

Schema și interogările unui magazin demonstrativ

Fișă publică de proiect · exerciții și criterii de acceptare

Proiectezi comenzi, poziții și produse, apoi verifici relațiile și planurile de interogare. Datele de test sunt separate de producție.

Livrabile

- Schemă
- Migrații
- Relații
- Raport N+1
- EXPLAIN înainte și după

Criterii de acceptare

- Prețul istoric rămâne stabil
- Pivotalul nu permite duplicate
- Optimizarea are măsurători

Schema pornită de la relații și constrângeri

Proiectează tabelul după întrebările aplicației și invariante, nu numai după formularul curent. O cerere aparține unui utilizator, iar un utilizator poate avea mai multe cereri. Cheia străină protejează legătura, dar regula la ștergere trebuie aleasă deliberat. Nu șterge în cascadă date care trebuie păstrate fără să verifici politica produsului. Pentru laborator, compară restrict și nullability înainte de a decide.

Exemplu de lucru

```
users: id, email unic.  
service_requests: id, user_id, service, status, created_at.  
Migrare exemplu:  
$table->foreignId('user_id')->constrained()->restrictOnDelete();  
$table->index(['user_id', 'status']);  
Întrebări: cine deține cererea; ce se întâmplă la ștergerea userului; cum căutăm cererile active?
```

Interpretarea rezultatului

Indexul compus urmărește o interogare concretă, nu se adaugă doar pentru că două câmpuri există. restrictOnDelete poate bloca ștergerea utilizatorului cu cereri; aplicația trebuie să trateze acest rezultat. Verifică motorul DB și constrângerile efective în laborator.

Exercițiu de consolidare

Propune o strategie de anonimizare pentru istoricul păstrat, fără a pretinde că simpla schimbare a emailului rezolvă toate obligațiile.

1. Proiectarea schemei

Propune o strategie de anonimizare pentru istoricul păstrat, fără a pretinde că simpla schimbare a emailului rezolvă toate obligațiile.

Verificare orientativă

Indexul compus urmărește o interogare concretă, nu se adaugă doar pentru că două câmpuri există. `restrictOnDelete` poate bloca ștergerea utilizatorului cu cereri; aplicația trebuie să trateze acest rezultat. Verifică motorul DB și constrângerile efective în laborator.

De evitat

O comandă veche nu trebuie recalculată după modificarea catalogului.

Relație unu-la-mulți și interogare limitată

Relația definește cum se leagă modelele, iar interogarea decide ce date sunt citite. Pentru cererile utilizatorului curent, pornește din relația lui, nu din toate cererile și filtrare în PHP. Aceasta reduce riscul de acces accidental și volumul transferat. Totuși, relația nu înlocuiește policy pentru toate operațiile. Păstrează paginarea, fiindcă un utilizator poate acumula multe înregistrări.

Exemplu de lucru

```
În User:
public function serviceRequests(): HasMany { return $this->hasMany(ServiceRequest::class); }
În ServiceRequest:
public function user(): BelongsTo { return $this->belongsTo(User::class); }
Listă:
$request = $request->user()->serviceRequests()->latest()->paginate(20);
```

Interpretarea rezultatului

Lista conține numai resursele utilizatorului autentificat și maximum pagina cerută. Importă tipurile de relație potrivite. Testează cu doi utilizatori și cereri diferite; un test cu un singur utilizator nu poate detecta o scurgere între conturi.

Exercițiu de consolidare

Adaugă filtrul `status=new` și verifică faptul că totalul paginării se referă la setul filtrat.

2. Relații unu-la-mulți

Adaugă filtrul `status=new` și verifică faptul că totalul paginării se referă la setul filtrat.

Verificare orientativă

Lista conține numai resursele utilizatorului autentificat și maximum pagina cerută. Importă tipurile de relație potrivite. Testează cu doi utilizatori și cereri diferite; un test cu un singur utilizator nu poate detecta o scurgere între conturi.

De evitat

O relație opțională trebuie tratată explicit la afișare.

Pivot cu unicitate și attribute proprii

O relație mulți-la-mulți poate avea date proprii: de exemplu, un utilizator participă la mai multe cursuri și un curs are mai mulți utilizatori. Pivotul poate păstra data înscrierii și statusul. O constrângere unică pe pereche previne duplicatele la nivel DB. Nu presupune că un if în aplicație este suficient în condiții de concurență. Alege attach, sync sau syncWithoutDetaching după sensul operației, nu după numele cel mai scurt.

