Database Schema Documentation

Overview

The Acrux MVP uses PostgreSQL as its primary database with Prisma as the ORM. The schema is designed to support objective and key result (OKR) management, team collaboration, pulse checks, and blocker tracking.

Core Models

1. User

Table: `users`

The central model for all system users (admins and team members).

Fields

Field	Type	Description	Default	Constraints
id	String	Unique user identifier	cuid()	Primary Key
name	String?	Full name of the user	-	Optional
email	String?	User’s email address	-	Unique
emailVerified	DateTime?	Email verification timestamp	-	Optional
image	String?	Profile image URL	-	Optional
password	String?	Hashed password	-	Optional
firstName	String?	User’s first name	-	Optional
lastName	String?	User’s last name	-	Optional
role	String	User role in system	“team_member”	admin, team_member
organizationId	String?	Organization identifier	-	Optional
isActive	Boolean	Account status	true	-
createdAt	DateTime	Account creation timestamp	now()	-
updatedAt	DateTime	Last update timestamp	Auto	Auto-updated

Relations

- **accounts**: One-to-many → Account (OAuth accounts)
- **sessions**: One-to-many → Session (Active sessions)
- **objectives**: One-to-many → Objective (Created objectives)
- **pulseChecks**: One-to-many → PulseCheck (Pulse check responses)
- **teamMemberships**: One-to-many → TeamMember (Team relationships)
- **objectiveAssignments**: One-to-many → ObjectiveAssignment (Assigned objectives)
- **sentPulseRequests**: One-to-many → PulseRequest (Sent pulse requests as admin)
- **receivedPulseRequests**: One-to-many → PulseRequest (Received pulse requests as member)

- **sentInvitations**: One-to-many → TeamInvitation (Sent invitations as admin)
- **receivedInvitations**: One-to-many → TeamInvitation (Received invitations)

---

2. Objective

Table: `objectives`

Represents company or team objectives/goals with progress tracking.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique objective identifier	cuid()	Primary Key
title	String	Objective title	-	Required
description	String?	Detailed description	-	Text, Optional
progress	Float	Progress percentage	0	0-100
healthScore	Float	Objective health score	75	0-100
status	String	Current status	“active”	active, completed, paused
targetDate	DateTime?	Target completion date	-	Optional
ownerId	String?	Responsible person ID	-	Optional
userId	String	Creator ID	-	Foreign Key → User
createdAt	DateTime	Creation timestamp	now()	-
updatedAt	DateTime	Last update timestamp	Auto	Auto-updated

Relations

- **user**: Many-to-one → User (Creator)
- **metrics**: One-to-many → Metric (Key metrics)
- **pulseChecks**: One-to-many → PulseCheck (Team pulse checks)

- **blockers**: One-to-many → Blocker (Issues/blockers)
- **assignments**: One-to-many → ObjectiveAssignment (Team assignments)
- **pulseRequests**: One-to-many → PulseRequest (Related pulse requests)

---

3. Metric

Table: `metrics`

Key results/metrics that measure objective progress.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique metric identifier	cuid()	Primary Key
name	String	Metric name	-	Required
description	String?	Metric description	-	Text, Optional
currentValue	Float	Current metric value	0	-
targetValue	Float	Target metric value	100	-
unit	String	Unit of measurement	“units”	units, percentage, count, etc
objectiveId	String	Parent objective ID	-	Foreign Key → Objective
createdAt	DateTime	Creation timestamp	now()	-
updatedAt	DateTime	Last update timestamp	Auto	Auto-updated

Relations

- **objective**: Many-to-one → Objective (Parent objective)

---

4. PulseCheck

Table: `pulse_checks`

Team member sentiment and confidence ratings for objectives.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique pulse check identifier	cuid()	Primary Key
sentiment	Int	Emotional sentiment rating	-	1-5 scale (1=😞, 5=😊)
confidence	Int	Confidence in success	-	1-5 scale
feedback	String?	Optional feedback text	-	Text, Optional
isAnonymous	Boolean	Anonymous submission	true	-
userId	String	Submitter ID	-	Foreign Key → User
objectiveId	String	Related objective ID	-	Foreign Key → Objective
createdAt	DateTime	Submission timestamp	now()	-

Relations

- **user**: Many-to-one → User (Submitter)
- **objective**: Many-to-one → Objective (Related objective)

---

5. Blocker

Table: `blockers`

Issues or obstacles preventing objective progress.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique blocker identifier	cuid()	Primary Key
title	String	Blocker title	-	Required
description	String	Detailed description	-	Text, Required
severity	String	Issue severity level	“medium”	low, medium, high, critical
status	String	Current status	“open”	open, in-progress, resolved
isAnonymous	Boolean	Anonymous submission	true	-
objectiveId	String	Related objective ID	-	Foreign Key → Objective
createdAt	DateTime	Creation timestamp	now()	-
updatedAt	DateTime	Last update timestamp	Auto	Auto-updated

