Creating High-Quality Seed Data for an Asana RL Environment

Author: Yogesh Saini
Context: Research Scientist Internship – Take-Home Assignment

Overview

This document describes the design and implementation of a realistic, enterprise-grade seed dataset that simulates an Asana workspace used by a large B2B SaaS organization (5,000–10,000 employees). The dataset is intended to serve as initialization data for reinforcement learning (RL) environments that evaluate computer-use AI agents on project management workflows.

The primary goal of this work is to ensure **data realism**, **methodological rigor**, and **consistency**, so that downstream agents cannot exploit synthetic shortcuts and must instead learn behaviors that generalize to real enterprise environments.

Section A: Database Schema

A.1 Schema Overview

The dataset is backed by a relational schema implemented in SQLite. The schema is designed to closely mirror core Asana entities and their relationships, while remaining simple enough to support reproducible data generation and efficient querying.

Key design goals of the schema include:

- Faithful representation of Asana's core objects (projects, tasks, users, teams)
- Support for hierarchical tasks and flexible metadata
- Strong referential integrity and temporal consistency
- Scalability to tens of thousands of tasks

A.2 Entity List

The schema includes the following entities:

- **organizations** – top-level workspace container
- **teams** – functional groups within the organization
- **users** – members of the workspace
- **team_memberships** – many-to-many mapping between users and teams

- **projects** – collections of tasks organized around goals or initiatives
- **sections** – workflow subdivisions within projects
- **tasks** – units of work (including subtasks via hierarchy)
- **comments** – discussion and activity on tasks
- **custom_field_definitions** – project-specific metadata definitions
- **custom_field_values** – values of custom fields attached to tasks
- **tags** – reusable labels across projects
- **task_tags** – many-to-many mapping between tasks and tags

