TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

Khoa Công nghệ thông tin

Báo cáo Hệ Quản Trị Cơ Sở Dữ Liệu

OrientDB

Nhóm 14

Nguyễn Đăng Nam	18020931
Nguyễn Thành Đạt	18020291
Tạ Quang Ngọc	18020952

1. Tổng quan về NoSQL

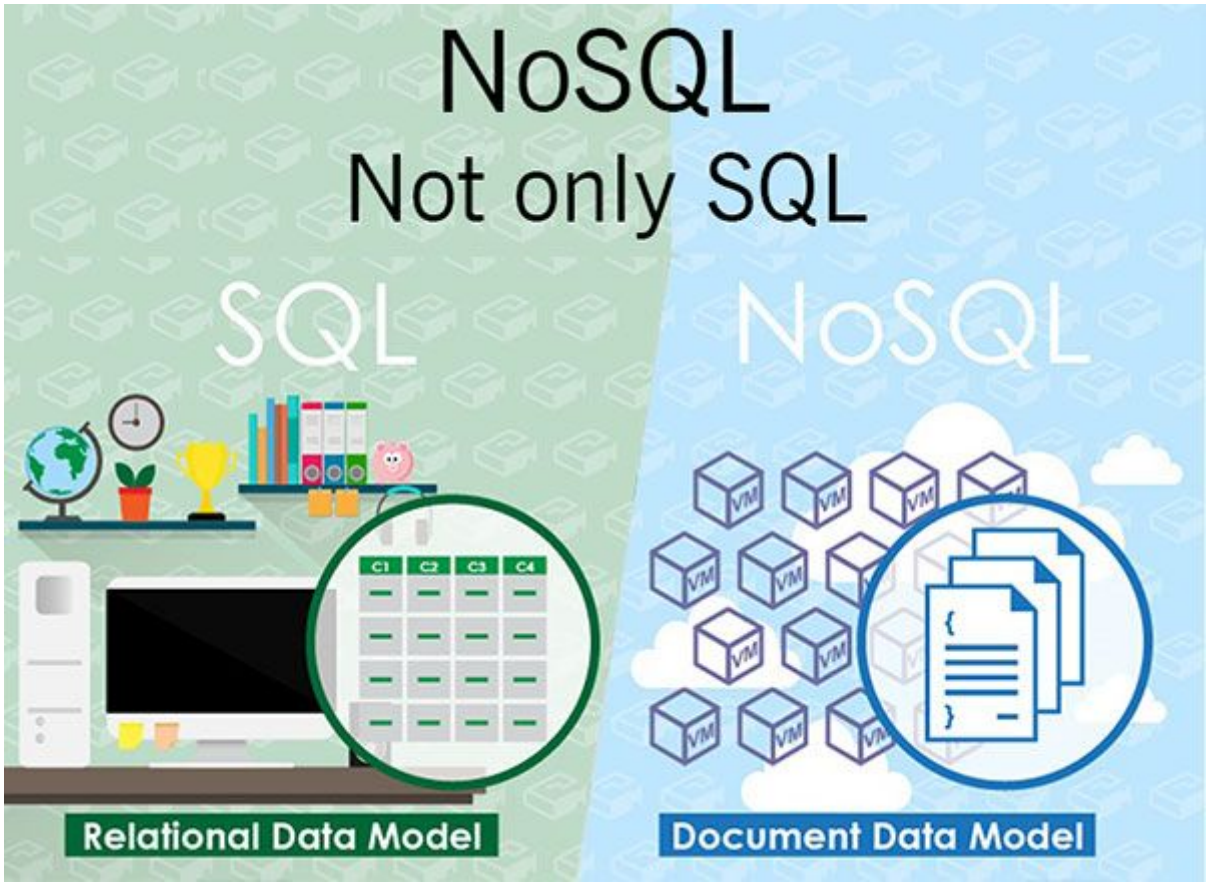
a. Giới thiệu về NoSQL

Relational database - RDBMS ra đời vào những năm 70 của thế kỷ trước, cho phép các ứng dụng lưu trữ dữ liệu thông qua ngôn ngữ truy vấn và mô hình hóa dữ liệu tiêu chuẩn (Structured Query Language - SQL). SQL nói chung hay cụ thể như RDBMS là một sản phẩm đã có bề dày sử dụng từ nhiều thập kỷ phát triển công nghệ, cho thấy khả năng ứng dụng và đáp ứng tốt trong thử nghiệm ứng suất thực tế. Vào thời điểm đó, việc lưu trữ dữ liệu khá tốn kém, tuy nhiên các lược đồ dữ liệu cũng tương đối đơn giản, dễ hiểu nên nhu cầu về một công cụ mới là chưa cần thiết.

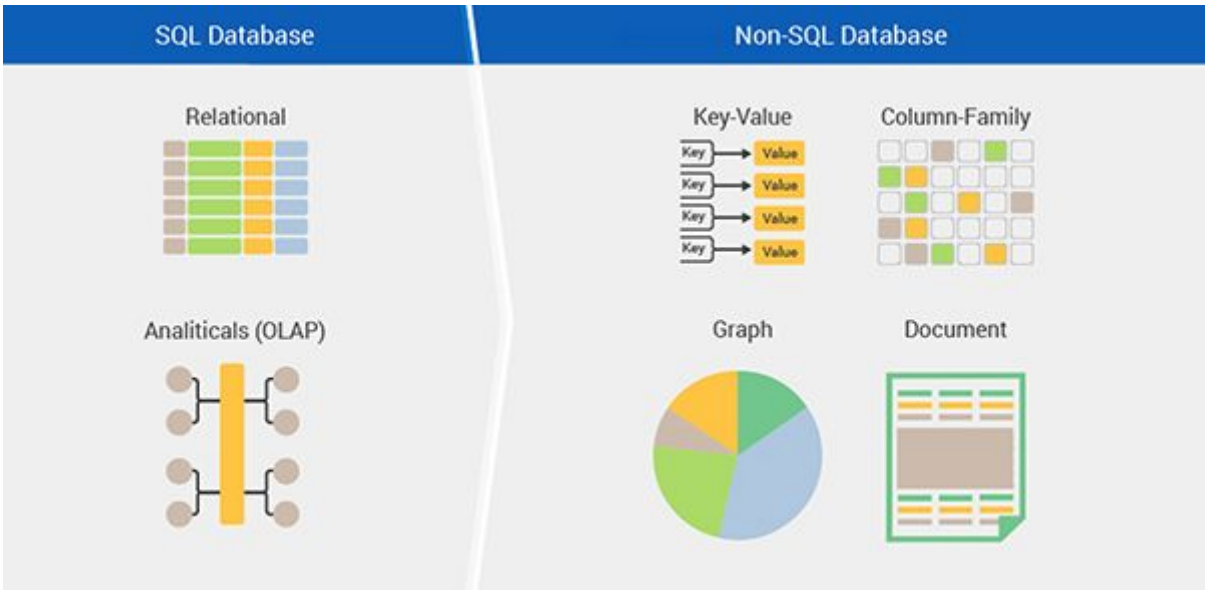
Càng về sau, công nghệ ngày càng phát triển, nhất là từ khi web nổi lên, khối lượng dữ liệu, thông tin về người dùng, về sản phẩm, đối tượng, sự kiện mà các hệ thống cần phải xử lý giờ đây ngày 1 lớn. Ví dụ như Google, Facebook phải lưu trữ và xử lý một lượng dữ liệu cực lớn mỗi ngày. Ngay cả việc hiển thị một trang web hoặc trả lời một yêu cầu API có thể làm mất hàng chục hoặc hàng trăm yêu cầu cơ sở dữ liệu khi các ứng dụng xử lý thông tin ngày một phức tạp.