Exemplu de lucru

```
course_user: user_id, course_id, enrolled_at.  
Migrare: $table->unique(['user_id', 'course_id']);  
Relație: return $this->belongsToMany(Course::class)->withPivot('enrolled_at');  
Înscriere fără eliminarea altor cursuri:  
$user->courses()->syncWithoutDetaching([$courseId => ['enrolled_at' => now()]]);
```

Interpretarea rezultatului

Operația trebuie verificată și repetată: perechea rămâne unică. sync poate elimina relații neincluse în lista transmisă, ceea ce este greșit pentru o simplă înscriere într-un curs nou. Nu folosi pivotul de înscriere ca dovadă de plată fără un contract separat.

Exercițiu de consolidare

Testează înscrierea în două cursuri, apoi repetarea primei înscrieri. Verifică păstrarea ambelor relații.

3. Relații mulți-la-mulți și pivot

Testează înscrierea în două cursuri, apoi repetarea primei înscrieri. Verifică păstrarea ambelor relații.

Verificare orientativă

Operația trebuie verificată și repetată: perechea rămâne unică. sync poate elimina relații neincluse în lista transmisă, ceea ce este greșit pentru o simplă înscriere într-un curs nou. Nu folosi pivotul de înscriere ca dovadă de plată fără un contract separat.

De evitat

Fără constrângere, o etichetă poate apărea duplicat.

N+1 observat prin numărul de interogări

Dacă afișezi 20 de cereri și citești utilizatorul separat pentru fiecare, poți produce o interogare pentru listă și încă una pentru fiecare rând. Eager loading reduce acest tipar, dar trebuie cerute numai relațiile necesare. Nu încărca toate relațiile posibile din precauție. Măsoară numărul de interogări într-un mediu de test și compară aceeași pagină înainte și după. Un timp mai mic într-o singură rulare nu este suficientă dovadă.

Exemplu de lucru

```
Fără eager loading:  
$requests = ServiceRequest::latest()->limit(20)->get();  
foreach ($requests as $r) echo $r->user->name;  
Cu eager loading:  
$requests = ServiceRequest::with('user:id,name')->latest()->limit(20)->get();  
Păstrează user_id în selecția cererilor dacă limitezi și coloanele principale.
```

Interpretarea rezultatului

Selecția user:id,name trebuie să includă cheia necesară relației. Verifică lipsa accesului accidental la o relație neîncărcată în view. În dezvoltare, poți folosi mecanismele frameworkului pentru detectarea lazy loadingului, adaptate proiectului.

Exercițiu de consolidare

Adaugă o relație suplimentară în template și măsoară dacă apare un nou tipar N+1.

4. Eager loading și N+1

Adaugă o relație suplimentară în template și măsoară dacă apare un nou tipar N+1.

Verificare orientativă

Selecția `user:id,name` trebuie să includă cheia necesară relației. Verifică lipsa accesului accidental la o relație neîncărcată în view. În dezvoltare, poți folosi mecanismele frameworkului pentru detectarea lazy loadingului, adaptate proiectului.

De evitat

Eager loading trebuie ales după relațiile folosite efectiv.

EXPLAIN pentru o întrebare precisă

Un index se alege după filtrare, sortare și distribuția datelor. Pentru cererile unui utilizator după status, un index compus poate ajuta, dar trebuie verificat cu planul de execuție și cu date reprezentative. EXPLAIN arată o estimare a planului; nu este un benchmark complet. Nu adăuga indexuri pe fiecare coloană, deoarece acestea ocupă spațiu și influențează scrierile. Păstrează comparația pe aceeași interogare.

Exemplu de lucru

```
EXPLAIN SELECT id, status, created_at
FROM service_requests
WHERE user_id = 42 AND status = 'new'
ORDER BY created_at DESC
LIMIT 20;
Candidat de verificat: INDEX(user_id, status, created_at).
Măsurători: plan, rânduri estimate, latență repetată, cost de scriere.
```

Interpretarea rezultatului

Ordinea coloanelor contează. Un index util pentru filtrul complet nu este automat potrivit pentru căutarea numai după status. Pentru baze mici, diferența de timp poate fi nesemnificativă; nu publica rezultate numerice inventate ca dovadă de performanță.

Exercițiu de consolidare

Compară planul cu și fără filtrul `user_id` într-o bază de laborator și explică schimbarea observată.

5. Indexuri și EXPLAIN

Compară planul cu și fără filtrul `user_id` într-o bază de laborator și explică schimbarea observată.

