

CRM Schema Analysis for Multi-Tenant SaaS Platform

Part 1: Shadow Users and Groups Integration

Background: *Shadow users* and *shadow groups* are representations of tenant service users and groups within the CRM context. They enable the CRM to reference tenant-wide users and teams for assignment, visibility, and automation, without the CRM owning the identity data. The CRM's schema (modeled after a lightweight HubSpot) currently includes entities such as pipeline, pipeline_stage, list, list_membership, deal, activity, contact, company, lead, and association. We assess each for integration points with shadow users/groups, identify schema gaps, and recommend changes:

Pipeline

- **Integration Points:** Pipelines might be tied to teams or owners for visibility. Currently, a pipeline is likely global per tenant. Introducing an **owner (shadow user)** or **owning group** field could allow associating a pipeline with a specific user (e.g. pipeline creator) or a team (e.g. sales team pipeline). This could control who can modify or access the pipeline.
- **Schema Gaps:** The schema likely lacks any user/group reference on pipelines (no owner or team field). There is no direct link to tenant users or groups.
- **Recommendations:** If pipelines should be private or departmental, add an optional `owner_user_id` (shadow user reference) and/or `owner_group_id` (shadow group reference). This would let a pipeline be owned by a user or restricted to a group. If pipelines are meant to be tenant-wide, ensure at least a `created_by_user` is stored for audit. Consider using tenant group assignments events (e.g. `tenant_group_assignment.created` carrying `user_id` and `group_id` ¹) to auto-create or permission pipelines per group as needed.

Pipeline Stage

- **Integration Points:** Pipeline stages are typically a sub-entity of pipelines and not directly user-specific. However, stage transitions might trigger user/group notifications or tasks. For example, entering a stage could assign a Flowable task to a group or user.
- **Schema Gaps:** Likely no user/group fields on stages, which is acceptable as stages are pipeline-scoped. However, there is no way to mark a stage as a group's responsibility or final approval by a user.
- **Recommendations:** Generally no schema change needed on pipeline_stage. Instead, leverage automation: e.g., when a record enters a stage, use the stage's configuration (or naming conventions) to assign tasks via Flowable to certain groups. Ensure the automation subsystem can map a stage to a shadow group or user for notifications (this can be handled in configuration or process logic rather than the stage schema). If needed, a metadata field on stages (like `auto_assignment_group_key`) could store a group identifier to auto-assign tasks at that stage.

List

- **Integration Points:** Lists (segments) group contacts (or other records) and could be **owned by a user** (who created or manages the list) or **shared with a group** for collaboration. In a multi-tenant scenario with role-based access, a list might need an owner user and possibly permissions for certain groups.
- **Schema Gaps:** The current schema likely has `list(id, tenant_id, name, ...)` but no `created_by` or `owner_user_id`. There's no notion of which team can see a list. All lists may be global within the tenant.
- **Recommendations:** Add a `created_by_user_id` (shadow user) to track who owns the list. If access control is needed, add an `owner_group_id` or a separate join table to share lists with groups. This allows, for example, marketing lists only visible to the marketing group. Absent that, all users in a tenant will see all lists. Also, consider a `visibility` flag (public/private) if needed. These changes ensure that list definitions honor tenant service user permissions (e.g. only the creator or their group can modify).

List Membership

- **Integration Points:** List membership links contacts (or other objects) to lists. There's minimal direct user/group involvement since it's essentially a mapping. However, membership could implicitly reflect user actions (who added/removed a contact).
- **Schema Gaps:** No user reference in `list_membership` (likely just `list_id` and `contact_id` or similar). No tracking of `added_by_user`.
- **Recommendations:** Generally, this can remain as-is for static lists. If auditing is needed, a timestamp and `added_by_user_id` could be added to each membership entry to record which user (shadow) performed the inclusion. For dynamic lists (if implemented later), the system would manage memberships, so user attribution is less relevant. Ensure membership updates (adds/removals) trigger appropriate events for automation, possibly carrying the acting user ID for logging.

Deal

- **Integration Points:** Deals (sales opportunities) typically have an **owner (assignee)** which should map to a shadow user. They may also be assigned to a *team/queue*, especially for round-robin assignments or when unowned. Shadow users represent sales reps, and shadow groups can represent teams or regions. The CRM should integrate by linking each deal to its owner user and possibly an owning group.
- **Schema Gaps:** The current deal schema might lack an explicit owner field. HubSpot's deals have `hubspot_owner_id` for the owner ². If our schema doesn't have `owner_user_id`, that's a gap in assignment and visibility control. Additionally, there may be no mechanism to assign deals to a group (e.g. a sales queue) for group ownership.
- **Recommendations:** Add an `owner_user_id` (foreign key to shadow user) to the deal table so each deal can be owned by a tenant user. Also consider an `owner_group_id` for scenarios like team-based selling or shared ownership. Alternatively, use the association table to associate deals with multiple users (collaborators) or groups, but a direct field for primary owner simplifies queries. With these in place, automation can, for instance, assign new inbound deals to a group (e.g. "SDR Team") – the Flowable workflow could set the `owner_group_id` and leave `owner_user_id` null until a user claims it. Ensure that deal-related processes (stage changes, close won/lost) incorporate

user context (e.g. who moved the stage). Tenant service user events should be consumed to update deal ownership when a user leaves or changes (possibly reassign deals of a deactivated user to another user or to that user's group for redistribution).