Giờ đây thì SQL lại gây một số cản trở với những hạn chế - cụ thể là lược đồ/schema cứng nhắc, thiếu linh hoạt - khiến chúng trở nên ít phù hợp hơn với các loại ứng dụng khác.

Từ những năm 2000 đến nay, một thứ gọi là NoSQL ra đời nhằm giải quyết các hạn chế của cơ sở dữ liệu SQL, đặc biệt liên quan đến quy mô, nhân rộng và lưu trữ dữ liệu phi cấu trúc. NoSQL không có nghĩa là đối lập hoàn toàn lại với SQL, một vài hệ quản trị NoSQL vẫn hỗ trợ SQL, vậy nên NoSQL có tên gọi thân thiện và đúng nghĩa nhất là “Not only SQL”.



b. Phân loại NoSQL



i. Dạng column-family

Dữ liệu được lưu trữ trong các cột thay vì các hàng như trong một hệ thống SQL thông thường. Bất kỳ số lượng cột nào (và do đó nhiều loại dữ liệu khác nhau) có thể được nhóm hoặc tổng hợp khi cần cho truy vấn hoặc chế độ xem dữ liệu. Ví dụ: HBase, Cassandra

Ưu điểm: Tìm kiếm nhanh, Phân tán dữ liệu tốt.
Nhược điểm: Hỗ trợ được với rất ít phần mềm.

ii. Dạng key-value

Các giá trị dạng tự do, từ số nguyên hoặc chuỗi đơn giản đến các tài liệu JSON phức tạp, được truy cập trong cơ sở dữ liệu bằng các khóa. Ví dụ: Redis, Riak

Ưu điểm: Tìm kiếm rất nhanh.
Nhược điểm: Lưu dữ liệu không theo khuôn dạng (schema) nhất định.

iii. Dạng document

Dữ liệu được thêm vào lưu trữ dưới dạng cấu trúc JSON tự do hoặc "tài liệu", trong đó dữ liệu có thể là bất kỳ kiểu nào, từ số nguyên đến chuỗi hay đến các văn bản tự do. Ví dụ: CouchDB, MongoDB

Ưu điểm: Dùng khi dữ liệu nguồn không được mô tả đầy đủ.
Nhược điểm: Hiệu năng truy vấn, Không có cú pháp chuẩn cho câu truy vấn dữ liệu.

iv. Dạng graph

Dữ liệu được biểu diễn dưới dạng mạng hoặc đồ thị của các thực thể và các mối quan hệ của thực thể đó, với mỗi node trong biểu đồ là một khối dữ liệu ở dạng tự do. Ví dụ: Neo4j, OrientDB

Ưu điểm: Ứng dụng các thuật toán trên đồ thị như Đường đi ngắn nhất, liên thông,…
Nhược điểm: Phải duyệt nội bộ đồ thị, để trả lời lại các truy vấn. Không dễ để phân tán.

2. Tổng quan về OrientDB

a. Giới thiệu chung

OrientDB là một Hệ quản trị cơ sở dữ liệu mô hình NoSQL được viết bằng Java. Nó là một cơ sở dữ liệu nhiều mô hình (multi-model) hỗ trợ các mô hình Graph, Document, Key - Value và mô hình đối tượng, nhưng các mối quan hệ giữa các mô hình được quản lý trong cơ sở dữ liệu Graph bởi các liên kết giữa các bản ghi. Nó hỗ trợ các chế độ giản đồ, lược đồ đầy đủ và lược đồ hỗn hợp. Nó có khả năng bảo mật hệ thống tốt dựa trên người dùng và các quyền (roles), đồng thời hỗ trợ các câu lệnh truy vấn với Gremlin cùng với SQL cho việc duyệt Graph. OrientDB sử dụng một vài kỹ thuật đánh chỉ mục (index) dựa trên B-tree và Hash Index. Mỗi bản ghi có một Surrogate key để xác định vị trí của bản ghi trong mảng. Các liên kết giữa các bản ghi được lưu trữ trong bộ nhớ như một giá trị của vị trí của bản ghi hoặc như một B-tree của vị trí bản ghi (được gọi là RIDs), giúp cho việc truy vấn các bản ghi nhanh hơn. OrientDB là cơ sở dữ liệu đồ thị phổ biến thứ ba theo xếp hạng cơ sở dữ liệu đồ thị của DB-Engines, tính đến tháng 9 năm 2017.

Sự phát triển của OrientDB vẫn dựa vào một cộng đồng mã nguồn mở do công ty OrientDB LTD dẫn đầu bởi tác giả ban đầu của nó là Luca Garulli. Dự án sử dụng GitHub để quản lý các tài nguyên, cộng tác viên và các phiên bản, Google Group và Stack Overflow để cung cấp hỗ trợ miễn phí cho người dùng trên toàn thế giới. OrientDB cũng cung cấp một khóa học Udemy miễn phí cho những ai muốn tìm hiểu những kiến thức cơ bản và bắt đầu với OrientDB.

b. Các khái niệm trong OrientDB

- **Record**

Là đơn vị nhỏ nhất có thể tải và lưu trữ trong cơ sở dữ liệu, các bản ghi có thể được lưu trữ trong loại:

1. Document
2. Record Bytes
3. Vertex
4. Edge

- **Record ID**

Khi tạo một bản ghi, máy chủ sẽ tự động gán một mã định danh cho bản ghi, được gọi là Record ID, nó có dạng # <cluster> : <position>.

Trong đó:

- cluster: Số nhận dạng cụm
- position: vị trí tuyệt đối của bản ghi trong cụm

- **Document**

Tài liệu là một dạng bản ghi linh hoạt, có thể xử lý tài liệu bằng cách nhập xuất dưới dạng JSON, ví dụ:

```
{
  "id"       : "1201",
  "name"     : "Jay",
```

```
  "job"      : "Developer",
  "creations" : [
    {
      "name"    : "Amiga",
      "company" : "Commodore Inc."
    },

    {
      "name"    : "Amiga 500",
      "company" : "Commodore Inc."
    }
  ]
}
```

● **Record Bytes**

Giống như dạng BLOB(Binary Large Object) trong cơ sở dữ liệu quan hệ.

● **Vertex**

Cơ sở dữ liệu OrientDB không chỉ là cơ sở dữ liệu tài liệu mà còn là cơ sở dữ liệu đồ thị. Các khái niệm mới như Vertex và Edge được sử dụng để lưu trữ dữ liệu dưới dạng đồ thị. Trong cơ sở dữ liệu đồ thị, đơn vị dữ liệu cơ bản nhất là nút, trong OrientDB được gọi là đỉnh. Vertex lưu trữ thông tin cho cơ sở dữ liệu.

● **Edge**

Là một dạng bản ghi đặc biệt để kết nối các đỉnh (vertex) trong đồ thị với nhau, hay còn được gọi là cạnh của đồ thị, các cạnh có 2 chiều.

● **Class**

Lớp là một kiểu mô hình dữ liệu và là khái niệm được rút ra từ mô hình lập trình hướng đối tượng. Dựa trên mô hình cơ sở dữ liệu tài liệu truyền thống, dữ liệu được lưu trữ dưới dạng tập hợp, trong khi trong mô hình cơ sở dữ liệu quan hệ dữ liệu được lưu trữ trong các bảng. OrientDB tuân theo API tài liệu cùng với mô hình OPPS. Theo một khái niệm, lớp trong OrientDB có mối quan hệ gần nhất với bảng trong cơ sở dữ liệu quan hệ, nhưng (không giống như bảng) các lớp có thể là lược đồ ít hơn, lược đồ đầy đủ hoặc hỗn hợp. Các lớp có thể kế thừa từ các lớp khác, tạo cây của các lớp. Mỗi lớp có cụm hoặc cụm riêng, (được tạo theo mặc định, nếu không có cụm nào được định nghĩa).

