ImpactCV - Database Schema Documentation

Overview

ImpactCV uses PostgreSQL (hosted on Supabase) as its primary database with Prisma ORM for type-safe database access. The schema is designed to support user authentication, resume management, and resume sharing functionality.

Database Architecture

Technology Stack

- **Database**: PostgreSQL 15+
- **Hosting**: Supabase (Cloud PostgreSQL)
- **ORM**: Prisma 6.19.0
- **Connection Pooling**: PgBouncer (Transaction Mode)

Connection Configuration

- **Pooled Connection** (Port 6543): Used by application for queries
- **Direct Connection** (Port 5432): Used for migrations and schema changes

---

Schema Diagram

```
┌──────────────────────┐
│        users         │
├──────────────────────┤
│ id (PK)              │
│ name                 │
│ email (UNIQUE)       │
│ password_hash        │
│ created_at           │
│ updated_at           │
└──────────────────────┘
         │
         │ 1:N
         ▼
┌──────────────────────┐
│       resumes        │
├──────────────────────┤
│ id (PK)              │
│ user_id (FK)         │
│ title                │
│ theme                │
│ data (JSONB)         │
│ created_at           │
│ updated_at           │
└──────────────────────┘


┌──────────────────────┐
│   shared_resumes     │
├──────────────────────┤
│ id (PK)              │
│ share_id (UNIQUE)    │
│ data (JSONB)         │
│ created_at           │
└──────────────────────┘
```

Table Definitions

1. users

Stores user account information with authentication credentials.

Columns

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing user identifier
name	VARCHAR(255)	NOT NULL	User's full name
email	VARCHAR(255)	UNIQUE, NOT NULL	User's email (login identifier)
password_hash	VARCHAR(255)	NOT NULL	Bcrypt hashed password
created_at	TIMESTAMP(6)	DEFAULT NOW()	Account creation timestamp
updated_at	TIMESTAMP(6)	DEFAULT NOW()	Last account update timestamp

Indexes

- `idx_users_email` on `email` - Optimizes login queries

Relationships

- **One-to-Many** with `resumes` table

Security

- Passwords are hashed using bcrypt (10 salt rounds)
- Email uniqueness enforced at database level
- No plain text passwords stored

Example Data

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john@example.com",
  "password_hash": "$2b$10$...",
  "created_at": "2024-01-15T10:30:00.000Z",
  "updated_at": "2024-01-15T10:30:00.000Z"
}
```

---

2. resumes

Stores resume data for authenticated users.

Columns

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing resume identifier
user_id	INTEGER	FOREIGN KEY, NULLABLE	Reference to users.id
title	VARCHAR(255)	NOT NULL	Resume title/name
theme	VARCHAR(100)	NOT NULL	Visual theme identifier
data	JSONB	NOT NULL	Complete resume content (JSON)
created_at	TIMESTAMP(6)	DEFAULT NOW()	Resume creation timestamp
updated_at	TIMESTAMP(6)	DEFAULT NOW()	Last modification timestamp

Indexes

- `idx_resumes_user_id` on `user_id` - Optimizes user resume queries

Relationships

- **Many-to-One** with `users` table
- **Foreign Key**: `user_id` REFERENCES `users(id)` ON DELETE CASCADE

Cascade Behavior

- When a user is deleted, all their resumes are automatically deleted

Data Structure (JSONB)

The `data` column stores the complete resume content as JSON:

