<https://claude.ai/share/f57dd4ad-13ec-4e2f-a31c-8bcfb39cbc09> important link for omplete db

Perfect! I've created a com

prehensive database schema and architecture documentation for your InterPay payment system. Here's what I've delivered:

**📦 What's Included:**

**1. Complete Database Schema (8 Microservices)**

* **User Service**: Authentication, profiles, KYC
* **Wallet Service**: Balance management, transactions, holds
* **Payment Service**: Payment processing, cards, refunds
* **Transfer Service**: P2P transfers, payouts
* **Merchant Service**: Merchant accounts, API keys, settlements
* **Fraud Detection**: Rule engine, risk scoring, blacklists
* **Notification Service**: Multi-channel notifications
* **Audit Service**: Comprehensive audit logging

**2. NoSQL Implementations**

* **Cassandra**: Session storage, logs, time-series analytics
* **Redis**: Caching patterns, rate limiting, distributed locks

**3. Architecture Features**

✅ **Consistency & Reliability**:

* Optimistic locking with version columns
* Saga pattern for distributed transactions
* Idempotency for payment operations
* Double-entry bookkeeping for wallets

✅ **Scalability**:

* Database sharding strategies
* Read replicas for load distribution
* Connection pooling
* Partition strategies for large tables

✅ **Performance**:

* Multi-level caching (Memory → Redis → DB)
* Composite indexes for common queries
* Materialized views for analytics
* Query optimization patterns

✅ **Security**:

* PCI DSS compliance (tokenization)
* Encryption at rest and in transit
* JWT authentication
* Audit logging

✅ **Fraud Prevention**:

* Real-time risk scoring
* Velocity checks
* Device fingerprinting
* Rule-based engine

**🎯 Key Design Decisions:**

1. **Low Latency**: Redis caching, indexed queries, connection pooling
2. **High Availability**: Read replicas, multi-region Cassandra
3. **Data Consistency**: ACID transactions within services, eventual consistency across services
4. **Fault Tolerance**: Circuit breakers, retry mechanisms, graceful degradation

Would you like me to:

1. Create API endpoint specifications for these services?
2. Design the event-driven message flow between services?
3. Create deployment configurations (Docker, Kubernetes)?
4. Add more specific fraud detection algorithms?