Activity

- **Integration Points:** Activities (calls, meetings, tasks, emails, etc.) are often assigned to users or teams. For example, a task might be assigned to *John Doe* (shadow user) or to a *Support Group* (shadow group). They are also logged by users when completed. Integration requires linking activities to their **owner/assignee user**, the **creator user**, and possibly a **responsible group** for unassigned tasks.
- **Schema Gaps:** The current activity schema is likely minimal (e.g. `activity(id, type, timestamp, details, associated_record)` etc.) with no `assigned_to` or `created_by` fields. This prevents tracking who is responsible or who performed the activity. Without user integration, activities cannot be used for personal task lists or group work queues.
- **Recommendations:** Extend the activity schema with:
 - `created_by_user_id` - the shadow user who logged or created the activity (for audit).
 - `assigned_to_user_id` - the shadow user responsible (for tasks or follow-ups).
 - `assigned_group_id` - a shadow group responsible (for activities placed in a team queue).

These allow, for example, a Flowable automation to create a follow-up task assigned to a *group* instead of a specific person, which all group members can see and claim. (Flowable supports assigning tasks to multiple candidates or groups ³.) An activity like a call log would have `created_by_user_id` (the rep who made the call) and maybe no assignee since it's completed. A scheduled task could have an `assigned_to_user_id` or group for completion. By linking shadow users, the CRM can display activity timelines with the user's name and filter activities by owner or team (e.g. "Show me all tasks assigned to *Team Alpha*"). Ensure association entries exist linking activities to relevant records (contact, deal, etc.) so that user-specific access control (e.g. only owners see certain activities) can be enforced if needed.

Contact

- **Integration Points:** Contacts (persons) in CRM are usually owned by a rep (user) and possibly accessible only to certain teams. A shadow user link is needed for contact owner. Additionally, contacts might be shared with groups (e.g. a buying committee might be visible to both sales and support teams).
- **Schema Gaps:** If modeled after HubSpot, contact ownership might not have been fully built out originally (HubSpot uses a property for owner). The schema may lack `owner_user_id` on contact. It likely contains basic fields (name, email, etc.) and `tenant_id`, but not the user assignment or team visibility controls.
- **Recommendations:** Add an `owner_user_id` to contacts to designate the primary owner (shadow user). This allows assigning each contact to a tenant user responsible for that relationship. Optionally, include an `owner_group_id` if contacts can be owned by a team (or use group assignment via association table). Also, update any *lead routing* or *contact creation* logic to utilize tenant group assignments: e.g., new contacts from a particular source could default to an owner group (like "Unassigned Leads" group) for triage. With this, when a user leaves, contacts can be reassigned based on group membership or another user. The tenant service's user deletion events would trigger such reassignments or deactivation in CRM to prevent orphaned contacts.

Company

- **Integration Points:** Companies (accounts) similarly should have an owner user (account manager) and possibly be linked to a group (account team). Shadow user integration means linking the company to a user for accountability. Group integration might mean certain teams manage certain companies (e.g. an enterprise account managed by a team).
- **Schema Gaps:** The current schema likely lacks an explicit owner for companies as well. Without an `owner_user_id`, all company records may be effectively unowned or implicitly owned by the tenant. This is a gap for accountability and territory management.
- **Recommendations:** Include `owner_user_id` on the company entity. In multi-tenant CRMs, this is crucial for sales alignment. Also consider a `customer_success_group_id` or similar if different groups (sales, support) have ownership roles on a company. Alternatively, use the association table to attach multiple users to a company with roles (e.g. one user as “Sales Owner”, another as “Support Rep”). If the schema uses an association/label model for ownership, define association types for ownership links. Ensure that any user or group referenced respects the tenant context (only shadow users of that same tenant are valid owners). These links will allow automation (like assignment rules or escalations) to reference the correct people – for instance, notifying the account owner’s group when a high-value deal is created.

Lead

- **Integration Points:** If leads are separate from contacts (as in some CRMs), they should integrate with users/groups for assignment during the qualification stage. Often, leads are in an *open queue (group)* until claimed by a user, or directly assigned to a user. Shadow users would correspond to sales development reps, and groups might represent queues like “Inbound Leads”.
- **Schema Gaps:** The lead schema likely mirrors contact (name, contact info, etc.) but might not have an owner field if it was simplified. Without user assignment, distributing and tracking leads is problematic.
- **Recommendations:** Add `owner_user_id` to lead, similar to contact. Additionally, a `lead_status` field often exists; if not, consider adding it (though not directly about users, it helps automation). To use groups, incorporate a concept of a *lead queue*: a lead could have an `owner_group_id` when unassigned. Workflow can assign new leads to a group (e.g. *inside sales*) who then claim it, at which point `owner_user_id` is set. This approach mirrors common CRM patterns where new leads go to a round-robin or queue. Leverage tenant group assignment events to manage these queues – e.g., if a user is removed from the “Inbound Leads” group, the CRM could auto-reassign that user’s open leads to the group or a replacement user.

Association

- **Integration Points:** The association table likely stores relationships between core objects (e.g. contact-company, deal-contact, deal-company). With user integration, we can extend associations to link records to users or groups as well. For example, multiple sales reps (users) associated to a deal (beyond the primary owner) as collaborators, or a support group associated to a company as the service team.
- **Schema Gaps:** Originally, user and group were not CRM entities, so the association model might not include them. There may be an enum or object type field in association that doesn’t account for a “user” or “group” object type.

- **Recommendations:** Update the association schema to allow `user` and `group` as valid associated object types. Define new association types for these links (for instance, an association type “COLLABORATOR” to link a deal to a user who helps on it, or “TEAM_RESPONSIBLE” to link a group to an account). This is more flexible than adding multiple explicit fields on the main tables for every secondary relationship. By leveraging the generic association table, the CRM can represent complex relationships (e.g. multiple contacts on a deal, or multiple reps on an account) just as HubSpot does via association definitions ⁴. Ensure that any such association respects tenant boundaries and is only created in response to valid events (like a user assignment event or an explicit user action in the CRM UI).