● **Cluster**

Cluster là một khái niệm quan trọng được sử dụng để lưu trữ các bản ghi, tài liệu hoặc các đỉnh. Nói một cách dễ hiểu, Cluster là nơi lưu trữ một nhóm các bản ghi. Theo mặc định, OrientDB sẽ tạo một cụm cho mỗi lớp. Tất cả các bản ghi của một lớp được lưu trữ trong cùng một cụm có cùng tên với lớp. Bạn có thể tạo tối đa 32.767 * (2^15 - 1) cụm trong cơ sở dữ liệu. CREATE là một lệnh được sử dụng để tạo một cụm với tên cụ thể. Khi cụm được tạo, bạn có thể sử dụng cụm để lưu bản ghi bằng cách chỉ định tên trong quá trình tạo bất kỳ mô hình dữ liệu nào.

● **Relationships**

OrientDB hỗ trợ hai loại mối quan hệ: tham chiếu và nhúng. Mối quan hệ tham chiếu có nghĩa là nó lưu trữ liên kết trực tiếp đến các đối tượng mục tiêu của mối quan hệ. Các mối quan hệ được nhúng có nghĩa là nó lưu trữ mối quan hệ trong bản ghi nhúng nó. Mối quan hệ này mạnh hơn mối quan hệ tham chiếu.

● **Database**

Cơ sở dữ liệu là một giao diện để truy cập vào bộ nhớ thực. CNTT hiểu các khái niệm cấp cao như truy vấn, lược đồ, siêu dữ liệu, chỉ số, v.v. OrientDB cũng cung cấp nhiều kiểu cơ sở dữ liệu.

c. Ưu, nhược điểm

i. Các ưu điểm

1. Cơ sở dữ liệu đa mô hình (Graph, Document, Object).

   OrientDB chủ yếu, hoặc về mặt lịch sử, được coi là cơ sở dữ liệu đồ thị. Tuy nhiên, nó thực sự là một cơ sở dữ liệu đa mô hình hỗ trợ nhiều mô hình NoSQL (key-value, document, v.v.).
2. Mã nguồn mở (Apache2).

   Giấy phép Apache 2 là một trong những giấy phép tự do nhất. Bạn có thể sử dụng OrientDB cho bất kỳ mục đích nào miễn phí.
3. Hỗ trợ phân cụm miễn phí.

   Không giống như các cơ sở dữ liệu NoSQL khác, OrientDB cung cấp hỗ trợ phân cụm miễn phí. Hơn nữa, nó không phải là Master / Slave cơ bản, nhưng nó hỗ trợ sao chép Master-Master + Sharding.
4. Cơ sở dữ liệu đồ thị phân tán với kiểu quan hệ Document.
5. Hỗ trợ SQL API và các ngôn ngữ truy vấn khác.

   OrientDB có thể được truy vấn theo một số cách, nhưng truy vấn bằng SQL làm cho nó quen thuộc hơn khi sử dụng.
6. Multi-master replication.

   OrientDB có khả năng mở rộng đáng kinh ngạc nhờ khả năng multi-master replication. Tất cả các nodes trong một cluster có thể đọc và viết.
7. Tìm kiếm full-text

   Tìm kiếm full-text được hỗ trợ khi sử dụng thuật toán lập chỉ mục dựa trên Lucene.
8. Hiệu năng.

   OrientDB rất nhanh, đặc biệt là khi nói đến mối quan hệ giữa các bản ghi (hoặc "liên kết").
9. Có phiên bản doanh nghiệp.

   Phiên bản Cộng đồng miễn phí bao gồm nhiều tính năng, nhưng phiên bản Doanh nghiệp tuyệt hơn (giá cả minh bạch, được công bố trực tuyến) với hỗ trợ 24x7.

ii. Các nhược điểm.

1. Không được biết đến nhiều

   Không có nhiều người biết đến cơ sở dữ liệu này, vì vậy đôi khi rất khó tìm được sự hỗ trợ của cộng đồng. Nó cũng gây khó khăn cho việc thu hút những người mua sắm các sản phẩm theo thương hiệu hoặc tiếp thị. Điều đó cũng có nghĩa là có ít tài liệu, thư viện và framework hơn. Mặc dù khả năng tương tác của nó rất tốt với sự hỗ trợ của SQL và Tinkerpop, nhưng thực tế là ít nhà phát triển biết đến cơ sở dữ liệu này.

2. Khó để thành thục.

   Khá khó để sử dụng OrientDB thành thục (chẳng hạn như truy vấn SQL). Tuy nhiên, điều quan trọng là phải hiểu một chút cơ sở dữ liệu và cách nó hoạt động nói chung (ví dụ: các liên kết). Điều này dẫn đến một chút khó khăn khi làm quen, mặc dù nó không quá lớn.

3. Chèn hàng loạt có thể gây ra tràn bộ nhớ.

   Khi thực hiện các thao tác chèn trong OrientDB, bạn phải cẩn thận thực hiện từng lần một. Chèn số lượng lớn có thể gây ra lỗi hết bộ nhớ.

d. Các câu lệnh, truy vấn trong OrientDB

i. Các câu lệnh truy vấn SQL (Basic CRUD)

1. `INSERT`

   Câu lệnh `INSERT` để thêm/chèn một/nhiều bản ghi mới vào cơ sở dữ liệu.

   Cấu trúc câu lệnh :

```
INSERT INTO
[CLASS:]<class>|CLUSTER:<cluster>|INDEX:<index>
  [(<field>[,]*) VALUES (<expression>[,]*)[,]*]|
  [SET <field> = <expression>|<sub-command>[,]*]|
  [CONTENT {<JSON>}]
  [RETURN <expression>]
  [FROM <query>]
```

   Trong đó :

   - **`CONTENT`** xác định dữ liệu truyền vào
   - **`RETURN`** xác định một biểu thức để trả về thay vì số lượng bản ghi được chèn vào. Bạn có thể sử dụng bất kỳ biểu thức SQL hợp lệ nào. Các trường hợp sử dụng phổ biến nhất,
     - `@rid` trả về Record Id của bản ghi mới
     - `this` trả về toàn bộ bản ghi
   - **`FROM`** xác định nơi bạn muốn chèn tập hợp kết quả vào.

   Ví dụ :

   Chèn 1 bản ghi vào bảng Customer.

```
INSERT INTO Customer (id, first_name, last_name,
city_id) VALUES (1, “John”, “Wick”, 3);
```

2. `UPDATE`

   Câu lệnh `UPDATE` để cập nhật một/nhiều bản ghi.

