📘 Njango ORM Documentation

Njango ORM is a lightweight Django-inspired ORM built in Node.js using SQLite as a backend.   
It provides Django-like APIs such as `Model`, `QuerySet`, and `Manager` for database operations,   
allowing developers to interact with databases using Pythonic, chainable syntax in JavaScript.

# 1. Model Definition

Each model represents a table in the database. You define it by extending the base `Model` class.  
Example:

import { Model } from "../../../core/orm/models.js";  
import { User } from "../../auth/models/user.js";  
  
export class Blog extends Model {  
 static table = "posts";  
 static fields = {  
 id: Blog.fields.IntegerField({ primaryKey: true }),  
 title: Blog.fields.CharField({ max\_length: 200 }),  
 body: Blog.fields.TextField({ null: true }),  
 author: Blog.fields.ForeignKey(User, { related\_name: "blogs", on\_delete: "CASCADE" }),  
 };  
  
 constructor(data = {}) {  
 super();  
 Object.assign(this, data);  
 }  
}  
  
await Blog.init();

# 2. QuerySet Methods

## all()

Fetch all records from the table.

Example:

const blogs = await Blog.objects.all();

## filter()

Filter rows based on given conditions.

Example:

const blogs = await Blog.objects.filter({ author\_id: 1 }).all();

## exclude()

Exclude records that match certain filters.

Example:

const blogs = await Blog.objects.exclude({ title: 'Draft' }).all();

## get()

Retrieve a single record that matches the filter.

Example:

const blog = await Blog.objects.get({ id: 5 });

## create()

Insert a new record into the table.

Example:

await Blog.objects.create({ title: 'New Post', body: 'Content', author\_id: 1 });

## update()

Update existing records.

Example:

await Blog.objects.update({ id: 1 }, { title: 'Updated Title' });

## delete()

Delete records from the table.

Example:

await Blog.objects.delete({ id: 1 });

## count()

Count records matching the filter.

Example:

const total = await Blog.objects.filter({ author\_id: 1 }).count();

## first()

Return the first record (ordered by ID).

Example:

const firstBlog = await Blog.objects.first();

## last()

Return the last record (ordered by ID DESC).

Example:

const lastBlog = await Blog.objects.last();

## exists()

Check if any record exists for a given condition.

Example:

const exists = await Blog.objects.filter({ author\_id: 1 }).exists();

## orderBy()

Order query results by fields.

Example:

const blogs = await Blog.objects.orderBy('-id').all();

## limit()

Limit number of records fetched.

Example:

const blogs = await Blog.objects.limit(5).all();

## offset()

Skip a number of records before returning results.

Example:

const blogs = await Blog.objects.offset(10).limit(5).all();

## distinct()

Return distinct records based on fields.

Example:

const titles = await Blog.objects.values('title').distinct().all();

## aggregate()

Perform aggregate queries like COUNT, SUM, AVG.

Example:

const stats = await Blog.objects.aggregate({ total: { fn: 'COUNT', field: 'id' } });

## annotate()

Add computed fields to queryset (e.g., count of related items).

Example:

const blogs = await Blog.objects.annotate({ comment\_count: { fn: 'COUNT', field: 'comments' } }).all();

## union()

Combine results of two querysets.

Example:

const combined = await Blog.objects.filter({ author\_id: 1 }).union(otherQuerySet).all();

## intersect()

Return records common to two querysets.

Example:

const common = await Blog.objects.filter({ author\_id: 1 }).intersect(otherQuerySet).all();

## except()

Return records in one queryset but not in another.

Example:

const diff = await Blog.objects.filter({ author\_id: 1 }).except(otherQuerySet).all();

# 3. Related Fields

Njango supports `ForeignKey`, `OneToOneField`, and `ManyToManyField` relations like Django.  
  
Example Many-to-Many relation:

export class Group extends Model {  
 static table = "groups";  
 static fields = {  
 id: Group.fields.AutoField(),  
 name: Group.fields.CharField({ max\_length: 100 }),  
 users: Group.fields.ManyToManyField(User, { related\_name: "groups" }),  
 };  
}

This automatically creates an intermediate table like `group\_users`   
to handle the many-to-many relationship.

# 4. Aggregations and Annotations

Example:  
const stats = await Order.objects.aggregate({ total: { fn: 'SUM', field: 'amount' } });  
console.log(stats.total);

# 5. Filtering with Complex Lookups

You can perform advanced queries using multiple filters and operators.  
Example:  
const results = await Blog.objects.filter({ author\_id: 1, title: 'Hello' }).exclude({ id: 5 }).orderBy('-id').limit(3).all();

# 6. Query Composition

You can combine querysets using union, intersect, and except operations.

Example:  
const q1 = Blog.objects.filter({ author\_id: 1 });  
const q2 = Blog.objects.filter({ author\_id: 2 });  
const union = await q1.union(q2).all();

# 7. Custom Query

ORM has option to custom query

const rows = await runQuery("SELECT \* FROM users WHERE id = ?", [1]);

await runExecute("UPDATE users SET firstname = ? WHERE id = ?", ["Alice", 1]);

import { runExecute , runQuery } from "../../core/orm/db.js"; “According to your path”