CS307-DATABASE_Project1

Group Members： 刘达洲12311004，魏宇晴12311043

Submission folder structure：

> 1. |——src_java
> 2. |——src_python
> 3. |——src_sql
> 4. |——resource
> 5. |——files

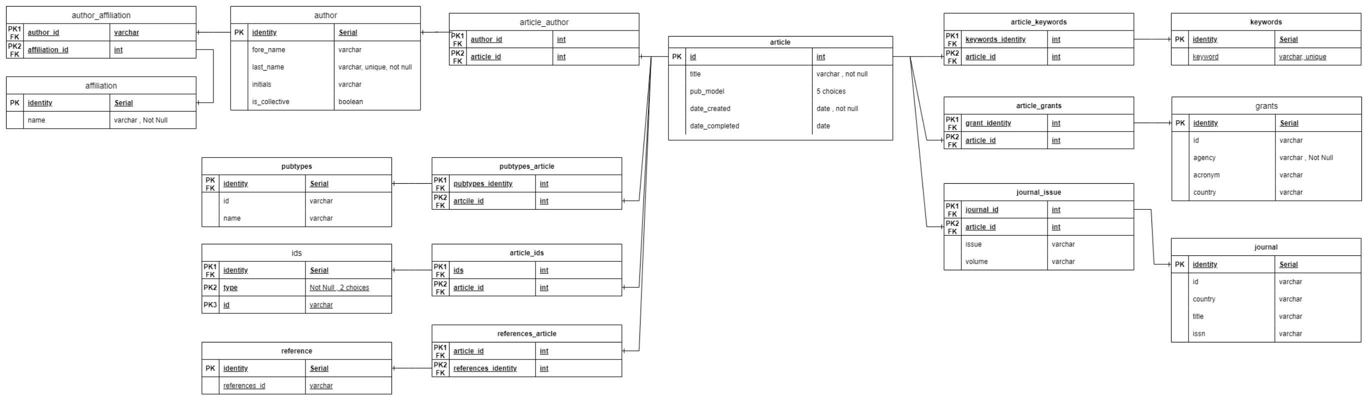
- `src_java` : Java source code folder, contains `DatabaseInserter.java` and `NDJSONImporter.java` and `DatabaseCleaner.java` and `testSQL.java` and `NDJSONImportertest.java` and `fileJournal.java` and `NDJSONreader.java` files.
- `src_python` : Python source code folder, contains `read_large_ndjson.py` .
- `src_sql` : DDL and DML statement files required for project operation.
- `resource` : project original data `.json` file and configuration files.
- `files` : Contains E-R diagram files, `.txt` files, screenshots of the tables and graphs, etc. during the project process.

Project precautions：

- The DBSM used in the project is `PostgreSQL 12`. Please create a username and password before running.
- The Java import script source file for this project is `DatabaseInserter.java`, `NDJSONImporter.java` , `DatabaseCleaner.java` .
- The operating system developed and tested is `Windows 10 Home 22H2`, Java is `JDK 17.0.3`, and Python version is `3.12.7`.

Task 1 : E-R Diagram

Our group uses drawio to draw the E-R diagram, the screenshot is as follow :



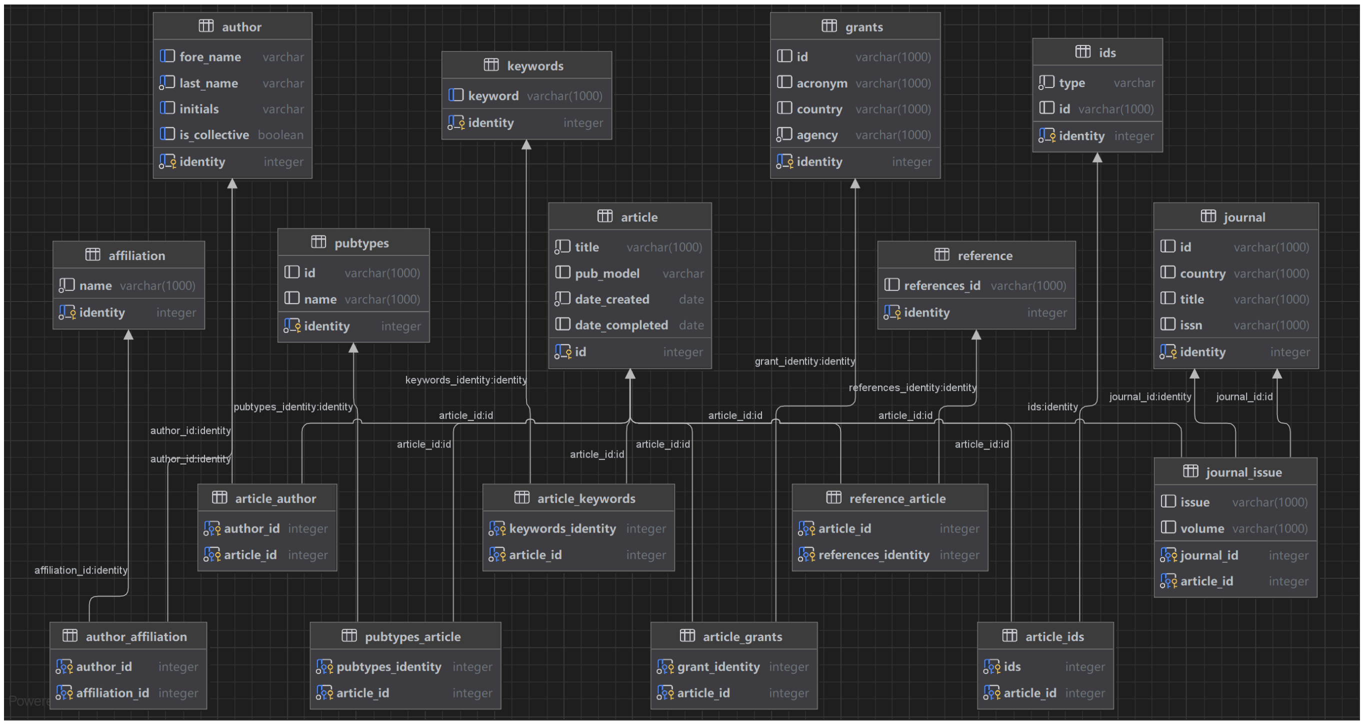
From the above E-R diagram, we obtained the entities and relationships in the context of