Cấu trúc câu lệnh:

```
UPDATE <class>|CLUSTER:<cluster>|<recordID>
  [SET|REMOVE <field-name> =
<field-value>[,]*]|[CONTENT|MERGE <JSON>]
  [UPSERT]
  [RETURN <returning> [<returning-expression>]]
  [WHERE <conditions>]
  [LOCK default|record]
  [LIMIT <max-records>] [TIMEOUT <timeout>]
```

Trong đó:

- **`SET`** xác định trường muốn cập nhật.
- **`REMOVE`** xóa một bản ghi trong collection
- **`CONTENT`** thay thế bản ghi với bản ghi kiểu JSON.
- **`MERGE`** trộn bản ghi với bản ghi kiểu JSON.
- **`LOCK`** Chỉ định cách khóa bản ghi giữa tải và cập nhật. Bạn có thể sử dụng một trong các chiến lược khóa sau:
    - `DEFAULT` Không khóa. Sử dụng trong trường hợp cập nhật đồng thời, MVCC ném một ngoại lệ.
    - `RECORD` khóa bản ghi trong khi update.
- **`UPSERT`** Cập nhật bản ghi nếu nó tồn tại hoặc chèn một bản ghi mới nếu nó không tồn tại. Điều này tránh sự cần thiết phải thực hiện hai lệnh, (một lệnh cho mỗi điều kiện, chèn và cập nhật).
  `UPSERT` yêu cầu một mệnh đề `WHERE` và một đối tượng class.
- **`RETURN`** Chỉ định một biểu thức để trả về thay vì bản ghi và phải làm gì với tập kết quả được trả về bởi biểu thức. Các toán tử trả lại có sẵn là:
    - `COUNT` Trả về số lượng bản ghi đã cập nhật. Đây là toán tử trả về mặc định.
    - `BEFORE` Trả về bản ghi trước khi update.
    - `AFTER` Trả về bản ghi sau khi update.
- `LIMIT` xác định số lượng bản ghi tối đa để update.
- `TIMEOUT` xác định thời gian bạn muốn cho phép bản cập nhật chạy trước khi hết thời gian.

Ví dụ:

Cập nhật tên của một bản ghi trong bảng Customer.

```
UPDATE Customer SET first_name = "Jack" WHERE
first_name = "John";
```

3. `DELETE`

Câu lệnh `DELETE` để xóa một/nhiều bản ghi trong cơ sở dữ liệu.

Cấu trúc câu lệnh:

```
DELETE FROM <Class>|CLUSTER:<cluster>|INDEX:<index>
[LOCK <default|record>] [RETURN <returning>]
```

```
  [WHERE <Condition>*] [LIMIT <MaxRecords>] [TIMEOUT
<timeout>]
```

Trong đó :

- **LOCK** Chỉ định cách khóa bản ghi giữa tải và xóa. Bạn có thể sử dụng một trong các chiến lược khóa sau:
  - DEFAULT Không khóa. Sử dụng trong trường hợp cập nhật đồng thời, MVCC ném một ngoại lệ.
  - RECORD khóa bản ghi trong khi update.
- **RETURN** Chỉ định một biểu thức để trả về thay vì bản ghi và phải làm gì với tập kết quả được trả về bởi biểu thức. Các toán tử trả lại có sẵn là:
  - COUNT Trả về số lượng bản ghi đã xóa. Đây là toán tử trả về mặc định.
  - BEFORE Trả về bản ghi trước khi xóa.
- **WHERE** lọc bản ghi bạn muốn xóa
- LIMIT xác định số lượng bản ghi tối đa để update.
- TIMEOUT xác định thời gian bạn muốn cho phép chạy trước khi hết thời gian.
- **UNSAFE** Cho phép xử lý XÓA trên Đỉnh hoặc Cạnh mà không có lỗi ngoại lệ. Không khuyến khích sử dụng! Nếu bạn phải xóa một Cạnh hoặc một Đỉnh, hãy sử dụng các lệnh tương ứng DELETE EDGE hoặc DELETE VERTEX.

Ví dụ:

Xóa một bản ghi trong bảng Customer.

```
DELETE FROM Customer WHERE last_name = "Wick";
```

4. SELECT

Câu lệnh SELECT dùng để đọc một/nhiều bản ghi từ cơ sở dữ liệu.

Cấu trúc câu lệnh :

```
SELECT [ <Projections> ] [ FROM <Target> [ LET
<Assignment>* ] ]
    [ WHERE <Condition>* ]
    [ GROUP BY <Field>* ]
    [ ORDER BY <Fields>* [ ASC|DESC ] * ]
    [ UNWIND <Field>* ]
    [ SKIP <SkipRecords> ]
    [ LIMIT <MaxRecords> ]
    [ FETCHPLAN <FetchPlan> ]
    [ TIMEOUT <Timeout> [ <STRATEGY> ]
    [ LOCK default|record ]
    [ PARALLEL ]
    [ NOCACHE ]
```

Trong đó :

- **WHERE** lọc nơi muốn lấy dữ liệu.
- **LET** Liên kết các biến ngữ cảnh để sử dụng trong các phép chiếu, điều kiện hoặc truy vấn phụ.
- **GROUP BY** xác định trường để nhóm kết quả.

- **ORDER BY** chỉ định trường để sắp xếp tập hợp kết quả. Sử dụng các toán tử ASC và DESC tùy chọn để xác định hướng của lệnh. Mặc định là tăng dần. Ngoài ra, nếu bạn đang sử dụng một phép chiếu, bạn cần bao gồm trường ORDER BY trong phép chiếu. Lưu ý rằng ORDER BY chỉ hoạt động trên các trường chiếu (trường được trả về trong tập kết quả) chứ không phải trên các biến LET.
- **UNWIND** chỉ định trường để giãn bộ sưu tập.
- **SKIP** Xác định số lượng bản ghi bạn muốn bỏ qua từ đầu tập kết quả. Bạn có thể thấy điều này hữu ích trong việc phân trang, khi sử dụng nó cùng với LIMIT
- LIMIT xác định số lượng bản ghi tối đa để đọc.
- **FETCHPLAN** xác định cách bạn muốn lấy kết quả.
- TIMEOUT xác định thời gian bạn muốn cho phép bản cập nhật chạy trước khi hết thời gian.
- **LOCK** Chỉ định cách khóa bản ghi giữa tải và đọc. Bạn có thể sử dụng một trong các chiến lược khóa sau:
    - DEFAULT Không khóa. Sử dụng trong trường hợp cập nhật đồng thời, MVCC ném một ngoại lệ.
    - RECORD khóa bản ghi trong khi update.
- **PARALLEL** thực thi truy vấn đối với x luồng đồng thời, trong đó x đề cập đến số lượng bộ xử lý hoặc lõi được tìm thấy trên hệ điều hành chủ của truy vấn. Bạn có thể thấy việc thực thi PARALLEL hữu ích trên các truy vấn chạy dài hoặc các truy vấn liên quan đến nhiều cụm. Đối với các truy vấn đơn giản, việc sử dụng PARALLEL có thể gây chậm do chi phí vốn có trong việc sử dụng nhiều luồng.
- **NOCACHE** xác định bạn muốn sử dụng cache hay không.

