ULTRA — Session Summary (2026-02-09)

This file is a **downloadable recap** of what was completed in the ULTRA project during this chat session.

Authoritative references used:

- `ULTRA_MASTER_PLAN.md` (architecture contract)
- `docs/SCHEMA_DECISIONS.md` (schema law)
- `docs/BACKEND_DECISIONS.md` (backend audit decisions)
- `docs/RLS_DESIGN.md` (authoritative RLS design)
- `docs/TERMINOLOGY.md` (authoritative naming map)

---

1) RLS helper + read policies (Step 4A)

Helper

- Created `public.visible_group_ids()` helper function (security definer + explicit `search_path`).
- Updated it later to **union** membership from both tables during transition:
    - `public.group_users` (terminology target; if present)
    - `public.user_group_links` (legacy; present in current DB)

Read-side RLS enabled + SELECT policies added

RLS was enabled and SELECT policies added to enforce **group-scoped visibility** (user sees only membership groups + descendants):

Core scope chain:

- `public.groups`
- `public.user_group_links`
- `public.yacht_group_links`
- `public.yachts`
- `public.task_categories`
- `public.task_category_map`
- `public.task_category_links` (legacy mapping protected)
- `public.tasks`
- `public.task_contexts`
- `public.task_results`
- `public.users` (removed permissive global directory policy; now group-scoped + self)
- `public.task_context_assignees`
- `public.yacht_tasks`
- `public.yacht_task_results`
- `public.yacht_user_links`

Reference / extra surfaces (hardened):

- `public.periods` (SELECT-only)
- `public.units_of_measure` (SELECT-only)
- `public.user_role_links` (self-only SELECT)
- `public.roles` (read-only SELECT)
- `public.apps` (read-only SELECT)
- `public.app_group_links` (group-scoped SELECT)
- `public.app_user_links` (self-only SELECT)

Note:

- `public.task_context_overrides` **does not exist** in the live DB at the time of this session, so RLS could not be applied yet.

---

2) Data fixes discovered while enabling RLS

Orphan group ids on categories

Symptom:

- Tasks disappeared after enabling RLS because `task_categories.group_id` values were **placeholder UUIDs** that did not exist in `public.groups`.

Evidence:

- `task_categories.group_id` contained:
    - `00000000-0000-0000-0000-000000000001`
    - `11111111-1111-1111-1111-111111111111`
- `public.groups` contained real UUIDs (e.g. Worthy Marine, Dockers, etc.).

Fix:

- Re-pinned `task_categories.group_id` to a real group ID so RLS could scope categories/templates correctly.

---

3) “Global task library + private group tasks” (Model 1)

Decision:

- Use **Model 1** to keep RLS design intact: “global” is a **real group** that **every user is a member of**.

Actions:

- Inserted a new group: **Global Library**
    - `groups.id = e0fff626-23b1-42d5-a111-9c08407335f7`
- Added **all users** into Global Library via `public.user_group_links`.
- Moved global categories into Global Library by updating `public.task_categories.group_id` to the Global Library group id.

Outcome:

- Everyone in the tenant sees the global template library.
- Private tasks remain possible by leaving some categories/templates pinned to a private group.

---

4) “One group per yacht” enforcement (SCHEMA_DECISIONS)

Goal:

- Enforce: “Yachts belong to exactly ONE group.”

Actions:

- Detected duplicate ownership rows in `public.yacht_group_links` for two yachts.
- Deleted extra rows so each yacht had exactly one owning group row (deterministic keep-one behavior).
- Created unique index:
    - `create unique index ... on public.yacht_group_links (yacht_id)`

---

5) Backend rewrite: invite endpoint (BACKEND_DECISIONS)

`api/invite-user.ts`

Rewritten to meet `docs/BACKEND_DECISIONS.md`:

- Requires explicit `groupId` in request body.
- Requires `Authorization: Bearer <access_token>` (authentication).
- Authorization rules implemented:
    - Caller must have **role `admin`** (via `user_role_links → roles`).
    - Caller must be a **member of the target group** (supports `group_users` if present; otherwise `user_group_links`).
- On successful invite/sync:
    - Upserts `public.users` directory row.
    - Creates membership for the invited user (prefers `group_users`, falls back to `user_group_links`).

`src/pages/NewUserPage.tsx`

Updated to support the new endpoint requirements:

- Loads visible groups and requires selecting a group.
- Fetches Supabase session and sends Bearer token.
- Sends `groupId` with the invite request.

---

6) Admin role reference doc

Added:

- `docs/ADMIN_ROLE.md`

Purpose:

- Records where admin role is stored (`roles` + `user_role_links`).
- Provides SQL snippets to add/remove admins.
- Clarifies admin does **not** bypass RLS; it is used by server endpoints (e.g. invite).

---

7) Write-side rules (Step 4 write enforcement)

Implemented **append-only results** and group-scoped writes (one-table-at-a-time):

- `public.task_results`: INSERT allowed for visible contexts; no UPDATE/DELETE policies.
- `public.yacht_task_results`: INSERT allowed for visible yacht_tasks; no UPDATE/DELETE policies.

Group-scoped writes:

- `public.task_categories`: INSERT/UPDATE/DELETE limited to visible groups.
- `public.task_category_map`: INSERT/DELETE limited to categories in visible groups.
- `public.task_contexts`: INSERT/UPDATE/DELETE limited to visible yachts (via yacht→group).
- `public.task_context_assignees`: INSERT/DELETE limited to visible contexts.
- `public.yacht_tasks`: INSERT/UPDATE/DELETE limited to visible yachts.
- `public.task_category_links` (legacy): INSERT/DELETE limited to categories in visible groups.

Known limitation (intentional for now):

- `public.tasks` (templates) **INSERT is not safely enforceable** under strict group scoping without:
  - a transactional server operation (RPC/endpoint) that creates the task + mapping in one operation, or
  - a schema change that anchors templates directly to a group/category.
    This was explicitly left for a later, deliberate step.

---

8) Smoke test performed

Verified "execution history blocks unassign":

- Inserted `task_results` rows (status check constraint requires `pass`/`fail`).
- Confirmed UI blocks unassign with message:
  - "This task cannot be unassigned because there is execution history (task_results) for this yacht."

---

9) Remaining / next steps (not done tonight)

Immediate next steps:

- Apply RLS policies to `public.task_context_overrides` **once the table exists**.
- Decide and implement the **approved** mechanism for creating templates under RLS:
  - likely a server endpoint/RPC for "create task template + link to category" (not yet implemented per `docs/BACKEND_DECISIONS.md`).
- Harden/align `migration_sync_public_users_from_auth.sql` with the new visibility model (explicitly listed under REWRITE in `docs/BACKEND_DECISIONS.md`).

Notes / constraints followed

- Followed authoritative documents as "law" once introduced.
- RLS was implemented **one table at a time** per instruction.
- SQL was intended for **manual execution** by the user in Supabase.