Summary of Part 1: Shadow users and groups should be woven into the CRM schema primarily through **ownership and assignment fields** on core records and by **extending the association model**. Key additions include owner user/group foreign keys on records (pipeline, deal, contact, company, etc.) and audit fields (created_by, assigned_to). There were no fundamental structural blockers (the schema supports tenant_id everywhere to isolate data), but the lack of user/group references is a gap. These schema changes, combined with listening to tenant service events (for user/group creation, updates, membership changes ⁵), will ensure the CRM stays in sync with the tenant directory. For example, when `tenant_mgr_srv.tenant_user.deleted` fires for a user leaving ⁶ ⁷, the CRM can find records owned by that shadow user and notify a group or reassign them, preventing orphaned records. Likewise, Flowable-driven processes can assign tasks to groups or users by leveraging these new links (Flowable user tasks support multiple candidates and groups ³). Overall, integrating shadow identities will enhance assignment workflows, data security (only authorized group members see certain records), and automation capabilities across the CRM.

Part 2: Feature Gap Analysis vs HubSpot (Pipeline, Stage, List, Deal, Activity, etc.)

The current CRM schema and features are compared against HubSpot’s CRM for the equivalent objects. We identify missing fields, states, or automation features for each entity, categorize each gap, and note any areas of strength. (We focus solely on HubSpot for comparison, as requested, not blending other CRMs.)

Pipeline

HubSpot Reference: In HubSpot, *pipelines* are used for deals, tickets, and other objects to track stages of a process. HubSpot supports multiple pipelines per object type (e.g., multiple deal pipelines for different sales processes) ⁸. Each pipeline has properties like a label (name), display order, and a set of stages. HubSpot pipelines can be activated/archived and are manageable via API (create, update, delete) ⁹ ¹⁰. They are also tied into forecasting (for deals) via probabilities on stages ¹¹.

- **Structural Gaps:** Our CRM likely has a basic `pipeline` table with `pipeline_id`, `name`, ... and is associated mainly with deals. It may not support multiple pipeline types or multiple pipelines per object. For example, HubSpot allows distinct pipelines for deals vs. tickets, etc., whereas our design might only implement a sales (deal) pipeline. If the current schema doesn’t indicate an object type or owner for a pipeline, that’s a limitation. Another structural gap is the absence of a *display order* field; HubSpot pipelines and stages use ordering for UI display ¹², which our schema might not capture explicitly. Also, HubSpot pipelines have an *internal ID* and a user-defined label – our

schema might be using an ID as primary key but might not expose a user-editable key or support pipeline *slug* (for API consistency).

- **Behavioral Gaps:** HubSpot allows reordering and editing pipelines via APIs and UI, and even tracking pipeline changes via events. If our CRM doesn't support creating additional pipelines or reordering, that's a behavioral gap. Another potential gap is *reference checks* on deletion – HubSpot's API prevents deleting a pipeline that has deals unless forced ¹⁰ – our CRM might not enforce this, risking accidental data loss. Also, HubSpot pipelines are associated with permissions (who can create/edit them); if our system lacks role-based controls on pipeline management, that's a behavioral difference.
- **Analytical Gaps:** Without multiple pipelines or pipeline types, our CRM might not handle scenarios like separate sales vs. renewals pipelines (common in HubSpot ¹³). Missing probability or category fields on pipeline stages (see Pipeline Stage below) means the pipeline can't directly support weighted forecasting. In HubSpot, the pipeline is central to forecasting revenue and identifying bottlenecks ¹⁴. If our schema doesn't integrate pipeline data into reports (e.g., no easy way to get count of deals per stage or track stage history), that's an analytical weakness. However, one area our schema could be "stronger" is simplicity – a single pipeline is easier to manage for small teams. Simplicity can aid automation alignment if all deals follow one flow, but it becomes a limitation as processes diversify.

Pipeline Stage

HubSpot Reference: Each pipeline in HubSpot has defined *stages*. Stages in HubSpot have a unique label, display order, and for deals, a **probability** (0–100% in UI or 0.0–1.0 in API) indicating likelihood of closing ¹¹. For ticket pipelines, stages include a status (Open/Closed) metadata ¹⁵. Stages can be marked as *Closed Won* or *Closed Lost* implicitly by their probability (1.0 or 0.0) for deals, and HubSpot's UI treats the last stage as closed won/lost categories for reporting. HubSpot also tracks stage **change dates** (when a deal moved into a stage) in property history for analytics like sales velocity.

- **Structural Gaps:** Our `pipeline_stage` schema likely has `stage_id, pipeline_id, name, order`. It probably lacks the **probability** field that HubSpot requires for deal stages ¹⁶. Without probability or an equivalent success indicator, the system cannot do weighted forecasting or easily identify which stages are terminal vs ongoing. Another gap is the *ticket state* concept (Open vs Closed) that HubSpot uses for service pipelines ¹⁵ – if our CRM will extend to cases or tickets, our stage model might not support marking certain stages as closed/resolved. Additionally, HubSpot stages have an internal ID separate from the name; if our system keys off stage name or doesn't allow stage re-labeling easily, that's less flexible.
- **Behavioral Gaps:** HubSpot's pipeline stages can be dynamically added and re-ordered through the API or UI. If our CRM doesn't allow stage reordering or adding new stages without a migration script, that is a usability gap. Another behavioral aspect: HubSpot automatically moves deals to Closed Won/Lost categories (or requires the user to pick a "Closed" type stage) – our CRM might not enforce any terminal stage logic, leaving it to user convention. Also, HubSpot can trigger **automation** (workflows) based on stage changes (e.g., send email when deal enters a stage). Without integration to an automation engine at the stage level, our CRM might rely on manual triggers. (However, since we have Flowable, we could configure process triggers on stage changes – but that might need a mechanism to detect stage transitions, which may not be baked in.)
- **Analytical Gaps:** The absence of probabilities means no native forecasting in reports (e.g., sum of deal amount * probability per stage). If stage change timestamps are not recorded (HubSpot keeps property history for `dealstage` changes ¹⁷), we cannot easily calculate metrics like *average time*

in stage or *conversion rates* between stages. These are critical for sales pipeline analysis. One strength of our schema might be that it's simpler to query current stage (fewer fields), but overall it's missing rich data. To align more with HubSpot's analytical capabilities, we'd need to add fields or track history to determine how deals progress through stages and with what outcomes.