Ví dụ :
Chọn đọc một bản ghi trong bảng Customer có id là 1000.

```
SELECT FROM Customer WHERE id = 1000;
```

ii. Câu lệnh duyệt đồ thị (Traverse)

Câu lệnh TRAVERSE (duyệt đồ thị) dùng để truy xuất các bản ghi được kết nối qua các mối quan hệ. Câu lệnh này hiệu quả với cả Graph và Document API's. Tuy nhiên, trong nhiều trường hợp, câu lệnh SELECT sẽ có lợi hơn do ngắn và dễ hiểu hơn.
Cấu trúc câu lệnh :

```
TRAVERSE <[class.]field>|*|any()|all()
          [FROM <target>]
          [
            MAXDEPTH <number>
             |
            WHILE <condition>
          ]
          [LIMIT <max-records>]
```

```
[STRATEGY <strategy>]
```

Trong đó :

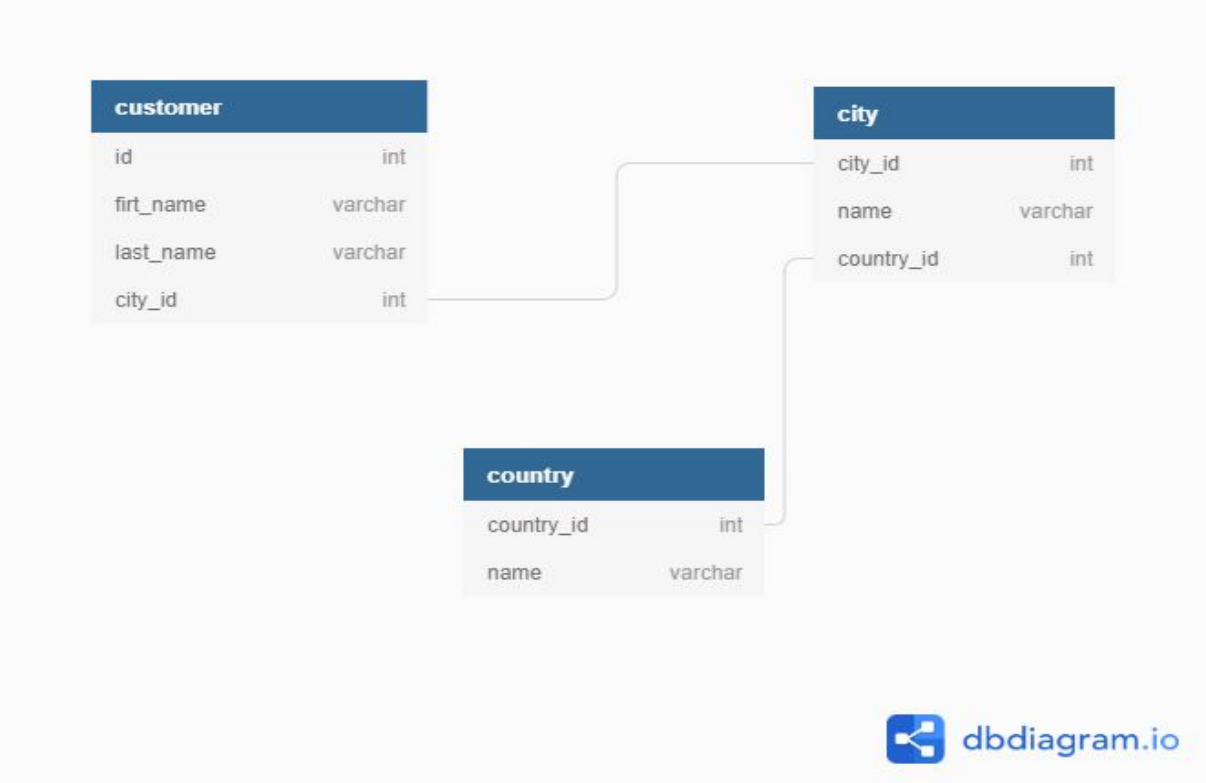
- `<fields>` xác định trường bạn muốn duyệt.
- `<target>` xác định mục tiêu bạn muốn duyệt. Đây có thể là một lớp, một hoặc nhiều cụm, một ID bản ghi, tập hợp các ID bản ghi hoặc một truy vấn phụ.
- `MAXDEPTH` xác định độ sâu tối đa của đường ngang. 0 chỉ ra rằng bạn chỉ muốn duyệt qua nút gốc. Giá trị âm không hợp lệ.
- `WHILE` xác định điều kiện để tiếp tục duyệt trong khi điều kiện là đúng.
- `LIMIT` xác định số lượng bản ghi tối đa có thể trả về.
- `STRATEGY` xác định chiến lược duyệt.

Ví dụ :
Duyệt tất cả các trường từ bản ghi gốc.

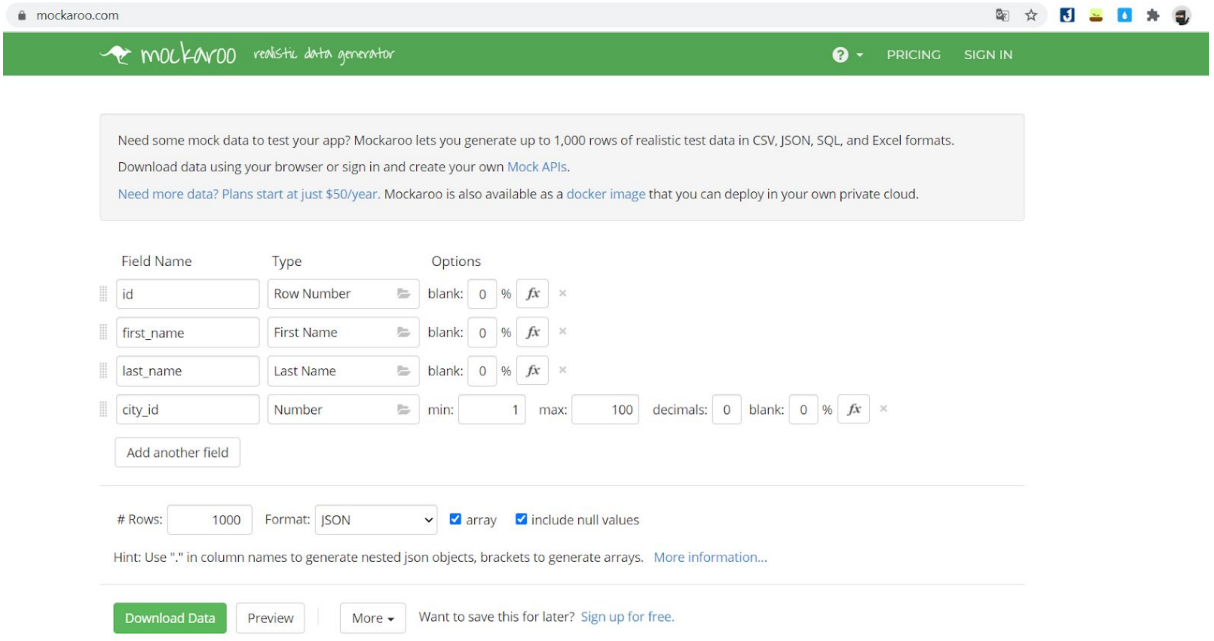
```
TRAVERSE * FROM #10:1234
```

3. Sinh dữ liệu

Dữ liệu lần này bọn em làm sẽ có 3 bảng để demo. Chúng em sẽ chèn 100.000.000 bản ghi vào bảng customer và 2 bảng còn lại chỉ chèn một vài dữ liệu mẫu để thực hiện truy vấn nối bảng và kiểm tra. Sau đây là cách chúng em chèn 100.000.000 bản ghi.



