Сервис QueriesHelper

`QueriesHelper` — локальный сервис-помощник для выполнения запросов к базе данных в сервисах, предоставляя единый интерфейс для вызова хранимых функций, выполнения SQL и работы с транзакциями.

1. Подключение в вашем сервисе

Ваш сервис должен наследоваться от `ServiceRequire` и указать `QueriesHelper` в списке зависимостей.

```
import { ServiceRequire } from '@morphcluster/core'
import { QueriesHelper } from '@morphcluster/carabi'

export default class MyService extends ServiceRequire {
  constructor(host, config) {
    super(host, config)
    this.requirements = ['QueriesHelper']
  }

  async start(log) {
    await super.start(log)
    // this.QueriesHelper уже доступен
  }
}
```

После вызова `super.start(log)` свойство `this.QueriesHelper` будет содержать готовый к использованию экземпляр.

2. Конфигурация

`QueriesHelper` получает тип подключения из реестра через `RegistryHelper`. Ключ конфигурации: `queries.type`.
Возможные значения:

- `"ora"` — работа через `OraQueries` (Oracle-совместимый адаптер поверх PostgreSQL).
- `"pg"` — прямое подключение через `PgQueryPool` (на данный момент не используется; при попытке использования в `_queryExec` выбрасывается ошибка).

3. Основные методы запросов

3.1. `queryRaw(queryName, params, count, offset, options)`

Базовый метод выполнения хранимой функции или процедуры.
Формат имени: `'PKG.FUN'` (пакет/схема и имя функции через точку). Для Postgres пакет заменен схемой.

Параметры:

- `queryName` (string) – полное имя функции, например `'csp_mainmenu.get_dashboard'`.
- `params` (object) – объект с параметрами, ключи соответствуют именам параметров функции (без префикса `:`).
- `count` (number, по умолчанию 1) – количество строк для извлечения из курсора (если возвращается курсор).
- `offset` (number, по умолчанию 0) – смещение для курсора.
- `options` (object) – дополнительные настройки (см. раздел «Опции запросов»).

Возвращает: `Promise<Array<{ paramName: string, type: string, value: any }>>`
Массив объектов, описывающих все выходные параметры функции. Для курсоров `value` будет содержать структуру `{ columns: Array<[string, string]>, list: Array<Array<any>> }`.

Пример:

```
const result = await this.QueriesHelper.queryRaw(
  'csp_mainmenu.get_items',
  { user_id: 42, role_id: 1 },
  10,
  0,
  { log, session }
);
// result[0] может быть { paramName: 'RESULT', type: 'CURSOR', value: {...} }
```

3.2. `query(queryName, params, count, offset, options)`

Обёртка над `queryRaw`, которая:

- Преобразует курсоры из сырого формата в массив объектов (через `convertCursor`).
- Упаковывает все выходные параметры в объект, где ключ — `paramName`, а значение — преобразованное значение.

Возвращает: `Promise<Object>`
Объект вида `{ PARAM1: value1, PARAM2: value2 }`. Курсоры превращаются в массив объектов, где ключи — имена колонок.

Пример:

```
const data = await this.QueriesHelper.query(
  'csp_mainmenu.get_user_info',
  { user_id: 100 },
  1, 0,
  { log, session }
);
// data.USER_INFO = [ { NAME: 'John', AGE: 30 } ]
// data.STATUS   = 'OK'
```

3.3. `select(queryName, params, count, offset, options)`

Упрощённый метод, когда ожидается ровно одно выходное значение. Фактически возвращает первое свойство из результата `query`.

Возвращает: значение первого выходного параметра (после преобразования курсора).

Пример:

```
const itemCount = await this.QueriesHelper.select(
  'csp_mainmenu.count_items',
  { category: 'books' },
  1, 0,
  { log, session }
);
// itemCount = 12 (если единственный out-параметр — число)
```

3.4. `selectRow(queryName, params, options)`

Используется, когда ожидается ровно одна строка из курсора. Возвращает первый элемент массива-результата или `null`.

