DOI: 10.15514/ISPRAS-2022-34(1)-12

Решение проблемы хранения траекторий молекулярной динамики в СУБД

И.В. Лихачев, ORCID: 0000-0002-4926-5654 <ilya_lihachev@mail.ru>
Институт математических проблем биологии РАН – филиал ИПМ им. М.В. Келдыша РАН
142290, Московская область, г. Пущино, ул. проф. Виткевича, д.1

Аннотация. В работе решается проблема хранения траекторий молекулярной динамики в реляционных и нереляционных базах данных. Традиционный подход организации структуры реляционных таблиц не подходит для хранения траекторий ввиду появления большого числа записей в одной таблице. Описано, каким образом можно структурировать данные в СУБД класса NoSQL. Затем эти идеи переносятся на реляционную СУБД MySQL.

Ключевые слова: СУБД; SQL; NoSQL; MongoDB; MySQL; моделирование молекулярной динамики; траектории молекулярной динамики; TAMD; анализатор траекторий молекулярной динамики; большие данные

Для цитирования: Лихачев И.В. Решение проблемы хранения траекторий молекулярной динамики в СУБД. Труды ИСП РАН, том 34, вып. 1, 2022 г., стр. 161-172. DOI: 10.15514/ISPRAS–2022–34(1)–12

Благодарности: Автор благодарит Балабаева Н.К. за критические замечания, а также Сычугова А.А. и Сулимову В.В. за то, что доверили вести курс «Технология хранения и обработки больших данных» в Тульском государственном университете.

Solving the problem of storing trajectories of molecular dynamics in a DBMS

I.V. Likhachev, ORCID: 0000-0002-4926-5654 <ilya_lihachev@mail.ru>
The Institute of Mathematical Problems of Biology RAS - the Branch of Keldysh IAM of RAS
1, Professor Vitkevich St., 142290, Pushchino, Moscow Region, Russia

Abstract. The paper solves the problem of storing molecular dynamics trajectories in relational and non-relational databases. The traditional approach to organizing the structure of relational tables is not suitable for storing trajectories due to the appearance of a large number of records in one table. It is described how best to place data in the NoSQL class DBMS. These ideas are then transferred to the MySQL relational DBMS.

Keywords: DBMS; SQL; NoSQL; MongoDB; MySQL; molecular dynamics simulation; molecular dynamics trajectories; TAMD; Trajectory Analyzer of Molecular Dynamics; Big Data

For citation: Likhachev I.V. Solving of the problem of storing trajectories of molecular dynamics in a DBMS. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 1, 2022, pp. 161-172 (in Russian). DOI: 10.15514/ISPRAS–2022–34(1)–12

Acknowledgements. The author thanks Balabaev N.K. for critical remarks, as well as Sychugov A.A. and Sulimova V.V. for being entrusted to conduct the course "Technology of storage and processing of big data" at Tula State University.

1. Введение

Метод моделирования молекулярной динамики (МД) является общепринятым методом исследования биологических структур. Результатами МД расчётов служат траектории молекулярной динамики – записи координат атомов через определённые, обычное одинаковые, промежутки времени. К настоящему моменту времени во многих лабораториях накоплено большое количество файлов траекторий. С ними работают при помощи программ анализа траекторий, таких как VMD [1], [2], TAMD [3-5], PyMol [6], [7]. Программы анализа траекторий, как правило, способны воспроизводить траекторию в виде молекулярного кино, а также рассчитывать некоторые характеристики вдоль траектории. Характеристики могу быть простыми, такие как расстояние между атомами, так и более сложными, например, вычисления матрицы ковариации.

К настоящему моменту времени ощущается нужда в средстве централизованного хранения и анализа траекторий. Это естественный этап развития вида программного обеспечения. Многие в нынешнем известные как гигантские распределенные, сетевые, клиент-серверные (в данном контексте это синонимы) программные комплексы изначально создавались как приложения для одного компьютера. И лишь с развитием локальных вычислительных сетей программные продукты стали поддерживать сетевой режим работы.

Когда речь идёт о большом количестве информации, которую необходимо сделать доступной для большого числа пользователей, разумно задуматься в первую очередь о хранении этой информации в системе управления базами данных (СУБД). Настоящая статья посвящена именно проблеме хранения траекторий молекулярной динамики в различных СУБД.

Предполагается, что программы анализа траекторий МД – не являющиеся предметом данной работы – могут подключаться напрямую к СУБД по сети в случае десктоп-приложений, либо быть разделены на клиент и сервер, который, в свою очередь, будет иметь подключение к серверу баз данных.