Thông qua npm, thêm bằng code JavaScript trên môi trường NodeJs.
Nguồn sinh dữ liệu từ web app mockaroo: https://www.mockaroo.com/



Ta sẽ lấy không gian sinh là ba thuộc tính với thuộc tính id duy nhất, ba thuộc tính còn lại được random và tất cả thuộc tính và giá trị được xuất ra file json như sau:

Lưu ý: *Mockaroo chỉ cho sinh 1000 dữ liệu random miễn phí. Tuy nhiên ta vẫn có thể yên tâm về tính độc nhất của dữ liệu của 10^8 bản ghi khi insert vào cơ sở dữ liệu.*

Chứng minh:

Có 1000 dữ liệu khác nhau ở ba trường first_name, last_name, city_id. Khi sinh dữ liệu cho các bản ghi, ta sẽ có tối đa 1000*1000*1000 (10^9) bản ghi tối đa có dữ liệu khác nhau. Hoàn toàn đủ đáp ứng 10^8 dữ liệu sinh theo đề bài yêu cầu.

a. Cách chèn 100 triệu bản ghi vào OrientDB

Bước 1: Mở thư mục chứa code thực thi, chạy command line: ***npm init --yes***

```
E:\DBMS-Nam>npm init --yes
Wrote to E:\DBMS-Nam\package.json:

{
  "name": "DBMS-Nam",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Bước 2: Chạy ***npm install orientjs*** để cài node package orientjs

```
+ orientjs@3.0.11
added 106 packages from 108 contributors and audited 121 packages in 10.58s
```

Bước 3: Thực hiện kết nối đến OrientDB

```
const OrientDBClient = require("orientjs").OrientDBClient;
```

Bước 4: Sử dụng module fs của JavaScript để đọc dữ liệu từ file dữ liệu đã sinh(db.json) vào mảng data

```
const fs = require("fs");
const data = JSON.parse(fs.readFileSync("db.json", { encoding: "utf-8" }));
```

Bước 5: Thực hiện đoạn code sinh dữ liệu và thêm vào OrientDB như sau:

```
var fs = require("fs");
var data = JSON.parse(fs.readFileSync("db.json", { encoding: "utf-8" }));
const OrientDBClient = require("orientjs").OrientDBClient;
(async function () {
  let client;
  try {
    client = await OrientDBClient.connect({ host: "localhost" });
    console.log("Connected");
  } catch (e) {
    console.log(e);
  }

  if (client) {
    var session = await client.session({
      name: "Demo",
      username: "root",
      password: "25072707",
    });
    // use the session
    var i = 1;
    let startTime = new Date();
    let count = 1;
    for(let i=1; i<=10000; i++){
      let query = "INSERT INTO Customer(id, first_name, last_name, city_id) VALUES";
      for(let j=1;j<=1000;j++){
        let customer = data[i*j % 100];
        query = query + `(${count}, ${customer.first_name}, ${customer.last_name}, ${customer.city_id}),`
        count ++;
      }
      let mainQuery = query.slice(0,query.length-1);
      await session.command(mainQuery).all();
    }
    let endTime = new Date();
    console.log(`Insert successfully in ${(endTime - startTime)/1000}s`);
    // close the session
    await session.close();
  }
})();
```

b. Cách chèn 100 triệu bản ghi vào MySQL

Bước 1: Thực hiện như bên trên với OrientDB

```
E:\DBMS-Nam>npm init --yes
Wrote to E:\DBMS-Nam\package.json:

{
  "name": "DBMS-Nam",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Bước 2: Chạy ***npm install mysql*** để cài node package mysql

```
+ mysql@2.18.1
added 9 packages from 13 contributors and audited 121 packages in 2.693s
```

Bước 3: Cấu hình kết nối với mysql

```
const mysql      = require('mysql');
const connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : '18020931',
  database : 'demodbms'
});
```

Bước 4: Sử dụng module fs của JavaScript để đọc dữ liệu từ file dữ liệu đã sinh(db.json) vào mảng data

```
const fs = require("fs");
const data = JSON.parse(fs.readFileSync("db.json", { encoding: "utf-8" }));
```

Bước 5: Thực hiện đoạn code sinh vào MySQL như sau:

```
var connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "mydbms",
  port: 3306,
  connectTimeout: 1000000,
});
connection.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
var query = 'INSERT INTO customer VALUES ';
var data = JSON.parse(fs.readFileSync('./db.json', {encoding:"utf-8"}));

  for (var  i= 1;i <= 4000000;i++) {
    var rand1 = Math.floor(Math.random()*100);
    var rand2 = Math.floor(Math.random()*100);
    var rand3 = Math.floor(Math.random()*10);
    query =  query + "(" + i + "," + '"' + data[rand1].first_name +'"' + "," + '"' + data[rand2].last_name +'"' + "," + data[rand3].city_id+ "),";
  }
 var finalQuery = query.slice(0,query.length-1);
 var startTime = new Date();
 connection.query(finalQuery, (error, result, field) => {
   if (error) {
     console.log("Error");
     connection.end();
   }
   console.log("Done!");
   var endTime = new Date();
   console.log((endTime - startTime)/1000);
   connection.end();
 })
```

c. Thời gian chèn

i. OrientDB

100.000 bản ghi

```
let startTime = new Date();
for(let i=0; i<100; i++){
  let query = "INSERT INTO Customer(id, first_name, last_name, city_id) VALUES";
  for(let j=1;j<=1000;j++){
    let customer = data[i*j % 100];
    query = query + `(${i*j}, ${customer.first_name}, ${customer.last_name}, ${customer.city_id}),`
  }
  let mainQuery = query.slice(0,query.length-1);
  await session.command(mainQuery).all();
}
let endTime = new Date();
console.log(`Insert successfully in ${(endTime - startTime)/1000}s`);
```

```
PS E:\DemoDBMS> node orientdb.js
Connected
Insert successfully in 3.991s
```

1.000.000 bản ghi

```
let startTime = new Date();
for(let i=0; i<1000; i++){
  let query = "INSERT INTO Customer(id, first_name, last_name, city_id) VALUES";
  for(let j=1;j<=1000;j++){
    let customer = data[i*j % 100];
    query = query + `(${i*j}, ${customer.first_name}, ${customer.last_name}, ${customer.city_id}),`
  }
  let mainQuery = query.slice(0,query.length-1);
  await session.command(mainQuery).all();
}
let endTime = new Date();
console.log(`Insert successfully in ${(endTime - startTime)/1000}s`);
```

```
PS E:\DemoDBMS> node orientdb.js
Connected
Insert successfully in 43.365s
```

100.000.000 bản ghi

```
var i = 1;
let startTime = new Date();
for(let i=0; i<100000; i++){
  let query = "INSERT INTO Customer(id, first_name, last_name, city_id) VALUES";
  for(let j=1;j<=1000;j++){
    let customer = data[i*j % 100];
    query = query + `(${i*j}, ${customer.first_name}, ${customer.last_name}, ${customer.city_id}),`
  }
  let mainQuery = query.slice(0,query.length-1);
  await session.command(mainQuery).all();
}
let endTime = new Date();
console.log(`Insert successfully in ${(endTime - startTime)/1000}s`);
```

```
PS E:\DemoDBMS> node orientdb.js
Connected
Insert successfully in 6473.685s
```

ii. MySQL

100.000 bản ghi

```
var query = 'INSERT INTO customer VALUES ';
var data = JSON.parse(fs.readFileSync('./db.json', {encoding:"utf-8"}));

 for (var  i= 1;i <= 100000;i++) {
   var rand1 = Math.floor(Math.random()*100);
   var rand2 = Math.floor(Math.random()*100);
   var rand3 = Math.floor(Math.random()*10);
   query =  query + "(" + i + "," + '"' + data[rand1].first_name +'"' + "," + '"' + data[rand2].last_name +'"' + "," + data[rand3].city_id+ "),";
 }
var finalQuery = query.slice(0,query.length-1);
var startTime = new Date();
connection.query(finalQuery, (error, result, field) => {
  if (error) {
    throw error;
    connection.end();
  }
  console.log("Done!");
  var endTime = new Date();
  console.log((endTime - startTime)/1000);
})
```

```
PS C:\Users\ADMIN\Dev\DBMS> node index1.js
Connected!
Done!
0.983
```