List & List Membership

HubSpot Reference: HubSpot *lists* (now also called segments) are used to group contacts (or companies) by criteria. HubSpot supports **Static lists** (manual) and **Active lists** (dynamic, rule-based) ¹⁸, as well as an intermediate *SNAPSHOT* type ¹⁹. Lists have properties like a name, an object type (e.g., contacts or companies), processing type (manual/dynamic) ²⁰, and filter criteria (for dynamic lists). *List membership* in HubSpot is the relationship indicating a contact belongs to a list. HubSpot's dynamic lists automatically add/remove contacts based on criteria in the background, whereas static list membership changes are user-driven or via API. Lists can also be used in marketing emails, workflows, etc., and have analytics (e.g., size of list, growth over time).

- **Structural Gaps:** Our CRM's `list` is likely a straightforward table with `list_id`, `tenant_id`, `name`. It probably does **not** have a field for *processing type* (no notion of dynamic vs static), nor a place to store filter criteria (which in HubSpot is a complex JSON of rules). This means we likely only support static lists. Also, if the schema is contact-centric, it may not allow lists of companies or deals, whereas HubSpot can have company lists as well ²¹. The list membership (`list_membership`) table in our schema likely has `list_id`, `contact_id` (or `lead_id`) as a composite key. It might lack a primary key or timestamps.
- **Behavioral Gaps:** Without dynamic lists, our users must manually maintain list membership or rely on external processes to update lists. HubSpot's dynamic lists continuously evaluate contacts against filters and update membership in real-time or on a schedule ²². That automated behavior is not present – our CRM can't auto-segment contacts by properties (e.g., "all contacts from Company X" or "leads from last month"), unless custom code or Flowable processes are implemented. Additionally, HubSpot provides a feature to convert active (dynamic) lists to static ²³ ¹⁸, and to refresh or recalc lists; our system likely has no parallel, since dynamic lists aren't supported. There may also be no concept of *archiving* or *deleting* lists in the UI (whereas HubSpot allows deletion and even restoring lists ²⁴).
- **Analytical Gaps:** HubSpot lists can be used to drive dashboards (e.g., measure how a segment grows over time, or use list membership as criteria in reports). If our lists are static only and lack metadata (like created date, criteria used), their analytical value is limited to simple counts. Also, if we don't timestamp list membership entries, we can't analyze *when* contacts joined a list or segment growth. One area our schema might be simpler/stronger is that a static list is straightforward to understand – no black-box criteria – which can be more *predictable*. Also, because we have a unified approach (likely just contact lists), it's directly analogous to HubSpot's static lists. But the lack of dynamic segmentation is a significant gap in *behavior* (automation capability) and *analysis*. A possible mitigating strength is that we could use our automation engine (Flowable) to simulate dynamic list behavior (periodically evaluating criteria and updating list memberships), but that would be a custom solution rather than built-in feature.

Deal

HubSpot Reference: In HubSpot, a *deal* represents a sales opportunity. Key HubSpot deal properties include: **amount**, **close date** (`closedate`), **deal stage** (`dealstage` which ties to a pipeline), **pipeline**

(which pipeline it belongs to), **deal name**, **deal owner** (`hubspot_owner_id` referencing a HubSpot user), among others ²⁵. HubSpot deals also have lifecycle properties like *deal probability* (inferred from stage, or custom), *deal type* (New Business vs Renewal), and *deal forecast category* (in Sales Hub Enterprise). Deals in HubSpot move through pipeline stages until Closed Won or Lost, and those outcomes are usually recorded via the stage or a separate property (`hs_stage_probability` hitting 0 or 1, or a `closed_won` boolean behind the scenes). HubSpot supports associating multiple contacts and a primary company to a deal ²⁶ ²⁷, and allows tasks/activities to be attached to deals.

- **Structural Gaps:** Our CRM's `deal` schema likely has basic fields like `id`, `name`, `pipeline_id`, `stage_id`, `amount`. Some probable omissions relative to HubSpot:
- **Close date:** If not present, that's a gap. HubSpot's `closedate` is important for forecasting and pipeline reviews ²⁸.
- **Deal owner:** Without an `owner_user_id` (as noted in Part 1, originally missing), there's no built-in assignment.
- **Deal type:** HubSpot has a default property for deal type (often *New Business* or *Existing Business*) to categorize deals. Our schema likely doesn't have this, meaning all deals are treated uniformly.
- **Probability/Forecast:** No field to override stage probability or indicate confidence. HubSpot auto-derives probability from stage but allows override or custom rollups. Our schema likely doesn't include any probability or forecast category field per deal.
- **Associations count:** HubSpot doesn't store foreign keys for associated contacts/companies on the deal record itself; instead it uses association tables. Our schema might have taken a lightweight approach – possibly storing a single `primary_contact_id` or `company_id` on deal (which is simpler but limits multiple associations). If we did not implement a join table for deals to contacts, that's a structural deviation from HubSpot's flexible association model ²⁶.
- **Behavioral Gaps:** HubSpot deals benefit from many CRM features:
- **Stage automation:** In HubSpot, moving a deal to certain stages can trigger workflows (e.g., send quote, create task). Unless our CRM's integration with Flowable explicitly covers these, we may lack out-of-the-box triggers.
- **Closing process:** HubSpot requires specifying Closed Won or Closed Lost by moving to a stage (and can capture a *Closed Lost reason* property). If our CRM doesn't enforce a specific process for closing deals (like simply marking inactive), it may lack clarity and data on lost reasons or win/loss dates.
- **Editing & Permissions:** HubSpot allows granular permissions (who can edit deals, view deals owned by team, etc.). Our CRM might not have such refined controls implemented yet (especially before shadow user integration). That's a behavioral gap in multi-user environments.
- **Multi-currency and multi-pipeline support:** HubSpot supports multiple currencies for amount (Enterprise tiers) and multiple pipelines as mentioned. If we support only a single currency globally or have no currency field (just a number), that's a gap for international usage.
- **Deal creation requirements:** HubSpot requires at least `dealname`, `dealstage`, and `pipeline` when creating via API ²⁹. Our CRM might not enforce such required fields (e.g., it might allow a deal without a name or without assigning to a pipeline if not carefully constrained). This could lead to incomplete data.
- **Analytical Gaps:** HubSpot's deals feed into robust reports – e.g., sales funnel, forecast, conversion rates, etc. Gaps in our schema that affect analysis:
 - Without close dates, you cannot easily build a *forecast by month or quarter*.
 - Without probabilities or forecast categories, weighted pipeline and forecast categories (Commit, Best Case, etc.) are absent.

