Sample application

The following sample app implementations are all designed to connect to the IAM database located at the TypeDB server with `0.0.0.0:1729` address. Please make sure to have TypeDB server running with the `iam` database created and both **schema** and **data** loaded. Use the Installation guide to prepare the server and the Quickstart guide to prepare the database.

The sample application has the following implementations:

- Python
- Java
- Node.js

Implementation

Java | Python | Node.js

The following Javascript code executes four simple requests to the `iam` database.

We can save it locally or clone the repository.

Make sure to install **Node.js** version **16+**, `npm`, and TypeDB Node.js driver with `npm install typedb-client`.

To run this sample application use the following command from the directory with the source code:

```
node sample.js
```

Use the source code below or the Explanation section below to explore four requests performed in the sample app.

```
const { TypeDB } = require("typedb-driver/TypeDB");
const { SessionType } = require("typedb-driver/api/connection
/TypeDBSession");
const { TransactionType } = require("typedb-driver/api/connection
/TypeDBTransaction");
const { TypeDBOptions } = require("typedb-driver/api/connection
/TypeDBOptions");

async function main() {
    console.log("IAM Sample App");

    console.log("Connecting to the server");
    const driver = await TypeDB.coreDriver("0.0.0.0:1729"); // driver
is connected to the server
    console.log("Connecting to the `iam` database");
    let k; // define counter
    let session // define session for later use
    try {
        session = await driver.session("iam", SessionType.DATA); //
session is open

        console.log("");
        console.log("Request #1: User listing");
        let transaction;
        try {
            transaction = await
session.transaction(TransactionType.READ); // READ transaction is
open
            let get_query = "match $u isa user, has full-name $n, has
email $e; get;"; // TypeQL query
            let iterator = transaction.query.get(get_query); //
Executing query
            let answers = await iterator.collect();
            let result = await Promise.all(
                answers.map(answer =>
                    [answer.map.get("n").value,
                     answer.map.get("e").value]
                )
            );
            k = 0; // reset the counter
            for(let i = 0; i < result.length; i++) {
                k++;
                console.log("User #" + k + ": " + result[i][0] + ",
has E-mail: " + result[i][1]);
            };
            console.log("Users found: " + k);
        } finally {
            transaction?.close();
        }
        console.log("");
        console.log("Request #2: Files that Kevin Morrison has access
to");
        try {
            transaction = await
session.transaction(TransactionType.READ); // READ transaction is
open
            get_query = "match $u isa user, has full-name 'Kevin
Morrison'; $p($u, $pa) isa permission; $o isa object, has path $fp;
$pa($o, $va) isa access; get $fp;";
            iterator = transaction.query.get(get_query); // Executing
query
            answers = await iterator.collect();
            result = await Promise.all(
                answers.map(answer =>
                    [answer.map.get("fp").value]
                )
            );
            k = 0; // reset the counter
            for(let i = 0; i < result.length; i++) {
                k++;
                console.log("File #" + k + ": " + result[i]);
            }
            console.log("Files found: " + k);
        } finally {
            await transaction.close();
        };

        console.log("");
        console.log("Request #3: Files that Kevin Morrison has view
access to (with inference)");
        let options = TypeDBOptions();
        options.infer = true; // set option to enable inference
        try {
            transaction = await
session.transaction(TransactionType.READ, options); // READ
transaction is open
            get_query = "match $u isa user, has full-name 'Kevin
Morrison'; $p($u, $pa) isa permission; $o isa object, has path $fp;
$pa($o, $va) isa access; $va isa action, has name 'view_file'; get
$fp; sort $fp asc; offset 0; limit 5;"
            iterator = transaction.query.get(get_query); // Executing
query
            answers = await iterator.collect();
            result = await Promise.all(
                answers.map(answer =>
                    [answer.map.get("fp").value]
                )
            );
            k = 0; // reset the counter
            for(let i = 0; i < result.length; i++) {
                k++;
                console.log("File #" + k + ": " + result[i]);
            };
            get_query = "match $u isa user, has full-name 'Kevin
Morrison'; $p($u, $pa) isa permission; $o isa object, has path $fp;
$pa($o, $va) isa access; $va isa action, has name 'view_file'; get
$fp; sort $fp asc; offset 5; limit 5;"
            iterator = transaction.query.get(get_query); // Executing
query
            answers = await iterator.collect();
            result = await Promise.all(
                answers.map(answer =>
                    [answer.map.get("fp").value]
                )
            );
            for(let i = 0; i < result.length; i++) {
                k++;
                console.log("File #" + k + ": " + result[i]);
            };
            console.log("Files found: " + k);
        } finally {
            await transaction.close();
        };

        console.log("");
        console.log("Request #4: Add a new file and a view access to
it");
        const today = new Date(Date.now());
        try {
            transaction = await
session.transaction(TransactionType.WRITE); // WRITE transaction is
open
            let filepath = "logs/" + today.toISOString() + ".log";
            let insert_query = "insert $f isa file, has path '" +
filepath + "';";
            console.log("Inserting file: " + filepath);
            transaction.query.insert(insert_query); // Executing
query
            insert_query = "match $f isa file, has path '" + filepath
+ "'; $vav isa action, has name 'view_file'; insert ($vav, $f) isa
access;";
            console.log("Adding view access to the file");
            await transaction.query.insert(insert_query); //
Executing query
            await transaction.commit(); // to persist changes, a
'write' transaction must be committed
        } finally {
            if (transaction.isOpen()) {await transaction.close()};
        };
    } finally {
        await session?.close(); // close session
        driver.close(); // close server connection
    };
};

main();
```

