Hướng dẫn tạo Model bằng thư viện `vdb` trong Go

Thư viện `vdb` cung cấp một cách tiện lợi để định nghĩa các model trong Go, ánh xạ chúng với cơ sở dữ liệu quan hệ. Hướng dẫn này sẽ giải thích cách tạo model, cấu hình các trường, chỉ mục, khóa ngoại, và đăng ký model với `vdb`.

1. Cấu trúc cơ bản của một Model

Mỗi model là một struct Go được nhúng với `vdb.Model[T]` để chỉ định loại của model. Struct này thường bao gồm các trường ánh xạ tới các cột trong cơ sở dữ liệu, cùng với các tag `db` để định nghĩa thuộc tính của cột.

Ví dụ về một model đơn giản:

```
type Department struct {
    vdb.Model[Department]
    ID       int    `db:"pk;auto"`
    Name     string `db:"size:100;uk:uq_dept_name"`
    Code     string `db:"size:20;uk:uq_dept_code"`
    ParentID *int
    BaseModel
}
```

- `vdb.Model[Department]`: Nhúng để đánh dấu struct là một model của `vdb`.
- Các trường như `ID`, `Name`, `Code`, `ParentID` ánh xạ tới các cột trong bảng cơ sở dữ liệu.
- `BaseModel`: Một struct cơ sở có thể được nhúng để thêm các trường chung như `CreatedAt`, `UpdatedAt`.

2. Các tag `db` phổ biến

Tag `db` được sử dụng để cấu hình các thuộc tính của cột trong cơ sở dữ liệu. Dưới đây là các tag phổ biến:

- `pk`: Đánh dấu trường là khóa chính.
- `auto`: Cho biết khóa chính được tự động tăng (auto-increment).
- `size:N`: Xác định độ dài tối đa của trường (dùng cho `string`).
- `type:<type>`: Chỉ định kiểu dữ liệu của cột (ví dụ: `date`, `time`, `decimal(15,2)`, `char(7)`).
- `idx`: Tạo chỉ mục (index) cho trường.
- `idx:<index_name>`: Tạo chỉ mục với tên cụ thể.
- `uk:<unique_key_name>`: Tạo ràng buộc duy nhất (unique key) với tên cụ thể.
- `fk(<Table.Column>)`: Định nghĩa khóa ngoại trỏ đến cột của bảng khác.
- `default:<value>`: Đặt giá trị mặc định (ví dụ: `default:now` cho thời gian hiện tại).

Ví dụ:

```
type Salary struct {
    vdb.Model[Salary]
    ID        int     `db:"pk;auto"`
    UserID    int     `db:"idx:idx_salary_user"`
    Month     string  `db:"type:char(7);idx:idx_salary_month"`
    Base      float64 `db:"type:decimal(15,2)"`
}
```

3. Nhúng BaseModel

`BaseModel` là một struct cơ sở chứa các trường chung thường được sử dụng trong nhiều model, chẳng hạn như thời gian tạo và cập nhật.

```
type BaseModel struct {
    CreatedAt   *time.Time `db:"default:now;idx"`
    UpdatedAt   *time.Time `db:"default:now;idx"`
    Description *string    `db:"size:255"`
}
```

- Nhúng `BaseModel` vào model để kế thừa các trường này.
- Các trường trong `BaseModel` sẽ tự động được ánh xạ vào bảng cơ sở dữ liệu.

4. Định nghĩa khóa ngoại (Foreign Key)

Khóa ngoại được cấu hình bằng cách sử dụng phương thức `AddForeignKey` trong hàm `init()` của package. Cú pháp:

```
(&ModelName{}).AddForeignKey(
    "FieldName",      // Tên trường trong model
    &ReferenceModel{}, // Model tham chiếu
    "ReferenceField", // Trường tham chiếu trong model tham chiếu
    &vdb.CascadeOption{ // Tùy chọn cascade (nếu có)
        OnDelete: false,
        OnUpdate: false,
    },
)
```

Ví dụ:

```
func init() {
    (&Department{}).AddForeignKey("ParentID", &Department{}, "ID", &vdb.CascadeOption{
        OnDelete: false,
        OnUpdate: false,
    })
}
```

Trong ví dụ trên, `ParentID` trong `Department` tham chiếu đến `ID` của chính bảng `Department`, tạo mối quan hệ tự tham chiếu.

