

NoSQL:

Описание сущностей:

1.City - информация о городе.

Содержит:

- «cityId» - int, идентификатор, 4 байта.
- «cityName» - string, название города, 50*2 байта.

Итого: 104 байта.

2.Roadwork - информация о дорожной работе. Содержит:

- «workId» - int, идентификатор, 4 байта.
- «workName» - string, название города, 50*2 байта.
- «workAddress» - string, название города, 50*2 байта.
- «workDate» - string, название города, формат: xx\xx\xx = 8*2 байта.
- «type» - string, название города, 50*2 байта.

Итого: 320 байта.

Существует связь между сущностями:

HAS - City->Roadwork.

Расчет объема:

Имеется N городов и M дорожных работ.

Объем городов - $N \cdot 104$ байта.

Объем дорожных работ - $M \cdot 320$ байта.

Итого: $R1 = (N \cdot 104 + M \cdot 320)$ байта

Фактически объем:

HAS(k) - 34 байта

Города - $M \cdot 232$ байта

Дорожные работа - $N \cdot 720$ байт

Итого: $R2 = (N \cdot 232 + M \cdot 720 + k \cdot 34)$ байта

Избыточность модели: $I = (N \cdot 232 + M \cdot 720 + k \cdot 34) / (N \cdot 104 + M \cdot 320)$

Пусть имеется $M = 1000000$ и

$k=3000000$. Получаем:

Чистый объем от количества городов:

$N \cdot 104 + 1000000 \cdot 320$ (байт) =

$N \cdot 104 + 320000000$ (байт) = $N \cdot 104$ (байт)

+ 0,32 (гигабайт)

Фактический объем от количества

городов:

$$\begin{aligned} & N \cdot 104 + 1000000 \cdot 324 + 3000000 \cdot 34 \text{ (байт)} \\ &= N \cdot 104 + 324000000 + 102000000 \text{ (байт)} = \\ & N \cdot 104 \text{ (байт)} + 0,426 \text{ (гигабайт)} \end{aligned}$$

Примеры запросов:

- Create (work{...}) - добавить работу
- Match (a {_id:34})-[:HAS]-(e) Return e.workName - найти название работ по id города

SQL

Описание сущностей:

1.City - информация о городе.

Содержит:

- «cityId» - int, идентификатор, 4 байта.
- «cityName» - string, название города, 50*2 байта.

Итого: 104 байта.

2.Roadwork - информация о дорожной работе. Содержит:

- «workId» - int, идентификатор, 4 байта.

- «workName» - string, название города, 50×2 байта.
- «workAddress» - string, название города, 50×2 байта.
- «workDate» - string, название города, формат: $xx \backslash xx \backslash xx = 8 \times 2$ байта.
- «cityId» - int, идентификатор города, 4 байта.
- «type» - string, название города, 50×2 байта.

Итого: 324 байта.

Расчет объема:

Имеется N городов и M дорожных работ.

Объем городов - $N \times 104$ байта.

Объем дорожных работ - $M \times 320$ байта.

Итого: $R3 = N \times 104 + M \times 320$ байта

Фактический объем:

Объем городов - $N \times 104$ байта.

Объем дорожных работ - $M \times 324$ байта.

Итого: $R4 = N \times 104 + M \times 324$ байта

Избыточность модели: $I = (N \times 104 +$

$M \cdot 324 \text{ байта}) / (N \cdot 104 + M \cdot 320) \text{ байта}$

Пусть имеется $M = 1000000$. Получаем:

Чистый объем от количества городов:

$N \cdot 104 + 1000000 \cdot 320 \text{ (байт)} =$

$N \cdot 104 + 320000000 \text{ (байт)} = N \cdot 104 \text{ (байт)}$

$+ 0,32 \text{ (гигабайт)}$

Фактический объем от количества

городов: $N \cdot 104 + 1000000 \cdot 324 \text{ (байт)} =$

$N \cdot 104 + 324000000 \text{ (байт)} = N \cdot 104 \text{ (байт)}$

$+ 0,324 \text{ (гигабайт)}$

Примеры запросов:

- `INSERT INTO Work VALUES(...)` -
добавить работу
- `SELECT * FROM City INNER JOIN
Work ON Work.workId = City.cityId` -
найти название работ по id города

Вывод:

Исходя из сделанного анализа Neo4j занимает больше памяти по сравнению с MySQL. Параметр избыточности у Neo4j выше, чем у его конкурента. Это объясняется большим объемом памяти.

При этом все запросы на Neo4j выглядят менее емкими и более простыми в сравнении с MySQL. Мы делаем вывод в сторону Neo4j.