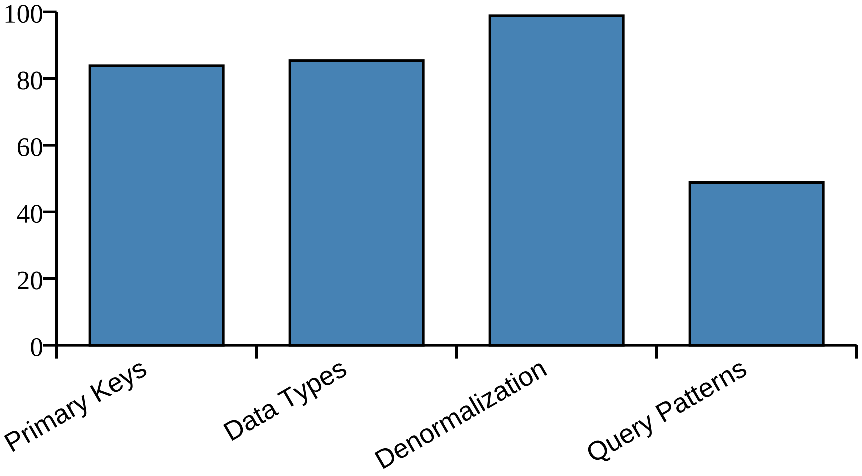
Cassandra Schema Optimization Report

Overall Schema Score: 79.2/100

GOOD: This schema can work with Cassandra but needs moderate optimizations.

Executive Summary

Category	Score	Assessment
Primary Keys	83.8/100	Excellent
Data Types	85.4/100	Excellent
Denormalization	98.8/100	Excellent
Query Patterns	48.8/100	Fair



Schema Overview

Total Tables: 26
Total Columns: 208
Total Relationships: 43
Query Patterns Analyzed: 42

Top Recommendations

Data Types Recommendations

- **products:** Replace decimal(10,2) with a more Cassandra-friendly type
  Consider using bigint with scaled integers instead
  Suggested solution: Convert to 'bigint' and multiply values by 100 to preserve precision
- **products:** Replace decimal(8,2) with a more Cassandra-friendly type
  Consider using bigint with scaled integers instead
  Suggested solution: Convert to 'bigint' and multiply values by 100 to preserve precision
- **product_variants:** Replace decimal(10,2) with a more Cassandra-friendly type
  Consider using bigint with scaled integers instead
  Suggested solution: Convert to 'bigint' and multiply values by 100 to preserve precision

Query Patterns Recommendations

- **user_addresses:** Align table design with query patterns
  Columns frequently used in WHERE clauses (user_id) are not part of the primary key
  Suggested solution: Consider a composite key with 'address_id' and 'user_id' or create a secondary table with 'user_id' as partition key

Many-to-Many Relationships Recommendations

- **product_categories:** Replace junction table with duplicated data
  Junction table 'product_categories' connects products, categories
  Suggested solution: Create a collection in 'categories' to store related 'products' IDs and duplicate data from 'product_categories'
- **wishlist_items:** Replace junction table with duplicated data
  Junction table 'wishlist_items' connects wishlist, products
  Suggested solution: Create a collection in 'products' to store related 'wishlist' IDs and duplicate data from 'wishlist_items'
- **promotion_categories:** Replace junction table with duplicated data
  Junction table 'promotion_categories' connects promotions, categories
  Suggested solution: Create a collection in 'categories' to store related 'promotions' IDs and duplicate data from 'promotion_categories'

Hierarchical Data Recommendations

- **categories:** Restructure hierarchical data
  Table 'categories' has a self-reference on column 'parent_category_id'
  Suggested solution: For hierarchical data in 'categories', consider: 1) Materialized paths: store the full path to each node; 2) Adjacency lists: store all children IDs in a collection; 3) Nested sets: store left/right indexes for efficient subtree queries

Detailed Schema Analysis

Table Structure Analysis

Tables with Problematic Data Types for Cassandra:

Table	Column	Current Type	Issue
products	price	decimal(10,2)	Consider using bigint with scaled integers instead
products	weight	decimal(8,2)	Consider using bigint with scaled integers instead
product_variants	price_adjustment	decimal(10,2)	Consider using bigint with scaled integers instead
cart_items	price_at_addition	decimal(10,2)	Consider using bigint with scaled integers instead
orders	total_amount	decimal(10,2)	Consider using bigint with scaled integers instead
orders	tax_amount	decimal(10,2)	Consider using bigint with scaled integers instead
orders	shipping_amount	decimal(10,2)	Consider using bigint with scaled integers instead
orders	discount_amount	decimal(10,2)	Consider using bigint with scaled integers instead
order_items	price	decimal(10,2)	Consider using bigint with scaled integers instead
order_items	discount	decimal(10,2)	Consider using bigint with scaled integers instead
order_items	tax	decimal(10,2)	Consider using bigint with scaled integers instead
order_items	total	decimal(10,2)	Consider using bigint with scaled integers instead
payments	amount	decimal(10,2)	Consider using bigint with scaled integers instead
promotions	discount_value	decimal(10,2)	Consider using bigint with scaled integers instead
promotions	minimum_order_amount	decimal(10,2)	Consider using bigint with scaled integers instead
coupons	discount_value	decimal(10,2)	Consider using bigint with scaled integers instead
coupons	minimum_order_amount	decimal(10,2)	Consider using bigint with scaled integers instead
coupon_usages	discount_amount	decimal(10,2)	Consider using bigint with scaled integers instead
price_history	price	decimal(10,2)	Consider using bigint with scaled integers instead