2. Представление данных в реляционной СУБД

Реляционные СУБД завоевали свою популярность ещё с конца 1970-х годов. Среди самых популярных стоит отметить Oracle [8], MySQL [9], [10], Microsoft SQL Server [11], PostgreSQL [12] и даже Microsoft Access. Все эти СУБД поддерживают размещение данных в связанных таблицах, доступ к которым может быть получен с применением языка SQL (Structured Query Language, структурированный язык запросов).

Попытка хранить данные в реляционной СУБД изначально выглядит естественной. Многие программисты владеют навыками работы именно с реляционными базами данных. При применении языка SQL доступ к этим базам в первом приближении выглядит практически одинаково. Под доступом понимаются операции CRUD, как их сейчас принято называть (create, read, update, delete – создание, выборка/чтение, обновление, удаление).

Табл. 1. Структура таблицы Names
Table 1. The structure of table Names

Поле	Тип	Описание
Id	Int, Первичный ключ	Идентификатор имени
Name	строка	Название эксперимента
Natoms	Int	Количество атомов
Nframes	Int	Количество кадров
Typename	char	Тип элемента: dir – каталог exp – данные об эксперименте (траектория молекулярной динамики)
Parent	Int	Ссылка на родителя

Мы будем использовать СУБД MySQL [9], [10] для иллюстрации. Выбор обусловлен бесплатной лицензией и широкой распространённостью данной СУБД.

Итак, траектория молекулярной динамики – это прежде всего файл – именованная область данных. Предлагается использовать отдельную таблицу Names (табл. 1) для хранения сведений о самой траектории. Остальные таблицы будут ссылаться на траекторию по её первичному ключу – уникальному идентификатору id.

Здесь следует рассказать о траектории более подробно. Траектория – это информация о координатах каждого атома в конкретный момент времени. Атомов фиксированное количество *natoms*. У каждого атома три координаты *x, y, z*. Моменты времени в траектории фиксируются только определённые, с необходимой частотой, заданной в МД-эксперименте. Зафиксированный в траектории момент времени будем называть кадром (фреймом). Всего кадров *nframes*.

Разрабатываемая система Trajectory Commander поддерживает хранение траекторий в древовидном каталоге, подобном организации файлов в файловой системе. Для отделения траекторий от каталогов служит поле *typename*, которое принимает значение *dir* в случае каталога, либо *exp* в случае хранения траектории. Поле *parent* ссылается на идентификатор (поле *id*) родительского каталога. В случае корневого каталога используется значение 0 в качестве родителя.

Таблица *Trj* (табл. 2) будет содержать данные непосредственно самих траекторий. Для данной траектории *id_name*, в заданном кадре *nframe*, у атома под номером *natom* будут храниться координаты *x, y, z*.

Табл. 2. Структура таблицы Trj
Table 2. The structure of table Trj

Поле	Тип	Описание
id_name	Int, внешний ключ	Идентификатор имени
Nframe	int	Номер кадра
Natom	int	Номер атома
X	float	координаты
Y		
Z		

SQL-запрос получения координат второго кадра определенной траектории может выглядеть как:

```
SELECT x, y, z,
FROM Trj
WHERE id_name=$trj_id AND nframe=2
ORDER BY natom;
```

Для выбора координат только двух атомов *atom1* и *atom2* можно написать запрос:

```
SELECT x, y, z,
FROM Trj
WHERE id_name=$trj_id AND (natom=$atom1 OR natom=$atom2)
ORDER BY nframe, natom;
```

Сортировка обязательна, чтобы кадры и атомы шли строго по порядку.

Как правило, траектории молекулярной динамики не хранят сведения о типах атомов, аминокислот и прочую метаинформацию. Она будет идентична от кадра к кадру. Вместо того в директориях с МД-расчетами всегда присутствуют PDB-файлы – файлы Protein Data Bank в одноимённом формате. Здесь мы не будет рассматривать формат PDB. Хранение его в СУБД не вызывает трудностей. Приведём лишь соответствующую таблицу (табл. 3), каждая строчка которой должна ссылаться своим внешним ключом на траекторию.

Табл. 3. Структура таблицы PDB
Table 3. The structure of table PDB

Поле	Тип	Описание
id_name	Int, внешний ключ	Идентификатор имени
Natom	int	Номер атома
Atomstring	Str	Строчка из файла без координат

Хранение информации о PDB-файлах необходимо для программ анализа траекторий. Будем считать, что такой файл удобно подавать этим программам в неизменном виде.