Explanation

List names and e-mails for all users that have them

TypeQL query used:

```
match $u isa user, has full-name $n, has email $e;
get;
```

Simple explanation: we go through all entities of `user` subtype (assigning a variable `$u` for those) that have `full-name` attribute (variable `$n` assigned for those) and `email` attribute (variable `$e`). Since we don't have an explicit `get` statement it is assumed that we get all the variables that were assigned in the query.

Note

Note that users that do not have `full-name` or `email` attributes added to them will not be shown in the results of this request. Additionally, users that have multiple full-names or emails will be mentioned more than once.

For more information, see the matching patterns explanation.

List all the files that Kevin Morrison has access to

TypeQL syntax | Java Builder syntax

TypeQL query used:

```
match $u isa user, has full-name 'Kevin Morrison'; $p($u, $pa) isa
permission;
      $o isa object, has path $fp; $pa($o, $va) isa access;
get $fp;
```

Simple explanation: we look for a `user` (variable `$u`) with attribute `full-name` of value `Kevin Morrison` assigned. Then we search for a `permission` relation (`$p`) in between this user `$u` and potential access `$pa`. Finally, we state that an `object` (`$o`) with a path `$fp` should be a part of `$pa` `access` relation, without having to specify what kind of action `$va` it should be. From all the variables requested, we only want it to return the `path` attributes (`$fp`) of any `object` that the `user` has `permission` to `access`.

Note

Note that users and files don't have a singular relation that connects them directly. According to the `iam` schema we need two relations to connect them: `permission` and `access`. Both relations must be used to make a decision of whether a user has access to a file.

List all the files Kevin has view_file access to (with inference)

TypeQL syntax | Java Builder syntax

TypeQL query used:

```
match $u isa user, has full-name 'Kevin Morrison'; $p($u, $pa) isa
permission;
      $o isa object, has path $fp; $pa($o, $va) isa access;
      $va isa action, has name 'view_file';
get $fp;
sort $fp asc; offset 0; limit 5;
```

Simple explanation: This is a similar request to the previous one. The difference is we set the type of action (`$va`) that the user has access to the `view_file`. We still get only `path` (`$fp`), but can now sort in ascending order and get it in two portions: this particular request gets the very first five entries. Later ones will get another five, starting from number six.

Note

Note that Kevin has been assigned only `modify_file` access, and the `view_file` access is being inferred by a rule. To use inference in this query we modify TypeDB options and send the modified set of options to the transaction call.

Note

To make things a bit more interesting we split this into two separate queries by using an `offset` keyword: we get the first five results and then five more results with an offset of five. To be able to do that we apply sorting of the results by `path` variable. Otherwise, we can't guarantee the results will be in the same order every time we send a request.

Insert a new file and then insert an access relation to it

First, we generate a new value for the `path` attribute of the query (stored locally in the variable called filepath), consisting of `logs/` prefix, current date and time in compact format, and `.log` ending.

The following queries are displayed in their final form, with the `path` attribute generated value.

Query #1:

TypeQL syntax | Java Builder syntax

```
insert $f isa file, has path 'logs/2023-06-30T12:04:36.351.log';
```

Simple explanation: we insert a `file` entity (instance of data) that has an attribute `path` with the value we generated before.

Query #2:

TypeQL syntax | Java Builder syntax

```
match
  $f isa file, has path 'logs/2023-06-30T12:04:36.351.log';
  $vav isa action, has name 'view_file';
insert
  ($vav, $f) isa access;
```

Simple explanation: we look for a `file` entity that has an attribute `path` with the value we generated before. And we find an `action`, that has a `name` attribute with the value of `view_file`. Then we insert an `access` relation in between the `file` and the `action`.

Note

Note that we create `file` first. If we try to set `access` to a nonexistent `file` our request will succeed (if we don't make any mistakes in the syntax of the query) but will not insert any new data (relation). After both requests are done we commit the write transaction. It is important not to forget to commit changes.