مخطط قاعدة البيانات (DB Diagram) وشرح تفصيلي

DB diagram

هذا المستند يشرح المخطط الموجود أعلاه (الذي أدرجته) بالتفصيل — جداول، حقول، علاقات، قواعد سلامة البيانات، واعتبارات للتنفيذ في لارافيل.

نظرة عامة سريعة

الـ schema يحتوي على الجداول الأساسية التالية:

- `users`
- `products`
- `carts` (تمثل عناصر الكارت — كل صف = عنصر واحد في كارت مستخدم)
- `orders`
- `order_items`

العلاقات الأساسية:

- user (1) — (many) cart rows
- user (1) — (many) orders
- product (1) — (many) cart rows
- product (1) — (many) order_items
- order (1) — (many) order_items

شرح الجداول (حقل بحقل)

users

- `id (PK)` — المفتاح الأساسي.
- `name` — اسم المستخدم.
- `email (unique)` — البريد الإلكتروني مع قيد التفرّد.
- `email_verified_at` — طابع زمني للتحقق (nullable).
- `password` — هاش الباسورد.
- `remember_token` — للتذكّر (optional).
- `timestamps` — `created_at` و `updated_at`.

تعليقات: الجدول يمثّل المستخدمين والمسؤول عن الربط مع الكارت والأوردرز.

---

products

- `id (PK)`
- `name` — اسم المنتج.
- `description` — وصف (nullable).

- `price` — `decimal(10,2)` — سعر الوحدة.
- `quantity` — كمية المخزون الحالية (integer).
- `sku (unique)` — رمز المنتج إن وُجد.
- `is_active` — boolean لتفعيل/تعطيل المنتج.
- `timestamps`

تعليقات: `quantity` هو الحقل الذي نخفضه عند انشاء الأوردر. يجب مراعاة التزامن (transactions/locking) عند تعديله لتفادي oversell.

---

carts

- `id (PK)`
- `user_id (FK -> users.id)` — حالة `constrained()->onDelete('cascade')` في المايجريشن.
- `product_id (FK -> products.id)` — من نفس القاعدة.
- `quantity` — عدد الوحدات في هذا العنصر.
- `timestamps`
- `UNIQUE (user_id, product_id)` — هذا يضمن أن كل زوج (مستخدم، منتج) يظهر مرة واحدة فقط.

مفهوم الداتا: كل صف في `carts` هو عنصر فردي داخل كارت المستخدم. لذلك كارت المستخدم هو مجموعة الصفوف التي تشترك في `user_id` واحد.

ملاحظة تصميمية: إن أردت فصل مفهوم "كارت" ككيان مستقل (مثلاً لخصائص الكارت كـ `currency`, `coupon_code`, `expires_at` ...) يمكنك إضافة جدول `shopping_carts` يحتوي على `id, user_id, meta...` ثم `cart_items` تشير إلى `shopping_cart_id` بدلاً من `user_id`.

---

orders

- `id (PK)`
- `order_number (unique)` — رقم الطلب، يجب أن يكون فريد (مثلاً `ORD-XXXX`).
- `address` — عنوان الشحن.
- `phone` — رقم العميل.
- `total` — المجموع `decimal(10,2)`.
- `timestamps`

تعليقات: لا يوجد حقل `user_id` في المايجريشن التي عرضتها — لو تطالب التطبيق، يجب عادة ربط الأوردر بالمستخدم (إضافة `foreignId('user_id')->constrained()->onDelete('cascade')`) حتى نعرف صاحب الطلب. إن لم يكن مقصودًا، فتأكد أن متطلباتك تسمح بأوردرات غير مرتبطة بمستخدم.

---

order_items

- `id (PK)`
- `order_id (FK -> orders.id) onDelete cascade`
- `product_id (FK -> products.id) onDelete cascade`
- `quantity` — عدد الوحدات في هذا الطلب.
- `price` — السعر في وقت الشراء (هام لتثبيت السعر لاحقًا).

- `timestamps`

تعليقات: حفظ السعر داخل `order_items.price` مهم لأنه يضمن أن السجل يحتفظ بسعر بيع تلك الحصة حتى لو تغير سعر المنتج لاحقًا.

علاقات ER (شرح العلاقات بالتفصيل)

- **User -> Carts (1 - many)**
- يملك المستخدم عدة صفوف في جدول `carts`، كل صف منتج واحد في الكارت.
- فعليًا هذه تمثل "عناصر الكارت"، وليست "كارتات متعددة".
- **Product -> Carts (1 - many)**
- كل منتج يمكن أن يكون موجودًا في كارتات عدة مستخدمين.
- **Order -> OrderItems (1 - many)**
- كل أوردر يحتوي على عدة عناصر؛ `order_items` يخزن التفاصيل لكل عنصر.
- **Product -> OrderItems (1 - many)**
- المنتج نفسه يمكن أن يظهر في العديد من طلبات العملاء.

نقاط مهمة للهندسة والتنفيذ في لارافيل

1. **ربط أوردر بالمستخدم**
2. أنصح بإضافة `user_id` إلى جدول `orders` حتى تضمن القدرة على استرجاع أوردرات المستخدم.
3. مثال في المايجريشن: `\$table->foreignId('user_id')->constrained()->onDelete('cascade');`
4. **التحكم بالمخزون (Stock)**
5. عند إنشاء الأوردر: نفذ كل شيء داخل `DB::transaction()` كما فعلت.
6. لتجنب حالات السباق (race conditions) عند الـ high concurrency، استخدم قفل صف (row locking): في Eloquent

```
\$product = Product::where('id', \$id)->lockForUpdate()->first();
```

7. هذا يضمن أن عمليتي كتابة متزامنتين لن تفرضا قيمة stock خاطئة.
8. **التخزين المؤقت (caching) والـ indexes**
9. ضع index على الحقول التي تُستدل كثيرًا مثل `user_id` و `product_id` (foreign keys عادة تعمل index تلقائياً).

