■ Cardinality Analysis by Module

1. USER & ORGANIZATION RELATIONSHIPS **One-to-Many (1:N)**
Organization → Users: One organization can have many users
organizations.id → users.organizationId
Business logic: Organizations can have multiple members

One-to-One (1:1)
User → Organization Owner: One user can own one organization
users.id → organizations.ownerId
Business logic: Each organization has exactly one owner

2. MEMBERSHIP SYSTEM RELATIONSHIPS **One-to-Many (1:N)**
Organization → Plans: One organization can create many membership plans
organizations.id → plans.organizationId
Business logic: Organizations offer multiple membership tiers
User → Subscriptions: One user can have multiple subscriptions
users.id → subscriptions.userId
Business logic: Users can subscribe to different plans over time
Plan → Subscriptions: One plan can have many subscribers
plans.id → subscriptions.planId
Business logic: Multiple users can subscribe to the same plan
User → Applications: One user can submit multiple applications
users.id → applications.userId
Business logic: Users can apply to different plans
Plan → Applications: One plan can receive many applications
plans.id → applications.planId
Business logic: Multiple users can apply to the same plan

3. PAYMENT & BILLING RELATIONSHIPS **One-to-Many (1:N)**
User → Payments: One user can make multiple payments
users.id → payments.userId
Business logic: Users pay for subscriptions, fees, etc.
Plan → Payments: One plan can have many payments
plans.id → payments.planId
Business logic: Multiple payments can be made for the same plan
Subscription → Payments: One subscription can have multiple payments
subscriptions.id → payments.subscriptionId
Business logic: Recurring payments for ongoing subscriptions
User → Invoices: One user can have multiple invoices
users.id → invoices.userId
Business logic: Users receive invoices for various charges
Plan → Invoices: One plan can generate many invoices
plans.id → invoices.planId
Business logic: Multiple invoices for the same plan
Payment → Invoice: One payment can have one invoice
payments.id → invoices.paymentId
Business logic: Each payment generates one invoice

4. DEBT & REMINDER RELATIONSHIPS **One-to-Many (1:N)**
User → Debts: One user can have multiple outstanding debts
users.id → debts.userId
Business logic: Users can owe money for different reasons
Plan → Debts: One plan can have multiple debt records
plans.id → debts.planId

Business logic: Debts can be associated with specific plans
Subscription → Debts: One subscription can have multiple debts
subscriptions.id → debts.subscriptionId
Business logic: Subscription-related outstanding amounts
User → Reminders: One user can have multiple reminders
users.id → reminders.userId
Business logic: Various reminder types for users
Plan → Reminders: One plan can have multiple reminders
plans.id → reminders.planId
Business logic: Plan-specific reminders
Subscription → Reminders: One subscription can have multiple reminders
subscriptions.id → reminders.subscriptionId
Business logic: Subscription-related notifications

5. SOCIAL FEATURES RELATIONSHIPS **One-to-Many (1:N)**
User → Spaces (Owner): One user can own multiple spaces
users.id → spaces.ownerId
Business logic: Users can create multiple communities
User → Posts: One user can create multiple posts
users.id → posts.userId
Business logic: Users can share multiple pieces of content
Space → Posts: One space can have multiple posts
spaces.id → posts.spaceId
Business logic: Communities contain multiple posts
User → Comments: One user can make multiple comments
users.id → comments.userId
Business logic: Users can comment on multiple posts
Post → Comments: One post can have multiple comments
posts.id → comments.postId
Business logic: Posts can receive multiple comments
User → Follows: One user can follow multiple entities
users.id → follows.userId
Business logic: Users can follow multiple users/spaces
User → Likes: One user can like multiple items
users.id → likes.userId
Business logic: Users can like multiple posts/comments

6. CAREER CENTER RELATIONSHIPS **One-to-Many (1:N)**
User → Jobs (Employer): One user can post multiple jobs
users.id → jobs.userId
Business logic: Companies can post multiple job openings
User → Job Applications (Applicant): One user can apply to multiple jobs
users.id → job_applications.applicantId
Business logic: Job seekers can apply to multiple positions
Job → Job Applications: One job can receive multiple applications
jobs.id → job_applications.jobId
Business logic: Job postings can receive multiple applications

7. MANY-TO-MANY (M:N) RELATIONSHIPS **Junction Tables:**
Users ↔ Spaces (via memberships): Users can join multiple spaces, spaces can have multiple members
Users ↔ Jobs (via saved_jobs): Users can save multiple jobs, jobs can be saved by multiple users

8. DIGITAL CARD RELATIONSHIPS **One-to-Many (1:N)**
User → Digital Cards: One user can have multiple digital cards
users.id → digital_cards.userId

Business logic: Users can have cards for different subscriptions
Subscription → Digital Cards: One subscription can have multiple cards
subscriptions.id → digital_cards.subscriptionId
Business logic: Subscription-specific membership cards

9. APPLICATION FORM RELATIONSHIPS **One-to-Many (1:N)**
Organization → Application Forms: One organization can have multiple application forms
organizations.id → application_forms.organizationId
Business logic: Organizations can create custom application forms