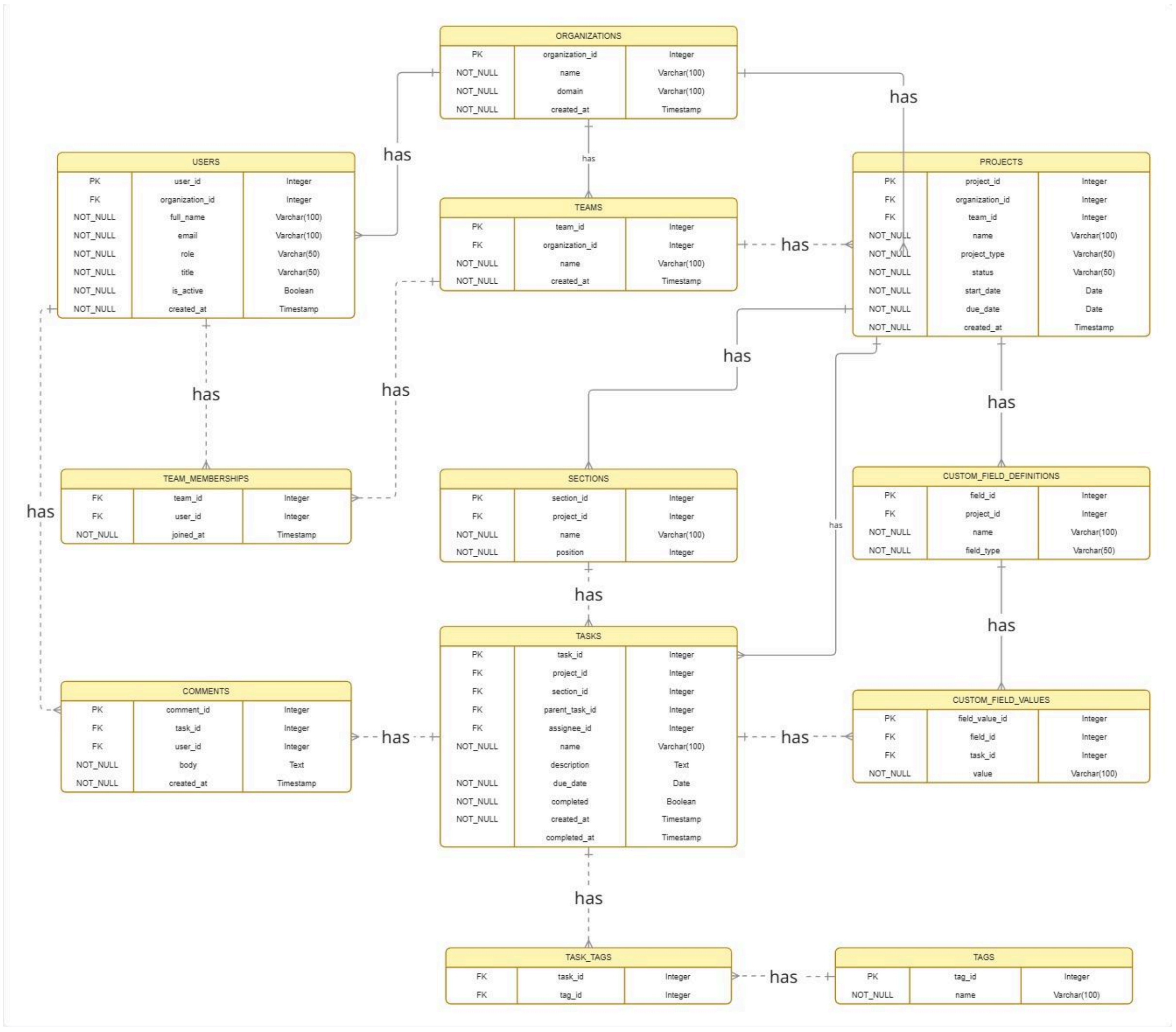
A.3 Entity–Relationship Diagram

The following Entity–Relationship (ER) diagram illustrates the structure of the simulated Asana workspace and the relationships between core entities.

Diagram Description:

- A single **organization** contains many **teams**, **users**, and **projects**
- **Users** can belong to multiple **teams** via `team_memberships`
- **Projects** optionally belong to a primary **team**
- **Projects** contain multiple **sections**
- **Tasks** belong to projects and sections
- **Subtasks** are represented as tasks with a `parent_task_id`
- **Tasks** can have many **comments**
- **Custom fields** are defined per project and attached to tasks
- **Tags** can be applied to tasks across projects

ER Diagram:


ORGANIZATIONS
PK organization_id Integer
NOT_NULL name Varchar(100)
NOT_NULL domain Varchar(100)
NOT_NULL created_at Timestamp
USERS
PK user_id Integer
FK organization_id Integer
NOT_NULL full_name Varchar(100)
NOT_NULL email Varchar(100)
NOT_NULL role Varchar(50)
NOT_NULL title Varchar(50)
NOT_NULL is_active Boolean
NOT_NULL created_at Timestamp
TEAMS
PK team_id Integer
FK organization_id Integer
NOT_NULL name Varchar(100)
NOT_NULL created_at Timestamp
PROJECTS
PK project_id Integer
FK organization_id Integer
FK team_id Integer
NOT_NULL name Varchar(100)
NOT_NULL project_type Varchar(50)
NOT_NULL status Varchar(50)
NOT_NULL start_date Date
NOT_NULL due_date Date
NOT_NULL created_at Timestamp
TEAM_MEMBERSHIPS
FK team_id Integer
FK user_id Integer
NOT_NULL joined_at Timestamp
SECTIONS
PK section_id Integer
FK project_id Integer
NOT_NULL name Varchar(100)
NOT_NULL position Integer
CUSTOM_FIELD_DEFINITIONS
PK field_id Integer
FK project_id Integer
NOT_NULL name Varchar(100)
NOT_NULL field_type Varchar(50)
COMMENTS
PK comment_id Integer
FK task_id Integer
FK user_id Integer
NOT_NULL body Text
NOT_NULL created_at Timestamp
TASKS
PK task_id Integer
FK project_id Integer
FK section_id Integer
FK parent_task_id Integer
FK assignee_id Integer
NOT_NULL name Varchar(100)
description Text
NOT_NULL due_date Date
NOT_NULL completed Boolean
NOT_NULL created_at Timestamp
completed_at Timestamp
CUSTOM_FIELD_VALUES
PK field_value_id Integer
FK field_id Integer
FK task_id Integer
NOT_NULL value Varchar(100)
TASK_TAGS
FK task_id Integer
FK tag_id Integer
TAGS
PK tag_id Integer
NOT_NULL name Varchar(100)
has

A.4 Key Schema Design Decisions

A.4.1 Task Hierarchy (Tasks vs Subtasks)

Tasks and subtasks are stored in a single `tasks` table. Subtasks are represented by setting the `parent_task_id` field to reference another task.