1.000.000 bản ghi

```
var query = 'INSERT INTO customer VALUES ';
var data = JSON.parse(fs.readFileSync('./db.json', {encoding:"utf-8"}));

  for (var  i= 1;i <= 1000000;i++) {
    var rand1 = Math.floor(Math.random()*100);
    var rand2 = Math.floor(Math.random()*100);
    var rand3 = Math.floor(Math.random()*10);
    query =  query + "(" + i + "," + '"' + data[rand1].first_name +'"' + "," + '"' + data[rand2].last_name +'"' + "," + data[rand3].city_id+ "),";
  }
 var finalQuery = query.slice(0,query.length-1);
var startTime = new Date();
 connection.query(finalQuery, (error, result, field) => {
   if (error) {
     throw error;
     connection.end();
   }
   console.log("Done!");
   var endTime = new Date();
   console.log((endTime - startTime)/1000);
 })
```

```
PS C:\Users\ADMIN\Dev\DBMS> node index1.js
Connected!
Done!
10.048
```

100.000.000 bản ghi

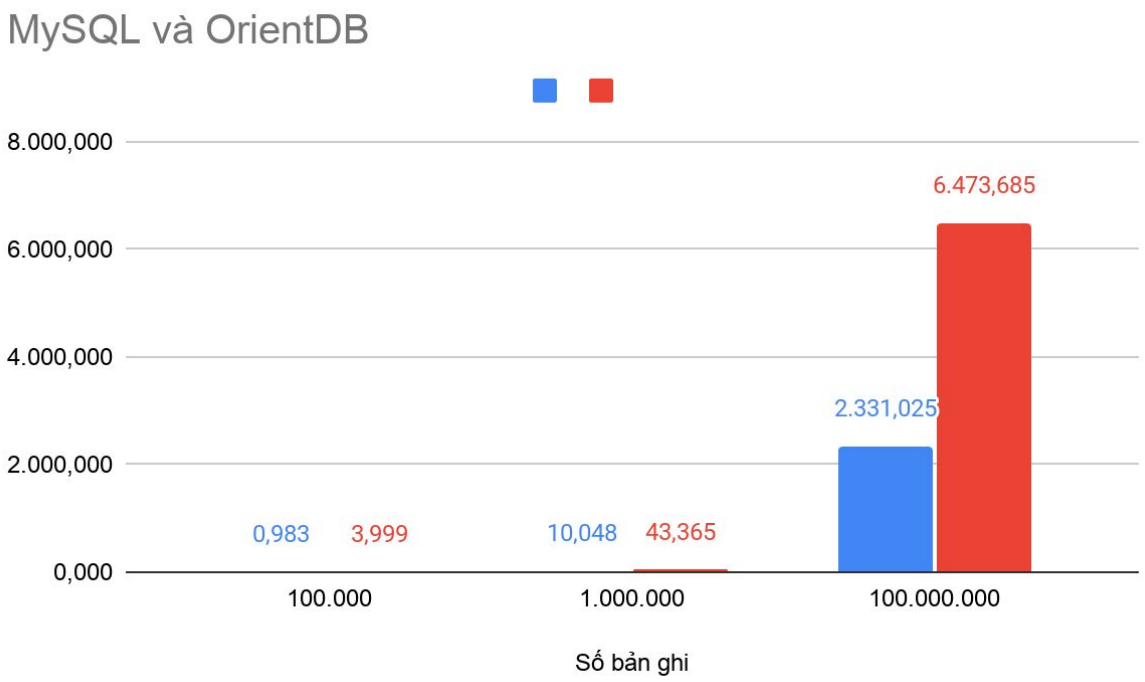
```
var query = 'INSERT INTO customer VALUES ';
var data = JSON.parse(fs.readFileSync('./db.json', {encoding:"utf-8"}));

  for (var  i= 1;i <= 4000000;i++) {
    var rand1 = Math.floor(Math.random()*100);
    var rand2 = Math.floor(Math.random()*100);
    var rand3 = Math.floor(Math.random()*10);
    query =  query + "(" + i + "," + '"' + data[rand1].first_name +'"' + "," + '"' + data[rand2].last_name +'"' + "," + data[rand3].city_id+ "),";
  }
 var finalQuery = query.slice(0,query.length-1);
 var startTime = new Date();
 connection.query(finalQuery, (error, result, field) => {
   if (error) {
     console.log("Error");
     connection.end();
   }
   console.log("Done!");
   var endTime = new Date();
   console.log((endTime - startTime)/1000);
   connection.end();
 })
```

Chèn 4.000.000 bản ghi lần lượt. Mỗi bản ghi hết xấp xỉ:

```
Connected!
Done!
93.241
```

Ước lượng chèn 100.000.000 bản ghi: 93.241 * 25 = 2331.025s

Số bản ghi	MySQL	OrientDB
100.000	0.983s	3.9991s
1.000.000	10.048s	43.365s
100.000.000	2331.025s	6473.685s



d. Nhận xét

+ Việc chèn bản ghi vào trong hai hệ quản trị là gần như tăng tuyến tính. Việc có sai số khỏi đường tuyến tính là do việc cấp phát biến trong khi chèn.
+ Việc chèn dữ liệu vào MySQL tăng nhanh hơn rất nhiều so với việc chèn bản ghi vào trong OrientDB. Để có thể chèn nhanh hơn thì có thể dùng đa luồng trong các ngôn ngữ như Java, tuy nhiên JavaScript không có khái niệm đa luồng, nên chèn dựa vào cơ chế bất đồng bộ. Có lẽ đây cũng là một lý do khiến cho việc chèn lâu. Có thể tham khảo bài viết ở link sau: https://stackoverflow.com/questions/29488241/orientdb-slow-insert

4. So sánh và đánh giá hiệu năng

a. Chưa đánh chỉ mục

i. Các truy vấn cơ bản

1. Select from

Câu lệnh thực hiện như sau:

MySQL: `SELECT COUNT(*) FROM customer`

OrientDB: `SELECT COUNT(*) FROM customer`

Số bản ghi	MySQL	OrientDB
100.000.000	23.465s	0.0451s

2. Select from where

Câu lệnh thực hiện như sau:

MySQL: `SELECT * FROM customer WHERE id = 18020931`

OrientDB: `SELECT FROM Customer WHERE id = 18020931`

Số bản ghi	MySQL	OrientDB
100.000.000	0.0027s	0.032s

3. Select from group by

Câu lệnh thực hiện như sau:

MySQL: `SELECT COUNT(*) FROM customer GROUP BY city_id`

OrientDB: `SELECT COUNT(*) FROM Customer GROUP BY city_id`

Số bản ghi	MySQL	OrientDB
100.000.000	68.557s	696.524s

ii. Nối các bảng

1. 2 bảng

Câu lệnh:

MySQL:

```
SELECT * FROM Customer cus JOIN city c ON cus.city_id =
c.city_id
WHERE c.name = "Ouagadougou";
```

OrientDB:

```
SELECT FROM Customer WHERE city_id.name = 'KOuagadougou';
```

Số bản ghi	MySQL	OrientDB
100.000.000	102.129s	607.234s

2. 3 bảng

Câu lệnh:

MySQL:

```
SELECT * FROM Customer cus JOIN city c ON cus.city_id =
c.city_id
JOIN country cou ON c.country_id = cou.country_id
WHERE cou.name = "Chile";
```

OrientDB:

```
SELECT FROM Customer WHERE city_id.country_id.name =
'Chile';
```

Số bản ghi	MySQL	OrientDB
100.000.000	104.734s	643.657s

iii. Thêm mới bản ghi

Câu lệnh thực hiện như sau:

```
INSERT INTO Customer(id, first_name, last_name, city_id)
  VALUES (100000001, "John", "Wick", 2);
```