2.1. Проблема выбора реальных данных

Объемы траекторий весьма разные. Различно как количество атомов, так и количество кадров. Так при количестве атомов 1000 и 1000 кадров (1 нс при частоте записи 1 кадр на 1 пс, либо 10 нс при частоте записи каждые 10 пс) мы получаем 1 миллион строк. Стоит загрузить 10 таких траекторий и размер таблицы Trj возрастает до 10 миллионов строк. А если хранить 100 траекторий по 1000 кадров, по 10 000 атомов в кадре, то получаем 1 миллиард строк.

Заметим, что приведённые числа служат скорее прикидкой сверху, чем снизу. Многогигабайтные файлы траекторий – это норма, равно как и системы из десятков тысяч атомов.

В тестовой базе порядка 24 млн строк таблицы *Trj*. Запрос выборки одного кадра без использования индексов занимает порядка 2 минут. При включении индексации по всем полям, что учувствуют в запросе это время сокращается до 5 секунд, что говорит о грамотном применении индексов. Однако для воспроизведения молекулярного кино это недостаточно. 1 кадр в секунду неприемлем. Особенно учитывая тот факт, то 24 млн строк – это совсем небольшая база.

Возможно, проблему можно решить, прибегнув к NoSQL базам данных?

3. СУБД класса NoSQL

Сам термин NoSQL можно воспринимать по-разному. Это и базы данных, не поддерживающие реляционную модель, и не поддерживающие язык запросов SQL. Нам больше нравится определение Not only SQL, подчёркивающее, что этот класс СУБД предлагают нечто иное, чем реляционные СУБД [13].

Стоит также отметить, что СУБД начала 1990-ых годов чем-то напоминали этот новый класс СУБД. Язык SQL не был в то время распространён. СУБД того времени поддерживали многие операции, доступ к которым программисты получали при помощи оригинального API, своего собственного языка, применяемого именно с конкретной СУБД. Яркий тому пример – FoxPro для DOS, поддерживающий и мульти индексные файлы, показывающий прекрасное быстродействие на вычислительных машинах класса Pentium. Эта СУБД (или язык, если угодно) перешел в версию Visual FoxPro for Windows. Стал поддерживать SQL, но потерял свою популярность, не выдержал конкуренции. Факт наличия статей в 2020-2021 годах [14], [15] вызывает удивление. Данный абзац лишь отражает одно из мнений авторов, а не четкие определения: к СУБД можно напрямую обращаться при помощи API, без использования SQL-запросов. Несомненно, это быстрее, т.к. не нужно тратить время на синтаксический разбор операторов.

Согласно рейтингу Top-5 СУБД [16], самая распространённая СУБД класса NoSQL – это MongoDB [17], [18]. Позиционируется как документно-ориентированная СУБД. Под документом понимается нечто, не поддающееся строгой классификации. Точнее, имеется в виду документ в формате JSON (JavaScript Object Notation), хранящийся в формате BSON (Binary JavaScript Object Notation), т.е. в бинарном виде для экономии дискового

пространства. Несмотря на своё название, данный формат может использоваться практически в любом языке программирования как ассоциативный массив.

БД для хранения траекторий может состоять из двух таблиц (двух коллекций, переходя на терминологию MongoDB):

Коллекция Names

```
/* Элемент директория */
{
    "_id" : ObjectId("610129f9ba76f8e36dc5da7d"),
    "name" : "dir",
    "Parent" : "0",
    "Type" : "dir",
    "PDB" : "",
    "aim" : 0,
    "NFrames" : 0
}

/* Элемент МД-эксперимент */
{
    "_id" : ObjectId("61012a04ba76f8e36dc5da7e"),
    "name" : "123",
    "Parent" : "610129f9ba76f8e36dc5da7d",
    "Type" : "exp",
    "PDB" : "ATOM      1   CT  Ac     1     -13.432  -5.484   ",
// содержимое поля целиком не приводится
    "aim" : 44689,
    "NFrames" : 145
}
```

Здесь и далее по тексту в каждой коллекции будут использоваться те же поля, как в реляционных СУБД. Это не требуется для работы с MongoDB, но необходимо для индексации.

Табл. 4. Описание полей коллекции Names
Table 4. The description of Names collection

Поле	Тип	Описание
_id_	ObjectId	Идентификатор имени
Name	String	Название эксперимента
PDB	String	Содержимое PDB-файла
Aim	Int	Количество атомов
NFrames	Int	Количество кадров
Parent	String	Ссылка на родителя

Коллекция *Names* аналогично таблице *Names* реляционной БД с тем различием, что поле *id* в MySQL числовое, а в MongoDB мы использовали внутренний идентификатор, который есть у каждого документа. Также таблица *Names* содержит строковое поле PDB, в котором целиком помещается содержимое PDB-файла. Как говорилось ранее, программы анализа траекторий ориентированы на чтение одного PDB файла, относящегося к траектории для распознавания типов атомов и первичной структуры белка. Координаты при этом берутся исключительно из траекторных файлов.

