"title\":\"GLOOM DIGITAL EDITION\",\"platform_id\":8,\"clas
sification_id\":2,\"genre_id\":3,\"released_id\":13,\"ratings_count\":5}
"
"{\"id\":14,\"title\":\"MORTAL KOMBAT DECEPTION\",\"platform_id\":9,\"cl
assification_id\":3,\"genre_id\":5,\"released_id\":14,\"ratings_count\":
65}"
```

Database with the fact table



```
select * from fact_game
```

	id [PK] integer	title character varying (255)	platform_id integer	classification_id integer	genre_id integer	released_id integer	ratings_count integer
1	1	CHRONOVOLT	1	1	1	1	20
2	2	MACHINIKA MUSEUM	2	2	2	2	10
3	3	BUD SPENCER & TERENCE HILL SLAPS AND BEANS	3	2	1	3	18
4	4	DARK MESSIAH OF MIGHT & MAGIC ELEMENTS	4	3	1	4	21
5	5	SEGA RALLY	5	1	1	5	35
6	6	HEROES TACTICS	6	1	1	6	1
7	7	OMERTA CITY OF GANGSTERS	6	3	3	7	10
8	8	REDNECK RAMPAGE COLLECTION	2	3	4	8	20
9	9	ATLANTIS 2 BEYOND ATLANTIS	2	2	2	9	8
10	10	CEL DAMAGE HD	7	2	1	10	19
11	11	CHRONO CROSS THE RADICAL DREAMERS EDITION	2	2	2	11	9
12	12	RED BOW	1	2	2	12	7
13	13	GLOOM DIGITAL EDITION	8	2	3	13	5
14	14	MORTAL KOMBAT DECEPTION	9	3	5	14	65
15	15	MARY SKELTER NIGHTMARES	10	3	2	15	18
16	16	KATAMARI FOREVER	5	1	6	16	25
17	17	SPACE INVADERS IG	3	1	4	17	8
18	18	DEFENCE TO DEATH	2	1	3	18	6
19	19	DESPOTS GAME	8	2	3	19	17
20	20	APOCALIPSIS THE TREE OF THE KNOWLEDGE OF GOOD AND EVIL	2	2	2	20	9
21	21	DEADLY PREMONITION 2 A BLESSING IN DISGUISE	2	3	1	21	15

Total rows: 1000 of 14707   Query complete 00:00:00.076   Ln 1, Col 24