Relationship Analysis

Tables with High Connectivity (potential query complexity):

Table	Incoming Refs	Outgoing Refs	Total
users	10	0	10
categories	3	1	4
products	12	1	13
product_variants	5	1	6
orders	3	3	6
order_items	1	3	4
reviews	1	3	4

Note: Tables with high connectivity often represent good candidates for denormalization in Cassandra.

Access Pattern Analysis

Most Frequently Queried Tables:

Table	Query Count
products	4
product_variants	3
users	2
product_images	2
categories	2

Most Common WHERE Conditions:

Column	Frequency
user_id	3
product_id	3
category_id	1
status	1
order_id	1

Note: Columns frequently used in WHERE clauses should be considered for partition keys in Cassandra.

Cassandra Best Practices Scorecard

Primary Key Design

Score: 83.8/100

Table	Primary Key Structure	Score	Issues
users	user_id	80/100	Single-column primary key is OK but could be improved with composite
user_addresses	address_id	80/100	Single-column primary key is OK but could be improved with composite
categories	category_id	80/100	Single-column primary key is OK but could be improved with composite
products	product_id	80/100	Single-column primary key is OK but could be improved with composite
product_categories	product_id, category_id	100/100	Good: Composite primary key
product_images	image_id	80/100	Single-column primary key is OK but could be improved with composite
product_attributes	attribute_id	80/100	Single-column primary key is OK but could be improved with composite
product_variants	variant_id	80/100	Single-column primary key is OK but could be improved with composite
variant_attributes	variant_id, attribute_name	100/100	Good: Composite primary key
carts	cart_id	80/100	Single-column primary key is OK but could be improved with composite
cart_items	cart_item_id	80/100	Single-column primary key is OK but could be improved with composite
orders	order_id	80/100	Single-column primary key is OK but could be improved with composite
order_items	order_item_id	80/100	Single-column primary key is OK but could be improved with composite
payments	payment_id	80/100	Single-column primary key is OK but could be improved with composite
reviews	review_id	80/100	Single-column primary key is OK but could be improved with composite
review_images	image_id	80/100	Single-column primary key is OK but could be improved with composite
wishlist	wishlist_id	80/100	Single-column primary key is OK but could be improved with composite
wishlist_items	wishlist_id, product_id	100/100	Good: Composite primary key
promotions	promotion_id	80/100	Single-column primary key is OK but could be improved with composite
promotion_categories	promotion_id, category_id	100/100	Good: Composite primary key
promotion_products	promotion_id, product_id	100/100	Good: Composite primary key
coupons	coupon_id	80/100	Single-column primary key is OK but could be improved with composite
coupon_usages	usage_id	80/100	Single-column primary key is OK but could be improved with composite
inventory_transactions	transaction_id	80/100	Single-column primary key is OK but could be improved with composite
price_history	history_id	80/100	Single-column primary key is OK but could be improved with composite
product_views	view_id	80/100	Single-column primary key is OK but could be improved with composite

Cassandra Primary Key Best Practices:

- Partition keys should distribute data evenly across nodes
- Avoid high-cardinality partition keys to prevent hotspots
- Use composite keys (partition key + clustering columns) for efficient data retrieval
- Order clustering columns based on query patterns
- Keep related data in the same partition to minimize reads

Data Type Selection

Score: 85.4/100
Cassandra Data Type Best Practices:

- Use text instead of varchar for string data
- Prefer bigint over decimal for numeric values requiring precision
- Use collections (list, set, map) for small groups of related data
- Use UUID type for globally unique identifiers
- Avoid using floating-point types for exact calculations

Denormalization Strategies

Score: 98.8/100
Cassandra Denormalization Best Practices:

- Design tables around query patterns, not entity relationships
- Duplicate data across tables to minimize joins
- Use collections for one-to-few relationships
- Create separate tables for each query pattern
- Accept data duplication to optimize read performance

Query Pattern Alignment

Score: 48.8/100
Cassandra Query Pattern Best Practices:

- Design tables based on specific query requirements
- Include all filtering columns in primary key
- Order clustering columns based on sorting needs
- Create separate tables for different access patterns
- Avoid secondary indexes except for low-cardinality columns