Database Schema Updates for Phase 2

Supabase SQL Migration Scripts | FSC Portal

■ Overview

This guide provides the exact SQL commands to update your FSC Portal database for revenue cycle management. Run these in Supabase SQL Editor in the order shown.

1■■ Update Sessions Table - Add Status Tracking

```
-- Add new columns to sessions table
ALTER TABLE public.sessions
ADD COLUMN IF NOT EXISTS billing_status TEXT DEFAULT 'completed',
ADD COLUMN IF NOT EXISTS date_submitted TIMESTAMPTZ,
ADD COLUMN IF NOT EXISTS date_paid TIMESTAMPTZ,
ADD COLUMN IF NOT EXISTS amount_billed DECIMAL(10,2),
ADD COLUMN IF NOT EXISTS amount_paid DECIMAL(10,2),
ADD COLUMN IF NOT EXISTS denial_reason TEXT,
ADD COLUMN IF NOT EXISTS insurance_portal_id UUID,
ADD COLUMN IF NOT EXISTS payment_reference TEXT,
ADD COLUMN IF NOT EXISTS last_status_change TIMESTAMPTZ DEFAULT NOW();

-- Create enum for billing status
CREATE TYPE billing_status_enum AS ENUM (
  'completed',
  'documented',
  'ready_to_bill',
  'submitted',
  'pending',
  'approved',
  'paid',
  'denied',
  'appealing'
);

-- Update column to use enum
ALTER TABLE public.sessions
ALTER COLUMN billing_status TYPE billing_status_enum
USING billing_status::billing_status_enum;

-- Add check constraint for logical amounts
ALTER TABLE public.sessions
ADD CONSTRAINT amount_paid_lte_billed
CHECK (amount_paid IS NULL OR amount_billed IS NULL OR amount_paid <= amount_billed);
```

2■■ Create Insurance Portals Table

```
-- Track which insurance portal each payer uses
CREATE TABLE IF NOT EXISTS public.insurance_portals (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  portal_name TEXT NOT NULL,
  portal_url TEXT,
  payer_id UUID REFERENCES public.payers(id),
  billing_route TEXT,
  login_instructions TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Add index for faster lookups
CREATE INDEX idx_portals_payer ON public.insurance_portals(payer_id);

-- Enable RLS
ALTER TABLE public.insurance_portals ENABLE ROW LEVEL SECURITY;

-- Policy: Admins and billing can read
CREATE POLICY "Admins and billing can view portals"
ON public.insurance_portals FOR SELECT
USING (
  auth.uid() IN (
    SELECT id FROM public.profiles
    WHERE role IN ('admin', 'billing', 'director')
  )
);
```

3■■ Create Payments Table

```
-- Track actual payments received
CREATE TABLE IF NOT EXISTS public.payments (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  payment_date DATE NOT NULL,
  amount DECIMAL(10,2) NOT NULL,
  check_number TEXT,
  payer_id UUID REFERENCES public.payers(id),
  payment_method TEXT, -- 'check', 'eft', 'credit_card'
  notes TEXT,
  created_at TIMESTAMPTZ DEFAULT NOW(),
  created_by UUID REFERENCES auth.users(id)
);

-- Create junction table for payment-session mapping
CREATE TABLE IF NOT EXISTS public.payment_sessions (
  payment_id UUID REFERENCES public.payments(id) ON DELETE CASCADE,
  session_id UUID REFERENCES public.sessions(id) ON DELETE CASCADE,
  amount_applied DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (payment_id, session_id)
);

-- Indexes
CREATE INDEX idx_payments_date ON public.payments(payment_date);
CREATE INDEX idx_payments_payer ON public.payments(payer_id);
CREATE INDEX idx_payment_sessions_payment ON public.payment_sessions(payment_id);
CREATE INDEX idx_payment_sessions_session ON public.payment_sessions(session_id);

-- Enable RLS
ALTER TABLE public.payments ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.payment_sessions ENABLE ROW LEVEL SECURITY;

-- Policies
CREATE POLICY "Admins and billing can manage payments"
ON public.payments FOR ALL
USING (
  auth.uid() IN (
    SELECT id FROM public.profiles WHERE role IN ('admin', 'billing')
  )
);

CREATE POLICY "Admins and billing can manage payment_sessions"
ON public.payment_sessions FOR ALL
USING (
  auth.uid() IN (
    SELECT id FROM public.profiles WHERE role IN ('admin', 'billing')
  )
);
```