Verificare orientativă

Ordinea coloanelor contează. Un index util pentru filtrul complet nu este automat potrivit pentru căutarea numai după status. Pentru baze mici, diferența de timp poate fi nesemnificativă; nu publica rezultate numerice inventate ca dovadă de performanță.

De evitat

Un index pe fiecare coloană nu este o strategie de optimizare.

Migrare expand-backfill-contract

Pe date reale, schema și codul pot rula în versiuni diferite pentru o perioadă. Adaugă întâi structura compatibilă, completează datele în loturi și verifică integritatea înainte restricției finale. Nu combina redenumirea, ștergerea și schimbarea logicii într-un singur pas imposibil de urmărit. O migrare reversibilă sintactic nu garantează recuperarea datelor șterse. Păstrează backup și testează restaurarea în altă destinație.

Exemplu de lucru

Etapa 1: adaugă reference nullable.
Etapa 2: codul scrie și noul câmp.
Etapa 3: backfill în loturi după id.
Etapa 4: verifică nulluri, duplicate și concordanță.
Etapa 5: restricție NOT NULL/unicitate după confirmare.
Etapa 6: elimină suportul vechi într-un deploy ulterior.

Interpretarea rezultatului

Un job vechi din coadă poate continua să scrie fără noul câmp, deci include workerii în plan. Dacă backfillul eșuează la jumătate, trebuie să poată fi reluat fără duplicare. Nu folosi offset pentru loturi care se modifică în timpul procesării fără a evalua consecințele.

Exercițiu de consolidare

Definește o verificare SQL care demonstrează că toate rândurile eligibile au fost completate înainte restricției.

6. Migrări pe date reale

Definește o verificare SQL care demonstrează că toate rândurile eligibile au fost completate înainte restricției.

Verificare orientativă

Un job vechi din coadă poate continua să scrie fără noul câmp, deci include workerii în plan. Dacă backfillul eșuează la jumătate, trebuie să poată fi reluat fără duplicare. Nu folosi offset pentru loturi care se modifică în timpul procesării fără a evalua consecințele.

De evitat

O migrare blocantă poate afecta traficul chiar dacă SQL-ul este corect.

Atelier final: Preț istoric și raport fără N+1

Timp de lucru estimat: 40 de minute. Folosește datele de laborator de mai jos și salvează rezultatul separat de orice proiect real.

Date de pornire

Produsul P costă acum 2.500 bani. Comanda veche conține 2 bucăți la 2.000 bani fiecare. Raportul listează 20 de comenzi și produsele lor.

1. Pregătire — 5 minute

Extrage intrările și condițiile din enunț. Notează ce rezultat trebuie produs și ce informații nu sunt confirmate. Pregătește fișierele sau tabelul în care vei păstra dovada lucrului.

2. Construire — 15 minute

Modelează prețul unitar în poziția comenzii. Construiește o listare cu relații încărcate explicit, apoi măsoară numărul de interogări pentru 1 și 20 de comenzi.

3. Variație și verificare — 15 minute

Aduagă o poziție cu produs retras și verifică raportul istoric. Explică ce informație trebuie păstrată în snapshotul poziției.

4. Autoevaluare — 5 minute

Compară rezultatul cu reperele de mai jos. Înregistrează cel puțin o eroare identificată sau un caz-limită verificat, plus corectura ori explicația comportamentului observat.

Repere de rezolvare

Comanda veche rămâne la 4.000 bani, indiferent de prețul actual. Numărul de interogări nu trebuie să crească cu fiecare rând din listare. Măsurarea se face pe aceeași structură de date.

Ce predai

- Schemă
- Migrații
- Relații
- Raport N+1
- EXPLAIN înainte și după

Criterii de acceptare

- Prețul istoric rămâne stabil
- Pivotul nu permite duplicate
- Optimizarea are măsurători

Păstrează o versiune inițială și una revizuită, cu explicația diferențelor. Dacă un criteriu nu este îndeplinit, revino la lecția relevantă și repetă verificarea. Durata este orientativă; instalarea mediului și aprofundarea pot necesita timp suplimentar.

Registrul de lucru

Pentru fiecare etapă notează intrarea, rezultatul, dovada verificării, eroarea găsită și corectura. Nu introduce date personale sau secrete în fișa partajată.

Documentație și surse

- Documentația Laravel 11 — versiunea aplicației demonstrative — Laravel
- EXPLAIN Statement — MySQL

<https://laravel.com/docs/11.x>

<https://dev.mysql.com/doc/refman/8.4/en/explain.html>