1. Entities (9 entities in total) :
    - article entity :
      Primary Key : id
      Attributes : title, pub_model, date_created, date_completed
    - grants entity :
      Primary Key : identity
      Attributes : id, acronym, country, agency
    - ids entity :
      Primary Key : identity
      Attributes : type, id
    - reference entity :
      Primary Key : identity
      Attributes : references_id
    - pubtypes entity :
      Primary Key : identity
      Attributes : id, name
    - author entity :
      Primary Key : identity
      Attributes : fore_name, last_name, initials, is_collective
    - affiliation entity :
      Primary Key : identity
      Attributes : name
    - journal entity :
      Primary Key : identity
      Attributes : id, country, title, issn
    - keywords entity :
      Primary Key : identity
      Attributes : keyword

2. Relationships :
   You may notice that the relationships between every entities are all many-to-many relationship, that is because when facing the problem of connecting huge amounts of data, many-to-many relationship can be faster. For example, in the relationship between publication_types and articles, each publication types contains many ids and names, if we use one-to-many relationship here, then we need to paste the same string type information again and again. However, if we use many-to-many relationship here, then each publication_type can generate an id, and we just need to link this int type id with the int type id of the article in the relationship table, which obviously saves time.`int is faster than String` (One of the optimizations)
   - Articles and grants have a many-to-many relationship.
   - Articles and article_ids have a many-to-many relationship.
   - Articles and references have a many-to-many relationship.
   - Articles and publication types have a many-to-many relationship.
   - Authors and affiliations have a many-to-many relationship.
   - Articles and authors have a many-to-many relationship.
   - journal_issue is linked between journals and articles, contains issue and volume besides the journal_id and article_id of the many-to-many relationship.
   - Articles and keywords have a many-to-many relationship

Task 2: Database Design

This project use `project1.sql` to create data table, written by `PostegreSQL DDL` syntax.
Database Design
Create tables using `DataGrip` and select all options, right click on `Diagrams > Show Diagram` to display the following data table design and relationships. Adjust it by importing `drawio` and then export it as a `.png` file.



List the data tables and their columns

A total of 17 data tables were created throughout the project, and the meanings of the data tables, columns, and foreign keys are as follows:

- `article` Table :
  `id` (Primary Key)
  `title` (Not Null)
  `pub_model`
  `date_created` (Not Null)
  `date_completed`
- `grants` Table :
  `identity` (Serial Primary Key)
  `id`
  `acronym`
  `country`
  `agency` (Not Null)
- `article_grants` Table :
  `grant_identity` (Foreign Key, from identity in grants)
  `article_id` (Foreign Key, from id in article)
  【Primary Key (grant_identity, article_id)】
- `ids` Table :
  `identity` (Serial Primary Key)
  `type`
  `id`

