☕ ChaiSQL Experience

First impression for the ChaiSQL typing system.

Hello!

1. My name is Dmitrii ✌️
2. Do you mind if we record this section? ⏺️

Session setup

1. Small introduction ~ 5-10 min
2. Case studies ~ 40 min
3. Retrospection ~ 5 min
4. Closing ~ 5 min

~~~~~~~~~~~~~~

~ 1 hour total
~~~~~~~~~~~~~~

Let’s get started :)

Test details and setup

Could you tell a bit about yourself?

1. How would you describe your current job/position?
2. How would you describe your experience with SQL?
3. What is your favorite programming language otherwise?

ChaiSQL goals

☕ ChaiSQL is a *static type checker* for *SQL SELECT* queries.

ChaiSQL goals

☕ ChaiSQL is a ***static*** *type checker* for *SQL SELECT* queries.

Static = ran before the SQL evaluation

ChaiSQL goals

☕ ChaiSQL is a *static* ***type checker*** for *SQL SELECT* queries.

Static = ran before the SQL evaluation

Type checker = asserts that type hints confirm to evaluated types

ChaiSQL goals

☕ ChaiSQL is a *static type checker* for ***SQL SELECT*** queries.

Static = ran before the SQL evaluation

Type checker = asserts that type hints confirm to evaluated types

SQL SELECT = the basic fragment of SQL supported by ChaiSQL

ChaiSQL goals

☕ ChaiSQL is a *static type checker* for *SQL SELECT* queries.

Static = ran before the SQL evaluation

Type checker = asserts that type hints confirm to evaluated types

SQL SELECT = the basic fragment of SQL supported by ChaiSQL

Other examples:

mypy for *Python*, *flow* for *JavaScript*, *sorbet* for *Ruby*

ChaiSQL base types (1/2)

Primitive types:

```
-- ... String

    > for VARCHAR(size), TEXT(size),
      etc...

-- ... Number

    > for INT(size), FLOAT(size, d),
      etc...

-- ... Boolean

    > for BOOL(size), BOOLEAN(size),
      and conditions in WHERE clauses
```

ChaiSQL base types (2/2)

Primitive types:

```
-- ... String
```

> for VARCHAR(size), TEXT(size), etc...

```
-- ... Number
```

> for INT(size), FLOAT(size, d), etc...

```
-- ... Boolean
```

> for BOOL(size), BOOLEAN(size), and conditions in WHERE clauses

Compound types:

```
-- ... DbView
<[notation: bag|set]>
{[key]: [type], ...}
```

> For results of a query, denoted by
> <bag> - result, maybe duplicates
> <set> - result, no duplicates
> {[key]: [type], ...}, columns with
  [key] - column name
  [type] - column primitive type

All ChaiSQL commands (1/3)

```
-- @chaisql:check
```

> instructs the ChaiSQL engine to check the SQL file.

All ChaiSQL commands (2/3)

```
-- @chaisql:check
```

> instructs the ChaiSQL engine to check the SQL file.

```
-- @chaisql:newtype [Alias] = [Primitive or compound type]
```

> to provide custom type names
> with [Alias] - the new type identifier
> and [Primitive or compound type] - assigned type value, must resolve to a Primitive
> for example

```
-- @chaisql:newtype Key = Number
-- @chaisql:newtype PersonView = DbView <bag> {id: Key, name: String}
```

All ChaiSQL commands (3/3)

```
-- @chaisql:check
```

> instructs the ChaiSQL engine to check the SQL file.

```
-- @chaisql:newtype [Alias] = [Primitive or compound type]
```

> to provide custom type names
> with [Alias] - the new type identifier
> and [Primitive or compound type] - assigned type value, must resolve to a Primitive
> for example

```
-- @chaisql:newtype Key = Number
-- @chaisql:newtype PersonView = DbView <bag> {id: Key, name: String}
```

```
-- @chaisql:returns [Primitive, compound, or alias type]
```