```
{
  "activeTheme": "modern",
  "basicInfo": {
    "name": "John Doe",
    "email": "john@example.com",
    "phone": "+1234567890",
    "location": "New York, NY",
    "linkedin": "linkedin.com/in/johndoe",
    "github": "github.com/johndoe",
    "website": "johndoe.com"
  },
  "summary": "Experienced software engineer...",
  "experience": [
    {
      "id": "exp1",
      "company": "Tech Corp",
      "position": "Senior Developer",
      "location": "New York, NY",
      "startDate": "2020-01",
      "endDate": "2024-01",
      "current": false,
      "description": "Led development of..."
    }
  ],
  "education": [
    {
      "id": "edu1",
      "institution": "University Name",
      "degree": "Bachelor of Science",
      "field": "Computer Science",
      "location": "City, State",
      "startDate": "2016-09",
      "endDate": "2020-05",
      "gpa": "3.8"
    }
  ],
  "skills": [
    {
      "id": "skill1",
      "category": "Programming Languages",
      "items": ["JavaScript", "TypeScript", "Python"]
    }
  ],
  "projects": [
    {
      "id": "proj1",
      "name": "Project Name",
      "description": "Description...",
      "technologies": ["React", "Node.js"],
      "link": "github.com/project"
    }
  ],
  "certifications": [
    {
      "id": "cert1",
      "name": "AWS Certified Developer",
      "issuer": "Amazon Web Services",
      "date": "2023-06",
      "credentialId": "ABC123"
    }
  ]
}
```

Example Record

```
{
  "id": 1,
  "user_id": 1,
  "title": "Software Engineer Resume",
  "theme": "modern",
  "data": { /* Full resume JSON */ },
  "created_at": "2024-01-15T11:00:00.000Z",
  "updated_at": "2024-01-20T14:30:00.000Z"
}
```

---

3. shared_resumes

Stores publicly shared resume snapshots accessible via unique URLs.

Columns

Column Name	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Auto-incrementing identifier
share_id	VARCHAR(255)	UNIQUE, NOT NULL	Unique share URL identifier
data	JSONB	NOT NULL	Resume snapshot (JSON)

created_at	TIMESTAMP(6) DEFAULT NOW()	Share creation timestamp

Indexes

- `idx_shared_resumes_share_id` on `share_id` - Optimizes share URL lookups

Relationships

- **Independent** - No foreign key relationships
- Stores snapshots, not live references

Share ID Format

- Generated using UUID or timestamp-based unique strings
- Example: `resume-1705320000000-abc123`

Data Structure

Same as `resumes.data` but represents a point-in-time snapshot

Example Record

```
{
  "id": 1,
  "share_id": "resume-1705320000000-abc123",
  "data": { /* Resume snapshot JSON */ },
  "created_at": "2024-01-15T12:00:00.000Z"
}
```

Prisma Schema

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider  = "postgresql"
  url       = env("DATABASE_URL")
  directUrl = env("DIRECT_URL")
}

model User {
  id           Int       @id @default(autoincrement())
  name         String    @db.VarChar(255)
  email        String    @unique @db.VarChar(255)
  passwordHash String    @map("password_hash") @db.VarChar(255)
  createdAt    DateTime? @default(now()) @map("created_at") @db.Timestamp(6)
  updatedAt    DateTime? @default(now()) @map("updated_at") @db.Timestamp(6)

  resumes      Resume[]

  @@index([email], map: "idx_users_email")
  @@map("users")
}

model Resume {
  id        Int       @id @default(autoincrement())
  userId    Int?      @map("user_id")
  title     String    @db.VarChar(255)
  theme     String    @db.VarChar(100)
  data      Json
  createdAt DateTime? @default(now()) @map("created_at") @db.Timestamp(6)
  updatedAt DateTime? @default(now()) @map("updated_at") @db.Timestamp(6)

  user      User?     @relation(fields: [userId], references: [id], onDelete: Cascade, onUpdate: NoAction)

  @@index([userId], map: "idx_resumes_user_id")
  @@map("resumes")
}

model SharedResume {
  id        Int       @id @default(autoincrement())
  shareId   String    @unique @map("share_id") @db.VarChar(255)
  data      Json
  createdAt DateTime? @default(now()) @map("created_at") @db.Timestamp(6)

  @@index([shareId], map: "idx_shared_resumes_share_id")
  @@map("shared_resumes")
}
```

Database Operations

Common Queries

1. User Registration

```
INSERT INTO users (name, email, password_hash, created_at, updated_at)
VALUES ($1, $2, $3, NOW(), NOW())
RETURNING id, name, email, created_at;
```

2. User Login

```
SELECT id, name, email, password_hash
FROM users
WHERE email = $1;
```

3. Get User's Resumes

```
SELECT id, title, theme, data, created_at, updated_at
FROM resumes
WHERE user_id = $1
ORDER BY updated_at DESC;
```