- `article_ids` Table :
  `ids` (Foreign Key, from identity in ids)
  `article_id` (Foreign Key, from id in article)
  【Primary Key (ids, article_id)】
- `reference` Table :
  `identity` (Serial Primary Key)
  `references_id`
- `references_article` Table :
  `article_id` (Foreign Key, from id in article)
  `references_identity` (Foreign Key, from identity in reference)
  【Primary Key (article_id, references_identity)】
- `pubtypes`Table :
  `identity` (Serial Primary Key)
  `id`
  `name`
- `pubtypes_article` Table :
  `pubtypes_identity` (Foreign Key, from identity in pubtypes)
  `article_id` (Foreign Key, from id in article)
  【Primary Key (article_id, pubtypes_identity)】
- `author` Table :
  `identity`(Serial Primary Key)
  `article_id` (Foreign Key, from id in article)
  `fore_name`
  `last_name` (Unique) (Not Null)
  `initials`
  `is_collective·
- `article_author` Table :
  author_id (Foreign Key, from identity in author)
  article_id (Foreign Key, from id in article)
  【Primary Key (author_id, article_id)】
- `affiliation` Table :
  `identity` (Serial Primary Key)
  `name` (Not Null)
- `author_affiliation` Table :
  `author_id` (Foreign Key, from identity in author)
  `affiliation_id` (Foreign Key, from identity in affiliation)
  【Primary Key (author_id, affiliation_id)】
- `journal` Table :
  `identity` (Serial Primary Key)

`id`
`country`
`title`
`issn`

- `jounal_issue` Table :
  `issue`
  `volume`
  `article_id` (Foreign Key, from id in article)
  `journal_id` (Foreign Key, from identity in journal)
  【Primary Key (journal_id, article_id)】
- `keywords` Table :
  `identity` (Serial Primary Key)
  `keyword` (Unique)
- `article_keywords` Table :
  `keywords_identity`(Foreign Key, from identity in keywords)
  `article_id`(Foreign Key, from id in article)
  【Primary Key (article_id, keywords_identity)】

Role of each constraint

1. check_pub_model CHECK
   This constraint ensures that the pub_model data of the aticles must be predifined values ('Print', 'Print-Electronic', 'Electronic', 'Electronic-Print', 'Electronic-eCollection') to prevent invalid pub_model input.

```
   CONSTRAINT check_pub_model CHECK ( pub_model IN ('Print', 'Print-Electronic',
'Electronic', 'Electronic-Print', 'Electronic-eCollection') )
```

2. valid_type CHECK
   This constraint ensures that the type data of the article_ids must be predifined values ('pubmed', 'doi') to prevent invalid type input.

```
   CONSTRAINT valid_type CHECK (type IN ('pubmed', 'doi')),
```

3. unique_name_quadruple UNIQUE
   This constraint ensures that every combination of this (fore_name, last_name, initials, is_collective) is unique.

```
   CONSTRAINT unique_name_quadruple UNIQUE (fore_name, last_name, initials,
is_collective),
```

The rationality of database construction

- Meet the three major paradigms
- Satisfied all the details required by the project requirements document

Task 3: Data Import

Descriptions of how our script imports data

1. Steps
    - `NDJSONImporter.java`
      Used to read data from the NDJSON file, convert them into Java objects through the Jackson library, and then use the DatabaseInserter class to insert the data into a PostgreSQL database.
        - `import com.fasterxml.jackson.databind.ObjectMapper;` Import the ObjectMapper class from Jackson library for JSON parsing and generation.
        - `ObjectMapper objectMapper = new ObjectMapper();` Create an Object Mapper instance to convert JSON strings into Java objects.
        - Use the *try-with-resources* statement to automatically manage resources, ensuring that BufferedReader closes automatically after reading the specified file at the specified path.
        - Use *objectMapper* to parse and convert a JSON formatted string (stored in the line variable) into a *Map<String, Object>object* in Java.
        - Call `insertDataIntoDatabase` method to insert data into database
    - `DatabaseInserter.java`
      By constructing parent and child tables and associated tables, data is inserted into a PostgreSQL database, including Authors, Journals, articleId,etc. During the insertion process, hash tables and deduplication keys are used to handle data deduplication, while transaction management is also performed.
        - Insert *Journal* data (parent table) , *Authors* data (parent table) , *PublicationTypes* data (parent table) , *Grants* data (parent table) , *Article* data (subtable) , then insert associated table data , including *References* , *ArticlePublicationTypes* , *ArticleGrants* , *ArticleIds*.
        - Every method,
            - uses Hashmaps to deal with deduplication
            - handles possible SQLException through try-catch blocks.
            - uses Transaction management . Start the transaction through `conn.setAutoCommit (false)` , and call `stmt.addbatch()` to commit the

    transaction after the operation succeeds, or call `conn. rollback ()` to rollback the transaction after catching exceptions.