- Without a lost reason or status field, analyzing why deals are lost or how many were lost is harder (HubSpot allows a custom Lost Reason property often).
- If we don't track stage history (dates of stage changes), we can't compute sales velocity metrics (average days to close, etc.). HubSpot implicitly provides this via property history and reports like *deal funnel*.

One area where our current schema might be *more automation-aligned* is that it's presumably simpler and could be tightly integrated with our automation engine: e.g., since our design is new, we might have structured it to easily trigger Flowable processes on deal events. Also, if we implemented a generic association table early, that's a plus – it means we can already handle multiple contacts per deal in a flexible way, something older CRMs struggled with (HubSpot only added flexible associations in later API versions). If our association model exists, it is a **strength**: it allows linking deals to contacts/companies freely, similar to HubSpot's approach ³⁰. Another potential strength is if our schema included an *AI-driven task subsystem* – e.g., automatically creating follow-up tasks for deals. That is not a standard HubSpot feature without user-defined workflows, so having it baked in (with AI suggestions) could be an innovation we have over HubSpot. Nonetheless, purely on fields and CRM capabilities, we have some catching up to do in order to match HubSpot's deal management depth.

Activity

HubSpot Reference: HubSpot activities (calls, emails, meetings, notes, and tasks) are collectively part of the **CRM timeline (Engagements)**. Each activity type in HubSpot has specific properties, but there are general ones common to all: - **Activity date/time** (when the interaction occurred), - **Activity type** (call, email, etc.), - **Activity owner/assigned to** (for tasks, who it's assigned to) ³¹, - **Created by** and **Last modified by** (user IDs of who logged or edited it) ³¹ ³², - **Associated records** (which contact, company, deal it's linked to), - Possibly outcome/status fields (e.g., call outcome, task status). HubSpot also has default values for each type: e.g., calls have **call direction, duration, outcome** ³³, meetings might have a location or type, tasks have a status (completed or not) and a due date, notes just have the body content. Activities are not first-class objects in HubSpot's main data model (they're available via a separate Engagements API), but they are critical for user productivity and analytics (e.g., number of calls made, tasks completed).

- **Structural Gaps:** Our CRM's `activity` table is likely a single generic table for all interaction logs. It might have fields like `id`, `type`, `timestamp`, `content`, `linked_contact_id`, `linked_deal_id` etc. Compared to HubSpot, probable missing fields include:
- **Assigned to (owner):** as noted, originally missing – now needed as `assigned_to_user_id` for tasks ³⁴.
- **Created by user:** initially absent – we should have `created_by_user_id` ³⁵.
- **Activity sub-type fields:** We probably don't have dedicated columns for call outcome, call duration, meeting type, task due date, task status, etc. HubSpot's model has a variety of properties for each activity type (as listed in their default properties docs ³⁶ ³⁷). Our single table may have to accommodate these via a generic structure (e.g., a JSON payload or a few nullable columns).
- **Attachments:** HubSpot can associate file attachments to engagements (with property "Attached file IDs") ³⁸. If our schema doesn't allow linking files to an activity (say via an attachment table or file IDs), that's a gap.
- **Behavioral Gaps:** In HubSpot:
 - Logging an activity triggers timeline updates and can trigger follow-up tasks or workflow actions. Our CRM might not have such triggers yet (unless Flowable is configured for it).

- Tasks in HubSpot can send reminders and appear in the user's task list in-app. If our tasks (as activities) don't have a concept of due date or reminder, users might not get notified of upcoming tasks – a usability gap.
- Completing activities in HubSpot sometimes auto-updates status (e.g., marking a call outcome as “completed call” or a task as done). Our CRM might treat activities as static logs with no status toggle (except possibly tasks).
- Also, HubSpot's email activities can automatically log email content and track opens/clicks if sent through HubSpot. We likely do not have an email integration of that sophistication; any email activity would be manually logged or BCCed into the system.
- **Analytical Gaps:** HubSpot provides reports on activity metrics: calls made per user, tasks completed vs overdue, average call duration, etc. Gaps impacting this:
 - If we lack structured fields (call duration, outcome, task status), we can't easily report how many calls were connected or how many tasks are overdue.
 - Without `activity_date` stored properly (and separate from create date), scheduling and historical trend analysis is difficult (HubSpot distinguishes the date the activity happened vs when it was logged ³⁹).
 - If we don't track `updated_by_user_id` or last modified time (HubSpot has these for activities ³² ⁴⁰), auditing changes (who edited a note) or calculating how often activities get updated is not possible.
- However, our unified activity table could be a strength in simplicity: all interactions in one place with a common structure might simplify building a consolidated timeline. HubSpot historically had separate engagement types but has unified them in their API. If our design was from scratch, we might already have a unified model (plus an `activity_type` field). This aligns well with automation – e.g., an AI module could scan the `activity` table for any type of interaction and analyze text or timings.
- Another possible strength is if our CRM uses AI for activity logging or follow-ups. For instance, if the AI-driven communication subsystem creates an activity when it sends an email or suggests a next step, this is a feature beyond HubSpot's basic logging (HubSpot has a relatively manual logging process or requires integration with their email/calendar).

