Базы данных


PostgreSQL

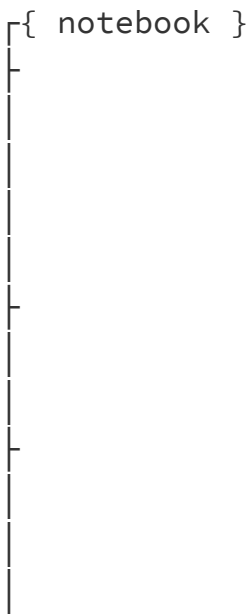
1. Устанавливаем PostgreSQL

2. Подключаемся:
   - psql
   - DataGrip
   - pgAdmin

База данных для приложения "Заметки"

Структура базы данных

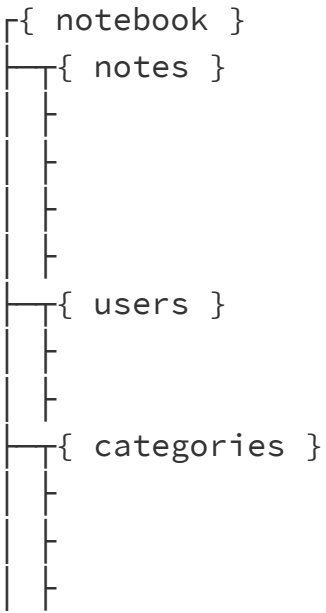
database



database

```
CREATE DATABASE notebook;
```

tables



tables

```
CREATE TABLE notes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    text TEXT NOT NULL,
    owner_id INTEGER
);
```

tables

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,

    name VARCHAR(255) NOT NULL
);

CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    best_note_id INTEGER
);
```

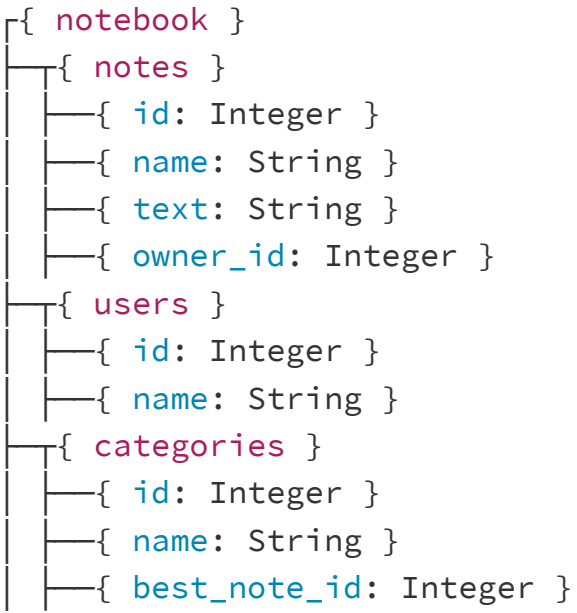
tables

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,

    name VARCHAR(255) NOT NULL
);

CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    best_note_id INTEGER
);
```

PRIMARY KEY

- уникальность
- создается индекс
- ограничение NOT NULL

fields


{ notebook }
{ notes }
{ id: Integer }
{ name: String }
{ text: String }
{ owner_id: Integer }
{ users }
{ id: Integer }
{ name: String }
{ categories }
{ id: Integer }
{ name: String }
{ best_note_id: Integer }

Типы данных

Строковые

VARCHAR(4)	"abc"
TEXT	"abcdef"

Числовые. Целые

SMALLINT	$[-2^{15}, 2^{15} - 1]$
INTEGER	$[-2^{31}, 2^{31} - 1]$
BIGINT	$[-2^{63}, 2^{63} - 1]$

Числовые. С плавающей точкой

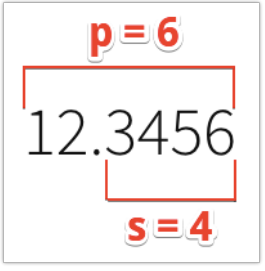
REAL	$[10^{-37}, 10^{37}]$
DOUBLE PRECISION	$[10^{-307}, 10^{308}]$

Числовые. С произвольной точностью

NUMERIC [(p[,s])]

DECIMAL [(p[,s])]