4■■ Create Status History Table (Audit Trail)

```
-- Track every status change for audit purposes
CREATE TABLE IF NOT EXISTS public.session_status_history (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  session_id UUID REFERENCES public.sessions(id) ON DELETE CASCADE,
  old_status TEXT,
  new_status billing_status_enum NOT NULL,
  changed_by UUID REFERENCES auth.users(id),
  changed_at TIMESTAMPTZ DEFAULT NOW(),
  notes TEXT
);

-- Index for fast session lookups
CREATE INDEX idx_status_history_session ON public.session_status_history(session_id);
CREATE INDEX idx_status_history_date ON public.session_status_history(changed_at);

-- Enable RLS
ALTER TABLE public.session_status_history ENABLE ROW LEVEL SECURITY;

-- Policy: Anyone can view history
CREATE POLICY "All authenticated users can view status history"
ON public.session_status_history FOR SELECT
USING (auth.role() = 'authenticated');
```

5■■ Create Alerts Table

```
-- Store system-generated alerts
CREATE TABLE IF NOT EXISTS public.alerts (
  id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
  alert_type TEXT NOT NULL, -- 'aging', 'denial', 'missing_doc', etc
  severity TEXT NOT NULL, -- 'low', 'medium', 'high', 'critical'
  session_id UUID REFERENCES public.sessions(id),
  message TEXT NOT NULL,
  assigned_to UUID REFERENCES public.profiles(id),
  status TEXT DEFAULT 'open', -- 'open', 'acknowledged', 'resolved'
  created_at TIMESTAMPTZ DEFAULT NOW(),
  resolved_at TIMESTAMPTZ,
  resolved_by UUID REFERENCES auth.users(id)
);

-- Indexes
CREATE INDEX idx_alerts_status ON public.alerts(status);
CREATE INDEX idx_alerts_assigned ON public.alerts(assigned_to);
CREATE INDEX idx_alerts_session ON public.alerts(session_id);

-- Enable RLS
ALTER TABLE public.alerts ENABLE ROW LEVEL SECURITY;

-- Policy: Users see their assigned alerts + admins see all
CREATE POLICY "Users see assigned alerts"
ON public.alerts FOR SELECT
USING (
  assigned_to = auth.uid() OR
  auth.uid() IN (SELECT id FROM public.profiles WHERE role = 'admin')
);
```

6■■ Create Helpful Views

```
-- View: Aging Report (sessions by age bracket)
CREATE OR REPLACE VIEW public.aging_report AS
SELECT
  s.id,
  s.date,
  s.billing_status,
  c.name as client_name,
  pr.name as provider_name,
  pa.payer_name,
  s.amount_billed,
  s.amount_paid,
  CASE
    WHEN s.billing_status = 'paid' THEN 0
    WHEN s.date_submitted IS NULL THEN
```

```
      EXTRACT(DAY FROM NOW() - s.date)
    ELSE
      EXTRACT(DAY FROM NOW() - s.date_submitted)
  END as days_outstanding,
  CASE
    WHEN s.billing_status = 'paid' THEN 'Paid'
    WHEN s.date_submitted IS NULL THEN 'Not Submitted'
    WHEN EXTRACT(DAY FROM NOW() - s.date_submitted) <= 30 THEN '0-30 days'
    WHEN EXTRACT(DAY FROM NOW() - s.date_submitted) <= 60 THEN '31-60 days'
    WHEN EXTRACT(DAY FROM NOW() - s.date_submitted) <= 90 THEN '61-90 days'
    ELSE '90+ days'
  END as age_bracket
FROM public.sessions s
LEFT JOIN public.clients c ON s.client_id = c.id
LEFT JOIN public.providers pr ON s.provider_id = pr.id
LEFT JOIN public.payers pa ON s.payer_id = pa.id
WHERE s.billing_status != 'paid';

-- View: Dashboard Metrics
CREATE OR REPLACE VIEW public.dashboard_metrics AS
SELECT
  COUNT(*) FILTER (WHERE billing_status = 'completed') as completed_sessions,
  COUNT(*) FILTER (WHERE billing_status = 'ready_to_bill') as ready_to_bill,
  COUNT(*) FILTER (WHERE billing_status = 'submitted') as submitted_claims,
  COUNT(*) FILTER (WHERE billing_status = 'paid') as paid_sessions,
  COUNT(*) FILTER (WHERE billing_status = 'denied') as denied_claims,
  SUM(amount_billed) FILTER (WHERE billing_status NOT IN ('paid', 'denied')) as outstanding_amount,
  SUM(amount_paid) as total_collected,
  AVG(EXTRACT(DAY FROM date_paid - date_submitted)) FILTER (WHERE date_paid IS NOT NULL) as avg_days_to_payment
FROM public.sessions
WHERE date >= DATE_TRUNC('month', NOW());
```