5. Đăng ký Model với `vdb.ModelRegistry`

Để `vdb` nhận diện và xử lý model, bạn cần đăng ký model bằng cách sử dụng `vdb.ModelRegistry.Add`. Điều này thường được thực hiện trong hàm `init()`.

```
func init() {
    vdb.ModelRegistry.Add(&ModelName{})
}
```

Ví dụ:

```
func init() {
    vdb.ModelRegistry.Add(&LeaveRequest{})
    (&LeaveRequest{}).AddForeignKey("EmployeeId", &Employee{}, "ID", nil)
}
```

Bạn cũng có thể đăng ký nhiều model cùng lúc:

```
func init() {
    vdb.ModelRegistry.Add(
        &Contract{},
        &User{},
        &Department{},
        &Position{},
    )
}
```

6. Xử lý các trường tùy chọn

Các trường có thể là con trỏ (`*int`, `*string`, `*time.Time`) để cho phép giá trị `NULL` trong cơ sở dữ liệu. Ví dụ:

```
type User struct {
    vdb.Model[User]
    ID           int     `db:"pk;auto"`
    UserId       *string `db:"size:36;unique"`
    Email        string  `db:"uk:uq_email;size:150"`
    Phone        string  `db:"size:20"`
    Username     *string `db:"size:50;unique"`
    HashPassword *string `db:"size:100"`
    BaseModel
}
```

- `UserId`, `Username`, `HashPassword` là con trỏ, cho phép giá trị `NULL`.
- `Email` và `Phone` là kiểu `string` thông thường, không cho phép `NULL`.

7. Ví dụ đầy đủ: Tạo Model Employee

Dưới đây là một ví dụ đầy đủ về cách định nghĩa một model `Employee`, bao gồm khóa ngoại và đăng ký với `vdb`:

```
package models

import "vdb"

type Employee struct {
    vdb.Model[Employee]
    ID           int    `db:"pk;auto"`
    FirstName    string `db:"size:50;idx"`
    LastName     string `db:"size:50;idx"`
    DepartmentID int    `db:"fk(Department.ID)"`
    PositionID   int    `db:"fk(Position.ID)"`
    UserID       int    `db:"fk(User.ID)"`
    BaseModel
}

func init() {
    vdb.ModelRegistry.Add(&Employee{})
    (&Employee{}).AddForeignKey("DepartmentID", &Department{}, "ID", nil).
        AddForeignKey("PositionID", &Position{}, "ID", nil).
        AddForeignKey("UserID", &User{}, "ID", nil)
}
```

- `Employee` có các khóa ngoại trỏ đến `Department`, `Position`, và `User`.
- Các trường `FirstName` và `LastName` có chỉ mục để tối ưu hóa tìm kiếm.

8. Lưu ý quan trọng

- **Định nghĩa bảng**: Sử dụng tag `db:"table:<table_name>"` nếu tên bảng không giống tên struct.
- **Kiểu dữ liệu**: Đảm bảo ánh xạ đúng giữa kiểu Go và kiểu cơ sở dữ liệu (ví dụ: `time.Time` cho `date` hoặc `timestamp`).
- **Đăng ký model**: Luôn đăng ký model với `vdb.ModelRegistry.Add` để `vdb` có thể quản lý schema.
- **Khóa ngoại**: Sử dụng `AddForeignKey` để đảm bảo tính toàn vẹn tham chiếu.
- **Cascade**: Cân nhắc sử dụng `CascadeOption` cho các hành vi `ON DELETE` hoặc `ON UPDATE` nếu cần.

9. Kết luận

Thư viện `vdb` cung cấp một cách mạnh mẽ và linh hoạt để định nghĩa các model trong Go, với hỗ trợ đầy đủ cho khóa chính, khóa ngoại, chỉ mục, và các ràng buộc cơ sở dữ liệu. Bằng cách tuân theo các bước trên, bạn có thể dễ dàng tạo và quản lý các model ánh xạ tới cơ sở dữ liệu một cách hiệu quả.