1 <= p <= 1000, 0 <= s <= p



Числовые. Последовательные

SMALLSERIAL	$[1, 2^{15} - 1]$
SERIAL	$[1, 2^{31} - 1]$
BIGSERIAL	$[1, 2^{63} - 1]$

Числовые. Последовательные

```
CREATE TABLE users (
    id SERIAL
);
```

```
CREATE SEQUENCE users_id_seq;
CREATE TABLE users (
    id integer NOT NULL DEFAULT nextval('users_id_seq')
);
ALTER SEQUENCE users_id_seq OWNED BY users.id;
```

Логический тип. BOOLEAN

TRUE	FALSE
't'	'f'
'true'	'false'
'y'	'n'
'yes'	'no'
'on'	'off'
'1'	'0'

Даты

timestamp [without time zone]	8 bytes
timestamp with time zone	8 bytes
date	4 bytes

Даты

time [without time zone]	8 bytes
time with time zone	12 bytes
interval	16 bytes

Даты. Примеры

timestamp	'2004-10-19 10:23:54'
timestamp with time zone	'2004-10-19 10:23:54+02'
date	'2018-04-03'
time	'04:05:06.789'
time with time zone	'04:05:06.789-3'
interval	'1 12:59:10'

Другие

Массивы

- integer[]
- integer[][]
- text[][]

Массивы

```
'{ значение1 разделитель значение2 разделитель ... }'
```

```
integer[][] -> '{{1,2,3},{4,5,6},{7,8,9}}'
```

```
text[] -> '{"apple", "orange", "cheese"}'
```

JSON(B) vs TEXT

- получение конкретного значения
- быстрый поиск
- много функций и операторов
- B → BINARY

JSON(B)

```
'{ "bar": "baz", "number": 7, "active": false }'
```

Модификация

```
CREATE TABLE notes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,

    text VARCHAR(255)
);

ALTER TABLE notes ADD COLUMN owner_id INTEGER;
ALTER TABLE notes DROP COLUMN owner_id;

ALTER TABLE notes ALTER COLUMN text TYPE TEXT;
ALTER TABLE notes ALTER COLUMN text SET NOT NULL;

ALTER TABLE notes RENAME TO personal_notes;

DROP TABLE notes;
```

Миграции

Механизм модификации структуры и данных в БД. Очень похоже на систему контроля версий. Если что-то пошло не так, то миграции можно откатить.

CRUD

от англ. create, read, update, delete — «создать, прочесть, обновить, удалить»

Create

```
INSERT INTO notes
(id, name, text)
VALUES
(1, 'Books', 'Books to read');
```

Read

```
SELECT id, name AS title, text FROM notes;
```

```
+----+-------+---------------+
| id | title |     text      |
+----+-------+---------------+
|  1 | Books | Books to read |
+----+-------+---------------+
```

Read

```
SELECT * FROM notes;
```

```
+----+-------+---------------+----------+
| id | name  |     text      | owner_id |
+----+-------+---------------+----------+
|  1 | Books | Books to read |   NULL   |
+----+-------+---------------+----------+
```

Auto increment

```
INSERT INTO notes (name, text)
VALUES ('Films', 'Films to watch');

ERROR:  duplicate key value violates
        unique constraint "notes_pkey"
DETAIL:  Key (id)=(1) already exists.
```

Auto increment

```
CREATE TABLE notes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    text TEXT NOT NULL,
    owner_id INTEGER
);

INSERT INTO notes (name, text)
VALUES ('Films', 'Films to watch'); // id сгенерирует последовательность

SELECT * FROM notes;
```

```
+----+-------+----------------+----------+
| id | name  |      text      | owner_id |
+----+-------+----------------+----------+
|  1 | Books | Books to read  |   NULL   |
|  2 | Films | Films to watch |   NULL   |
+----+-------+----------------+----------+
```

Read. Where

```
SELECT id, name, text FROM notes
WHERE name = 'Films';
```

```
+----+-------+----------------+
| id | name  |      text      |
+----+-------+----------------+
|  2 | Films | Films to watch |
+----+-------+----------------+
```

Read. Where + AS

```
SELECT id, name AS title, text FROM notes
WHERE title = 'Films';

ERROR:  column "title" does not exist
```