Параметры:

- `queryName`, `params`, `options` (count и offset не принимаются – под капотом используется `count=1, offset=0`).

Возвращает: объект строки или `null`.

Пример:

```
const user = await this.QueriesHelper.selectRow(
  'csp_mainmenu.get_user_by_id',
  { user_id: 55 },
  { log, session }
```

```
);
// user = { NAME: 'Alice', EMAIL: 'alice@example.com' } или null, если не найден
```

3.5. `querySql(log, SQL, rawParams, options)`

Выполняет произвольный SQL-запрос (не хранимую функцию). Поддерживает как простые запросы, так и выполнение в транзакции.

Параметры:

- `log` (Logger) – обязательно.
- `SQL` (string) – текст запроса с плейсхолдерами в нотации Oracle (`:param_name`).
- `rawParams` (Array) – массив объектов параметров: `{ name: 'PARAM', type: 'number', value: 123 }`.
- `options` (object) – `{ userId, session, trxId, count, offset }`.

Возвращает: `Promise<Array<{ paramName, type, value }>>` (как `queryRaw`, но только для SQL).

Пример:

```
const result = await this.QueriesHelper.querySql(
  log,
  `UPDATE users SET name = :NEW_NAME WHERE id = :ID`,
  [
    { name: 'NEW_NAME', type: 'varchar2', value: 'Bob' },
    { name: 'ID', type: 'number', value: 100 }
  ],
  { session, trxId: currentTrx } // если нужно в транзакции
);
```

3.6. `querySqlCursor(log, SQL, rawParams, options)`

То же, что `querySql`, но ожидает, что единственный выходной параметр — курсор. Сразу преобразует его через `convertCursor` и возвращает массив объектов (строк).

Возвращает: массив объектов (строк курсора).

Пример:

```
const rows = await this.QueriesHelper.querySqlCursor(
  log,
  `SELECT id, name FROM users WHERE role = :ROLE`,
  [{ name: 'ROLE', type: 'varchar2', value: 'admin' }],
  { log, session }
```

```
);
// rows = [ { ID: 1, NAME: 'Alice' }, { ID: 2, NAME: 'Bob' } ]
```

3.7. `convertCursor(cursor)`

Вспомогательный метод, который вы можете использовать отдельно, если получили сырой курсор. Преобразует колонки и значения в массив объектов, попутно парся числа.

Сигнатура:

```
convertCursor(rawCursor: { columns: Array<[string, string]>, list: Array<Array<any>> }): Array<Object>
```

4. Опции запросов

Почти все методы принимают объект `options` со следующими необязательными полями:

- `log` — экземпляр `Logger` (обязателен для методов `querySql`, `querySqlCursor`).
- `session` — объект сессии текущего пользователя (обычно `{ userId, token, isAdmin }`).
- `userId` — числовой ID пользователя; если передан, он будет добавлен в `session.userId`.
- `trxId` — ID открытой транзакции, если запрос нужно выполнить в её контексте.
- `count`, `offset` — для курсоров, если не переданы отдельными аргументами (в `querySql` они берутся из options).

5. Работа с транзакциями

5.1. `transactionWrap({ log, callback, trxId, session })`

Выполняет переданную функцию в рамках транзакции. Если `trxId` не указан, создаёт новую транзакцию, после успешного выполнения коммитит, при ошибке — откатывает.

Параметры:

- `callback` (async function) — принимает `trxId` и выполняет запросы.
- `trxId` — можно передать существующую транзакцию, тогда коммит/откат не управляется автоматически.
- `session` — сессия для создания транзакции.
- `log` — логгер.