7■■ Create Automated Triggers

```
-- Function: Auto-create status history entry
CREATE OR REPLACE FUNCTION log_status_change()
RETURNS TRIGGER AS $$
BEGIN
  IF OLD.billing_status IS DISTINCT FROM NEW.billing_status THEN
    INSERT INTO public.session_status_history (
      session_id, old_status, new_status, changed_by
    ) VALUES (
      NEW.id,
      OLD.billing_status::TEXT,
      NEW.billing_status,
      auth.uid()
    );

    NEW.last_status_change = NOW();
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Attach trigger
CREATE TRIGGER sessions_status_change
BEFORE UPDATE ON public.sessions
FOR EACH ROW
EXECUTE FUNCTION log_status_change();

-- Function: Auto-generate alerts for aging claims
CREATE OR REPLACE FUNCTION check_aging_claims()
RETURNS void AS $$
BEGIN
  -- Alert for claims >30 days without update
  INSERT INTO public.alerts (alert_type, severity, session_id, message, assigned_to)
  SELECT
    'aging_claim',
    CASE
      WHEN EXTRACT(DAY FROM NOW() - date_submitted) > 90 THEN 'critical'
      WHEN EXTRACT(DAY FROM NOW() - date_submitted) > 60 THEN 'high'
      ELSE 'medium'
    END,
    id,
    'Claim has been pending for ' ||
      EXTRACT(DAY FROM NOW() - date_submitted) || ' days',
    (SELECT id FROM public.profiles WHERE role = 'billing' LIMIT 1)
  FROM public.sessions
  WHERE billing_status = 'submitted'
    AND date_submitted < NOW() - INTERVAL '30 days'
    AND id NOT IN (
      SELECT session_id FROM public.alerts
      WHERE alert_type = 'aging_claim' AND status = 'open'
    );
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Schedule: Run daily (configure in Supabase Dashboard -> Database -> Cron)
-- SELECT cron.schedule('check-aging-claims', '0 8 * * *', 'SELECT check_aging_claims()');
```

8■■ Update Existing Data (Migration)

```
-- Backfill existing sessions with default status
UPDATE public.sessions
SET billing_status = 'completed'
WHERE billing_status IS NULL;

-- If you have sessions with "submitted" in notes, mark them
UPDATE public.sessions
SET billing_status = 'submitted'
WHERE status = 'submitted' OR notes ILIKE '%submitted%';

-- Mark sessions with payments as paid (if you track this)
UPDATE public.sessions s
SET
  billing_status = 'paid',
  date_paid = NOW()
WHERE EXISTS (
  SELECT 1 FROM public.payment_sessions ps
  WHERE ps.session_id = s.id
);
```

■ Verification Queries

```
-- Check all tables were created
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public'
  AND table_name IN (
    'insurance_portals', 'payments', 'payment_sessions',
    'session_status_history', 'alerts'
  );

-- Check sessions table has new columns
SELECT column_name, data_type
FROM information_schema.columns
WHERE table_name = 'sessions'
  AND column_name IN (
    'billing_status', 'date_submitted', 'date_paid',
    'amount_billed', 'amount_paid'
  );

-- Test views
SELECT * FROM public.dashboard_metrics LIMIT 1;
SELECT * FROM public.aging_report LIMIT 5;
```

■ Next Steps After Running SQL

1. Run each SQL block in order in Supabase SQL Editor 2. Verify all tables created successfully 3. Test the views with SELECT statements 4. Update your CSV import script to set initial billing_status 5. Build UI for status updates (buttons to change status) 6. Create forms for logging payments 7. Build dashboard to show aging_report view 8. Set up email alerts for critical aging claims