10. فكر في cache لصفحة المنتج أو قائمة المنتجات لكن تأكد من إبطال الكاش عند تغيير `quantity` أو `is_active`.

11. **المعاملات (transactions) والـ rollback**

12. عند فشل أي خطوة أثناء إنشاء الأوردر، قم بعمل `DB::rollBack()` ومسح أي تعديل على الـ stock.

13. أنت بالفعل تستخدم هذا النمط — ممتاز.

14. **unique constraint في carts**

15. `UNIQUE(user_id, product_id)` مفيد يمنع تكرار صفين لنفس المنتج في نفس كارت. بدلاً من `updateOrCreate()` عند إضافة نفس المنتج، قم بزيادة الـ quantity.

16. **حقل السعر في order_items**

17. تأكد من نسخ `product.price` إلى `order_items.price` وقت الإنشاء — هذا يمنع تغيّر التاريخي للأسعار.

18. **Model relations (Eloquent)**

19. Users: `hasMany(Cart::class)` , `hasMany(Order::class)` (لو أضفت user_id إلى orders).
20. Cart: `belongsTo(User::class)` , `belongsTo(Product::class)` .
21. Order: `hasMany(OrderItem::class)` , `belongsTo(User::class)` (إن وُجد).
22. OrderItem: `belongsTo(Order::class)` , `belongsTo(Product::class)` .

تدفق مثال عملي (Flow) — من Add to Cart إلى Create Order

1. المستخدم يضيف منتجًا: `POST /api/auth/cart` مع `product_id` و `quantity` .
2. في السيرفر: إذا وجد صف في `carts` ، زِد `quantity` أو غَيّره حسب الطلب.
3. المستخدم يعرض الكارت: `GET /api/auth/cart` → تُعيد كل صفوف `carts` مع منتجاتها المرتبطة.
4. عند إنشاء الأوردر: `POST /api/auth/orders` مع `address` و `phone` .
5. السيرفر يجلب كل عناصر `carts` للمستخدم.
6. داخل `DB::transaction()` لكل منتج:
   - جلب المنتج مع `lockForUpdate()` .
   - التحقق من `quantity` في الـ stock.
   - إن كافٍ: `decrement('quantity', qty)` في نفس الترانزاكشن.
   - تحضير `order_items` مع `price` و `quantity` .
7. إنشاء السجل في `orders` ثم `order_items` .
8. حذف صفوف الـ `carts` للمستخدم (clear cart).
9. إرجاع JSON فيه: `order_number` , `total` , `items summary` .

أمثلة استعلامية (SQL / Eloquent)

- **جلب كارت المستخدم مع تفاصيل المنتج**:

```
\$cart = Cart::with('product')->where('user_id', \$userId)->get();
```

- **قفل المنتج وخصم الـ stock (داخل تTransaction)**:

```
DB::transaction(function() use (\$productId, \$qty) {
    \$product = Product::where('id', \$productId)->lockForUpdate()->first();
    if (!\$product || \$product->quantity < \$qty) {
        throw new \Exception('Not enough stock');
    }
    \$product->decrement('quantity', \$qty);
});
```

- **إنشاء order مع order_items** (مبسّط):

```
\$order = Order::create([...]);
foreach (\$items as \$it) {
    OrderItem::create(array_merge(\$it, ['order_id' => \$order->id]));
}
```

حالات الـ Edge / Error Handling التي يجب تغطيتها

- المنتج محذوف أو غير موجود في وقت إنشاء الأوردر (تُرجع 404).
- عدم توافر كمية كافية (تُرجع 400 مع رسالة واضحة).
- محاولة إضافة منتج في الكارت بعد أن يصبح `is_active=false` .
- سباق على الـ stock — حلّها بالـ `lockForUpdate()` أو باستخدام صفقة `select ... for update` .
- التحقق من صلاحيات المستخدم: الوصول لـ cart وordersg يتطلب تحقق JWT.

اختبارات مقترحة (tests)

1. **Unit tests** على موديل Product.checkStock() وعمليات decrement.
2. **Feature tests**:
3. إضافة عنصر للكارت، تحديث الكمية، حذف العنصر.
4. إنشاء الأوردر: نجاح المسار (decrements, clears cart, returns order_number).
5. محاولة إنشاء أوردر عندما المخزون غير كافٍ (assert 400).

ملاحظات نهائية

- إن أردت أن أعدّل المايجريشنز لربط `orders` بـ `users` أو أضيف `shopping_carts` منفصل، أقدر أكتب لك الـ migration و الـ models.
- لو تحب، أحول هذا المخطط إلى **Mermaid diagram** لوضعه مباشرة داخل `README.md` كنص (سهل القراءة ويدعم GitHub). أيضاً أقدر أعمل صورة عالية الجودة للـ diagram.

هل تريد التحويل إلى Mermaid diagram لنسخه في README؟ أو أعمل التغييرات التالية مباشرة في الكود:
(ضع علامة ✓ على الخيارات): - [] أضيف `user_id` إلى `orders` - [] أعمل جدول `shopping_carts` منفصل و `cart_items` - [] أضف قفل الصف عند خصم المخزون (`lockForUpdate()`) في الكود الحالي