## Contents


**SECTION 45: VERSIONED SUBSCRIPTIONS & GRANDFATHERING (v4.15.0)** **1**

**1**

45.1 GRANDFATHERING PRINCIPLES . . . . . 1
45.2 DATABASE SCHEMA . . . . . 1
packages/infrastructure/migrations/045_versioned_subscriptions.sql . . . . . 1
45.3 GRANDFATHERING SERVICE . . . . . 4
packages/functions/src/services/grandfathering.ts . . . . . 4

**7**


# SECTION 45: VERSIONED SUBSCRIPTIONS & GRANDFATHERING (v4.15.0)

**Version: 4.15.0 | Preserves original pricing/features when plans change**

---

## 45.1 GRANDFATHERING PRINCIPLES

- **Existing subscribers keep original terms** when plans change
- **Price increases don't affect** current subscribers
- **Migration offers with incentives** encourage voluntary upgrades
- **Complete audit trail** of all plan changes

---

## 45.2 DATABASE SCHEMA

**packages/infrastructure/migrations/045_versioned_subscriptions.sql**

```
-- ============================================================================
-- RADIANT v4.15.0 - Versioned Subscriptions & Grandfathering
-- ============================================================================

-- Subscription Plan Versions
CREATE TABLE subscription_plan_versions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tier_id VARCHAR(50) NOT NULL REFERENCES subscription_tiers(id),
    version_number INTEGER NOT NULL,

    display_name VARCHAR(100) NOT NULL,
    description TEXT,

    price_monthly_cents INTEGER,
```

```
    price_annual_cents INTEGER,
    price_per_user BOOLEAN DEFAULT FALSE,

    included_credits_per_user DECIMAL(10,2) NOT NULL,
    features JSONB NOT NULL,
    rate_limits JSONB NOT NULL,

    effective_from TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    effective_until TIMESTAMPTZ,

    change_reason TEXT,
    changed_by UUID REFERENCES administrators(id),

    created_at TIMESTAMPTZ DEFAULT NOW(),

    UNIQUE(tier_id, version_number)
);

CREATE INDEX idx_plan_versions_tier ON subscription_plan_versions(tier_id);
CREATE INDEX idx_plan_versions_effective ON subscription_plan_versions(effective_from, effecti

-- Grandfathered Subscriptions
CREATE TABLE grandfathered_subscriptions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    subscription_id UUID NOT NULL REFERENCES subscriptions(id),
    plan_version_id UUID NOT NULL REFERENCES subscription_plan_versions(id),

    locked_price_monthly_cents INTEGER,
    locked_price_annual_cents INTEGER,
    locked_features JSONB NOT NULL,
    locked_rate_limits JSONB NOT NULL,
    locked_credits_per_user DECIMAL(10,2) NOT NULL,

    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'opted_out', 'mig

    migration_offered BOOLEAN DEFAULT FALSE,
    migration_offer_date TIMESTAMPTZ,
    migration_incentive JSONB,
    migration_response VARCHAR(20),
    migration_response_date TIMESTAMPTZ,

    grandfathered_at TIMESTAMPTZ DEFAULT NOW(),
    grandfathered_reason TEXT,

    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);
```

```sql
CREATE INDEX idx_grandfathered_subscription ON grandfathered_subscriptions(subscription_id);
CREATE INDEX idx_grandfathered_status ON grandfathered_subscriptions(status);

-- Plan Change Audit
CREATE TABLE plan_change_audit (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tier_id VARCHAR(50) NOT NULL REFERENCES subscription_tiers(id),

    old_version_id UUID REFERENCES subscription_plan_versions(id),
    new_version_id UUID NOT NULL REFERENCES subscription_plan_versions(id),

    change_type VARCHAR(30) NOT NULL CHECK (change_type IN (
        'price_increase', 'price_decrease', 'feature_add', 'feature_remove',
        'limit_increase', 'limit_decrease', 'credit_change', 'terms_change'
    )),

    change_summary TEXT NOT NULL,
    affected_subscribers INTEGER NOT NULL DEFAULT 0,
    grandfathered_count INTEGER NOT NULL DEFAULT 0,

    changed_by UUID REFERENCES administrators(id),
    approved_by UUID REFERENCES administrators(id),

    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Function: Get effective plan for subscription
CREATE OR REPLACE FUNCTION get_effective_plan(p_subscription_id UUID)
RETURNS TABLE (
    tier_id VARCHAR(50),
    version_number INTEGER,
    price_monthly_cents INTEGER,
    price_annual_cents INTEGER,
    features JSONB,
    rate_limits JSONB,
    credits_per_user DECIMAL(10,2),
    is_grandfathered BOOLEAN
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.tier_id,
        COALESCE(pv.version_number, 0),
        COALESCE(gs.locked_price_monthly_cents, st.price_monthly::INTEGER * 100),
        COALESCE(gs.locked_price_annual_cents, st.price_annual::INTEGER * 100),
        COALESCE(gs.locked_features, st.features),
        COALESCE(gs.locked_rate_limits, jsonb_build_object(
            'requestsPerMinute', st.requests_per_minute,
```

```
            'tokensPerMinute', st.tokens_per_minute,
            'concurrentRequests', st.concurrent_requests
        )),
        COALESCE(gs.locked_credits_per_user, st.included_credits_per_user),
        (gs.id IS NOT NULL) as is_grandfathered
    FROM subscriptions s
    JOIN subscription_tiers st ON st.id = s.tier_id
    LEFT JOIN grandfathered_subscriptions gs ON gs.subscription_id = s.id AND gs.status = 'acti
    LEFT JOIN subscription_plan_versions pv ON pv.id = gs.plan_version_id
    WHERE s.id = p_subscription_id;
END;
$$ LANGUAGE plpgsql;
```