Коллекция Trj:

```
{
    "_id" : ObjectId("5eb2dd86895408ca51dd9b4e"),
// собственный идентификатор
    "trj_id" : ObjectId("5eb2dd84895408ca51dd9b4d"),
// ссылка на имя траектории
    "frame" : 0,
    "data" : [
        25.80421,
...
}
```

Табл. 5. Описание полей коллекции Trj
Table 5. The description of Trj collection

Поле	Тип	Описание
_id_	ObjectId	Собственный уникальный идентификатор кадра
trj_id	ObjectId	Идентификатор имени, ссылка на имя траектории
Frame	int	Номер кадра
Data	Array	Массив координат в формате (*x1, y1, z1, x2, y2, z2, …, xn, yn, zn*)

Как видно из описания полей, одна таблица коллекции (таблицы) *Trj* хранит кадр целиком, а не координату каждого атома в каждой строке. Существование Поля *_id_* является требованием СУБД, а не продиктовано архитектурой конкретной базы.

В итоге количество строк в таблице (коллекции) уменьшается на 3-4 порядка. А запрос на выборку одного кадра занимает доли секунды (0,15-0,2 с).

Мы достигли своей цели – получили кадр траектории за приемлемое время. Надо заметить, что при этом мы возможности выбора координат конкретных атомов. Большая ли это плата за быстродействие? С одной стороны, программы анализа траекторий молекулярной динамики читают кадр целиком, даже когда нужно построить кривую расстояния между двумя атомами. С другой стороны, многие СУБД, и MongoDB в частности, поддерживают хранимые процедуры. Это небольшие программы, исполняемы на серверной стороне. Именно от такой программы можно потребовать выборку конкретных координат из массива координат текущего кадра (поля *data* таблицы *Trj*). Либо же можно написать функцию, которая вернёт сразу расстояние между заданными номерами атомов.

3.1 Подсказка от NoSQL: реализация хранения кадра в одном поле реляционной СУБД

Выдвинем гипотезу о том, что успех СУБД класса NoSQL связан не с самим классом баз данных, а со способом хранения кадра целиком в одном поле.

Начиная с версии 5.7.8, СУБД, MySQL поддерживает тип данных JSON (JavaScript Object Notation). В таком поле удобно будет хранить линеаризованный массив координат одного кадра траектории. Структура таблицы хранения траектории примет следующий вид:

Табл. 6. Новая структура таблицы Trj
Table 6. The new structure of Trj table

Поле	Тип	Описание
id_name	Int, внешний ключ	Идентификатор имени
Nframe	int	Номер кадра
Data	JSON	Координаты кадра в формате [*x1, y1, z1, x2, y2, z2, …]*

Индексируемыми полями будут *id_name* как ссылка на название траектории и *nframe* для быстрой выборки номера кадра. При этом экономится место на поле, отвечающего за номер атома в кадре. Причем как в таблице, так и в индексе.

В новой структуре теряется возможность обращения к конкретному номеру атому в кадре. Возможно только получение кадра целиком. Однако получать доступ к конкретному кадру и

не требуется, если мы используем Буфер случайного доступа к траектории Анализатора траекторий молекулярной динамики. Анализатор всегда считывает кадр целиком. Программы, такие как VMD и PyMol считывают траекторию целиком, что не экономично использует оперативную память.

4. Программная реализация – приложение Trajectory Commander как часть Анализатора траекторий молекулярной динамики

В рамках Анализатора траекторий молекулярной динамики (TAMD) было создано приложение в стиле двухпанельного менеджера, позволяющего оперировать траекториями на диске и сразу в двух базах данных – MySQL и MongoDB.