- Includes processing of dates, such as the createDate and completedDate methods, which convert years, months, and days to java. sql Date object, and handled the situation of date format errors.
- Especially in the *insertAuthor* function , we
    - Create an empty ArrayList `authorLastNames` to store the last_names of authors inserted into the database.
    - Define three kind of SQLstatement , either used to insert author information and whether there is institutional information, or used to insert institution information and return institution_id, or used to insert the association between author_id and institution_id.
    - Create a PreparedStatement using conn to execute authorSQL. Set RETURN-GENERATED_KEYS to obtain the ID automatically generated after inserting the author record.
    - Extract the author's fore_name, last_name, initials, and identification of whether they are a collective author from the Map. Build a deduplication key based on this information to avoid inserting duplicate author records.
    - Use three HashMaps, namely *collectiveAuthors*, *lastNameOnlyAuthors*, and *fullAuthors*, to store the deduplicationkey of authors that have already been processed, in order to determine whether the current author has been inserted.
    - If the author has not been inserted before (the deduplicationkey is not in the HashMap), insert the author record.
    - Obtain the automatically generated id of the author record that was just inserted using the getGeneratedKeys method.
    - If the author has associated institutional information, insert the institutional information first and obtain the institution_id.
    - Then insert the association record between the author and the institution using the author_id and institution_id.
    - Return the arraylist `authorLastNames` .

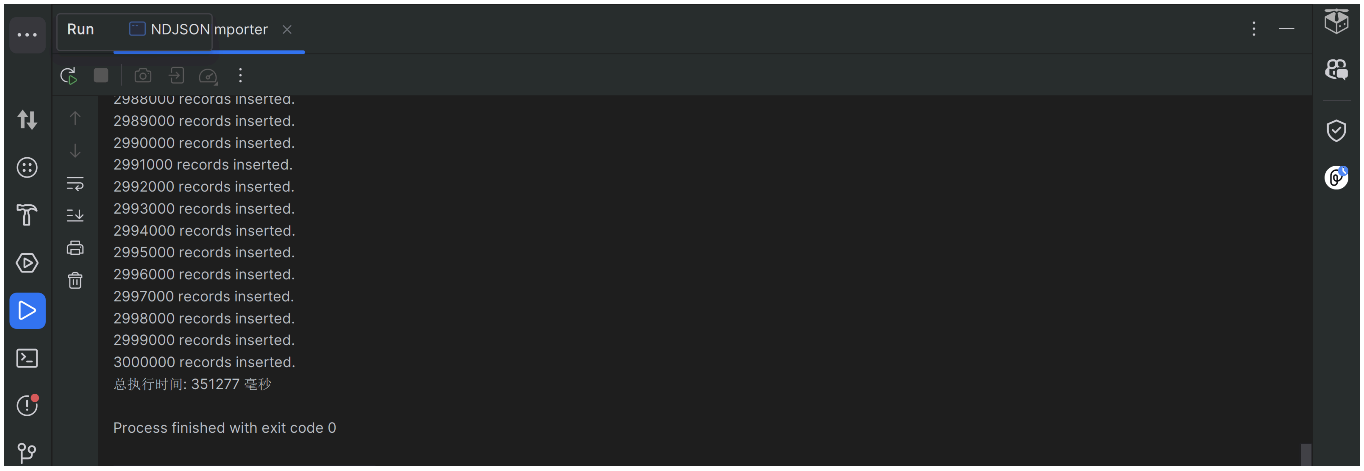
- `DatabaseCleaner.java`
  Clear all specified tables from the database.
  Recreate these tables.
  Equivalent to running database code once in a Java program.
  Clearing the tables can ensure that there is no former data in the database, which can avoid confusion between newly imported data and previous data, and ensure data consistency and accuracy.

    If there were errors during the previous import process, DatabaseCleaner can help restore to a known clean state, facilitating error troubleshooting and data validation.

2. Necessary prerequisites
    - Configure IDEA
        - In the created project directory， choose *Maven* in *Build System*.
            - In `pom.xml`, add dependencies.
            - Click on the search button in the upper right corner to search for *Maven*.
            - Choose *Reload All Maven Projects*
            - If there is still a conflict in the version, follow the prompts to make modifications.
    - Connect To Database
      - Copy the URL on the interface for establishing a connection between the data source and database。
      - Modify the *URL, USER*, and *PASSWORD* in Java code to our own information.
      - Create a table in the database
      `create table users(`
      `name varchar,`
      `age INT`
      `);`
      - Run the Java code
3. Cautions to run the script and import data correctly
    - Environment configuration:
        - Ensure that the Java development environment is properly configured, including JDK and necessary libraries such as Jackson library.
        - Confirm that the PostgreSQL is running and accessible.
    - Confirm that the path to the. ndjson file is correct and that the program has sufficient permissions to read the file.
    - Check if the database URL, username, and password are correct to ensure that the database connection can be successfully established.
    - Our program includes exception handling mechanisms to capture and handle possible IO or SQL exceptions when reading files or performing database operations
    - Confirm whether the transaction management logic is correct, ensure that transactions can be rolled back in case of errors, and avoid data inconsistency.
        - `conn.setAutoCommit(false);`
        - `conn.commit();`
        - `conn.rollback();`
    - The program uses deduplication logic to avoid inserting duplicate data.
        - `boolean isCollective = collectiveName != null;`

