**Cursor Prompts for MINGUS Database Migration**

**📋 PHASE 1: DATABASE ASSESSMENT (Day 1)**

**Prompt 1: Create Database Assessment Script**

Create a Python script called `assess\_databases.py` that will analyze my current MINGUS SQLite databases and generate a comprehensive assessment report.

Requirements:

- Analyze these 5 databases: mingus.db, business\_intelligence.db, cache.db, performance\_metrics.db, alerts.db

- For each database, extract table names, schemas, and record counts

- Identify potential table name conflicts between databases

- Generate a JSON report with the assessment results

- Include error handling for missing databases

- Print a summary to console showing total records per database

The script should help me understand what data I have before migrating to PostgreSQL. Make it robust and handle cases where some databases might not exist.

**Prompt 2: Create Conflict Analysis Script**

Create a Python script called `conflict\_analysis.py` that analyzes potential conflicts when consolidating my 5 SQLite databases into a single PostgreSQL database.

Requirements:

- Read the assessment data from `database\_assessment.json` (created by previous script)

- Identify table name conflicts across different databases

- Check for schema conflicts (same table name, different structures)

- Suggest resolution strategies for conflicts

- Generate a conflict resolution plan

- Output both console summary and detailed JSON report

Focus on identifying issues that would prevent a clean migration, such as duplicate table names or incompatible schemas.

**📋 PHASE 2: POSTGRESQL SCHEMA DESIGN (Day 2)**

**Prompt 3: Create Unified PostgreSQL Schema**

Create a comprehensive PostgreSQL schema file called `unified\_schema.sql` for the MINGUS personal finance application.

Requirements:

- Design for a financial wellness app targeting African American professionals

- Include these core table categories:

\* User management (users, user\_profiles)

\* Subscription/billing system (subscriptions, feature\_access, billing\_history)

\* Financial data (encrypted\_financial\_profiles, user\_income\_due\_dates, user\_expense\_due\_dates)

\* Health tracking (user\_health\_checkins, health\_spending\_correlations)

\* Career data (job\_security\_analysis)

\* Analytics (user\_analytics, performance\_metrics)

\* System management (system\_alerts, important\_dates)

- Use PostgreSQL-specific features:

\* UUID primary keys

\* JSONB for flexible data

\* Proper foreign key relationships

\* Timestamps with timezone

\* Decimal types for financial data

\* Constraints and indexes

- Include missing critical fields identified in the analysis:

\* first\_name, last\_name, zip\_code, dependents in user\_profiles

\* Complete subscription management system

\* Feature access control with usage limits

\* Enhanced health and career tracking

Make this production-ready with proper constraints, indexes, and relationships.

**Prompt 4: Create SQLAlchemy Models**

Create SQLAlchemy models in a `models/` directory that correspond to the PostgreSQL schema for the MINGUS application.

Requirements:

- Create separate model files: user.py, subscription.py, health.py, financial.py, analytics.py

- Use PostgreSQL-specific SQLAlchemy types (UUID, JSONB, etc.)

- Include proper relationships between models

- Add validation methods and properties

- Include these critical models:

\* User (with profile relationship)

\* UserProfile (with missing fields: first\_name, last\_name, zip\_code, dependents)

\* Subscription (new - for billing system)

\* FeatureAccess (new - for tier-based access control)

\* UserHealthCheckin (enhanced health tracking)

\* EncryptedFinancialProfile (secure financial data)

\* UserAnalytics (business intelligence)

- Create an \_\_init\_\_.py file that imports all models and sets up the database connection

- Include methods for common operations (create, update, delete)

- Add string representations and JSON serialization methods

Make these models production-ready with proper validation and security considerations.

**📋 PHASE 3: DATA MIGRATION SCRIPTS (Day 2-3)**

**Prompt 5: Create Data Migration System**

Create a comprehensive data migration system to move data from my 5 SQLite databases to the new PostgreSQL database.

Requirements:

- Create a main migration class `DatabaseMigrator` in `data\_migration.py`

- Handle these databases: mingus.db, business\_intelligence.db, cache.db, performance\_metrics.db, alerts.db

- Map SQLite tables to PostgreSQL equivalents

- Handle data type conversions (UUID, datetime, JSON)

- Include field mapping for renamed/restructured fields

- Handle missing data gracefully

- Provide detailed logging and progress tracking

- Include rollback capabilities

- Generate migration reports

Key features needed:

- SQLite to PostgreSQL connection management

- Batch processing for large datasets

- Data validation during migration

- Conflict resolution for duplicate data

- Support for encrypted financial data migration

The migrator should be able to run the entire migration automatically while providing detailed feedback on what's happening.