---

## 45.3 GRANDFATHERING SERVICE

**packages/functions/src/services/grandfathering.ts**

```typescript
/**
 * Grandfathering Service
 * @version 4.15.0
 */

import { Pool } from 'pg';

export class GrandfatheringService {
  constructor(private pool: Pool) {}

  async createPlanVersion(
    tierId: string,
    changeType: string,
    changeSummary: string,
    changedBy: string,
    approvedBy: string
  ): Promise<{ versionId: string; grandfatheredCount: number }> {
    const client = await this.pool.connect();

    try {
      await client.query('BEGIN');

      // Get current tier and latest version
      const tierResult = await client.query(`SELECT * FROM subscription_tiers WHERE id = $1`,
      const tier = tierResult.rows[0];

      const versionResult = await client.query(`
        SELECT COALESCE(MAX(version_number), 0) + 1 as next_version
        FROM subscription_plan_versions WHERE tier_id = $1
      `, [tierId]);
```

```
const nextVersion = versionResult.rows[0].next_version;

// Close previous version
await client.query(`
  UPDATE subscription_plan_versions SET effective_until = NOW()
  WHERE tier_id = $1 AND effective_until IS NULL
`, [tierId]);

// Create new version
const newVersionResult = await client.query(`
  INSERT INTO subscription_plan_versions (
    tier_id, version_number, display_name, description,
    price_monthly_cents, price_annual_cents, price_per_user,
    included_credits_per_user, features, rate_limits, change_reason, changed_by
  ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12)
  RETURNING id
`, [
  tierId, nextVersion, tier.display_name, tier.description,
  tier.price_monthly ? tier.price_monthly * 100 : null,
  tier.price_annual ? tier.price_annual * 100 : null,
  tier.price_per_user, tier.included_credits_per_user, tier.features,
  JSON.stringify({ requestsPerMinute: tier.requests_per_minute, tokensPerMinute: tier.to
  changeSummary, changedBy
]);

const newVersionId = newVersionResult.rows[0].id;

// Get previous version for grandfathering
const prevVersionResult = await client.query(`
  SELECT id FROM subscription_plan_versions WHERE tier_id = $1 AND version_number = $2 -
`, [tierId, nextVersion]);

let grandfatheredCount = 0;

if (prevVersionResult.rows.length > 0) {
  const prevVersionId = prevVersionResult.rows[0].id;

  // Grandfather all active subscriptions
  const grandfatherResult = await client.query(`
    INSERT INTO grandfathered_subscriptions (
      subscription_id, plan_version_id, locked_price_monthly_cents, locked_price_annual_
      locked_features, locked_rate_limits, locked_credits_per_user, grandfathered_reason
    )
    SELECT s.id, $1, pv.price_monthly_cents, pv.price_annual_cents,
      pv.features, pv.rate_limits, pv.included_credits_per_user, $2
    FROM subscriptions s
    JOIN subscription_plan_versions pv ON pv.id = $1
    WHERE s.tier_id = $3 AND s.status IN ('active', 'trialing')
```

```
          AND NOT EXISTS (SELECT 1 FROM grandfathered_subscriptions gs WHERE gs.subscription_
        `, [prevVersionId, changeSummary, tierId]);

        grandfatheredCount = grandfatherResult.rowCount || 0;
      }

      // Record audit
      await client.query(`
        INSERT INTO plan_change_audit (tier_id, old_version_id, new_version_id, change_type, ch
        VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
      `, [tierId, prevVersionResult.rows[0]?.id, newVersionId, changeType, changeSummary, grand

      await client.query('COMMIT');
      return { versionId: newVersionId, grandfatheredCount };
    } catch (error) {
      await client.query('ROLLBACK');
      throw error;
    } finally {
      client.release();
    }
  }

  async getEffectivePlan(subscriptionId: string): Promise<{
    grandfathered: boolean;
    priceMonthly: number | null;
    features: Record<string, any>;
    rateLimits: Record<string, number>;
    creditsPerUser: number;
  }> {
    const result = await this.pool.query(`SELECT * FROM get_effective_plan($1)`, [subscriptionI
    const row = result.rows[0];

    return {
      grandfathered: row.is_grandfathered,
      priceMonthly: row.price_monthly_cents,
      features: row.features,
      rateLimits: row.rate_limits,
      creditsPerUser: row.credits_per_user,
    };
  }

  async offerMigration(subscriptionId: string, incentive: { bonusCredits?: number; discountPerc
    await this.pool.query(`
      UPDATE grandfathered_subscriptions
      SET migration_offered = TRUE, migration_offer_date = NOW(), migration_incentive = $2, upd
      WHERE subscription_id = $1 AND status = 'active'
    `, [subscriptionId, JSON.stringify(incentive)]);
  }
```

```
  async acceptMigration(subscriptionId: string): Promise<void> {
    await this.pool.query(`
      UPDATE grandfathered_subscriptions
      SET status = 'migrated', migration_response = 'accepted', migration_response_date = NOW()
      WHERE subscription_id = $1 AND status = 'active'
    `, [subscriptionId]);
  }
}
```

---