Credit System Database Migrations - Application Guide

Overview

This guide walks you through applying the credit/billing system migrations to your VerifyLens database in Supabase.

Migration Files:
- `004_create_credit_system_tables.sql` - Creates core credit tables
- `005_add_stripe_customer_id_to_customers.sql` - Adds Stripe integration column
- `006_create_credit_indexes.sql` - Creates performance indexes
- `007_seed_credit_packages.sql` - Inserts initial credit packages

Total Estimated Time: 5-10 minutes

Prerequisites

Before starting, ensure you have:
- ✅ Access to your Supabase project dashboard
- ✅ SQL Editor permissions in Supabase
- ✅ Backup of your database (recommended)
- ✅ All previous migrations (001-003) already applied

Migration Order (MUST be followed)

IMPORTANT: Run migrations in this exact order. Each migration depends on the previous ones.

1. **004** → Create credit tables
2. **005** → Add Stripe customer ID
3. **006** → Create indexes
4. **007** → Seed credit packages

Step-by-Step Instructions

Option 1: Using Supabase SQL Editor (Recommended)

This is the easiest and safest method.

Step 1: Open Supabase SQL Editor

1. Go to your Supabase project dashboard: https://app.supabase.com
2. Navigate to **SQL Editor** in the left sidebar
3. Click **New Query** button

Step 2: Apply Migration 004 - Create Credit Tables

1. Open the file: `database/migrations/004_create_credit_system_tables.sql`
2. Copy ALL contents of the file
3. Paste into the Supabase SQL Editor
4. Click **Run** (or press Ctrl+Enter / Cmd+Enter)
5. Wait for completion (should take 2-3 seconds)
6. ✅ Verify success message appears:

```
NOTICE: ✓ Migration 004 completed successfully
   NOTICE: ✓ Created tables: credit_packages, customer_credits, credit_transactions,
stripe_payments
   NOTICE: → Next: Run migration 005 to add stripe_customer_id to customers table
```

If you see an error: Stop here and troubleshoot before proceeding.

Step 3: Apply Migration 005 - Add Stripe Customer ID

1. Create a **new query** in Supabase SQL Editor
2. Open the file: `database/migrations/005_add_stripe_customer_id_to_customers.sql`
3. Copy ALL contents of the file
4. Paste into the SQL Editor
5. Click **Run**
6. ✅ Verify success message:

```
NOTICE: ✓ Migration 005 completed successfully
   NOTICE: ✓ Added column: customers.stripe_customer_id (VARCHAR(255) UNIQUE)
   NOTICE: → Next: Run migration 006 to create performance indexes
```

Step 4: Apply Migration 006 - Create Indexes

1. Create a **new query** in Supabase SQL Editor
2. Open the file: `database/migrations/006_create_credit_indexes.sql`
3. Copy ALL contents of the file
4. Paste into the SQL Editor
5. Click **Run**
6. ✅ Verify success message (should mention number of indexes created):

```
NOTICE: ✓ Migration 006 completed successfully
   NOTICE: ✓ Created XX indexes for credit system tables
   NOTICE: → Next: Run migration 007 to seed initial credit packages
```

Step 5: Apply Migration 007 - Seed Credit Packages

1. Create a **new query** in Supabase SQL Editor
2. Open the file: `database/migrations/007_seed_credit_packages.sql`
3. Copy ALL contents of the file
4. Paste into the SQL Editor
5. Click **Run**
6. ✅ Verify success message shows the 4 credit packages:

```
NOTICE: ✓ Migration 007 completed successfully
   NOTICE: ✓ Credit packages seeded: 4 active packages
   NOTICE:
   NOTICE: Available Credit Packages:
   NOTICE:   Starter Pack: 10 credits = $1000
   NOTICE:   Professional Pack: 50 credits = $5000
   NOTICE:   Business Pack: 100 credits = $10000
   NOTICE:   Enterprise Pack: 200 credits = $20000
   NOTICE:
   NOTICE: → All credit system migrations completed!
```

Option 2: Using psql Command Line

If you prefer command line:

```
# Set your database URL
export DATABASE_URL="postgresql://postgres:[YOUR-PASSWORD]@db.[YOUR-PROJECT-
REF].supabase.co:5432/postgres"

# Navigate to migrations folder
cd /home/ubuntu/github_repos/roblox-tool/database/migrations

# Run migrations in order
psql "$DATABASE_URL" -f 004_create_credit_system_tables.sql
psql "$DATABASE_URL" -f 005_add_stripe_customer_id_to_customers.sql
psql "$DATABASE_URL" -f 006_create_credit_indexes.sql
psql "$DATABASE_URL" -f 007_seed_credit_packages.sql
```

Verification Steps

After applying all migrations, verify everything is correct:

1. Verify Tables Exist

Run this query in Supabase SQL Editor:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
  AND table_name IN (
    'credit_packages',
    'customer_credits',
    'credit_transactions',
    'stripe_payments'
  )
ORDER BY table_name;
```

Expected Result: 4 rows (all 4 tables)

2. Verify Stripe Column Exists

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'customers'
  AND column_name = 'stripe_customer_id';
```