**Prompt 6: Create Migration Validation Script**

Create a validation script called `validate\_migration.py` that verifies the database migration was successful.

Requirements:

- Connect to the new PostgreSQL database

- Verify all expected tables exist

- Check record counts match source databases

- Validate data integrity (foreign keys, constraints)

- Test sample queries to ensure data is accessible

- Verify relationships between tables work correctly

- Check for any data corruption or loss

- Generate a comprehensive validation report

Include these specific tests:

- User profile completeness

- Financial data integrity

- Health checkin data accuracy

- Subscription system functionality

- Feature access control validation

- Performance benchmarking of key queries

Output both console results and a detailed JSON validation report.

**📋 PHASE 4: APPLICATION CONFIGURATION (Day 3-4)**

**Prompt 7: Update Application Configuration**

Update the MINGUS application configuration to use PostgreSQL instead of SQLite.

Requirements:

- Create/update `config.py` with PostgreSQL connection strings

- Support both development (local) and production (Digital Ocean) environments

- Include proper connection pooling and SSL configuration

- Add environment variable management for database credentials

- Configure SQLAlchemy for PostgreSQL optimization

- Include backup and monitoring configuration

- Add security configurations for production

Key configurations needed:

- Database URLs for different environments

- Connection pool settings optimized for financial app workload

- SSL/security settings for production

- Logging configuration for database operations

- Performance tuning parameters

Make this production-ready with proper security and performance considerations.

**Prompt 8: Create Database Initialization System**

Create a database initialization system that sets up the PostgreSQL database with initial data and configurations.

Requirements:

- Create `init\_db.py` script that initializes the database

- Set up all tables using SQLAlchemy models

- Create default subscription tiers (Budget $10, Mid-tier $20, Professional $50)

- Initialize feature access configurations

- Set up default system settings

- Create admin user if needed

- Include data seeding for testing

- Add database health checks

The initialization should:

- Create all tables and relationships

- Populate pricing tiers with proper features and limits

- Set up feature access control rules

- Configure system defaults

- Verify everything is working correctly

This should be idempotent - safe to run multiple times without causing issues.

**📋 PHASE 5: TESTING & PERFORMANCE (Day 4-5)**

**Prompt 9: Create Performance Testing Suite**

Create a comprehensive performance testing suite for the new PostgreSQL database setup.

Requirements:

- Create `performance\_test.py` with realistic MINGUS application queries

- Test these key operations:

\* User registration and login

\* Financial profile updates

\* Health checkin submissions

\* Cash flow calculations

\* Income comparison queries

\* Subscription status checks

\* Feature access validation

- Include these test scenarios:

\* Single user operations

\* Concurrent user simulation

\* Bulk data operations

\* Complex analytical queries

\* Database under load

- Measure and report:

\* Query execution times

\* Connection pool performance

\* Memory usage

\* CPU utilization

\* Concurrent user capacity

Target performance: <500ms for 95% of queries, support for 100+ concurrent users.

Generate detailed performance reports with recommendations for optimization.

**Prompt 10: Create Migration Testing Script**

Create a comprehensive testing script called `test\_migration.py` that validates the entire migration and new database setup.

Requirements:

- Test data integrity between old SQLite and new PostgreSQL

- Verify all user data migrated correctly

- Test subscription system functionality

- Validate feature access control

- Test financial calculations work correctly

- Verify health tracking system

- Test all API endpoints with new database

- Include automated regression testing

Test categories:

- Data consistency tests

- Functional tests for key features

- Performance regression tests

- Security validation tests

- User workflow end-to-end tests

The script should run automatically and generate a pass/fail report for each test category. Include detailed error reporting for any failures.

**📋 PHASE 6: PRODUCTION DEPLOYMENT (Day 5-7)**

**Prompt 11: Create Digital Ocean Deployment Configuration**

Create deployment configuration files for deploying MINGUS to Digital Ocean App Platform with the new PostgreSQL database.

Requirements:

- Create `app-spec.yaml` for Digital Ocean App Platform deployment

- Configure environment variables for production database connection

- Set up proper scaling and resource allocation

- Include health checks and monitoring

- Configure SSL/security settings

- Set up automated deployments from GitHub

Key components:

- Web service configuration optimized for Python/Flask app

- Database connection to Digital Ocean Managed PostgreSQL

- Environment variable management

- Security configurations

- Monitoring and logging setup

- Backup and disaster recovery

Make this production-ready with proper security, monitoring, and scaling capabilities.