> the type hint for the query return type
> for example

```
-- @chaisql:returns DbView <bag> {name: String}
SELECT name FROM person;
```

Case study setup

5 cases with examples of ChaiSQL.
Each case has **2 stages**:
specification & evaluation

Consistent DB schema

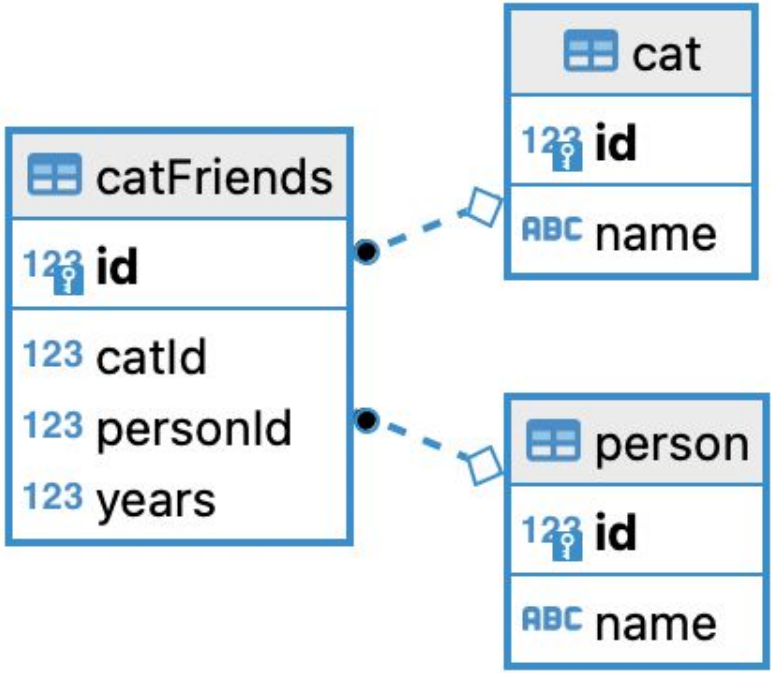
Simple running database for all cases.

1. Two entity tables:

   cat, person

2. One intersection table:

   catFriends



Case <n>/<t>: <Description>

Case setup <n>

```
-- file: case_<n>.sql

-- @chaisql:check

-- @chaisql:newtype Key = Number
-- @chaisql:newtype Name = String

-- @chaisql:returns
     DbView <bag> {id: Key, name: Name}
SELECT id, name
FROM person;
```

1. Query to be checked

```
$ chaisql      < arguments >
```

Case setup <n>

```
-- file: case_<n>.sql

-- @chaisql:check

-- @chaisql:newtype Key = Number
-- @chaisql:newtype Name = String

-- @chaisql:returns
     DbView <bag> {id: Key, name: Name}
SELECT id, name
FROM person;
```

```
$ chaisql      < arguments >
```

2. CLI interaction with ChaiSQL

Case setup <n>

```
-- file: case_<n>.sql

-- @chaisql:check

-- @chaisql:newtype Key = Number
-- @chaisql:newtype Name = String

-- @chaisql:returns
     DbView <bag> {id: Key, name: Name}
SELECT id, name
FROM person;
```

```
$ chaisql      < arguments >

<

 ℹ️ ChaiSQL checking output

 ...

>
```

3. ChaiSQL type checking output:

<u>OK</u> if everything went well.

Otherwise, an <u>error description</u>

Before we begin!

1. There are **no right or wrong** answers
2. **All your insights** are very **valuable**
3. The **handout** should contain **background information** for all cases
4. Don’t hesitate to **ask questions if you feel like**

Let’s meet the cases

Ready when you are!

Case 1/5: Selections

Case 1

```
-- file: case_1.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <bag>
          {id: Number, name: String}
SELECT *
FROM cat;
```

```
$ chaisql      --check   case_1.sql \
               --db      test.db \
               --verbose
```

Case 1

```
-- file: case_1.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <bag>
          {id: Number, name: String}
SELECT *
FROM cat;
```

```
$ chaisql      --check   case_1.sql \
               --db      test.db \
               --verbose
```

✅ **All good!**

👌 Type checks were successful.