Read. Order

```
SELECT name FROM notes;
```

```
+----------+
|   name   |
+----------+
|  Books   |
|  Films   |
|  Music   |
|  Rules   |
| Markdown |
+----------+
```

```
SELECT name FROM notes
ORDER BY name DESC;
```

```
+----------+
|   name   |
+----------+
|  Rules   |
|  Music   |
| Markdown |
|  Films   |
|  Books   |
+----------+
```

Read. Order, offset, limit

```
SELECT name FROM notes
ORDER BY name DESC;
```

```
+----------+
|   name   |
+----------+
|  Rules   |
|  Music   |
| Markdown |
|  Films   |
|  Books   |
+----------+
```

```
SELECT name FROM notes
ORDER BY name DESC
OFFSET 2
LIMIT 2;
```

```
+----------+
|   name   |
+----------+
| Markdown |
|  Films   |
+----------+
```

Read. Count

```
SELECT count(*) FROM notes;
```

```
+-------+
| count |
+-------+
|   5   |
+-------+
```

Read. Group by

```
SELECT name, owner_id
FROM notes;
```

```
+----------+----------+
|   name   | owner_id |
+----------+----------+
|  Books   |     3    |
|  Films   |     1    |
|  Music   |     2    |
|  Rules   |   NULL   |
| Markdown |     1    |
+----------+----------+
```

```
SELECT owner_id, count(*)
FROM notes
GROUP BY owner_id;
```

```
+----------+-------+
| owner_id | count |
+----------+-------+
|    1     |   2   |
|    2     |   1   |
|    3     |   1   |
|   NULL   |   1   |
+----------+-------+
```

Подзапросы

name	owner_id
Books	3
Films	1
Music	2
Rules	NULL
Markdown	4

id	name
1	Антон
2	Михаил
3	Олег
4	Андрей

Подзапросы

```
SELECT notes.name
FROM notes
WHERE owner_id IN (
    SELECT id
    FROM users
    WHERE users.name LIKE 'A%'
);
```

```
+----------+
|   name   |
+----------+
|  Films   |
| Markdown |
+----------+
```

Update

```
UPDATE notes
SET text = 'My favorite books to read', owner_id = 4
WHERE id = 1;
```

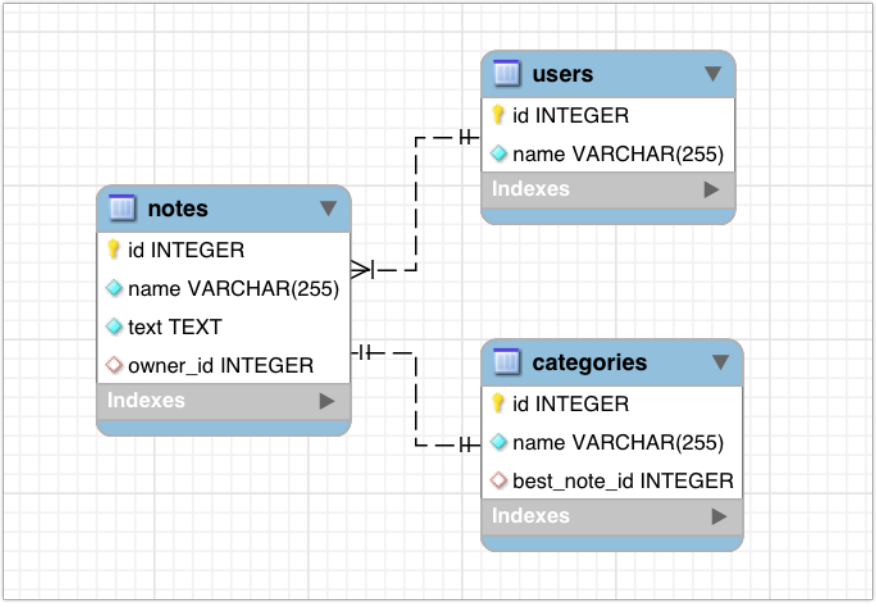
Delete

```
DELETE FROM notes
WHERE id = 1;
```

Транзакции

```
BEGIN;

UPDATE users SET account = account - 10000
WHERE id = 3;

UPDATE users SET account = account + 10000
WHERE id = 4;

{ COMMIT | ROLLBACK };
```