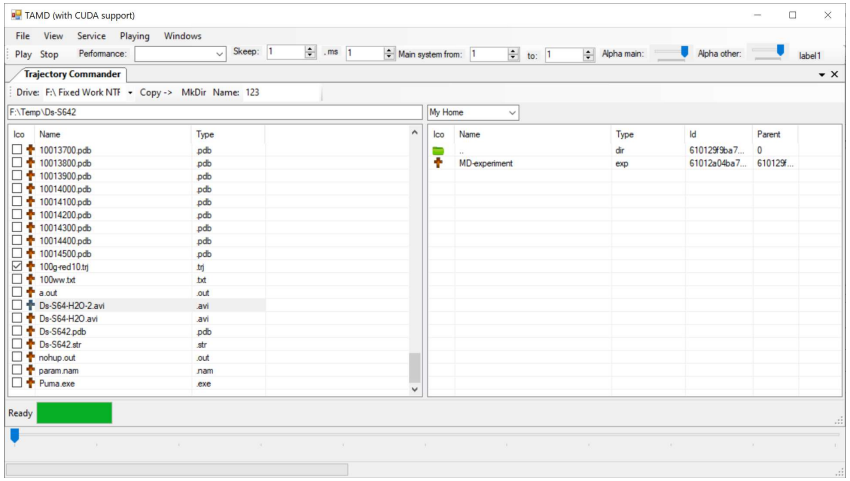

Рис. 1. Trajectory Commander в Анализаторе траекторий молекулярной динамики (TAMD)
Fig. 2. Trajectory Commander as part of Trajectory analyzer of molecular dynamics (TAMD)

На данном этапе работы основной целью было сравнение NoSQL и SQL подходов к хранению траекторий. Поэтому был реализован следующий список возможностей:

- отображение и позиционирование по локальной файловой системы на левой панели;
- хранение учетных записей подключений к различным СУБД;
- отображение и позиционирование по древовидной структуре траекторий, хранящихся в СУБД;
- создание каталогов в СУБД;
- копирование PDB и траекторных файлов из локальной файловой системы в СУБД;
- открытие и работа с траекториями молекулярной динамики напрямую из СУБД средствами Анализатора траекторий.

Анализатор траекторий молекулярной динамики изначально разрабатывался для работы с текстовыми траекториями формата PUMA, близкого к формату XYZ. Для чтения траекторий был разработан буфер случайного доступа к траектории, который хранит последние прочитанные файлы.

Для чтения траекторий напрямую из СУБД были добавлены функции чтения кадров не из текстового файла, а из базы данных. Таким образом сохранена возможность работы с траекторией через буфер случайного доступа к траекториям. Все возможности TAMD

сохранили работоспособность благодаря тому, что они обрабатывают координаты исключительно из модуля чтения траекторий.

4.1 Сравнение выбранных решений

Для сравнения выбранных решений была использована траектория, которая в текстовом формате PUMA занимает 535 МБ дискового пространства.

Табл. 7. Объемы информации
Table 7. Volumes of information

СУБД	Объем данных	Объем индексов	Общий объем данных
Текстовый файл	535 МБ		535 МБ
MySQL	324 МБ (61%)	410 МБ (+127%)	734 МБ (137%)
MySQL (JSON)	246 МБ (46%)	32 КБ (+0,013%)	246 МБ (46%)
MongoDB	281 МБ (53%)	45 КБ (+0,016%)	281 МБ (53%)

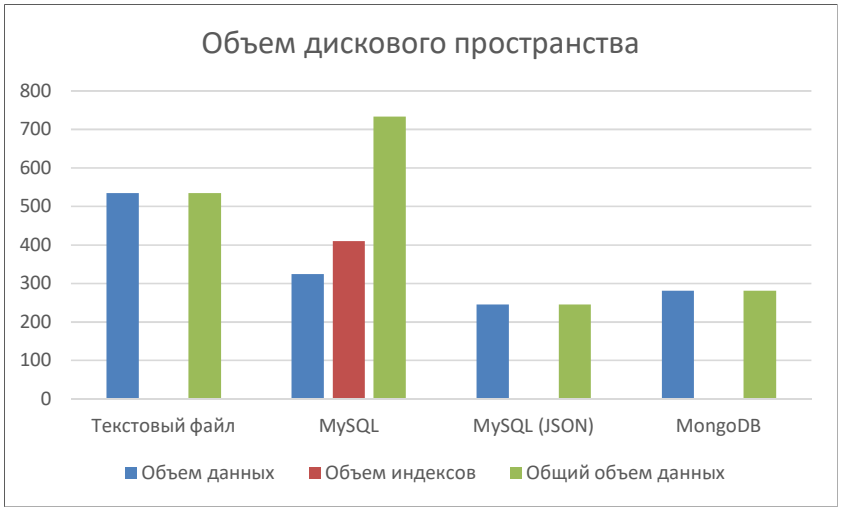

Рис. 3. Объем дискового пространства (меньше – лучше)
Fig. 4. Volume of disk space (less is better)

В столбце «объём данных» и в столбце «общий объем данных» процент приведен от размера исходного текстового траекторного файла, в столбце «объем индексов» от размера данных в соответствующей СУБД.

