# NoSQL injection

**NoSQL** (*not only SQL*, *не только SQL*) — термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL. Применяется к базам данных, в которых делается попытка решить проблемы масштабируемости (*scalability*) и доступности (*availability*) за счёт атомарности (*atomicity*) и согласованности данных (*consistency*).

**Атомарная** (атом от [греч.](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B5%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) atomos – неделимое) **операция** — операция, которая либо выполняется целиком, либо не выполняется вовсе; операция, которая не может быть частично выполнена и частично не выполнена.

В зависимости от модели данных и подходов к распределённости и репликации можно выделить четыре типа хранилищ:

1. «ключ-значение» (key-value store)
2. документно-ориентированные (document store)
3. хранилища семейств колонок (column database)
4. графовые базы данных (graph database).

**Хранилище «ключ-значение»**

Хранилище «ключ-значение» является простейшим хранилищем данных, использующим ключ для доступа к значению. Примеры: MemcacheDB, Redis, Riak, Tarantool

**Хранилища семейств колонок**

В этом хранилище данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование.  
**Документно-ориентированные**

Документо-ориентированные СУБД служат для хранения иерархических структур данных. В основе документоориентированных СУБД лежат документные хранилища ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *document store*), имеющие структуру [дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29) (иногда [леса](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%29#.D0.A1.D0.B2.D1.8F.D0.B7.D0.B0.D0.BD.D0.BD.D1.8B.D0.B5_.D0.BE.D0.BF.D1.80.D0.B5.D0.B4.D0.B5.D0.BB.D0.B5.D0.BD.D0.B8.D1.8F)). [Структура дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%28%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%29) начинается с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, что позволяет даже при достаточно сложной структуре находить место (путь) искомых данных. [API](https://ru.wikipedia.org/wiki/API) для поиска позволяет находить по запросу документы и части документов. Документы могут быть организованы (сгруппированы) в *коллекции*. Пример: MongoDB

**Графовые базы данных**

[Графовые базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_%D0%BD%D0%B0_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2) применяются для задач, в которых данные имеют большое количество связей, например, [социальные сети](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), выявление мошенничества.

Однако, сколько мы бы не выигрывали в производительности, мы всегда будем проигрывать в чем-то другом. В нашем случае это возможности построения взаимоотношений между данными и проверки консистенции. В NoSQL бд нет запросов как в обычной реляционной модели, но это не делает ее менее уязвимой к инъекциям. В данном случае атаки подобного рода могут быть исполнены процедурным языком, а не декларативным, как в случае с SQL. Причем последствия могут быть куда хуже, чем в случае sql injection.  
  
Запросы в NoSQL БД поступают либо через API, либо в форматированном виде (XML, JSON, ..), либо с помощью собственного языка программирования приложения. Вредоносный код может при этом не попадать под действия фильтров первичного приложения. Например, фильтрование стандартного набора спецсимволов HTML символов:  **< > & ;** не защитит от атак на JSON API, в котором спецсимволы включают в себя еще и такие: **/ { } :**.

На данный момент существует более 225 NoSQL БД, используемых с приложениями, предоставляющими API, которые в свою очередь написаны на разных языках и предоставляют разнообразные модели взаимодействий. Каждая предоставляет различные плюшки и ограничения. Поэтому поскольку не существует какой-то определенного общего языка общения, то инъекции к разным базам данных и API могут существенно отличаться. Тестирование на проникновение NoSQL DB, требует ознакомления с каждой спецификацией каждого используемого продукта.

Атаки типа NoSQL могут быть выполнены в различных областях(участках) приложения, в отличии от традиционной SQL inj. Тогда как обычные SQL inj будут выполняться в пределах движка sql, nosql инъекции могут быть выполнены и на уровне приложения и на уровне базы данных, в зависимости от используемых API и модели взаимодействия. Типичные NoSQL атаки будут исполняться во время парсинга передаваемых данных, исполнения их или конкатенации в NoSQL api вызов.

## Тестирование:

Пример 1:

Рассмотрим инъекцию на примере MongoDB. API этой бд ожидает на ход BSON (Binary JSON). Однако, согласно документации в некоторых параметрах запроса принимается десериализованные выражения JSON, JS. Самый часто используемый вызов к API, позволяющий ввод произволного JS кода - это оператор $where.

Обычно он используется в простых фильтрах, как в SQL:

|  |
| --- |
| db.myCollection.find( { $where: "this.credits == this.debits" } ); |

Опционально при этом может быть разрешено использование JS, для задания более сложных запросов:

|  |
| --- |
| db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } ); |

И если бы у злоумышленника была бы возможность манипулировать данными без должной фильтрации, например вот так:

|  |
| --- |
| db.myCollection.find(  {  active: true, $where: function()  {  return obj.credits - obj.debits < $userInput;  }  }  );; |

Тогда, в качестве теста можно проверить наличие уязвимости, передав спецсимволы:

|  |
| --- |
| ' " \ ; { } |

И если они не экранируются, то такой ввод приведет к ошибке базы данных. В результате в данном примере, проводя аналогию с SQL Inj, можно выполнить любой SQL запрос, который разрешен данному пользователю. Но, поскольку JS - полноценный язык, то кроме исполнения запросов, можно делать и другие интересные вещи, например, если передать вот такой код:

|  |
| --- |
| "(function(){var date = new Date(); do{curDate = new Date();}while(curDate-date<10000); return Math.max();})()" |

Мы заставим базу данных выполнить функцию, результатом которой будет число, но зато полученное большой ценой - нагрузкой на CPU. И это уже будет совмещенный тип атаки NoSQL + Deny of service.

Пример 2:

Даже если ввод пользовательских данных полностью фильтруется, все равно остается еще возможность сделать инъекцию. Многие экземпляры NoSQL имеют собственные зарезервированные имена переменных, которые не зависят от языка API.

Например, в случае с Mongodb, оператор $where сам по себе - зарезервированный оператор запроса. Он должен передаваться в запросе as-is. Любая модификация вызовет ошибку в базе данных. Но, кроме этого $where это валидная переменная языка PHP. И это дает возможность атакующему внедрить код в запрос, создавая PHP переменную, с именем $where. К этому моменту на сайте mongodb есть предупреждение, о том как правильно использовать параметры, начинающиеся с $. But who cares.

И даже, если запрос вообще не предполагает пользовательского ввода, атакующий все равно может попробовать проэксплуатировать, заменяя оператор вредоносными данными:

|  |
| --- |
| db.myCollection.find( { $where: function() { return obj.credits - obj.debits < 0; } } ); |

И один из потенциальных способов присвоить данные переменной в PHP - используя уязвимость HTTP Parameter Pollution. Эта уязвимость заключается в передаче приложению одних и тех же переменных несколько раз в рамках одного запроса. При этом может сработать ситуация, когда будет выбран именно значение, нужное атакующему, а не то, которое подразумевал автор.