Rationale:

- This mirrors Asana's internal representation, where subtasks are first-class tasks
- Enables arbitrary nesting depth without schema changes
- Simplifies querying and avoids redundant tables
- Allows consistent handling of metadata (assignees, due dates, comments)

A.4.2 Custom Fields Representation

Custom fields are modeled using two tables:

- `custom_field_definitions` – defines metadata fields at the project level
- `custom_field_values` – stores task-level values for those fields

Rationale:

- In real Asana usage, custom fields vary significantly between projects
- Avoids unrealistic global schemas where all projects share identical metadata
- Enables partial usage (not all tasks populate all fields)
- Prevents RL agents from exploiting uniform metadata patterns

---

A.4.3 Team Memberships and Ownership

Users and teams are connected through a many-to-many relationship via `team_memberships`. Projects optionally reference a primary owning team.

Rationale:

- Reflects real-world collaboration where users belong to multiple teams
- Supports cross-functional projects without artificial constraints
- Allows realistic assignment logic for tasks and comments

A.4.4 Temporal Fields and Consistency

All major entities include timestamp fields (`created_at`, `completed_at`, `joined_at`) to model temporal dynamics.

Rationale:

- Enables time-aware reasoning for RL agents
- Prevents invalid states (e.g., tasks completed before creation)
- Supports realistic historical growth patterns

A.4.5 Identifier Strategy

All primary keys are stored as UUID-based `TEXT` fields.

Rationale:

- Simulates Asana's Global ID (GID) system
- Avoids sequential ID shortcuts

- Ensures uniqueness across distributed generation

Section B:Seed Data Methodology

This section describes the methodology used to generate realistic, enterprise-grade seed data for simulating an Asana workspace. For each core entity, we outline the data generation strategy, real-world inspiration, distributional assumptions, and consistency constraints applied during generation.

The methodology is designed to avoid synthetic shortcuts (e.g., uniform timestamps or placeholder names) and instead reflect the partial, noisy, and heterogeneous nature of real enterprise Asana usage. All data is generated programmatically using deterministic rules, probabilistic distributions, and structured templates to ensure reproducibility and realism at scale.

B.1 Organizations

Column	Data Type	Source Strategy	Methodology & Justification
organization_id	TEXT (UUID)	Generated	UUIDv4 identifiers are used to simulate Asana Global IDs (GIDs) and avoid sequential ID shortcuts that agents could exploit.
name	TEXT	Curated	A realistic B2B SaaS company name is used to ground the dataset in a plausible enterprise context.
domain	TEXT	Derived	Email domain is derived directly from the organization name to ensure consistency with user email addresses.

created_at	TIMESTAMP	Synthetic (heuristic)	Set several years in the past to simulate a mature organization with sufficient historical depth.

Design Note:
Only a single organization is simulated. In real-world enterprise usage, companies typically operate within a single Asana organization, and modeling multiple organizations does not meaningfully improve RL evaluation for computer-use agents.

B.2 Teams

Column	Data Type	Source Strategy	Methodology & Justification
team_id	TEXT (UUID)	Generated	UUIDv4 identifiers simulate Asana GIDs and ensure uniqueness.
organization_id	TEXT (FK)	Derived	All teams belong to the single organization to preserve hierarchy.
name	TEXT	Template-based	Team names follow common functional patterns observed in SaaS organizations (e.g., Engineering, Marketing, Operations, Data).
created_at	TIMESTAMP	Synthetic	Team creation times are distributed across the organization’s lifetime to reflect organizational growth.

Distribution Assumptions:

- Total teams: ~20–40
- Engineering teams are more numerous than Operations or Support teams
- Team sizes are intentionally non-uniform

B.3 Users

Column	Data Type	Source Strategy	Methodology & Justification
user_id	TEXT (UUID)	Generated	UUIDv4 identifiers prevent sequential ID patterns.
organization_id	TEXT (FK)	Derived	All users belong to the same organization.
full_name	TEXT	Faker	Names are generated using Faker to reflect realistic human naming distributions.
email	TEXT	Derived	Emails follow a standard enterprise format (firstname.lastname@company.com).
role	TEXT	Categorical (weighted)	Roles are assigned with weighted probabilities (admins are rare, members common, guests present).
title	TEXT	Template-based	Job titles are chosen based on typical SaaS roles (engineer, PM, designer, marketer).
is_active	BOOLEAN	Synthetic	A small fraction of users are inactive to reflect churn and legacy accounts.
created_at	TIMESTAMP	Synthetic (distribution-based)	User creation timestamps are spread over multiple years, with increased hiring in recent periods.