Из табл. 7 видно, что хранение траекторий в СУБД практически в 2 раза выгоднее по сравнению с текстовым форматом. Однако для эффективного доступа к траектории необходима индексация полей. В MongoDB индексация идёт по идентификатору траектории и кадру. В MySQL к индексам ещё добавляется номер атома в кадре. Размер записей в реляционной базе данных увеличивается на количества атомов в системе (на три порядка при размере системы 1000 атомов). Именно этим объясняется столь существенный объем индексов.

Согласно табл. 8, по скорости считывания кадра лидирует MongoDB. Хранение одного кадра целиком в одной строке реляционной СУБД даёт десятикратный прирост в скорости выполнения запроса.

Табл. 8. Скорость чтения кадров траектории
Table 8. Trajectory frame reading speed

СУБД	Время чтения одного кадра
Текстовый файл	55 мс
MySQL	300 мс
MySQL (JSON)	30 мс
MongoDB	4 мс

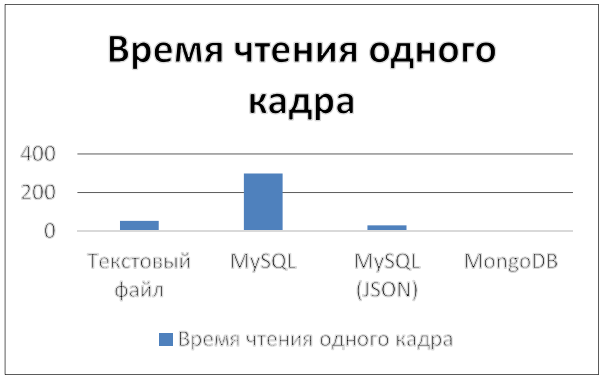

Рис. 5. Сравнение быстродействия при чтении траектории (меньше – лучше)
Fig. 6. Comparison of performance of reading the trajectory (less is better)

5. Обсуждение

В рамках рабочего программного продукта, активно используемого в НИИ для работы с траектория молекулярной динамики (Анализатора траекторий молекулярной динамики TAMD) были испытаны способы хранения и доступа к траекториям в СУБД различных видов.

Ввиду массового распространения нереляционных СУБД в последние годы важным и разумным казалось сравнивать двух представителей СУБД: реляционную СУБД MySQL и нереляционную, так называемого класса NoSQL, документо-ориентированную СУБД MongoDB. Обе СУБД полностью бесплатны, с открытым исходным кодом (open source), являются широко известными представителями СУБД в своём классе. Вместо конкретных СУБД можно было бы выбрать их конкурентов. Но задачей статьи является сравнение подходов, а не конкретных программных продуктов.

При построении **структуры данных** существенная разница проявилась в способе хранения одного кадра траектории. В реляционной СУБД естественным кажется хранение трёхмерных координат каждого атома в отдельной строчке. Количество строк соответствует количеству атомов в системе. В NoSQL подходе все координаты кадра хранятся в линеаризованном массиве, в одной ячейке.

В реляционной СУБД **дисковое пространство** уходит как на хранение самих данных, так и на хранение индексов. При хранении большого количества траекторий быстродействие реляционной СУБД ещё больше падает ввиду большого количества строк в одной таблице.

В работе с траекториями предполагается, что **кадр необходимо читать полностью**. Разумеется, есть множество характеристик, таких как расстояние между двумя отдельными атомами или валентные и торсионные углы, которые требуют выборки только конкретных номеров атомов. Но архитектура Анализатора траекторий такова, что кадр читается полностью. Так удобнее было реализовать Буфер случайного доступа к траектории, который

хранит в оперативной памяти последние кадры. Они могут понадобится как для нахождения характеристик, так и для визуализации кадра. В последнем случае необходимо иметь полный набор кадра.

Сравнивая характеристики MySQL- и MongoDB-подходов, выбор становится очевидным. MongoDB существенно выигрывает и в занимаемом дисковом пространстве, и в скорости доступа к траектории.

Встаёт вопрос, а можно ли перенести нереляционный подход на реляционную СУБД? Подход заключается лишь в способе хранения одного кадра траектории. Все координаты одного кадра разумно хранить в виде линейного массива.

Необходимо выбрать тип данных, поддерживаемый MySQL для хранения массива чисел. Можно было бы использовать бинарный формат, но MySQL начиная с версии 5.7, поддерживает тип JSON, который выглядит предпочтительнее ввиду простоты интерпретации данных человеком при поиске ошибок.