JOIN


users
id INTEGER
name VARCHAR(255)
Indexes
notes
id INTEGER
name VARCHAR(255)
text TEXT
owner_id INTEGER
Indexes
categories
id INTEGER
name VARCHAR(255)
best_note_id INTEGER
Indexes

Внешние ключи

FOREIGN KEY

```
CREATE TABLE notes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    text TEXT,
    owner_id INTEGER,
    CONSTRAINT fk_notes_users FOREIGN KEY (owner_id)
        REFERENCES users (id)
);
```

FOREIGN KEY

```
CREATE TABLE notes (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    text TEXT,
    owner_id INTEGER REFERENCES users (id)
);
```

FOREIGN KEY

```
ALTER TABLE notes
ADD CONSTRAINT fk_notes_users
FOREIGN KEY (owner_id)
REFERENCES users (id);
```

Виды JOIN

- INNER
- LEFT OUTER
- RIGHT OUTER
- FULL OUTER
- CROSS

Данные

```
  users       |            notes                |
              |                                 |
id   name     |     id    name     owner_id     |
--   ------   |     --    -----    --------     |
 1   Олег     |     1     Books       1         |
 2   Сергей   |     2     Films       1         |
 3   Михаил   |     3     Music       2         |
              |     4     Rules      NULL       |
              |     5    Markdown    NULL       |
```

INNER JOIN

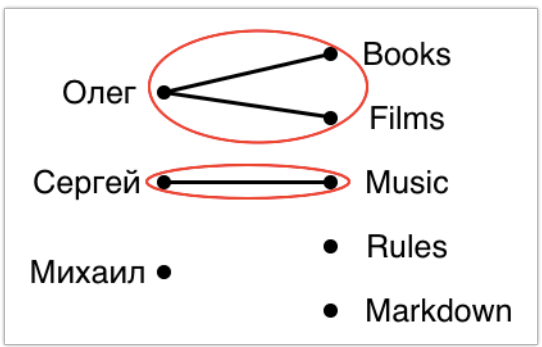
```
SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users
INNER JOIN notes
ON users.id = notes.owner_id;
```

```
  users       |            notes                |
              |                                 |
id   name     |     id    name     owner_id     |
--   ------   |     --    -----    --------     |
 1   Олег     |     1     Books       1         |
 2   Сергей   |     2     Films       1         |
 3   Михаил   |     3     Music       2         |
              |     4     Rules     NULL        |
              |     5    Markdown   NULL        |
```

users		notes		
id	name	id	name	owner_id
1	Олег	1	Books	1
2	Сергей	2	Films	1
3	Михаил	3	Music	2
		4	Rules	NULL
		5	Markdown	NULL

INNER JOIN

user_id	user_name	note_name
1	Олег	Books
1	Олег	Films
2	Сергей	Music



OUTER JOIN

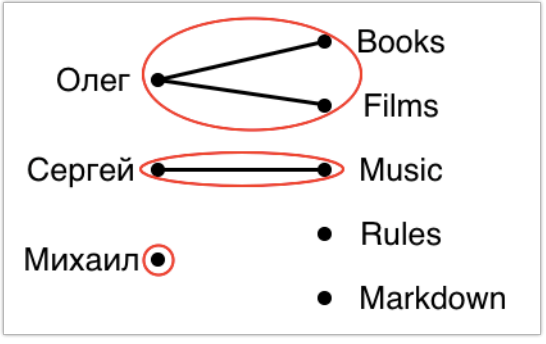
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT JOIN

```
SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users
LEFT JOIN notes
ON users.id = notes.owner_id;
```

LEFT JOIN

user_id	user_name	note_name
1	Олег	Books
1	Олег	Films
2	Сергей	Music
3	Михаил	NULL

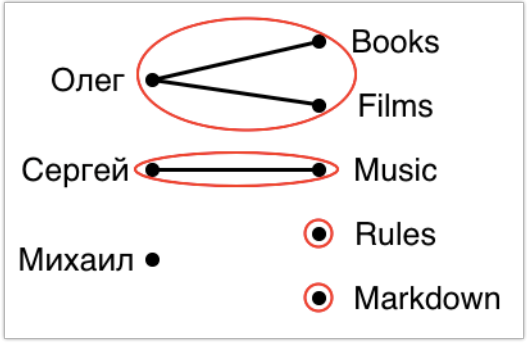


RIGHT JOIN

```
SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users
RIGHT JOIN notes
ON users.id = notes.owner_id;
```