In summary, across these entities, many **gaps are structural** – missing fields or entities (like no dynamic list support, missing owner fields, no probability on stages). **Behavioral gaps** often relate to automation and lifecycle: HubSpot's built-in automation (list processing, stage triggers) versus our reliance on manual or custom processes. **Analytical gaps** stem from not capturing data that HubSpot uses for analytics (stage timestamps, activity outcomes, etc.). There are a few areas of comparative strength in our current schema: - The likely presence of a **generic association** mechanism (if implemented) is forward-thinking and aligns with HubSpot's flexible data model (many legacy CRMs had rigid links, e.g., only one contact per deal, which HubSpot overcame; our schema listing an `association` table suggests we did too). - Our tight integration with an **automation engine (Flowable)** could allow more complex or customized workflows than HubSpot's out-of-the-box ones. For instance, HubSpot workflows are powerful but limited to what their UI allows; a Flowable process could, in theory, do anything (call external APIs, orchestrate multi-step tasks beyond HubSpot's scope). - The inclusion of **AI-driven subsystems** hints that our platform might automatically log or prompt activities (like suggesting a task after a meeting), which would be an innovative feature not inherently available in HubSpot without third-party tools.

To close the gaps, we should consider extending the schema (as discussed in Part 1 and above) and possibly building equivalent **features**: - Add needed fields (owner, dates, probabilities, etc. – many are straightforward additions). - Implement dynamic list processing (maybe via Flowable jobs or a rule engine)

to match HubSpot's segmentation power ²² . - Track historical data where relevant (stage changes, etc.) either via audit tables or at least last changed timestamps. - Leverage our automation and AI to compensate and even exceed what HubSpot does (e.g., auto-assign leads to lists or stages based on AI, which HubSpot would require user-defined workflow).

By addressing structural gaps first, we enable the behaviors and analytics to be built on a solid data foundation, bringing our lightweight HubSpot-like CRM closer to feature parity with HubSpot itself while maintaining the flexibility of our multi-tenant, automation-enhanced architecture.

Part 3: Company vs. Account Modeling

Understanding Terminology Across Platforms: Different CRMs use the terms "Company" and "Account" somewhat interchangeably, but with context-specific nuances: - **HubSpot:** Uses **Company** as the object representing a business or organization. In HubSpot's model, a Company record holds the information about an organization (name, domain, address, etc.) that you have a relationship with. HubSpot does not have an entity called "Account" in its CRM lexicon; instead, the Company serves that role (effectively equivalent to an account in other systems). HubSpot allows contacts to be associated to companies (many-to-many), and also supports a single **Parent Company** linkage to create a hierarchy ⁴¹ ⁴² . In practice, HubSpot's Company is the account-level entity – for example, HubSpot's Salesforce integration will sync a HubSpot Company to a Salesforce Account (with the caveat that HubSpot might not create a Salesforce Account if a company exists with no contacts ⁴³). - **Salesforce:** Uses **Account** as a first-class entity for the organization that you do business with. In Salesforce's data model, an Account is defined as a business entity or organization ⁴⁴ . Every contact in Salesforce is typically associated with an Account (except standalone leads). Salesforce is **account-centric**: the Account is at the top of the hierarchy, and opportunities (deals), contacts, cases, etc. roll up to the Account ⁴⁵ . Salesforce allows an Account hierarchy via a *Parent Account* field for roll-up (e.g., regional offices as separate Accounts with a parent HQ) ⁴⁶ . Notably, when you convert a Lead in Salesforce, it *must* create or attach to an Account (along with a Contact) ⁴⁷ – reflecting the idea that a qualified prospect should be tied to an account (company). The term *Account* in Salesforce often represents the **billing or customer account** – the entity with which contracts are signed. Salesforce also offers a concept of *Person Accounts* (for B2C, where an Account and Contact are merged into one record behind the scenes). - **Microsoft Dynamics (CRM):** Uses **Account** similarly to Salesforce, as the organization or company entity, and **Contact** as the individual. An Account in Dynamics represents a customer or a business relationship; it can also have a hierarchy (parent account field) and can aggregate contacts, opportunities, and cases under it. Dynamics often uses Accounts not only for customers but also for partners or vendors – essentially any company the business interacts with. The term *Company* might appear as a field (like company name on a lead or contact), but the object is Account.

In all these platforms, **Account = Company** in concept: a container for contacts and deals at an organizational level. HubSpot chooses the term "Company" for user-friendliness, whereas Salesforce/Dynamics use "Account," reflecting a more finance-centric term (customer account).