**Prompt 12: Create Production Migration Script**

Create a production-safe migration script called `production\_migration.py` that safely migrates data to the live Digital Ocean PostgreSQL database.

Requirements:

- Connect to Digital Ocean Managed PostgreSQL

- Include comprehensive backup procedures

- Implement zero-downtime migration strategy

- Add rollback capabilities

- Include extensive logging and monitoring

- Verify SSL connections and security

- Test database performance post-migration

Safety features:

- Pre-migration database backup

- Transaction-based migration with rollback

- Data validation at each step

- Performance monitoring during migration

- Automated rollback on failure

- Post-migration validation

This script will handle the final migration to production and must be extremely reliable and safe.

**Prompt 13: Create Post-Migration Cleanup System**

Create a cleanup and maintenance system for after the database migration is complete.

Requirements:

- Create `cleanup\_migration.py` for post-migration tasks

- Backup old SQLite databases safely

- Clean up temporary migration files

- Set up ongoing database maintenance tasks

- Configure monitoring and alerting

- Create database health check system

- Set up automated backup verification

Include:

- Automated backup of old databases

- Cleanup of migration artifacts

- Database optimization and indexing

- Monitoring setup for production

- Documentation of new database structure

- Maintenance procedures and schedules

This should ensure the new PostgreSQL system runs smoothly in production with proper monitoring and maintenance.

**📋 PHASE 7: INTEGRATION & VERIFICATION (Day 6-7)**

**Prompt 14: Update Application Routes and Services**

Update all MINGUS application routes and services to work with the new PostgreSQL database and consolidated schema.

Requirements:

- Update all Flask routes to use new SQLAlchemy models

- Modify authentication system for PostgreSQL

- Update subscription and billing integration

- Modify health checkin system

- Update financial analysis services

- Integrate feature access control throughout the app

- Update API endpoints for new schema

Key areas to update:

- User registration and profile management

- Subscription and billing workflows

- Health tracking and correlation services

- Financial planning and forecasting

- Career advancement features

- Analytics and reporting

Ensure all existing functionality works with the new database while maintaining API compatibility.

**Prompt 15: Create End-to-End Testing Suite**

Create a comprehensive end-to-end testing suite called `e2e\_tests.py` that validates the entire MINGUS application with the new PostgreSQL database.

Requirements:

- Test complete user workflows from registration to advanced features

- Validate subscription signup and billing

- Test health checkin and correlation features

- Verify financial planning and forecasting

- Test income comparison lead magnet

- Validate feature access control

- Test API endpoints

- Include browser automation tests if needed

Test scenarios:

- New user registration and onboarding

- Subscription tier selection and payment

- Health data entry and insights

- Financial goal setting and tracking

- Career advancement features

- Administrative functions

The test suite should simulate real user behavior and validate that all features work correctly with the new database system. Include performance testing under realistic user loads.

**🎯 EXECUTION ORDER**

**Copy these prompts into Cursor in sequence:**

**Day 1:**

1. Prompt 1 (Database Assessment)
2. Prompt 2 (Conflict Analysis)
3. Prompt 3 (PostgreSQL Schema)

**Day 2:**

1. Prompt 4 (SQLAlchemy Models)
2. Prompt 5 (Data Migration System)
3. Prompt 6 (Migration Validation)

**Day 3:**

1. Prompt 7 (Application Configuration)
2. Prompt 8 (Database Initialization)
3. Prompt 9 (Performance Testing)

**Day 4:**

1. Prompt 10 (Migration Testing)
2. Prompt 11 (Digital Ocean Deployment)
3. Prompt 12 (Production Migration)

**Day 5:**

1. Prompt 13 (Post-Migration Cleanup)
2. Prompt 14 (Application Integration)
3. Prompt 15 (End-to-End Testing)

**🔧 SUCCESS VALIDATION**

After completing all prompts, you should have:

* ✅ Single PostgreSQL database replacing 5 SQLite databases
* ✅ Complete subscription and billing system
* ✅ Enhanced user profiles with missing critical fields
* ✅ Production-ready deployment on Digital Ocean
* ✅ Comprehensive testing and validation
* ✅ Zero data loss during migration
* ✅ Performance meeting <500ms query targets

**This gets you from "95% complete" to "production-ready with revenue generation capability" in 5-7 days.**