Выбранный подход в таблицах и рисунках изображен как MySQL (JSON). При хранении дискового пространства требуется даже меньше, чем для MongoDB. Выборка данных всё же занимает большего количества времени, чем в MongoDB. По всей вероятности, сказывается устройство реляционной СУБД в целом, начиная с анализа запросов SQL, нежели структура. Однако выбранный подход выглядит гораздо более привлекательным, нежели классическое реляционное представление с некоторой избыточностью в виде доступа к конкретному атому кадра. При этом не тратится существенных дисковых ресурсов на хранение индексов. 30 мс на чтение кадра кажется вполне приемлемыми. И это время не будет стремительно расти, как в традиционном подходе с большим количеством строк в таблице.

В итоге делается вывод, что компактное хранение MySQL (JSON) может конкурировать с СУБД MongoDB. MongoDB продолжает выигрывать в скорости, но скорость MySQL уже приемлема. При этом MySQL выигрывает в расходе дискового пространства, не смотря даже на применение текстового формата JSON против бинарного BSON у MongoDB.

При разработке решения на языке C# MySQL выглядит более документированным, а сам API более привычной. При работе с MongoDB создатели отмечают в плюсах отсутствие схемы, что должно ускорить разработку. Автору работы эта возможность показалась и положительной, когда не надо создавать схему под новую таблицу, и отрицательной, т.к. нет под глазами четкой схемы данных. При использовании сторонних продуктов-утилит для работы с СУБД (таких как текстовая консоль, phpmyadmin, HeidiSQL, Robo 3T, MogoDBCompass) отмечается больших функционал у MySQL, отсутствие явных ошибок. При работе с MongoDB MogoDBCompass не смогла отображать данные коллекции с координатами кадров за приемлемое время (требовались десятки секунд).

6. Выводы

Построение классической структуры реляционной СУБД для хранения траекторий молекулярной динамики не подходит для хранения существенных объемов информации. Основная причина – возникновение большого количества строк в одной таблице. Следствие – низкая скорость выполнения запросов, существенный объем индексных файлов.

NoSQL-подход на базе MongoDB даёт лучше результаты по быстродействию. Кадр траектории хранится в одном поле. Подход пригоден для построения больших хранилищ траекторий молекулярной динамики.

Для MySQL возможно компромиссная ситуация, когда один кадр также хранится в одном поле. В работе для этого использован тип данных JSON, но для этой цели может использоваться и другой тип, пригодный для хранения массивов. Подход MySQL+JSON показывает приемлемые результаты по скорости доступа, нет быстрого падения производительности при увеличении объемов информации. Среди преимуществ отмечается наименьший объем дискового пространства.

Несмотря на очевидное преимущество MongoDB-подхода по скорости, при построении хранилища выбор может быть сделан в пользу MySQL+JSON подхода ввиду лучшего знакомства с этой СУБД, лучшей документированности и большего количество API и полезных утилит.

Табл. 9. Преимущества и недостатки
Table 9. Advantages and disadvantages

	MySQL	**MySQL (JSON)**	**MongoDB**
Использование дискового пространства	137% от исходного файла	46% от исходного файла	53% от исходного файла
Скорость чтения одного кадра	Не справляется с большим объёмом данных	Приемлемое при работе больших объемов данных	Быстрое при работе больших объемов данных
Рекомендуется использовать для организации хранилища траекторий молекулярной динамики	Нет. Ввиду большого количества записей в одной таблице	Да	Да
Легкость разработки	Одинаково легко		Изучение новых API
Выбор инструментальных средств	Широкий		Ограниченный

Список литературы / References

[1] W. Humphrey, A. Dalke, and K. Schulten. VMD: Visual molecular dynamics. Journal of Molecular Graphics, vol. 14, issue 1, 1996, pp. 33-38.

[2] J. Hsin, A. Arkhipov et al. Using VMD: an introductory tutorial. Current Protocols in Bioinformatics, issue Suppl. 24, 2008, pp. 5.7.1-5.7.48.

[3] I.V. Likhachev, N.K. Balabaev, and O.V. Galzitskaya. Available Instruments for Analyzing Molecular Dynamics Trajectories. The Open Biochemistry Journal, vol. 10, 2016, pp.1-11.

[4] I.V. Likhachev, N.K. Balabaev. Trajectory analyzer of molecular dynamics. Mathematical Biology and Bioinformatics, vol. 2, issue 1, 2007, pp. 120-129.

[5] I.V. Likhachev, N.K. Balabaev. Construction of Extended Dynamical Contact Maps by Molecular-Dynamics Simulation Data. Mathematical Biology and Bioinformatics, vol. 4, issue 1, 2009, pp. 36-45.