Expected Result:

column_name	data_type	is_nullable
stripe_customer_id	character varying	YES

3. Verify Indexes Were Created

```sql
SELECT
    tablename,
    indexname,
    indexdef
FROM pg_indexes
WHERE tablename IN (
    'credit_packages',
    'customer_credits',
    'credit_transactions',
    'stripe_payments'
)
ORDER BY tablename, indexname;
```

Expected Result: Multiple indexes (15-20 rows)

4. Verify Credit Packages Were Seeded

```sql
SELECT
    name,
    credits,
    price_cents,
    (price_cents::FLOAT / 100) as price_dollars,
    is_active
FROM credit_packages
ORDER BY credits;
```

Expected Result:

name	credits	price_cents	price_dollars	is_active
Starter Pack	10	100000	1000	true
Professional Pack	50	500000	5000	true
Business Pack	100	1000000	10000	true
Enterprise Pack	200	2000000	20000	true

5. Test Foreign Key Relationships

```sql
-- Test cascade relationships
SELECT
    tc.table_name,
    tc.constraint_name,
    tc.constraint_type,
    kcu.column_name,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM information_schema.table_constraints tc
JOIN information_schema.key_column_usage kcu
    ON tc.constraint_name = kcu.constraint_name
JOIN information_schema.constraint_column_usage ccu
    ON ccu.constraint_name = tc.constraint_name
WHERE tc.constraint_type = 'FOREIGN KEY'
  AND tc.table_name IN (
      'customer_credits',
      'credit_transactions',
      'stripe_payments'
  )
ORDER BY tc.table_name, tc.constraint_name;
```

Expected Result: Multiple foreign key constraints linking to customers, users, search_history, and stripe_payments tables.

Database Schema Summary

After all migrations, your database will have these **new tables**:

credit_packages

Purpose: Available credit packages for purchase
Key Columns:
- `id` - Package ID
- `name` - Package name (e.g., “Starter Pack”)
- `credits` - Number of credits in package
- `price_cents` - Price in cents (divide by 100 for dollars)
- `is_active` - Whether package is available

customer_credits

Purpose: Track credit balance for each customer
Key Columns:
- `customer_id` - FK to customers (UNIQUE)
- `balance` - Current available credits
- `total_purchased` - Lifetime credits purchased
- `total_used` - Lifetime credits used
- `last_purchase_at` - Last purchase timestamp

Important Constraint: `balance = total_purchased - total_used` (enforced by CHECK constraint)

credit_transactions

Purpose: Immutable log of all credit operations
Key Columns:
- `customer_id` - Customer who owns the transaction
- `user_id` - User who performed the action
- `transaction_type` - PURCHASE, USAGE, REFUND, ADJUSTMENT, PROMO
- `amount` - Credit change (positive or negative)
- `balance_before` - Balance before transaction
- `balance_after` - Balance after transaction
- `search_history_id` - FK to search (for USAGE)
- `payment_id` - FK to stripe_payments (for PURCHASE)

Important Constraint: `balance_after = balance_before + amount` (enforced by CHECK constraint)

stripe_payments

Purpose: Payment transaction records from Stripe
Key Columns:
- `customer_id` - Customer who made payment
- `user_id` - User who initiated payment
- `stripe_payment_intent_id` - Stripe PaymentIntent ID (UNIQUE)
- `stripe_customer_id` - Stripe Customer ID
- `amount_cents` - Payment amount in cents
- `status` - pending, processing, succeeded, failed, canceled, refunded
- `credits_purchased` - Credits bought with this payment

customers (modified)

New Column:
- `stripe_customer_id` - Stripe Customer ID (cus_xxx) - UNIQUE, nullable

Common Issues & Troubleshooting

Issue: “relation already exists”

Cause: Migration was already run or tables already exist.

Solution: If you’re sure the tables are correct, you can skip that migration. Otherwise, drop the tables first:

```
DROP TABLE IF EXISTS credit_transactions CASCADE;
DROP TABLE IF EXISTS stripe_payments CASCADE;
DROP TABLE IF EXISTS customer_credits CASCADE;
DROP TABLE IF EXISTS credit_packages CASCADE;
ALTER TABLE customers DROP COLUMN IF EXISTS stripe_customer_id;
```

Then re-run all migrations.

Issue: “column already exists” (for migration 005)

Cause: stripe_customer_id column was already added.

Solution: Safe to skip migration 005 if the column already exists. Verify with:

```sql
SELECT column_name FROM information_schema.columns
WHERE table_name = 'customers' AND column_name = 'stripe_customer_id';
```

Issue: Migration takes too long / times out

Cause: Large database or network issues.

Solution:
1. Run migrations one by one with breaks in between
2. Check your internet connection to Supabase
3. Try running during off-peak hours

Issue: Foreign key constraint violation

Cause: Data integrity issue in existing tables.