RIGHT JOIN

user_id	user_name	note_name
1	Олег	Books
1	Олег	Films
2	Сергей	Music
NULL	NULL	Rules
NULL	NULL	Markdown



FULL JOIN

```
SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users
FULL JOIN notes
ON users.id = notes.owner_id;
```

FULL JOIN

user_id	user_name	note_name
1	Олег	Books
1	Олег	Films
2	Сергей	Music
3	Михаил	NULL
NULL	NULL	Rules
NULL	NULL	Markdown

CROSS JOIN

```
SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users
CROSS JOIN notes;

SELECT users.id AS user_id,
       users.name AS user_name,
       notes.name AS note_name
FROM users, notes;
```

CROSS JOIN

user_id	user_name	note_name
1	Олег	Books
1	Олег	Films
1	Олег	Music
1	Олег	Rules
1	Олег	Markdown
2	Сергей	Books
2	Сергей	Films
...	...	...

Индексы

Выбираем самые важные колонки и переносим их:

- в оптимальную для поиска структуру (например, дерево или хэш-таблица)
- в оперативную память (для более быстрого доступа)

Создание индекса

```
CREATE INDEX user_name_idx ON users (name);
```

Индексы по выражениям

```
CREATE INDEX users_lower_name ON users (lower(name));

-- можно сделать индекс на значение поля JSON
CREATE INDEX users_data ON users (data ->> 'field');
```

Составной индекс

```
SELECT * FROM notes
WHERE name = 'Books' AND owner_id = 1;

CREATE INDEX users_name_owner_id ON notes (name, owner_id);
```

Пример: индекс на тройку (x,y,z)

Будет использоваться при таких запросах:

- x = 1
- x = 1 AND y = 2
- x = 1 AND y = 2 AND z = 3

Но **не будет** использоваться при таких:

- y = 2
- z = 3
- y = 2 AND z = 3

Уникальный индекс

```
SELECT * FROM notes WHERE name = 'Unique book';

CREATE UNIQUE INDEX notes_name_idx ON notes (name);
```

Нюанс: `NULL != NULL`

Плюсы индексов

- позволяют фильтровать и сортировать гораздо быстрее
- позволяют гарантировать уникальность значений в колонке

Минусы индексов

- значения хранятся и в таблице, и в индексе, поэтому операции изменения данных будут работать медленнее из-за необходимости обновления индекса
- занимает место в оперативной памяти, которая не бесконечна, поэтому нужно помещать в индексы только самые необходимые колонки
- чтобы эффективно использовать индексы, нужно хорошо знать структуру базы и иметь представление о том, какие запросы она будет обрабатывать

Relax 

ORM

Object-Relational Mapping


Sequelize

Установка

```
npm install pg

npm install sequelize

npm install sequelize-typescript

npm install reflect-metadata
```

Подключение к DB

```
import { Sequelize, SequelizeOptions } from 'sequelize-typescript';



const sequelizeOptions: SequelizeOptions = {
    // connection
    host: 'localhost',
    port: 5432,
    username: 'user',
    password: 'pass',
    database: 'dbname',

    // db options
    dialect: 'postgres' // 'mysql', 'sqlite', 'mariadb', 'mssql'
};

const sequelize = new Sequelize(sequelizeOptions);
```

Модели

Модели. Объявление

```
import { Model, Table } from 'sequelize-typescript';

@Table
class Note extends Model<Note> {}
```

Модели. Атрибуты

```
@Table
class Note extends Model<Note> {

    @Column(DataType.INTEGER)
    id: number;

}
```

Именование

```
@Table
class Note extends Model<Note> {

    @Column({
        type: DataType.INTEGER,
        field: 'owner_id'
    })
    ownerId: number;

}
```