[6] The PyMol Molecular Graphics System, Version 2.0. Schrödinger, 2015.

[7] R.E. Rigsby, A.B. Parker. Using the PyMOL application to reinforce visual understanding of protein structure. Biochemistry and Molecular Biology Education, vol. 44, issue 5, 2016, pp. 433-437.

[8] [8]Т. Кайт, Д. Кун. Oracle для профессионалов. Архитектура, методики программирования и основные особенности. Вильямс, 2020 г., 960 стр. / T. Kyte, D. Kuhn. Expert Oracle Database Architecture 3rd ed. Edition. Apress, 2014, 857 p.

[9] Р. Шелдон, Д. Мойе, MySQL. Базовый курс. Диалектика, 2007 г., 880 стр. / R. Sheldon, G. Moes. Beginning MySQL. Wrox, 2005, 864 p.

[10] Ш. Чаллавала, Дж. Лакхатария и др. MySQL 8 для больших данных. ДМК-Пресс, 2019 г., 226 стр. / Sh. Challawala, J. Lakhatariya et al. MySQL 8 for Big Data. Packt, 2017, 266 p.

[11] P.M. Михеев. MS SQL Server 2005 для администраторов. BHV, 2007 г., 534 стр. / R.M. Mikheev. MS SQL Server 2005 for administrators. BHV, 2007, 534 p. (in Russian).

[12] Г.-Ю. Шениг. PostgreSQL 11. Мастерство разработки. ДМК-Пресс, 2019 г., 352 стр. / H.-J. Schönig. Mastering PostgreSQL 11. Packt Publishing, 2018, 446 p.

[13] А.А. Зоткина, И.А. Подопригора, Е.И. Маркин. Сравнительный анализ NoSQL баз данных. Вестник современных исследований, вып. 9.1 (24), 2018 г., стр. 146-148 / A.A. Zotkina, I.A. Podoprigora, and E.I. Markin. Comparative analysis of NoSQL databases. Bulletin of Modern Research, issue. 9.1 (24), 2018, pp. 146-148 (in Russian).

[14] О.Н. Чопоров, О.Е. Работкина, А.Е. Капишников, Разработка баз данных в среде Субд Ms Access и Ms Visual FoxPro. Учебное пособие. Воронежский государственный технический университет, 2004 г., 171 стр. / O.N. Choporov, O.E. Rabotkina, A.E. Kapishnikov, Development of databases in the environment of DBMS Ms Access and Ms Visual FoxPro. Tutorial. Voronezh State Technical University, 2004, 171 p. (in Russian).

[15] Д.А. Шорохов. Создание меню и отчётности для информационной системы учёта игроков и тренеров футбольного клуба „Калининград“ в среде разработки Microsoft Visual Foxpro 9.0. Актуальные научные исследования в современном мире, вып. 7–1 (63), 2020 г., стр. 227-232 / D.A. Shorokhov. Creating a menu and reporting for the information system of accounting of players and trainers of the football club “Kaliningrad” in the environment of Microsoft Visual Foxpro 9.0 development. Actual scientific research in the modern world, vol. 7–1 (63), 2020, pp. 227-232 (in Russian).

[16] Top 5 NoSQL databases for Data Scientists in 2020. URL: https://content.techgig.com/top-5-nosql-databases-for-data-scientists-in-2020/articleshow/78330888.cms, accessed 29.09.2021.

[17] Ю.С. Порохненко, П.Н. Полежаев. Сравнительный нализ NoSQL баз данных. Актуальные направления научных исследований XXI века: теория и практика, т. 5, вып. 9 (35), 2017 г., стр. 25-30 / Yu.S. Porokhnenko, P.N. Polezhaev. Comparative analysis of NoSQL database. Actual Directions of Scientific Research of the 21st Century: Theory and Practice, vol. 5, no. 9 (35), 2017, pp. 25-30 (in Russian).

[18] К. Бэнкер. MongoDB в действии. ДМК Пресс, 2017 г., 394 стр. / K. Banker. MongoDB in Action. Manning Publications, 2011, 312 p.

Информация об авторе / Information about the author

Илья Вячеславович ЛИХАЧЕВ – кандидат физико-математических наук, старший научный сотрудник. Сфера научных интересов: моделирование молекулярной динамики, параллельное программирование, в т.ч. графических процессоров по технологии CUDA, большие данные, СУБД, NoSQL.

Ilya Vyacheslavovich LIKHACHEV – PhD in Physics and Mathematics, Senior Researcher Research interests: modeling of molecular dynamics, parallel programming, incl. GPUs using CUDA technology, Big Data, DBMS, NoSQL.