Realism Considerations:

- Not all users join at the same time
- Senior and admin roles are statistically rarer
- Guest users model contractors and external collaborators

B.4 Team Memberships

Column	Data Type	Source Strategy	Methodology & Justification
team_id	TEXT (FK)	Derived	References an existing team.
user_id	TEXT (FK)	Derived	References an existing user.
joined_at	TIMESTAMP	Synthetic	Membership timestamps are always after both the user and team creation times.

Assignment Rules:

- ~70% of users belong to exactly one team
- ~25% belong to two teams
- ~5% belong to three teams

This reflects real-world collaboration patterns where managers and senior contributors participate across teams.

B.5 Projects

Column	Data Type	Source Strategy	Methodology & Justification

project_id	TEXT (UUID)	Generated	UUIDv4 identifiers simulate Asana GIDs and prevent agents from exploiting sequential ordering.
organization_id	TEXT (FK)	Derived	All projects belong to the single organization, matching enterprise Asana usage.
team_id	TEXT (FK, nullable)	Derived	Most projects are owned by a primary team; a minority are cross-functional and intentionally have no owning team.
name	TEXT	Template-based	Project names follow realistic patterns inspired by public Asana templates and product roadmaps (e.g., “Q3 Platform Improvements”, “Customer Onboarding Revamp”).
project_type	TEXT	Categorical	Assigned from a controlled set (Sprint, Roadmap, Marketing Campaign, Operations, Bug Tracking, Ongoing) based on team function.
status	TEXT	Synthetic (state-based)	Project status is sampled from {active, completed, archived}, with older projects more likely to be completed or archived.
start_date	DATE	Synthetic (heuristic)	Typically precedes project creation by a small margin to reflect planning before formal kickoff.
due_date	DATE	Synthetic (distribution-based)	Due dates depend on project type (e.g., sprints 2–4 weeks, campaigns 1–3

			months). Some projects intentionally have no due date.
created_at	TIMESTAMP	Synthetic (temporal)	Project creation dates follow organizational growth, with higher density in recent months.

Realism Notes:

- Several hundred projects are generated
- Engineering teams produce more sprint and bug-tracking projects
- Marketing teams produce campaign-oriented projects
- Operations teams favor ongoing projects

B.6 Sections

Column	Data Type	Source Strategy	Methodology & Justification
section_id	TEXT (UUID)	Generated	UUIDv4 identifiers ensure uniqueness.
project_id	TEXT (FK)	Derived	Each section belongs to exactly one project.
name	TEXT	Template-based	Section names depend on project type (e.g., Backlog, To Do, In Progress, Review, Done).
position	INTEGER	Generated	Integer ordering preserves section order as displayed in the Asana UI.

Design Considerations:

- Sprint and bug-tracking projects typically contain 4–6 sections
- Some projects intentionally contain empty sections to reflect early-stage planning
- Section ordering is consistent within a project but varies across projects

B.7 Tasks and Subtasks

Column	Data Type	Source Strategy	Methodology & Justification
task_id	TEXT (UUID)	Generated	UUIDv4 identifiers simulate Asana GIDs.
project_id	TEXT (FK)	Derived	Each task belongs to a single project.
section_id	TEXT (FK, nullable)	Derived	Most tasks are assigned to a section; a small fraction remain unsectioned to model backlog or inbox-style tasks.
parent_task_id	TEXT (FK, nullable)	Derived	Null for top-level tasks; non-null indicates a subtask referencing another task.
assignee_id	TEXT (FK, nullable)	Derived	Assignees are chosen from relevant teams. Approximately 15% of tasks are intentionally unassigned.
name	TEXT	Template-based	Task names follow patterns inspired by real Asana usage and public issue trackers (e.g., “[Component] – [Action] – [Detail]”).
description	TEXT	Template-based	Descriptions vary in length: some empty, some short, and some detailed with structured text.

due_date	DATE	Synthetic (distribution-based)	Due dates follow realistic planning horizons; weekends are usually avoided. A small fraction of tasks are overdue.
completed	BOOLEAN	Synthetic (heuristic)	Completion probability varies by project type (higher for sprints, lower for ongoing work).
created_at	TIMESTAMP	Synthetic (temporal)	Task creation is more frequent on weekdays and peaks earlier in the week.
completed_at	TIMESTAMP	Derived	If completed, always occurs after creation and within a realistic cycle-time window.