Типы данных

PostgreSQL ⇒ Sequelize

Строковые

PostgreSQL	Sequelize
VARCHAR(255)	STRING(255)
TEXT	TEXT

Числовые

PostgreSQL	Sequelize
INTEGER	INTEGER
BIGINT	BIGINT
REAL	REAL
DOUBLE PRECISION	DOUBLE
DECIMAL	DECIMAL

Даты

PostgreSQL	Sequelize
TIMESTAMP WITH TIME ZONE	DATE / NOW
DATE	DATEONLY
TIME	TIME

Другие

PostgreSQL	Sequelize
ARRAY	ARRAY
JSON	JSON
JSONB	JSONB

Data types

Модели. Атрибуты

```
@Table
class Note extends Model<Note> {
    @AutoIncrement
    @PrimaryKey
    @Column(DataType.INTEGER)
    id: number;

    @AllowNull(false)
    @Column(DataType.STRING)
    name: string;
}
```

Модели. Getters & setters

```
@Table
class Note extends Model<Note> {

    @Column(DataType.STRING)
    get name(): string {
        return 'It is ' + this.getDataValue('name');
    }

    set name(value: string) {
        const text = value.replace(/\*(\w+)\*/g, '<b>$1</b>');
        this.setDataValue('name', text);
    }
}
```

```
'films to *watch*' -> 'films to <b>watch</b>'
```

Модели. Валидаторы

```
@Table
class Note extends Model<Note> {
    @Is('Length', (value) => {
        if (value.length > 20) {
            throw new Error(`Name is too long!`);
        }
    })
    @Column(DataType.STRING)
    name: string;
}
```

Модели. Валидаторы

```
@Table
class Note extends Model<Note> {
    @Length({ max: 20, min: 5 })
    @Column(DataType.STRING)
    name: string;
}
```

Модели. Валидаторы

```
@Table
class User extends Model<User> {
    @Contains('@yandex.ru')
    @IsEmail
    @Column(DataType.STRING)
    email: string;
}
```

Подробнее

Модели. Конфигурация таблиц

```
@Table({
    timestamps: false, // don't add 'created_at', 'updated_at'
    paranoid: true,   // add 'deleted_at'

    tableName: 'notes'
})
class Note extends Model<Note> {}
```

Импорт моделей

```
const sequelize = new Sequelize({
    ...
    models: ['/tables']
});

sequelize.addModels(['/tables']);
sequelize.addModels([User, Category, Note]);
sequelize.addModels([
    '/tables/user.ts',
    '/tables/note.ts',
    '/tables/category.ts'
]);
```

Создание таблиц

```
await sequelize.sync({ force: true }); // все таблицы

await Note.sync({ force: true }); // только конкретную
```

Удаление таблицы

```
await Note.drop();
```

CRUD

Create

```
await Note.create({
    name: 'Books',
    text: 'Books to read'
});
```

Create. bulk

```
await Note.bulkCreate([
    {
        name: 'Books',
        text: 'Books to read',
        ownerId: 3
    },
    {
        name: 'Films',
        text: 'Films to watch',
        ownerId: 1
    }
]);
```

Read

```
const note = await Note.findOne({
    where: {
        name: 'Films'
    }
});

const text = note.text;
```

Read. Projection

```
const note = await Note.findOne({
    where: {
        name: 'Films'
    },
    attributes: ['id', 'text', ['name', 'title']]
});
```

Read. Операторы

```
import { Op } from 'sequelize';

const userModel = await User.findOne({
    where: {
        [Op.or]: [
            { name: { [Op.like]: 'A%' } },
            { saveDate: { [Op.gt]: '2018-04-09 13:25:13' } }
        ]
    }
});
```

Подробнее

Read. Поиск по id

```
const note = await Note.findOne({
    where: { id: 23 }
});

const note = await Note.findByPk(23);
```

Read. Все записи

```
const notes = await Note.findAll({
    where: {
        ownerId: 1
    },
    attributes: ['id', 'name']
});
```

Read. Order, offset, limit

```
const notes = await Note.findAll({
    order: [
        ['name', 'DESC']
    ],
    offset: 2,
    limit: 2
});
```

Read. Count

```
const count = await Note.count({
    where: {
        ownerId: 1
    }
});
```

Read. Group by

```
const notesData = await Note.findAll({
    attributes: [
        'ownerId',
        [
            Sequelize.fn('count', Sequelize.col('name')),
            'countNotes'
        ]
    ],
    group: ['ownerId']
});
```