Solution: Run this to check for orphaned records:

```sql
-- Check for users without customers
SELECT id, username, customer_id
FROM users
WHERE role != 'SUPER_ADMIN'
  AND customer_id IS NULL;

-- Check for invalid customer references in search_history
SELECT DISTINCT sh.customer_id
FROM search_history sh
LEFT JOIN customers c ON sh.customer_id = c.id
WHERE c.id IS NULL AND sh.customer_id IS NOT NULL;
```

Fix any orphaned records before applying migrations.

Testing the Credit System

After migrations are complete, you can test the system:

1. Initialize a Customer with Credits

```sql
-- Give a test customer 100 credits
INSERT INTO customer_credits (customer_id, balance, total_purchased, total_used)
VALUES (
    2, -- Replace with actual customer ID
    100, -- Starting balance
    100, -- Total purchased
    0 -- Total used
);
```

2. Record a Test Purchase

```sql
BEGIN;

-- Create a test payment record
INSERT INTO stripe_payments (
    customer_id, user_id, stripe_payment_intent_id,
    amount_cents, status, credits_purchased
) VALUES (
    2, -- Customer ID
    6, -- User ID
    'pi_test_12345', -- Test payment intent
    500000, -- $5000 in cents
    'succeeded',
    50 -- Credits purchased
) RETURNING id;

-- Record the credit transaction (use payment ID from above)
INSERT INTO credit_transactions (
    customer_id, user_id, transaction_type, amount,
    balance_before, balance_after, payment_id
) VALUES (
    2, 6, 'PURCHASE', 50,
    100, 150, 1 -- Replace 1 with actual payment_id
);

-- Update customer credit balance
UPDATE customer_credits
SET balance = 150,
    total_purchased = 150,
    last_purchase_at = CURRENT_TIMESTAMP
WHERE customer_id = 2;

COMMIT;
```

3. Verify Transaction

```sql
-- Check customer credits
SELECT * FROM customer_credits WHERE customer_id = 2;

-- Check transaction history
SELECT * FROM credit_transactions WHERE customer_id = 2;

-- Check payment records
SELECT * FROM stripe_payments WHERE customer_id = 2;
```

Next Steps

After successfully applying migrations:

1. ✅ **Update Application Code**
   - Implement credit balance checks before searches
   - Add credit deduction logic for searches
   - Build Stripe payment integration
   - Create credit purchase UI

2. ✅ **Set Up Stripe Integration**
   - Create Stripe account (if not already)
   - Get API keys (test mode first)
   - Configure webhook endpoints
   - Test payment flow
3. ✅ **Add Credit Management Features**
   - Credit balance display on dashboard
   - Purchase credits page
   - Transaction history page
   - Low balance notifications
4. ✅ **Testing**
   - Test credit purchases (Stripe test mode)
   - Test credit deductions on searches
   - Test refund flows
   - Verify transaction logging

Rollback Instructions

If you need to rollback these migrations:

```sql
-- WARNING: This will delete all credit data!
BEGIN;

-- Drop foreign key from credit_transactions first
ALTER TABLE credit_transactions DROP CONSTRAINT IF EXISTS cred-
it_transactions_payment_id_fkey;

-- Drop tables in reverse dependency order
DROP TABLE IF EXISTS credit_transactions CASCADE;
DROP TABLE IF EXISTS stripe_payments CASCADE;
DROP TABLE IF EXISTS customer_credits CASCADE;
DROP TABLE IF EXISTS credit_packages CASCADE;

-- Drop indexes
DROP INDEX IF EXISTS idx_customers_stripe_customer_id;

-- Remove column from customers
ALTER TABLE customers DROP COLUMN IF EXISTS stripe_customer_id;

COMMIT;
```

⚠️ **WARNING:** This will permanently delete all credit packages, customer balances, transactions, and payment records. Only use this if you're absolutely sure!

Support

If you encounter any issues during migration:

1. Check the error message carefully

2. Review the troubleshooting section above
3. Verify prerequisites are met
4. Check Supabase logs for detailed errors
5. Ensure you’re running migrations in the correct order

Migration Checklist

Use this checklist to track your progress:

- [ ] Backup database
- [ ] Verify prerequisites
- [ ] Apply migration 004 (credit tables)
- [ ] Verify migration 004 success
- [ ] Apply migration 005 (Stripe column)
- [ ] Verify migration 005 success
- [ ] Apply migration 006 (indexes)
- [ ] Verify migration 006 success
- [ ] Apply migration 007 (seed packages)
- [ ] Verify migration 007 success
- [ ] Run all verification queries
- [ ] Test credit system with sample data
- [ ] Update application code
- [ ] Configure Stripe integration

Congratulations! 🎉 Your credit/billing system is now ready for VerifyLens!

Document Version: 1.0
Last Updated: October 28, 2025
Migration Files: 004-007
Database: PostgreSQL on Supabase