Hierarchy Rules:

- Tasks and subtasks are stored in a single table
- Approximately 25–35% of tasks are subtasks
- Subtasks always belong to the same project as their parent

B.8 Comments

Column	Data Type	Source Strategy	Methodology & Justification
comment_id	TEXT (UUID)	Generated	UUIDv4 identifiers ensure uniqueness.
task_id	TEXT (FK)	Derived	Each comment is attached to a valid task.

user_id	TEXT (FK)	Derived	Comment authors are typically the task assignee or collaborators from the same team.
body	TEXT	Template-based	Comment content includes updates, clarifications, approvals, and follow-ups inspired by real Asana discussions.
created_at	TIMESTAMP	Synthetic	Comment timestamps always occur after task creation and before task completion (if completed).

B.9 Custom Field Definitions

Column	Data Type	Source Strategy	Methodology & Justification
field_id	TEXT (UUID)	Generated	UUIDv4 identifiers simulate Asana Global IDs and ensure uniqueness across custom fields.
project_id	TEXT (FK)	Derived	Custom fields are defined at the project level, reflecting how Asana allows different metadata per project.
name	TEXT	Template-based	Field names are selected based on project type and team function (e.g., Priority, Effort, Sprint, Status, Impact), inspired by public Asana templates.
field_type	TEXT	Categorical	Field types are chosen from {enum, number, text, boolean}. Engineering projects favor numeric effort fields, while marketing projects favor enum-based priority or status fields.

Design Rationale:
Custom fields vary significantly between projects in real Asana usage. Defining them per project avoids unrealistic global schemas and prevents RL agents from exploiting uniform metadata across unrelated workflows.

B.10 Custom Field Values

Column	Data Type	Source Strategy	Methodology & Justification
field_value_id	TEXT (UUID)	Generated	UUIDv4 identifiers ensure uniqueness and mirror Asana’s internal object identifiers.
field_id	TEXT (FK)	Derived	References a valid custom field definition for the task’s project.
task_id	TEXT (FK)	Derived	Custom field values are attached only to tasks belonging to the same project as the field definition.
value	TEXT	Synthetic + constrained	Values are generated according to field type: enums from predefined option sets, numeric values within realistic bounds, and booleans for binary states.

Coverage Rules:

- Not all tasks populate all custom fields
- ~70–80% coverage for core fields (e.g., Priority)
- ~30–50% coverage for optional fields (e.g., Effort, Impact)

This reflects partial and inconsistent metadata usage observed in real enterprise Asana workspaces.

B.11 Tags

Colum n	Data Type	Source Strategy	Methodology & Justification
tag_id	TEXT (UUID)	Generated	UUIDv4 identifiers ensure unique tag identity.
name	TEXT	Curated	Tag names are short, reusable labels such as “bug”, “urgent”, “blocked”, “frontend”, “backend”, and “customer-request”, inspired by real Asana and issue-tracker usage.

Realism Notes:

- Tags are global within the organization
- The total number of tags is intentionally limited (typically fewer than 100)
- Tags are reused across many projects to encourage cross-cutting categorization

---

B.12 Task Tags

Column	Data Type	Source Strategy	Methodology & Justification
task_id	TEXT (FK)	Derived	References a valid task.
tag_id	TEXT (FK)	Derived	References a valid tag.

Assignment Rules:

- ~40–60% of tasks have at least one tag
- Most tagged tasks have 1–2 tags
- Bug-related and urgent tasks are more likely to be tagged

This reflects real-world usage where tags are applied selectively rather than universally.

B.13 Metadata Consistency & Integrity Rules

The following constraints are enforced for custom metadata and tagging:

- Custom field values only exist for tasks within the same project as the field definition
- Enum custom fields only use allowed option values
- Numeric custom fields follow realistic bounds (e.g., effort points between 1–13)
- Tags are never duplicated and are reused across tasks
- No orphaned custom field or tag references exist

B.14 Global Temporal & Relational Consistency Rules

The following constraints are enforced across all generated data:

- Tasks cannot be completed before they are created
- Subtasks always reference an existing parent task within the same project
- Comments always occur after task creation
- Task assignees belong to teams associated with the project
- Overdue tasks have a due date in the past and are not marked completed

---

Conclusion

This work demonstrates a methodology for constructing realistic, enterprise-grade seed data for simulating an Asana workspace in reinforcement learning environments. By grounding data generation decisions in real-world usage patterns, enforcing temporal and relational consistency, and avoiding synthetic shortcuts, the resulting dataset supports meaningful evaluation of computer-use AI agents.

The generated schema and data are intentionally designed to reflect the ambiguity, partial metadata, and heterogeneous workflows observed in real enterprise project management systems, enabling downstream agents to learn behaviors that generalize beyond synthetic environments.