Update

```
await Note.update(
    {
        text: 'My favorite books'
    },
    {
        where: { name: 'Books' }
    }
);
```

Update. Active record

```
const note = await Note.findOne({
    where: { name: 'Books' }
});

note.text = 'Important books to read';

await note.save();
```

Delete

```
await Note.destroy({
    where: { id: 23 }
});

const note = await Note.findOne({
    where: { id: 23 }
});

await note.destroy();
```

Комбинации. findOrCreate

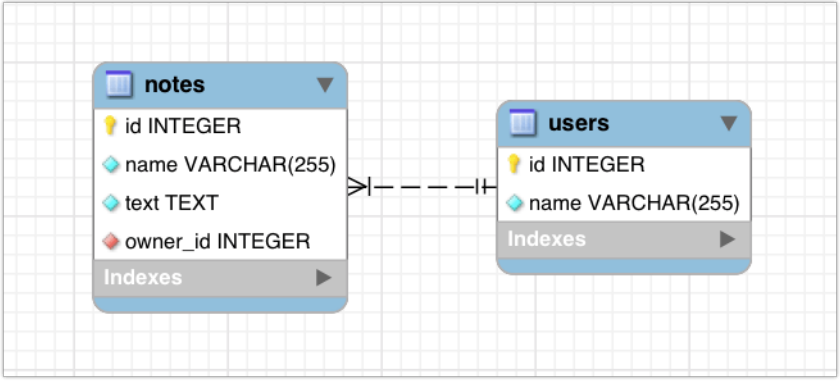
```
const note = await Note.findOrCreate({
    where: { name: 'Books' },
    defaults: {
        name: 'Books',
        text: 'Books to read',
        ownerId: 1
    }
});
```

JOIN

Внешние связи. ForeignKey

```
class Note extends Model<Note> {
    @ForeignKey(() => User)
    @Column({
        type: DataType.INTEGER,
        field: 'owner_id'
    })
    ownerId: number;
}
```

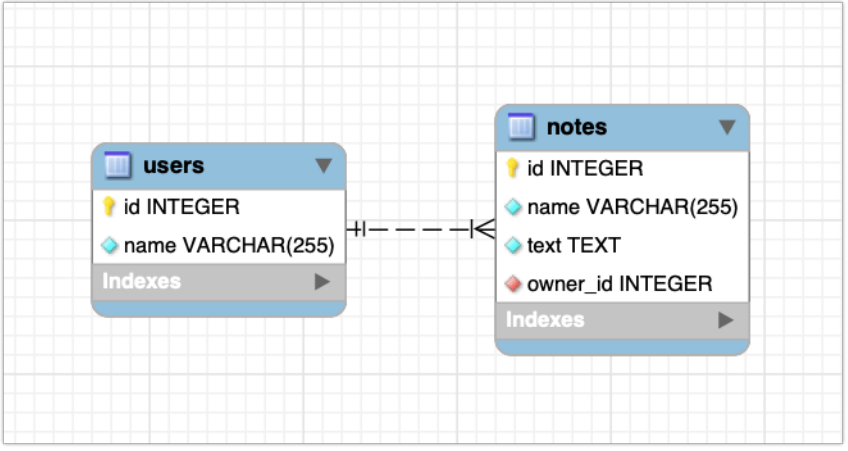
Внешние связи. BelongsTo



Внешние связи. BelongsTo

```
class Note extends Model<Note> {
    ...

    @ForeignKey(() => User)
    @Column({
        type: DataType.INTEGER,
        field: 'owner_id'
    })
    ownerId: number;

    @BelongsTo(() => User)
    user: User;
}
```