        - `String deduplicationKey;`
        - `boolean shouldInsert = false;`
    - Ensure that all relevant foreign keys and references are processed correctly when inserting data to maintain data integrity.
4. The number of records in each entity table
    - article 3000000
    - author 1671875
    - pubtypes 50
    - ids 5242281
    - reference 643080
    - affiliation 1018482
    - keywords 5518
    - grants 141975
    - journal 5639

Below is a screenshot of the time that the whole data import into database, it takes 351277ms.



```
2988000 records inserted.
2989000 records inserted.
2990000 records inserted.
2991000 records inserted.
2992000 records inserted.
2993000 records inserted.
2994000 records inserted.
2995000 records inserted.
2996000 records inserted.
2997000 records inserted.
2998000 records inserted.
2999000 records inserted.
3000000 records inserted.
总执行时间: 351277 毫秒

Process finished with exit code 0
```

Advanced Tasks

Description of how we optimize our script and analysis of how fast it is compared with your original script .

- When removing duplicates, use a *hashmap* instead of *on conflict do nothing*. This changes the complexity of originally $O(n)$ or even $O(n^2)$ to $O(1)$.

【 Just as in the course of data structure and algorithm analysis, a *hashmap* can provide an address for every data that needs to be inserted, which means when facing duplicate data, the time complexity is only to whether there has already developed an address in that specific position. However, if we use *on conflict do nothing* we need to traverse twice to find whether duplicate. 】

- We use batch processing in our code, which significantly improves performance when inserting large amounts of data by reducing the number of querys with the database.

- When selecting primary keys, we find that choosing int type is faster than string type.

【That is why you can find that even though *keywords* can be inserted directly, we still add an identity to it and tranfer the one-to-many relationship to the many-to-many relationship. The same logic applies when inserting *references*.
As a result, we convert all the relationships between entites to many-to-many relationship, which not only have great efficiency, but also reduce duplicate pasting of data and reduce redundancy.】
- We use journal_issue as the relationship table between jounal and article, which simplifies database structure.
- We try to use concurrency hash map to do the multithreading. But due to some process safety problems such as the conflict of multiusers inserting into the same table, we finally give it up.
- Create serial primary key.
【De duplication of journal to reduce storage space and increase efficiency.】
- We use the first one instead of the second one. This is particularly useful for tables that have columns such as' Serial 'or' Identity '(such as the' identity' in the author table). It is a performance optimization that avoids the need for additional SELECT queries to retrieve the ID of the newly inserted record.
`paredStatement authorStmt = conn.prepareStatement(authorSql, Statement.RETURN_GENERATED_KEYS)` and

```
    try(ResultSet rs = authorStmt.getGeneratedKeys()) {
        // ...
        }
```

```
    try(ResultSet rs = insertStmt.executeQuery()){}
            // ...
        }
```

Task 4: Compare DBMS with File I/O

1. Our test environment :
   a. Hardware specification

	LIU	WEI
CPU	AMD Ryzen 7 7840HS w/ Radeon 780M Graphics	Inter i7-8665U
RAM	16.0 GB	16.0 GB
DISK	1TB SSD	1TB PCIE SSD

b. Software specification

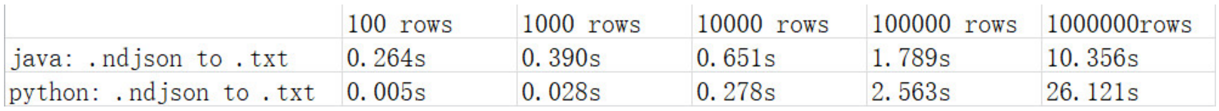
Requirements	Specification
version of DBMS	PostgreSQL_12
version of operating system （LIU）	Windows 11
version of operating system （WEI）	Windows 10 Home 22H2
programming language 1	Java
programming language 2	Python
version of compilers 1	JDK 21.0.1
version of compilers 2	3.12.7
version of libraries1	jackson 2.13.4.2
version of libraries2	org.postgresql 42.7.2

c. If someone else is going to replicate our experiment, they should be provided with some necessary information like:

- The specified version of DBMS, operating system, the type of programming language and the specified version of compilers and libraries.
- Operating environment of the programming language.