4. Create Resume

```
INSERT INTO resumes (user_id, title, theme, data, created_at, updated_at)
VALUES ($1, $2, $3, $4, NOW(), NOW())
RETURNING id, title, theme, created_at;
```

5. Update Resume

```
UPDATE resumes
SET title = $1, theme = $2, data = $3, updated_at = NOW()
WHERE id = $4 AND user_id = $5
RETURNING id, title, updated_at;
```

6. Share Resume

```
INSERT INTO shared_resumes (share_id, data, created_at)
VALUES ($1, $2, NOW())
ON CONFLICT (share_id) DO UPDATE SET data = $2
RETURNING share_id;
```

7. Get Shared Resume

```
SELECT data
FROM shared_resumes
WHERE share_id = $1;
```

Performance Considerations

Indexes

All critical lookup columns are indexed:

- `users.email` - Fast login queries
- `resumes.user_id` - Fast user resume retrieval
- `shared_resumes.share_id` - Fast share URL lookups

JSONB Performance

- JSONB format allows efficient querying of nested data
- Supports GIN indexes for complex JSON queries (if needed)
- Faster than TEXT-based JSON storage

Connection Pooling

- PgBouncer in transaction mode
- Reduces connection overhead
- Handles concurrent requests efficiently

Security Measures

1. Password Security

- Bcrypt hashing with 10 salt rounds
- No plain text passwords stored
- Password validation on application layer

2. SQL Injection Prevention

- Prisma ORM provides parameterized queries
- All user inputs are sanitized
- No raw SQL with user input

3. Data Access Control

- JWT-based authentication
- User can only access their own resumes

- Middleware validates ownership before operations

4. Email Uniqueness

- Database-level UNIQUE constraint
- Prevents duplicate accounts
- Application-level validation as well

Backup and Recovery

Supabase Automatic Backups

- Daily automated backups (free tier: 7 days retention)
- Point-in-time recovery available (paid tiers)
- Manual backup via pg_dump

Manual Backup Command

```
pg_dump -h [SUPABASE_HOST] -U postgres -d postgres > backup.sql
```

Restore Command

```
psql -h [SUPABASE_HOST] -U postgres -d postgres < backup.sql
```

Migration Strategy

Using Prisma Migrate

Create Migration

```
npx prisma migrate dev --name migration_name
```

Apply Migration (Production)

```
npx prisma migrate deploy
```

Reset Database (Development Only)

```
npx prisma migrate reset
```

Manual Migration

SQL scripts can be run directly in Supabase SQL Editor for schema changes.

Monitoring and Maintenance

Key Metrics to Monitor

1. **Query Performance**: Slow query log analysis
2. **Connection Pool**: Active connections vs. max connections
3. **Database Size**: Monitor JSONB column growth
4. **Index Usage**: Ensure indexes are being utilized

Supabase Dashboard

- Real-time query monitoring
- Connection pool statistics
- Database size and usage
- Slow query identification

Future Enhancements

Potential Schema Additions

1. **Resume Templates Table**
    - Store predefined resume templates
    - Allow users to start from templates
2. **Resume Versions Table**
    - Track resume edit history
    - Allow rollback to previous versions
3. **User Preferences Table**
    - Store user settings
    - Default theme, language, etc.
4. **Analytics Table**
    - Track resume views
    - Share link analytics

5. **Collaboration Table**
    - Allow resume sharing between users
    - Collaborative editing features

Appendix

Environment Variables

```
DATABASE_URL="postgresql://postgres.[PROJECT]:[PASSWORD]@[HOST]:6543/postgres?pgbouncer=true"
DIRECT_URL="postgresql://postgres.[PROJECT]:[PASSWORD]@[HOST]:5432/postgres"
```

Useful Prisma Commands

```
# Generate Prisma Client
npx prisma generate

# Open Prisma Studio (GUI)
npx prisma studio

# Validate schema
npx prisma validate

# Format schema
npx prisma format

# Pull schema from database
npx prisma db pull

# Push schema to database
npx prisma db push
```

Document Version: 1.0
Last Updated: January 2024
Database Version: PostgreSQL 15+
Prisma Version: 6.19.0