Roles of Account vs Company: - Billing/Contract Role: In Salesforce/Dynamics, an Account is typically the entity that signs deals and is billed. In HubSpot, the Company would serve this role if HubSpot is used for sales, though HubSpot itself doesn't handle billing natively. Our platform's *tenant* concept is different (that's the customer using our app), so within the CRM data, a "company" record would represent our tenants' clients. If we introduce a separate "Account" entity, one interpretation is that it could represent a higher-level grouping for billing or contractual purposes. For example, if one customer has multiple subsidiary

companies (each tracked as a Company in CRM), an Account could represent the umbrella contract or customer group. This is not explicitly needed if a parent-child hierarchy is sufficient. - **Parent Organization / Hierarchy:** All major CRMs allow linking companies in hierarchy (HubSpot's parent company, Salesforce's parent account, etc.). If our current `company` model lacks a way to link companies, it may be overloaded to represent any level of organization without hierarchy. Introducing an Account object could imply making "Account" the parent-level organization and "Company" a child (like location or division). However, this can often be handled by self-referencing links within one object. For instance, Salesforce didn't introduce a separate object for branch vs HQ; it uses parent account fields ⁴⁶. - **Aggregation:** Account often serves as an aggregation point for reporting – e.g., total sales to an account including all sub-accounts, or total open cases for an account. If our system needs to aggregate multiple companies under one umbrella for analytics, we currently might do that via a parent company link. A distinct Account entity could also serve that by grouping multiple company records.

Analysis of Current `company` Model: Our schema's `company` entity likely contains fields for company name, website domain, industry, etc., similar to HubSpot's company properties. It may currently be doing *all the work*: - Representing the organizations our users interact with (as prospects or customers). - Potentially also representing a top-level client if one organization has sub-entities (though without a hierarchy, we might just have multiple company records and no link). - Possibly being used for both customers and other organizations (partners, vendors) – all stored in the same table, distinguished perhaps by a type field or just by context.

If we say the `company` model is **overloaded**, it might mean: - We have no clear distinction between a **customer account** and a **business location**. For example, if a large enterprise customer has 5 regional offices, in our system we might enter all 5 as separate `company` records. Without a hierarchy, we can't easily tell they are related, except maybe by name. Users might create naming conventions or custom fields to link them (not ideal). - The `company` record might be used to store both what other CRMs would put in an Account (like billing details, client since date, contract value) and things that could be separate (like site/location info). This could lead to duplication or confusion if one actual client spans multiple records. - We might lack clarity on what a "Lead" conversion does in terms of company: if a new lead is qualified, do we create a company record for them always? If so, that company is acting as the account. But if multiple leads from the same real company come in, do we create duplicates? That's an area where Salesforce's account model shines (it would convert subsequent leads into the same Account if recognized). HubSpot handles this by matching domain to avoid duplicate companies.

Options to Consider:

- **Option 1: Keep Company as-is.** In this approach, we continue with a single `company` entity representing organizations, akin to HubSpot's model. We would then ensure to enhance it rather than split it:
- Add a self-referential link for parent-child (e.g., `parent_company_id` on the company table) if hierarchical relationships are needed. This would address the aggregation and parent organization use-case without introducing a new table.
- Use fields or tags to distinguish different roles of companies (e.g., a checkbox or enum for "Is Billing Account" or "Is Partner").
- Benefit: Simplicity and alignment with HubSpot – one less object for users to understand. HubSpot users are already comfortable with companies as the account analog.

- Drawback: If in future we want to attach specific billing info or contract data at an account level, we'd put it on the company record or a related contract table. That is workable, but we must be clear that one company could be the master account for others.
- **Recommendation if this path is chosen:** *Evolve the company schema* by adding an optional parent relationship and any needed fields like account number, customer tier, etc., rather than creating a separate account object.
- **Option 2: Evolve Company to behave more like Account.** This is a middling approach where we conceptually treat our `company` as the "Account" entity. This might simply be a terminology shift and schema enrichment:
 - We ensure every contact is linked to a company (like Salesforce linking contacts to accounts) to maintain an account-centric structure. (If currently contacts can exist without company, we might start encouraging linking or auto-creating company records from contact domains, similar to HubSpot's domain-based company creation feature.)
 - We might rename certain things in the UI or code to "Accounts/Companies" to indicate they're the same level as an account. We could also introduce an alias or view called "Account" if needed for integration with external systems (e.g., if syncing to an external CRM that expects accounts).
 - We implement features common to account management: for example, one company could have multiple addresses or child entries (we can implement the parent/child as above). We might add an "Account Owner" field (which is basically the same as company owner we already planned) to reinforce that someone is responsible for the overall account.
 - Essentially, this means **don't split the entity, but treat it as the top-level account record**, and adjust features around it to cover account use-cases (hierarchies, roll-up reporting, etc.).
 - This approach aligns well if our target users are more familiar with Salesforce-style thinking but we don't want to overhaul the data model. We can provide flexibility (one table, multiple roles).
- Drawback: Some may confuse terminology if coming from Salesforce ("where are my accounts? – oh they're called companies here"). This can be solved through documentation or even UI labels customization.
- **Option 3: Introduce a distinct `account` entity.** This would mean creating a new table, say `account`, and rethinking the relationship with `company`:
 - One possible model: **Account as a parent object, Company as a sub-object.** For example, an Account could represent the customer's overall relationship, and each Account could have multiple Company records (perhaps representing branches or divisions). In this design, deals might link to an Account (the ultimate billable entity) while companies represent delivery locations or child orgs. This is somewhat analogous to how Salesforce uses the Account Site field or multiple accounts with parent link – except here we'd formally separate them.
 - Another interpretation: We keep Company as currently and use Account for a different purpose, but that could be redundant. More likely we'd migrate to using Account for what we currently call Company, and perhaps repurpose Company for something else (like maybe internal companies or vendors).
 - Benefit: Clear differentiation of global account vs specific entity. For businesses with complex structures, it might model reality better (e.g., you have a "Global Account" for Acme Corp, and separate company records for Acme Corp East, Acme Corp West that link to that Account).

- It could also allow storing distinct data: e.g., Account could have fields for billing terms, Master Service Agreement info, total contract value, etc., while Company could have local office info, regional manager, etc.
- Drawbacks: This is a significant complexity increase. We'd need to migrate existing data (every Company might become an Account or be linked to one). Users would have to understand two concepts where previously one sufficed. We might inadvertently mirror Salesforce too closely, when our original model (like HubSpot's) was simpler and worked for our needs.
- Also, if not carefully planned, it could duplicate data or confuse usage (when do I create a new company vs a new account?).