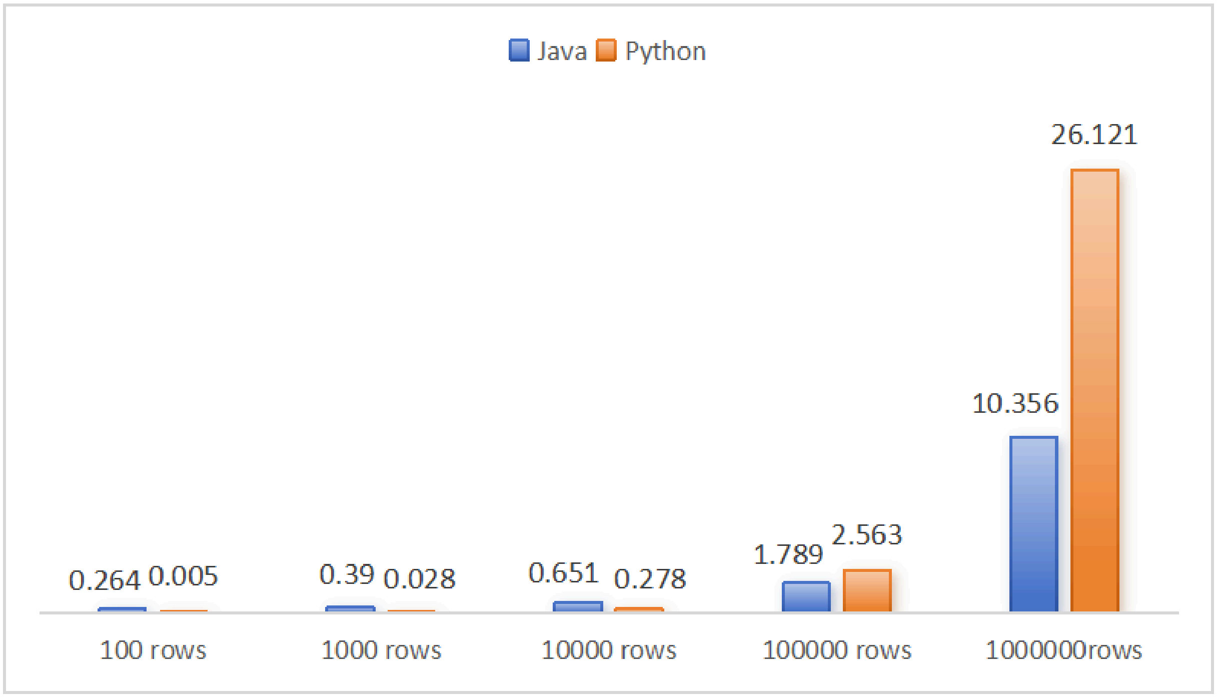
2. The way we organize the test data in the DBMS and the data file
    - the way we generate the test SQL statements:
      the `testSQL.java` and `NDJSONImportertest.java` are just part of the Java import script source file.
      the statements added to measure time is :
      `long startTime = System.currentTimeMillis();`
      `long endTime = System.currentTimeMillis();`
      `long duration = endTime - startTime;`
      `System.out.println("程序运行时长：" + duration + "毫秒");`
    - data form of the files :
      `journal1.txt` to `journal10.txt`: Stores the first 1000000 rows of the journal data of the NDJSON file, outputed by running `fileJournal.java` .
      `WholeData1.txt` to `WholeData10.txt`: Stores the first 1000000 rows of the NDJSON file, outputed by running `NDJSONreader.java`.
      `WholePython1.txt` to `WholePython10.txt` : Stores the first 1000000 rows of the NDJSON file, outputed by running `read_large_ndjson.py`.
3. Description of our test SQL script and the source code of our program
   We have 5 source code files, `testSQL.java`, `NDJSONImportertest.java`, `fileJournal.java`, `NDJSONreader.java`, `read_large_ndjson.py`. And here is the details of

all five source code files.
`testSQL.java`: only insert the journal, journal_issue, article, article_ids, ids data.
`NDJSONImportertest.java` : Used to read journal relevant data from the NDJSON file, convert them into Java objects through the Jackson library, and then use the DatabaseInserter class to insert the data into a PostgreSQL database.
`fileJournal.java` : This java script reads the NDJSON file, write the first one million journal data of the JSON objects to a text file, while calculating and outputting the program's runtime.
- For each line, use the Object Mapper to parse it into a `Map<String, Object>object`.
- Extract the journal field from the Map object, and further extract the id, country, issn, and title fields, writing them to the output file separated by commas.
- Check if the journal_issue field exists, and if so, continue to determine if the volume and issue fields exist. If present, extract the volume and issue fields and write them to the output file as well. If a field does not exist, write 'null'.
- After processing each record, write a line break character.
`NDJSONreader.java` : This Java script reads the NDJSON file, convert the first one million JSON objects into string format and write them to a text file, while calculating and outputting the program's runtime. We can compare the output time of Java, Python, and DataGrip by changing the required number of output lines and recording the time.
`read_large_ndjson.py` : This Python script reads the NDJSON file, parses the first one million JSON objects into a Python dictionary (json obj), converts it into a string, and writes it to another text file. It also measures and outputs the program's execution time, which can be used to compare the efficiency of different tools in handling the same task.