Case 2/5: More selections

Case 2

```
-- file: case_2.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <bag>
          {name: String}
SELECT *
FROM person;
```

```
$ chaisql      --check   case_2.sql \
               --db      test.db \
               --verbose
```

Case 2

```sql
-- file: case_2.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <bag>
          {name: String}
SELECT *
FROM person;
```

```
$ chaisql      --check   case_2.sql \
               --db      test.db \
               --verbose
```

⛔ **Type errors found:**

```
-- @chaisql:returns
     DbView <bag>
          {name: String}
          /\~~~~~~~~~~~~
          Expected id: Number, ...
SELECT * ...
```

☹️ Type checks have failed.

Case 3/5: Set notations

Case 3

```sql
-- file: case_3.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <set> {catId: Number}
SELECT DISTINCT catId
FROM catFriends;
```

```
$ chaisql      --check   case_3.sql \
               --db      test.db \
               --verbose
```

Case 3

```sql
-- file: case_3.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <set> {catId: Number}
SELECT DISTINCT catId
FROM catFriends;
```

```
$ chaisql      --check   case_3.sql \
               --db      test.db \
               --verbose
```

✅ **All good!**

👌 Type checks were successful.

Case 4/5: Subqueries

Case 4

```sql
-- file: case_4.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <set> {id: Number, name: String}
SELECT DISTINCT p.id, p.name
FROM person AS p
WHERE p.name IN (
   -- @chaisql:returns
          DbView <set> {personId: Number}
   SELECT DISTINCT personId
   FROM catFriends
);
```

```
$ chaisql      --check   case_4.sql \
               --db      test.db \
               --verbose
```

Case 4

```sql
-- file: case_4.sql

-- @chaisql:check

-- @chaisql:returns
     DbView <set> {id: Number, name: String}
SELECT DISTINCT p.id, p.name
FROM person AS p
WHERE p.name IN (
   -- @chaisql:returns
          DbView <set> {personId: Number}
   SELECT DISTINCT personId
   FROM catFriends
);
```

```
$ chaisql      --check   case_4.sql \
               --db      test.db \
               --verbose
```

⛔ **Type errors found:**

```
WHERE p.name IN (
      /\~~~~~~~
      > name: String does not satisfy IN.
      > String !== Number in name: Sting.
      > IN expects equal argument types.
   -- @chaisql:returns
          DbView <set> {personId: Number}
   SELECT DISTINCT personId ...
```

☹️ Type checks have failed.

Case 5/5: Joins, aliases

Case 5

```sql
-- file: case_5.sql

-- @chaisql:check
-- @chaisql:newtype Duration = Number
-- @chaisql:returns
     DbView <set>
          {catName: String,
           personName: String,
           years: Duration}
SELECT DISTINCT c.name AS catName,
     p.name AS personName, cf.years
FROM cat AS c, person AS p, catFriends AS cf
WHERE c.id = cf.catId
AND   p.id = cf.personId;
```

```
$ chaisql      --check   case_5.sql \
               --db      test.db \
               --verbose
```

Case 5

```
-- file: case_5.sql

-- @chaisql:check
-- @chaisql:newtype Duration = Number
-- @chaisql:returns
     DbView <set>
          {catName: String,
           personName: String,
           years: Duration}
SELECT DISTINCT c.name AS catName,
     p.name AS personName, cf.years
FROM cat AS c, person AS p, catFriends AS cf
WHERE c.id = cf.catId
AND   p.id = cf.personId;
```

```
$ chaisql      --check   case_5.sql \
               --db      test.db \
               --verbose
```

✅ **All good!**

👌 Type checks were successful.

Case study retrospection

Thank you very much for participating 🤝

For more contact → d.orlov@student.tue.nl

If you would like to get any updates, let me know :)