Recommendation: Given our current schema is inspired by HubSpot (which successfully uses a single Company object for account-level data), it's likely best to **keep the Company model as-is (Option 1)**, but enhance it to cover any missing account-oriented features. This means: - Continue referring to the organization entity as "Company" in our CRM semantics, for consistency. - Add a self-referential parent-child relationship on Company to model hierarchies where needed (similar to HubSpot's parent company and Salesforce's parent account) ⁴⁶. This addresses the parent organization/aggregation requirement without a new entity. - Use the existing Company object to represent the "account" in all senses: it can be the billing entity, the top of hierarchy, etc., depending on how it's linked. We can add fields like *account_number* or *customer_category* to Company if distinguishing internal roles (billing account vs prospect) is required. - Clearly document that in our CRM, Company = Account (just different naming). If integration with systems like Salesforce is in scope, map our Company to their Account in integration code, which is straightforward (indeed, HubSpot's integration does exactly this mapping ⁴⁸). - If there is concern about overloading, we can introduce the notion of *Company Type* (for instance, a dropdown to mark a company record as a Customer vs. Vendor vs. Partner). This avoids a new entity while segmenting the data.

By not introducing a separate Account entity, we avoid unnecessary complexity for now. However, we should remain open to it if a future need arises – for example, if we decide to implement a module for **billing or contracts**, having a separate Account entity that ties to financial systems could be beneficial. In that case, we might revisit Option 3 but design Account as more of a financial account record linked 1-1 with a Company.

In conclusion, our current company model is not so much *wrong* as it is in need of some **clarification and extension**: - We will **treat Company as the Account** equivalent (which aligns with HubSpot and simplifies user understanding for those familiar with HubSpot's approach). - We will add capabilities (parent-child relationships for hierarchy, owner fields as discussed in Part 1, etc.) to ensure it can fulfill the aggregation and organizational structure roles that "Account" serves in other CRMs. - This approach leverages the strengths of our existing model (simplicity and consistency) and avoids a disruptive schema split. It will improve clarity as we implement those changes, ensuring that we're not *overloading* the Company with conflicting purposes but rather empowering it to handle multiple related purposes with proper relationships.

Finally, by keeping a single unified entity for organizations, our system stays closer to HubSpot's data model, which eases understanding for users and ensures that our **multi-tenant SaaS CRM** remains straightforward: each tenant manages Companies (their client organizations) and Contacts, with flexible grouping via shadow groups and association links, rather than having to juggle an extra layer of abstraction. Should we encounter scenarios that strain this model (e.g., very complex account hierarchies),

we can address them with targeted solutions (like hierarchy support or custom linking entities) rather than a wholesale new object at this stage.

Actionable Recommendations Summary: Update the CRM schema to integrate shadow users/groups (Part 1 changes such as adding owner and created_by fields), fill structural gaps vis-à-vis HubSpot (e.g., add missing fields like close dates, probabilities, list types), and enhance the Company model with a parent linkage and ownership to serve as the account. These changes will improve data integrity, alignment with industry practices, and support advanced automation and analytics, all while leveraging our platform's strengths in workflow (Flowable) and AI. With these schema foundations in place, we can iteratively build out behavioral features (dynamic segments, automated triggers on stage changes, etc.) and provide a CRM that meets or exceeds the capabilities of a lightweight HubSpot, tailored for our multi-tenant architecture.

Sources:

- HubSpot CRM Pipelines & Stages – multi-pipeline support, stage probabilities 49 11
- HubSpot Deals API – required properties (dealstage, pipeline, owner), associations 29 50
- HubSpot Lists (Segments) API – list types (Manual vs Dynamic) 18
- HubSpot Activity (Engagements) Properties – activity owner, dates, details 31 38
- Salesforce Data Model (Accounts & Contacts) – definition of Account as business entity, hierarchy via parent account 44 46
- Flowable BPMN User Task docs – tasks can be assigned to users and groups (supporting our group assignment design) 3

1 5 6 7 Tenant Service Integration Guide.pdf

file:///file-PDAGFjDTmLjdcMADH7pWgD

2 4 17 25 26 27 28 29 30 50 CRM API | Deals - HubSpot docs

<https://developers.hubspot.com/docs/api-reference/crm-deals-v3/guide>

3 User Task | Flowable Enterprise Documentation

<https://documentation.flowable.com/latest/reactmodel/bpmn/reference/user-task>

8 9 10 11 12 13 14 15 16 49 CRM API | Pipelines - HubSpot docs

<https://developers.hubspot.com/docs/api-reference/crm-pipelines-v3/guide>

18 19 20 22 23 24 Lists (Segments) API - HubSpot docs

<https://developers.hubspot.com/docs/api-reference/crm-lists-v3/guide>

21 Re: How to get all company lists with the v3 lists API?

<https://community.hubspot.com/t5/APIs-Integrations/How-to-get-all-company-lists-with-the-v3-lists-API/m-p/1026842>

31 32 33 34 35 36 37 38 39 40 HubSpot's default activity properties

<https://knowledge.hubspot.com/properties/hubspots-default-activity-properties>

41 42 48 Mirroring Company Hierarchies Between Salesforce and HubSpot for ABM

<https://blog.insycle.com/mirroring-company-hierarchies-salesforce-hubspot>

43 **Best Practices for Company Sync with Salesforce Integration**

<https://community.hubspot.com/t5/Tips-Tricks-Best-Practices/Best-Practices-for-Company-Sync-with-Salesforce-Integration/m-p/1038981>

44 45 46 47 **Leads vs. Account and Contacts - The Salesforce Data Model**

<https://www.shellblack.com/whiteboard/overview-of-leads-account-and-contacts-the-salesforce-data-model/>