Relations

- **objective**: Many-to-one → Objective (Related objective)

---

6. TeamMember

Table: `team_members`

Represents team membership relationships between users and admins.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique team member identifier	cuid()	Primary Key
role	String	Member role	“member”	member, lead, admin
joinedAt	DateTime	Join timestamp	now()	-
isActive	Boolean	Membership status	true	-
userId	String	Team member ID	-	Foreign Key → User
adminId	String	Admin who added member	-	Foreign Key → User

Relations

- **user**: Many-to-one → User (Team member)

Constraints

- Unique constraint on [userId, adminId] combination

---

7. ObjectiveAssignment

Table: `objective_assignments`

Assigns team members to specific objectives with defined roles.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique assignment identifier	cuid()	Primary Key
role	String	Assignment role	“member”	owner, member, viewer
assignedAt	DateTime	Assignment timestamp	now()	-
isActive	Boolean	Assignment status	true	-
userId	String	Assigned user ID	-	Foreign Key → User
objectiveId	String	Assigned objective ID	-	Foreign Key → Objective
assignedBy	String	Admin who made assignment	-	Foreign Key → User

Relations

- **user**: Many-to-one → User (Assigned member)
- **objective**: Many-to-one → Objective (Assigned objective)

Constraints

- Unique constraint on [userId, objectiveId] combination

---

8. PulseRequest

Table: `pulse_requests`

Admin-initiated requests for team member pulse checks.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique request identifier	cuid()	Primary Key
title	String	Request title	-	Required
message	String?	Optional message	-	Text, Optional
dueDate	DateTime?	Response due date	-	Optional
status	String	Request status	“pending”	pending, completed, expired
adminId	String	Admin sender ID	-	Foreign Key → User
memberId	String	Team member recipient ID	-	Foreign Key → User
objectiveId	String?	Related objective ID	-	Foreign Key → Objective, Optional
createdAt	DateTime	Creation timestamp	now()	-
completedAt	DateTime?	Completion timestamp	-	Optional

Relations

- **admin**: Many-to-one → User (Request sender)
- **member**: Many-to-one → User (Request recipient)
- **objective**: Many-to-one → Objective (Related objective, optional)

---

9. TeamInvitation

Table: `team_invitations`

Email-based team invitations sent by admins.

Fields

Field	Type	Description	Default	Constraints
id	String	Unique invitation identifier	cuid()	Primary Key
email	String	Invitee email address	-	Required
role	String	Invited role	“team_member”	team_member, admin
status	String	Invitation status	“pending”	pending, accepted, expired, cancelled
token	String	Unique invitation token	-	Unique
expiresAt	DateTime	Expiration timestamp	-	Required
adminId	String	Admin who sent invitation	-	Foreign Key → User
acceptedBy	String?	User who accepted	-	Foreign Key → User, Optional
createdAt	DateTime	Creation timestamp	now()	-
acceptedAt	DateTime?	Acceptance timestamp	-	Optional

Relations

- **admin**: Many-to-one → User (Invitation sender)
- **acceptedUser**: Many-to-one → User (Invitation acceptor, optional)

Authentication Models

10. Account

Table: `accounts`

NextAuth.js OAuth provider accounts.

Fields

Field	Type	Description	Constraints
id	String	Unique account identifier	Primary Key
userId	String	Related user ID	Foreign Key → User
type	String	Account type	Required
provider	String	OAuth provider	Required
providerAccountId	String	Provider account ID	Required
refresh_token	String?	OAuth refresh token	Text, Optional
access_token	String?	OAuth access token	Text, Optional
expires_at	Int?	Token expiration	Optional
token_type	String?	Token type	Optional
scope	String?	OAuth scope	Optional
id_token	String?	OAuth ID token	Text, Optional
session_state	String?	Session state	Optional

Relations

- **user**: Many-to-one → User (Account owner)

Constraints

- Unique constraint on [provider, providerAccountId] combination

---

11. Session

Table: `sessions`

Active user sessions for NextAuth.js.

Fields

Field	Type	Description	Constraints
id	String	Unique session identifier	Primary Key
sessionToken	String	Unique session token	Unique
userId	String	Session owner ID	Foreign Key → User
expires	DateTime	Session expiration	Required

Relations

- **user**: Many-to-one → User (Session owner)

---

12. VerificationToken

Table: `verificationtokens`

Email verification and password reset tokens.