Пример:

```
await this.QueriesHelper.transactionWrap({
  log,
  session,
  callback: async (trxId) => {
```

```
    // Передаём trxId в опции всех запросов
    await this.QueriesHelper.querySql(log,
      `INSERT INTO audit (user_id, action) VALUES (:UID, :ACT)`,
      [
        { name: 'UID', type: 'number', value: session.userId },
        { name: 'ACT', type: 'varchar2', value: 'update' }
      ],
      { trxId, session }
    );
    // ещё запросы...
  }
});
```

Если не передать `trxId`, транзакция будет создана и автоматически завершена.
Если вы передали внешний `trxId` (например, из длинной транзакции), обёртка не выполняет commit/rollback — управление остаётся за вами.

5.2. Прямое управление транзакциями

Вы можете самостоятельно создавать, коммитить и откатывать транзакции через сервис `OraLongTransactions` (доступен как `this.QueriesHelper.OraLongTransactions`). Однако обычно удобнее использовать `transactionWrap`.

6. Параметры запросов

Именованные параметры передаются в виде массива объектов:

```
interface QueryParam {
  name: string;   // имя параметра (без двоеточия)
  type: string;   // тип: 'varchar2', 'number', 'numeric', 'json' и т.д.
  value: any;     // значение
}
```

В тексте SQL плейсхолдеры указываются с двоеточием: `:USER_ID`. `QueriesHelper` сам преобразует их в позиционные (`$1`, `$2`, ...) в зависимости от типа БД.

Для вызова хранимых функций (`query`, `queryRaw`) параметры передаются объектом, где ключи соответствуют именам параметров функции (без префикса `p_` или других соглашений – смотрите документацию к вашим функциям). Внутренний механизм сам сопоставит их с аргументами функции.

7. Типы данных и преобразования

- **NUMBER** — PostgreSQL-значения числовых типов автоматически преобразуются в `float` при конвертации курсора (`convertCursor`).
- **DATE / TIMESTAMP** — возвращаются как строки в формате ISO (зависит от адаптера OraQueries).
- **VARCHAR2** — строки.
- **CURSOR** — преобразуется в массив объектов.

Если вы используете метод `queryRaw`, вы получаете сырые значения без дополнительных преобразований чисел.

8. Обработка ошибок

При ошибке выполнения запроса выбрасывается исключение. В него добавляются поля `query` и `queryParams` для упрощения отладки. Также ошибка автоматически логируется через `log.writeExceptionOnly(e)`.

Рекомендуется оборачивать вызовы в try/catch и при необходимости выбрасывать `ComplexError` для клиентов.

9. Полный пример сервиса

```
import { ServiceRequire } from '@morphcluster/core'
import { QueriesHelper } from '@morphcluster/carabi'

export default class ReportService extends ServiceRequire {
  constructor(host, config) {
    super(host, config)
    this.requirements = ['QueriesHelper']
  }

  async start(log) {
    await super.start(log)
    // Можно выполнить начальную загрузку
  }

  async getDailyReport({ date, session }, ws, log) {
    const rows = await this.QueriesHelper.querySqlCursor(log,
      `SELECT product, sum(amount) as total
       FROM sales
       WHERE sale_date = :SALE_DATE
       GROUP BY product`,
      [{ name: 'SALE_DATE', type: 'date', value: date }],
      { session, count: 1000 }
```

```
    );
    return rows;
  }

  async updateProductStock({ productId, quantity }, ws, log) {
    await this.QueriesHelper.transactionWrap({
      log,
      session: ws.session,
      callback: async (trxId) => {
        await this.QueriesHelper.querySql(log,
          `UPDATE products SET stock = stock - :QTY WHERE id = :ID`,
          [
            { name: 'QTY', type: 'number', value: quantity },
            { name: 'ID', type: 'number', value: productId }
          ],
          { trxId }
        );
        // Дополнительные действия...
      }
    });
    return { success: true };
  }
}
```

Revision #3
Created 22 June 2026 18:56:28 by Admin
Updated 26 June 2026 15:00:35 by Admin