Số bản ghi	MySQL	OrientDB
100.000.000	0.0202s	0.129s

b. Đánh chỉ mục (index)

Câu lệnh thực hiện như sau:

MySQL: `CREATE UNIQUE INDEX Customer_id ON customer (id);`

OrientDB: `CREATE INDEX Customer.id UNIQUE;`

Số bản ghi	MySQL	OrientDB
100.000.000	264.738s	7210s

i. Các truy vấn cơ bản

1. Select from

Câu lệnh thực hiện như sau:

MySQL: `SELECT COUNT(*) FROM customer GROUP BY city_id`

OrientDB: `SELECT COUNT(*) FROM Customer GROUP BY city_id`

Số bản ghi	MySQL	OrientDB
100.000.000	21.076s	0.015s

2. Select from where

Câu lệnh thực hiện như sau:

MySQL: `SELECT * FROM customer WHERE id = 18020931`

OrientDB: `SELECT FROM Customer WHERE id = 18020931`

Số bản ghi	MySQL	OrientDB
100.000.000	0.0024s	0.039s

3. Select from group by

Câu lệnh thực hiện như sau:

MySQL: `SELECT COUNT(*) FROM customer GROUP BY city_id`

OrientDB: `SELECT COUNT(*) FROM Customer GROUP BY city_id`

Số bản ghi	MySQL	OrientDB
100.000.000	35.8722s	534.812s

ii. Nối các bảng

1. 2 bảng

MySQL:

```
SELECT * FROM Customer cus JOIN city c ON cus.city_id =
c.city_id
WHERE c.name = "Ouagadougou";
```

OrientDB:

```
SELECT FROM Customer WHERE city_id.name = 'Ouagadougou';
```

Số bản ghi	MySQL	OrientDB
100.000.000	1.275s	601.584s

2. 3 bảng

MySQL:

```
SELECT * FROM Customer cus JOIN city c ON cus.city_id =
c.city_id
JOIN country cou ON c.country_id = cou.country_id
WHERE cou.name = "Chile";
```

OrientDB:

```
SELECT FROM Customer WHERE city_id.country_id.name =
'Chile';
```

Số bản ghi	MySQL	OrientDB
100.000.000	1.908s	663.234s

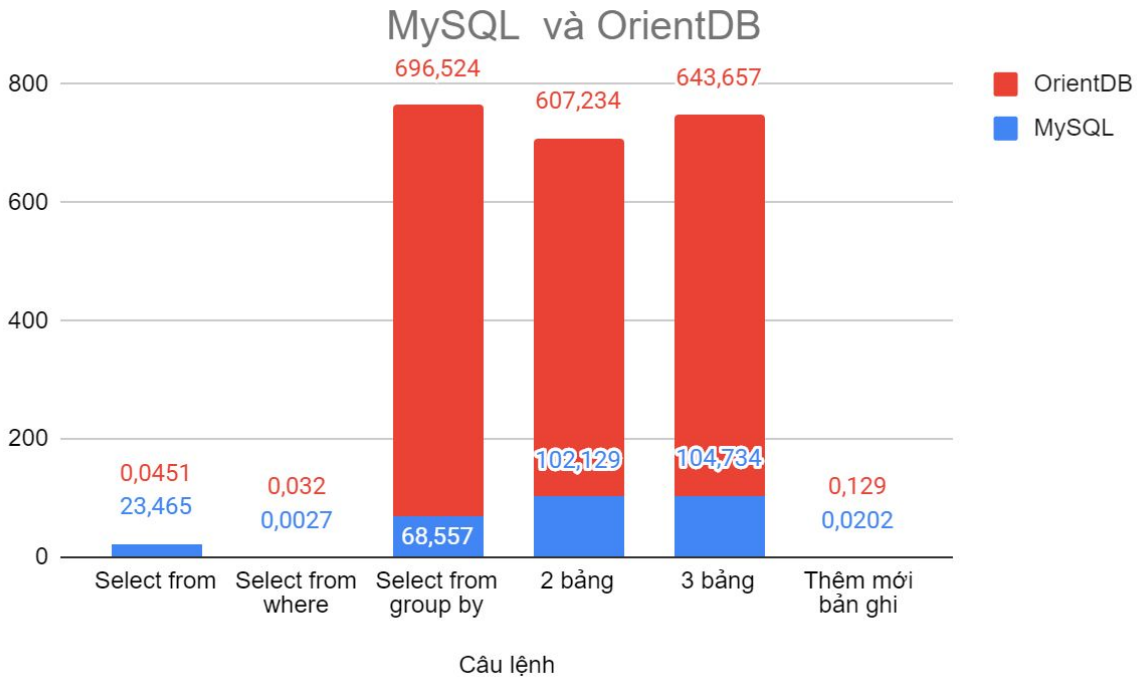
iii. Thêm mới bản ghi

Câu lệnh thực hiện như sau:

```
INSERT INTO Customer(id, first_name, last_name, city_id)
  VALUES (100000001, "John", "Wick", 2);
```

Số bản ghi	MySQL	OrientDB
100.000.000	0.024s	0.048s

5. Kết luận



Dựa vào dữ liệu thống kê được trên các bộ dữ liệu ở trên, ta nhận thấy **MySQL tỏ ra có ưu thế về thời gian rất rõ ràng so với OrientDB**. Tuy nhiên **sau khi đánh chỉ mục thì OrientDB có những bước đột phá về thời gian truy vấn**. Về mặt đếm số lượng bản ghi, OrientDB cho ta thấy một tốc độ vượt trội so với MySQL. Điều này cũng rõ ràng khi phiên bản hiện tại là bản 3.0, được đưa ra từ 18/12/2017

Released OrientDB v 3.0.0 RC1

Mon, 12/18/2017 - 17:51

London - December 18, 2017 The OrientDB Team has just released OrientDB v3.0.0 RC1 It is with great pleasure that we announce the first Release Candidate of OrientDB 3.0! Improving upon the foundations established in OrientDB 2.2, this latest release further consolidates all our strengths to establish a new standard for Multi-model Databases. As with all our releases, security is always of the utmost importance. Read More

và đến nay vẫn là bản 3.0 nhưng đã được sửa chữa lỗi và tối ưu 32 lần

OrientDB v 3.0.32 released

Tue, 06/16/2020 - 08:57

OrientDB Team has just released OrientDB v 3.0.32 Download The Latest OrientDB (v3.0.32) version:Download OrientDB Community Edition hereDownload OrientDB Enterprise Edition here If you are currently using a previous version of OrientDB, we recommend you upgrade using the link above. However, if you would like to download previous versions of OrientDB Community Edition, you may do so here: http://orientdb.com/download-previous/ Read More

Mong muốn trong tương lai OrientDB sẽ cải thiện được nhiều về hiệu năng trong việc chèn cũng như việc thực thi các truy vấn

6. Tài liệu tham khảo

[1] Document OrientDB: https://www.OrientDB.org/docs/3.0.x/
[2] Connect OrientDB-NodeJS: https://OrientDB.com/docs/3.0.x/orientjs/OrientJS.html
[3] Connect MySQL-NodeJS: https://www.npmjs.com/package/mysql
[4] Tổng quan về NoSQL:
https://toidicodedao.com/2015/09/24/nosql-co-gi-hay-ho-tong-quan-ve-nosql-phan-1/
https://quantrimang.com/co-so-du-lieu-phi-quan-he-nosql-160708
OrientDB - Wikipedia