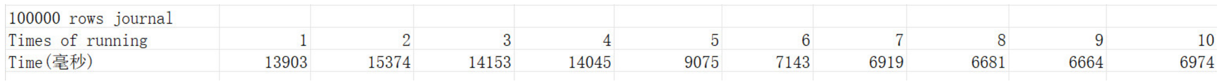
4. Comparative study of the running time.
    - data visualization of the result
      This table compares the time taken to convert ndjson format files with different specified line counts into txt files in Java and Python .
      

	100 rows	1000 rows	10000 rows	100000 rows	1000000rows
java: .ndjson to .txt	0.264s	0.390s	0.651s	1.789s	10.356s
python: .ndjson to .txt	0.005s	0.028s	0.278s	2.563s	26.121s

      We can see it more directly from the following bar chart.

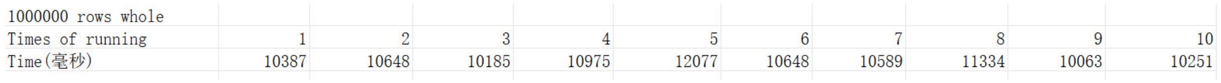


This table is the time of transversing 1000000rows of journal data into `.txt` files. We run this `fileJournal.java` java program ten times, and store the journal data into `journal1.txt` to `journal10.txt` while recording the time. The average time of running this program is 10.0931 s.



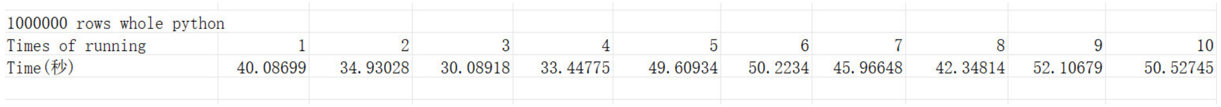
100000 rows journal										
Times of running	1	2	3	4	5	6	7	8	9	10
Time(毫秒)	13903	15374	14153	14045	9075	7143	6919	6681	6664	6974

This table is the time of transversing 1000000rows of the whole data into `.txt` files. We run this `NDJSONreader.java` java program ten times, and store the whole data into `WholeData1.txt` to `WholeData10.txt` while recording the time. The average time of running this program is 10.7157 s.



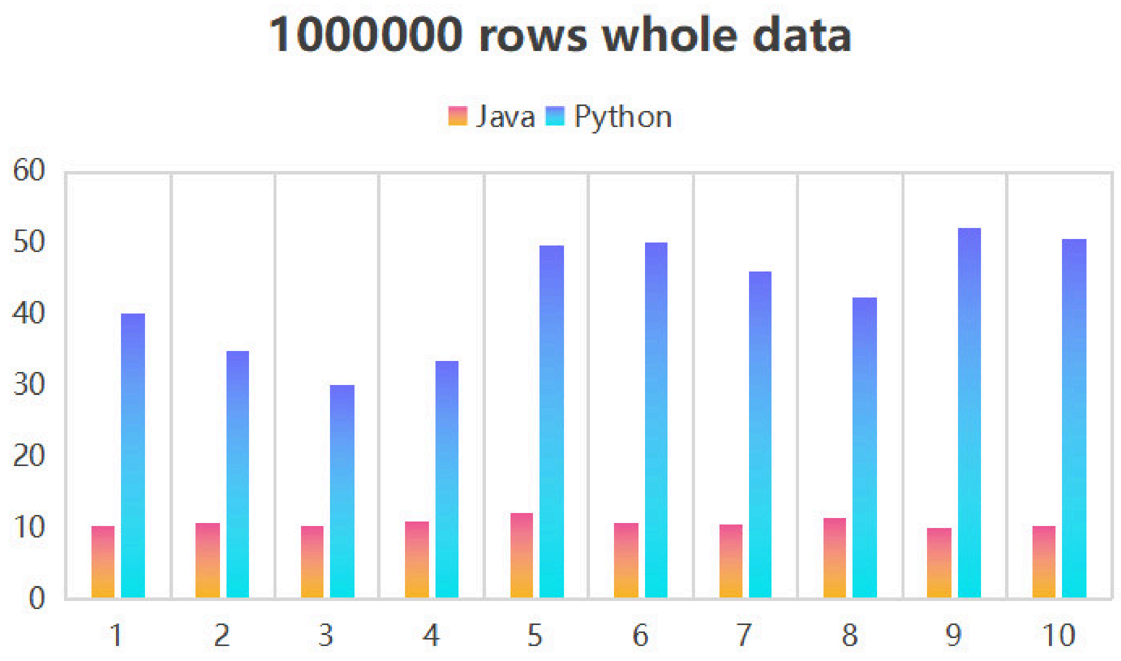
1000000 rows whole										
Times of running	1	2	3	4	5	6	7	8	9	10
Time(毫秒)	10387	10648	10185	10975	12077	10648	10589	11334	10063	10251

This table is the time of transversing 1000000rows of the whole data into `.txt` files. We run this `read_large_ndjson.py` python program ten times, and store the whole data into `WholePython1.txt` to `WholePython10.txt` while recording the time. The average time of running this program is 42.93358 s.

