# Database Query Tool Documentation

The Database Query Tool provides a convenient command-line interface for querying your PostgreSQL database using pre-built queries and raw SQL capabilities.

## 🚀 Quick Start

# Show available queries  
npm run db:query help  
  
# List all users  
npm run db:query users  
  
# Show database schema  
npm run db:query schema

## 📋 Available Queries

### 👥 User Queries

| Command | Description | Example Output |
| --- | --- | --- |
| users | List all users with basic info | Table with id, username, email, role, status |
| users:count | Count total users | Total users: 15 |
| users:active | List only active users | Table filtered by status=‘active’ |

npm run db:query users  
npm run db:query users:count  
npm run db:query users:active

### 🎭 Role Queries

| Command | Description | Example Output |
| --- | --- | --- |
| roles | List all roles | Table with id, name, description, is\_system |
| roles:system | List only system roles | Table filtered by is\_system=true |

npm run db:query roles  
npm run db:query roles:system

### 🔐 Permission Queries

| Command | Description | Example Output |
| --- | --- | --- |
| permissions | List all permissions | Table with id, name, category, description |
| permissions:by-category | Group permissions by category | Separate tables for each category |

npm run db:query permissions  
npm run db:query permissions:by-category

### 🔗 Relationship Queries

| Command | Description | Example Output |
| --- | --- | --- |
| user-roles | Show user-role assignments | Table with username, role name, assigned date |
| role-permissions | Show role-permission assignments | Table with role name, permission name, category |

npm run db:query user-roles  
npm run db:query role-permissions

### 📊 Database Information

| Command | Description | Example Output |
| --- | --- | --- |
| tables | List all database tables | Table with table names and types |
| schema | Show detailed table schemas | Column info for all tables |

npm run db:query tables  
npm run db:query schema

## 📖 Detailed Examples

### Example 1: Check User Data

# See all users  
npm run db:query users

**Output:**

📋 All Users:  
┌─────────┬────┬──────────┬─────────────────┬───────┬────────┐  
│ (index) │ id │ username │ email │ role │ status │  
├─────────┼────┼──────────┼─────────────────┼───────┼────────┤  
│ 0 │ 1 │ 'admin' │ 'admin@test.com'│'admin'│'active'│  
│ 1 │ 2 │ 'user1' │ 'user@test.com' │'user' │'active'│  
└─────────┴────┴──────────┴─────────────────┴───────┴────────┘

### Example 2: Analyze Permissions by Category

npm run db:query permissions:by-category

**Output:**

📂 Permissions by Category:  
  
📁 USERS:  
┌─────────┬────┬─────────────┬─────────┬──────────────────────────┐  
│ (index) │ id │ name │category │ description │  
├─────────┼────┼─────────────┼─────────┼──────────────────────────┤  
│ 0 │ 1 │'users:read' │ 'users' │'View user information' │  
│ 1 │ 2 │'users:write'│ 'users' │'Create and update users' │  
└─────────┴────┴─────────────┴─────────┴──────────────────────────┘  
  
📁 ADMIN:  
┌─────────┬────┬──────────────────┬─────────┬─────────────────────────┐  
│ (index) │ id │ name │category │ description │  
├─────────┼────┼──────────────────┼─────────┼─────────────────────────┤  
│ 0 │ 9 │'admin:dashboard' │ 'admin' │'Access admin dashboard' │  
└─────────┴────┴──────────────────┴─────────┴─────────────────────────┘

### Example 3: Check Role-Permission Assignments

npm run db:query role-permissions

**Output:**

🔗 Role-Permission Assignments:  
┌─────────┬────────┬──────────────┬──────────────────┬─────────┐  
│ (index) │ roleId │ roleName │ permissionName │category │  
├─────────┼────────┼──────────────┼──────────────────┼─────────┤  
│ 0 │ 1 │ 'admin' │ 'users:read' │ 'users' │  
│ 1 │ 1 │ 'admin' │ 'users:write' │ 'users' │  
│ 2 │ 2 │ 'user' │ 'users:read' │ 'users' │  
└─────────┴────────┴──────────────┴──────────────────┴─────────┘

### Example 4: Database Schema Inspection

npm run db:query schema

**Output:**

🏗️ Table Schemas:  
  
📋 USERS:  
┌─────────┬─────────────────┬─────────────────┬─────────────┬──────────────────┐  
│ (index) │ column\_name │ data\_type │ is\_nullable │ column\_default │  
├─────────┼─────────────────┼─────────────────┼─────────────┼──────────────────┤  
│ 0 │ 'id' │ 'integer' │ 'NO' │ 'nextval(...' │  
│ 1 │ 'username' │'character varying'│ 'NO' │ null │  
│ 2 │ 'email' │'character varying'│ 'YES' │ null │  
└─────────┴─────────────────┴─────────────────┴─────────────┴──────────────────┘

## 🛠️ Advanced Usage

### Raw SQL Queries

The tool uses the postgres client internally, so you can extend it with custom queries by modifying /api/scripts/db-query.js.

### Adding Custom Queries

To add your own queries, edit the queries object in /api/scripts/db-query.js:

const queries = {  
 // ... existing queries ...  
   
 'my-custom-query': async () => {  
 console.log('🔍 My Custom Query:');  
 const result = await db.select().from(users).where(eq(users.status, 'inactive'));  
 console.table(result);  
 },  
};

Then use it:

npm run db:query my-custom-query

## 🔧 Troubleshooting

### Connection Issues

If you get connection errors: 1. Check your .env file has correct database credentials 2. Ensure your database is running 3. Verify SSL settings match your database configuration

### Query Errors

If a specific query fails: 1. Check the error message for details 2. Verify the table/column names exist in your database 3. Ensure you have proper permissions

### Performance

For large datasets: - Use specific queries instead of users (which gets all users) - Consider adding LIMIT clauses for very large tables - Use users:count instead of users if you just need the count

## 📚 Related Tools

* **Database Seeder**: npm run db:seed - Populate database with default data
* **Duplicate Remover**: npm run db:remove-duplicates - Clean up duplicate records
* **Drizzle Studio**: npm run db:studio - Visual database browser

## 🎯 Best Practices

1. **Start with help** - Always check available queries first
2. **Use specific queries** - Prefer users:active over users when you only need active users
3. **Check schema first** - Use schema or tables to understand your database structure
4. **Combine with other tools** - Use with seeder and duplicate remover for complete database management

## 📝 Examples by Use Case

### Database Health Check

npm run db:query tables # See what tables exist  
npm run db:query users:count # Check user count  
npm run db:query roles # Verify roles are set up  
npm run db:query permissions # Check permissions exist

### RBAC Verification

npm run db:query user-roles # See who has what roles  
npm run db:query role-permissions # See what permissions each role has  
npm run db:query permissions:by-category # Organize permissions by category

### Data Analysis

npm run db:query users:active # See active users  
npm run db:query roles:system # Check system roles  
npm run db:query schema # Understand table structure

This tool provides a powerful way to inspect and understand your database without needing to connect to PostgreSQL directly or write custom SQL queries.