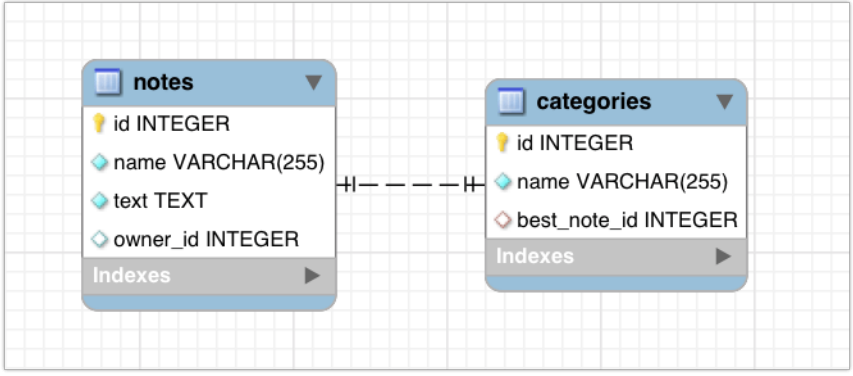
Внешние связи. HasMany



Внешние связи. HasMany

```
class User extends Model<User> {
    ...

    @HasMany(() => Note)
    notes: Note[];
}
```

Внешние связи. HasOne



Внешние связи. HasOne

```
class Note extends Model<Note> {
    ...

    @HasOne(() => Category)
    category: Category;
}
```

Внешние связи

Подробнее

Данные

```
  users     |         notes          |      categories
            |                        |
id   name    |  id  name    owner_id  | id  name   best_note_id
--   ------  |  --  -----   --------  | --  -----  ------------
 1   Олег    |  1   Books      1      | 1   study        1
 2   Сергей  |  2   Films      1      | 2    fun         3
 3   Михаил  |  3   Music      2      |
             |  4   Rules    NULL     |
             |  5  Markdown  NULL     |
```

Include

```
await User.findAll({
  attributes: ['name'],
  include: [
    {
      model: Note,
      attributes: ['name']
    }
  ]
});
```

```
[
  {
    name: 'Олег',
    notes: [
      { name: 'Films' },
      { name: 'Books' }
    ]
  },
  {
    name: 'Сергей',
    notes: [{ name: 'Music' }]
  },
  {
    name: 'Михаил',
    notes: []
  }
]
```

Include. Where

```
await User.findAll({
  attributes: ['name'],
  include: [
    {
      model: Note,
      where: {
        name: {
          [Op.or]: [
            { [Op.like]: 'F%' },
            { [Op.like]: 'M%' }
          ]
        }
      },
      attributes: ['name']
    }
  ]
});
```

```
[
  {
    name: 'Олег',
    notes: [
      { name: 'Films' }
    ]
  },
  {
    name: 'Сергей',
    notes: [
      { name: 'Music' }
    ]
  }
]
```

Include. Where + required

```
await User.findAll({
  attributes: ['name'],
  include: [
    {
      model: Note,
      where: {
        name: {
          [Op.or]: [
            { [Op.like]: 'F%' },
            { [Op.like]: 'M%' }
          ]
        }
      },
      attributes: ['name'],
      required: false
    }
  ]
});
```

```
[
  {
    name: 'Олег',
    notes: [{ name: 'Films' }]
  },
  {
    name: 'Сергей',
    notes: [{ name: 'Music' }]
  },
  {
    name: 'Михаил',
    notes: []
  }
]
```

Include. Multiple join

```
const users = await User.findAll({
    attributes: ['name'],
    include: [
        {
            model: Note,
            attributes: ['name'],
            include: [
                {
                    model: Category,
                    attributes: ['name']
                }
            ]
        }
    ]
});
```

Include. Multiple join

```
[
  {
    name: 'Олег',
    notes: [
      {
        name: 'Films',
        category: null
      },
      {
        name: 'Books',
        category: {
          name: 'study'
        }
      }
    ]
  },
  ...
```

```
  ...
  {
    name: 'Сергей',
    notes: [
      {
        name: 'Music',
        category: {
          name: 'fun'
        }
      }
    ]
  },
  {
    name: 'Михаил',
    notes: []
  }
]
```

Транзакции

```
import { Sequelize } from 'sequelize-typescript';

const sequelize = new Sequelize({ ... });

await sequelize.transaction(async function(t) {
    await User.increment(
        'account',
        { by: -50, where: { id: 1 }, transaction: t }
    );
    await User.increment(
        'account',
        { by: 50, where: { id: 2 }, transaction: t }
    );
});
```

На почитать:

- Первая и вторая лекции прошлого года
- Как думать на SQL?
- Нормальные формы
- Документация PostgreSQL
- Документация Sequelize
- Sequelize + TypeScript