Fields

Field	Type	Description	Constraints
identifier	String	User identifier (email)	Required
token	String	Unique verification token	Unique
expires	DateTime	Token expiration	Required

Constraints

- Unique constraint on [identifier, token] combination

Database Relationships Diagram

```
User
 ├─> Account (OAuth)
 ├─> Session (Active sessions)
 ├─> Objective (Created objectives)
 │    ├─> Metric (Key results)
 │    ├─> PulseCheck (Team pulse)
 │    ├─> Blocker (Issues)
 │    ├─> ObjectiveAssignment (Team assignments)
 │    └─> PulseRequest (Pulse requests)
 ├─> PulseCheck (Submitted pulse checks)
 ├─> TeamMember (Team relationships)
 ├─> ObjectiveAssignment (Assigned objectives)
 ├─> PulseRequest (Sent as admin)
 ├─> PulseRequest (Received as member)
 ├─> TeamInvitation (Sent as admin)
 └─> TeamInvitation (Accepted invitations)
```

Indexes and Performance Considerations

Recommended Indexes

1. **users table**
   - `email` (unique) - for login lookups
   - `role` - for role-based queries
   - `isActive` - for active user queries

2. **objectives table**
   - `userId` - for creator lookups
   - `status` - for status filtering
   - `createdAt` - for sorting

3. **pulse_checks table**
   - `objectiveId` - for objective pulse history
   - `userId` - for user pulse history
   - `createdAt` - for chronological ordering

4. **blockers table**
   - `objectiveId` - for objective blockers
   - `status` - for open blocker queries
   - `severity` - for priority queries

5. **objective_assignments table**
   - Composite index on `[userId, objectiveId]` (already enforced by unique constraint)
   - `isActive` - for active assignment queries

6. **pulse_requests table**
   - `memberId, status` - for pending request queries
   - `adminId` - for admin-sent requests
   - `objectiveId` - for objective-related requests

Data Integrity Rules

Cascade Delete Behaviors

1. **User deletion**: Cascades to all related records
   - Accounts, Sessions, Objectives, PulseChecks, TeamMembers, ObjectiveAssignments
2. **Objective deletion**: Cascades to
   - Metrics, PulseChecks, Blockers, ObjectiveAssignments
   - PulseRequests: Sets objectiveId to NULL (SetNull)
3. **TeamInvitation acceptance**: Sets NULL on delete of accepted user

Data Validation

1. **Sentiment and Confidence**: Must be 1-5 scale
2. **Progress and HealthScore**: Must be 0-100 percentage
3. **Status fields**: Must match predefined enum values
4. **Email addresses**: Must be unique across users
5. **Invitation tokens**: Must be unique

Security Considerations

1. **Password Storage**: Passwords are hashed using bcrypt
2. **Session Management**: Handled by NextAuth.js with secure tokens
3. **Anonymous Submissions**: PulseChecks and Blockers support anonymous flag
4. **Role-Based Access**: User roles control access to admin features
5. **Invitation Security**: Time-limited tokens with expiration
6. **Soft Deletes**: isActive flags allow soft deletion of relationships

---

Query Patterns

Common Queries

1. **Get user's objectives**

```
SELECT * FROM objectives
WHERE userId = ? OR id IN (
  SELECT objectiveId FROM objective_assignments
  WHERE userId = ? AND isActive = true
);
```

1. **Get pending pulse requests**

```
SELECT * FROM pulse_requests
WHERE memberId = ? AND status = 'pending'
ORDER BY dueDate ASC;
```

1. **Get objective health data**

```sql
SELECT o.*,
  AVG(pc.sentiment) as avg_sentiment,
  AVG(pc.confidence) as avg_confidence,
  COUNT(DISTINCT b.id) as open_blockers
FROM objectives o
LEFT JOIN pulse_checks pc ON pc.objectiveId = o.id
LEFT JOIN blockers b ON b.objectiveId = o.id AND b.status = 'open'
WHERE o.id = ?
GROUP BY o.id;
```

Migration Strategy

Initial Setup

```
# Generate Prisma client
npx prisma generate

# Run migrations
npx prisma migrate dev

# Seed database
npx prisma db seed
```

Schema Updates

```
# Create migration
npx prisma migrate dev --name description_of_change

# Apply to production
npx prisma migrate deploy
```

Backup and Maintenance

1. **Regular Backups**: Automated daily PostgreSQL backups
2. **Data Retention**: Historical pulse checks and blockers retained indefinitely
3. **Cleanup Tasks**:
   - Expired invitations (> 7 days)
   - Expired pulse requests (> 30 days after due date)
   - Inactive sessions (handled by NextAuth.js)

Last Updated: 2025-10-03
Database Version: 1.0