Database Migration Guide for Staging

Complete guide for managing database migrations in staging environment.

📋 Migration Workflow

1. Initial Setup

```
# Generate Prisma client
yarn prisma generate

# Check migration status
yarn prisma migrate status

# Deploy all pending migrations
yarn prisma migrate deploy
```

2. Schema Changes Process

For New Features (Safe)

```
# 1. Update schema.prisma
# 2. Create migration
yarn prisma migrate dev --name add_new_feature

# 3. Review generated SQL
cat prisma/migrations/[timestamp]_add_new_feature/migration.sql

# 4. Test on staging
yarn prisma migrate deploy
```

For Schema Changes (Careful)

```
# 1. ALWAYS backup first
curl -X POST $STAGING_URL/api/database/backups \
  -H "Content-Type: application/json" \
  -d '{"backupName":"pre-migration-backup","backupType":"pre-migration"}'

# 2. Review migration impact
yarn prisma migrate diff \
  --from-schema-datamodel prisma/schema.prisma \
  --to-schema-datasource prisma/schema.prisma

# 3. Deploy migration
yarn prisma migrate deploy

# 4. Verify data integrity
yarn tsx scripts/verify-migration.ts
```

3. Migration Safety Checks

Pre-Migration Checklist

- [ ] Backup created and verified

- [ ] Migration reviewed for data loss potential
- [ ] Staging environment has sufficient disk space
- [ ] No breaking changes to existing API endpoints
- [ ] Migration can be rolled back if needed

High-Risk Operations

```
-- 🚨 These require extra caution:
ALTER TABLE users DROP COLUMN email;          -- Data loss
ALTER TABLE posts ALTER COLUMN id TYPE bigint; -- Long lock time
CREATE UNIQUE INDEX ON large_table(column);    -- Performance impact
```

Safe Operations

```
-- ✅ These are generally safe:
ALTER TABLE users ADD COLUMN new_field TEXT;
CREATE INDEX CONCURRENTLY ON table(column);
ALTER TABLE users ALTER COLUMN field SET DEFAULT 'value';
```

🗄️ Staging Migration Commands

Deploy All Migrations

```
# Standard deployment
yarn prisma migrate deploy

# With verbose logging
yarn prisma migrate deploy --verbose

# Skip confirmation prompts
yarn prisma migrate deploy --accept-data-loss
```

Reset Staging Database (DESTRUCTIVE)

```
# ⚠️ THIS DELETES ALL DATA
yarn prisma migrate reset --skip-seed

# Then re-seed
yarn tsx scripts/seed-staging.ts
```

Rollback Migration (Manual Process)

```
# 1. Identify target migration
yarn prisma migrate status

# 2. Create rollback SQL (manual)
# Edit: prisma/migrations/rollback.sql

# 3. Apply rollback
psql $DATABASE_URL -f prisma/migrations/rollback.sql

# 4. Update migration history
yarn prisma migrate resolve --rolled-back [migration-name]
```

🛡️ Backup Before Migration

Automatic Pre-Migration Backup

```
// scripts/pre-migration-backup.ts
import { exec } from 'child_process';
import { promisify } from 'util';

const execAsync = promisify(exec);

async function createPreMigrationBackup() {
  const backupName = `pre-migration-${Date.now()}`;

  try {
    // Create backup via API
    const response = await fetch(`${process.env.STAGING_URL}/api/database/backups`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        backupName,
        backupType: 'pre-migration'
      })
    });

    if (!response.ok) {
      throw new Error('Backup creation failed');
    }

    console.log(`✅ Pre-migration backup created: ${backupName}`);
    return backupName;
  } catch (error) {
    console.error('❌ Backup failed:', error);
    process.exit(1);
  }
}

// Run before migration
createPreMigrationBackup().then(() => {
  console.log('✅ Safe to proceed with migration');
});
```

Manual Database Backup

```
# Direct PostgreSQL backup
pg_dump $DATABASE_URL > backup-$(date +%Y%m%d-%H%M%S).sql

# Compressed backup
pg_dump $DATABASE_URL | gzip > backup-$(date +%Y%m%d-%H%M%S).sql.gz

# Schema only
pg_dump --schema-only $DATABASE_URL > schema-backup.sql
```

📊 Migration Monitoring

Check Migration Status

```
# View migration history
yarn prisma migrate status

# Check for pending migrations
yarn prisma migrate diff

# Verify schema matches database
yarn prisma db pull && git diff prisma/schema.prisma
```