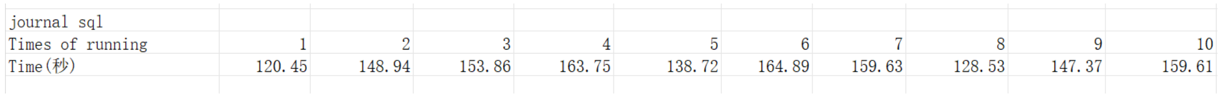


1000000 rows whole python										
Times of running	1	2	3	4	5	6	7	8	9	10
Time(秒)	40.08699	34.93028	30.08918	33.44775	49.60934	50.2234	45.96648	42.34814	52.10679	50.52745

We can observe a direct comparison of the data import time of Java and Python through the following bar chart.

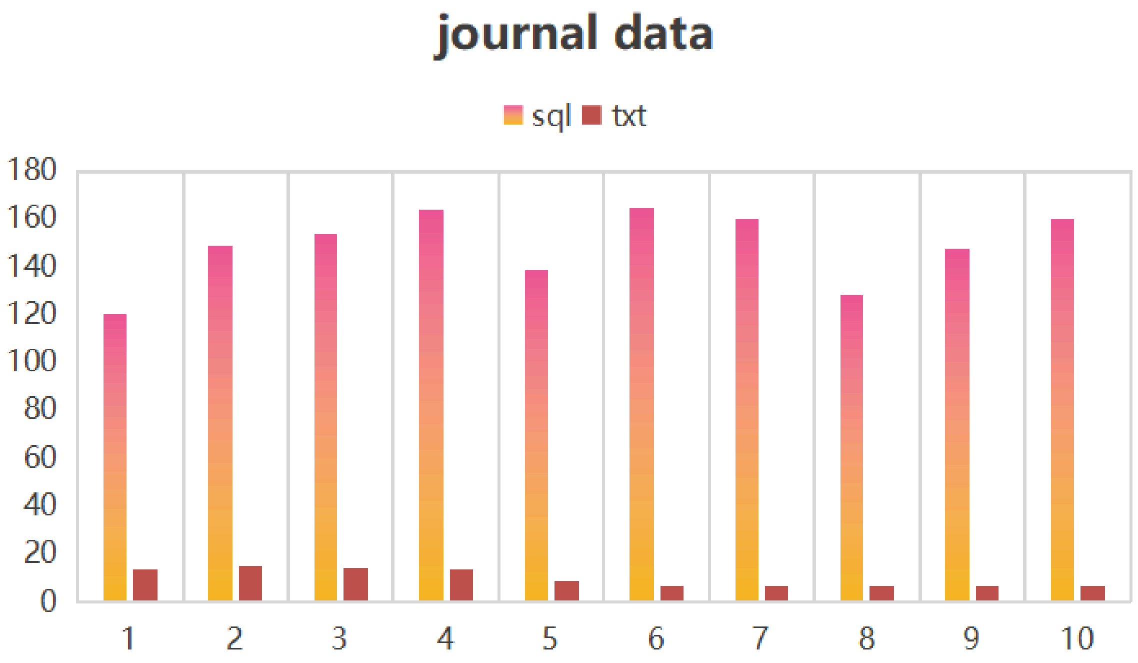


This table is the time of importing the journal data into SQL. We run this `testSQL.java` java program ten times, while recording the time. The average time of running this program is 148.575 s.



journal sql										
Times of running	1	2	3	4	5	6	7	8	9	10
Time(秒)	120.45	148.94	153.86	163.75	138.72	164.89	159.63	128.53	147.37	159.61

We can observe a direct comparison of the data import time to txt and to sql through the following bar chart.



- description of the major differences with respect to running performance and the interesting points within the result
  - From this bar chart, we can obviously notice that when the number of data rows in the database is relatively small, Python ought to be faster than Java.

However, when the number of data rows in the database is relatively huge, we use Monte Carlo method to calculate the average running time of the programme, and notice that Java has significant advantages in terms of data import speed.
By analyzing this result, we find out that in contrast, Java is a statically typed language where type checking and exception handling mechanisms are completed at compile time, which helps improve code reliability and runtime efficiency. And Java has advantages in concurrency and multithreading, which allows it to more effectively utilize system resources when processing large-scale data.

- SQL sometimes is slower than file I/O,
    - the cause may be that although both are implemented using code at the bottom level, file I/O converts eveything inserted into text forms, while SQL needs to transform the data into different types,including data, varchar, int, etc.
    - Another reeason may be that when processing data, SQL not only needs to convert the data into different data types, but also needs to handle database specific functions such as indexes, constraints, foreign keys, and primary keys, which introduce additional computation and storage overhead. And file I/O only reads and writes data in text form, without involving these complex data processing logics.