Performance Impact Monitoring

```
-- Monitor long-running queries during migration
SELECT
  query,
  state,
  query_start,
  NOW() - query_start AS duration
FROM pg_stat_activity
WHERE state != 'idle'
ORDER BY duration DESC;

-- Check table sizes after migration
SELECT
  schemaname,
  tablename,
  pg_size_pretty(pg_total_relation_size(schemaname||'.'||tablename)) AS size
FROM pg_tables
WHERE schemaname = 'public'
ORDER BY pg_total_relation_size(schemaname||'.'||tablename) DESC;
```

🔧 Common Migration Scenarios

Adding New Models (Phase 3.2-3.5)

```
// Example: Adding social embed tracking
model SocialEmbed {
  id           String   @id @default(cuid())
  platform     String
  url          String
  embed_id     String   @unique
  thumbnail    String?
  title        String?
  author       String?
  aspect_ratio String   @default("16:9")
  usage_count  Int      @default(0)
  status       String   @default("active")
  metadata     Json?
  created_at   DateTime @default(now())
  updated_at   DateTime @updatedAt

  @@map("social_embeds")
}
```

Adding Indexes for Performance

```
-- Create indexes concurrently (non-blocking)
CREATE INDEX CONCURRENTLY idx_blog_posts_published ON blog_posts(published, cre-
ated_at);
CREATE INDEX CONCURRENTLY idx_social_embeds_platform ON social_embeds(platform, status)
;
CREATE INDEX CONCURRENTLY idx_media_assets_type ON media_assets(file_type, created_at);
```

Data Migration Scripts

```
// scripts/migrate-data.ts
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

async function migrateExistingData() {
  console.log('🔄 Starting data migration...');

  // Example: Update existing records to match new schema
  const updatedPosts = await prisma.blogPost.updateMany({
    where: { featured_image: null },
    data: { featured_image: 'default-image.jpg' }
  });

  console.log(`✅ Updated ${updatedPosts.count} blog posts`);
}

migrateExistingData()
  .catch(console.error)
  .finally(() => prisma.$disconnect());
```

🚨 Emergency Procedures

Migration Failed Mid-Process

```
# 1. Check current state
yarn prisma migrate status

# 2. Identify failed migration
SELECT * FROM _prisma_migrations ORDER BY finished_at DESC LIMIT 5;

# 3. Mark migration as rolled back
yarn prisma migrate resolve --rolled-back [migration-name]

# 4. Fix the issue and retry
yarn prisma migrate deploy
```

Database Corruption

```
# 1. Stop application
vercel env add DATABASE_URL_BACKUP

# 2. Restore from latest backup
curl -X POST $STAGING_URL/api/database/backups/[backup-id]/restore

# 3. Verify data integrity
yarn tsx scripts/verify-data-integrity.ts

# 4. Update application to use restored database
vercel env rm DATABASE_URL_BACKUP
```

Rollback to Previous Schema

```
-- Example rollback SQL
-- (Generated manually based on migration)
ALTER TABLE social_embeds DROP COLUMN usage_count;
DROP INDEX idx_social_embeds_platform;
ALTER TABLE media_assets ALTER COLUMN file_type DROP NOT NULL;
```

✅ Post-Migration Verification

Data Integrity Checks

```
// scripts/verify-migration.ts
async function verifyMigration() {
  // Check record counts
  const counts = {
    blogPosts: await prisma.blogPost.count(),
    users: await prisma.user.count(),
    socialEmbeds: await prisma.socialEmbed.count(),
    mediaAssets: await prisma.mediaAsset.count()
  };

  console.log('📊 Record counts:', counts);

  // Verify foreign key relationships
  const orphanedRecords = await prisma.$queryRaw`
    SELECT 'blog_posts' as table_name, COUNT(*) as count
    FROM blog_posts bp
    WHERE NOT EXISTS (SELECT 1 FROM users u WHERE u.id = bp.author_id)
  `;

  console.log('🔍 Orphaned records:', orphanedRecords);
}
```

API Endpoint Testing

```
# Test all critical endpoints after migration
curl -f $STAGING_URL/api/health
curl -f $STAGING_URL/api/social-embeds
curl -f $STAGING_URL/api/media
curl -f $STAGING_URL/api/homepage-blocks
curl -f $STAGING_URL/api/database/stats
```

Performance Verification

```
-- Check query performance post-migration
EXPLAIN ANALYZE SELECT * FROM blog_posts WHERE published = true ORDER BY created_at DESC LIMIT 10;
EXPLAIN ANALYZE SELECT * FROM social_embeds WHERE platform = 'instagram' AND status = 'active';
EXPLAIN ANALYZE SELECT * FROM media_assets WHERE file_type = 'image' ORDER BY created_at DESC;
```

Remember: Always backup before migrations, test on staging first, and have a rollback plan ready!