ChatGPT

Complete NeonDB Schema for Marin Pest Control Dashboard

To support **Google Calendar integration**, **QuickBooks two-way sync**, **user authentication**, and an **employee time clock**, we will design a comprehensive Postgres schema (for NeonDB). The schema is organized into logical parts: **User Management**, **Google Calendar Data**, **QuickBooks Data**, and **Time Clock**. Each table includes standard timestamps (`created_at`, `updated_at`) for auditing. Below is a breakdown of the schema with tables and their key fields:

1. User Management (Users & Auth)

We use a `users` table to store employee login credentials and profile details, plus supporting tables for sessions and permissions. This allows Google OAuth login or traditional username/password.

users (Employee Accounts) [1] 【15†L23-L28**

- **id** – Primary key (serial).
- **first_name**, **last_name** – Employee's first and last name (varchar) [1].
- **address_line1**, **address_line2**, **city**, **state**, **zip_code**, **country** – Home address components (varchar) [2].
- **mobile_phone**, **home_phone** – Contact numbers (varchar).
- **email** – Unique email address (used for login/OAuth) [3].
- **username** – Unique username for login (if not using email) [1].
- **password_hash** – Hashed password for local login (bcrypt) [4].
- **google_id** – Google OAuth account ID (if logging in via Gmail) [5].
- **role** – User role for access control (e.g. `'admin'`, `'manager'`, `'user'`) [6]. Use `'admin'` for full access, `'manager'` for limited admin, `'user'` for regular employee.
- **is_active** – Boolean flag if the user is active (currently employed) [7]. Inactive could indicate on leave or terminated status.
- **employment_status** – Optionally, a status enum to distinguish `'active'` vs. `'on_leave'` (or other states) [8]. For example, mark an employee as on leave without deleting their account.
- **pay_rate** – Hourly pay rate (DECIMAL) [9].
- **hours_worked_this_week**, **hours_worked_last_week** – Total hours worked in the current and previous week (DECIMAL). These can be updated via time clock entries or calculated as needed.
- **admin_notes** – Internal notes about the employee (visible to admins only) [10].
- **employee_notes** – Notes by the employee (or general profile notes) [10].
- **last_login** – Timestamp of last login.
- **created_at** – Timestamp when the user was created (default `NOW()`).
- **updated_at** – Timestamp when the user record was last updated (auto-updated) [11].

user_sessions (Active Sessions) [12]

- **id** – Primary key (serial).
- **user_id** – References `users(id)`, the user who owns this session [12].
- **session_token** – Unique token for session (e.g. JWT or random string) [13].
- **expires_at** – Session expiration timestamp [13].
- **created_at** – Session creation timestamp (default `NOW()`).
- **last_accessed** – Timestamp of last usage (for session renewal/expiration) [13].

user_permissions (Granular Permissions) [14]

- **id** – Primary key (serial).
- **user_id** – References `users(id)` (cascade on delete) [15].
- **permission** – Permission name (e.g. `"manage_users"`, `"view_reports"`) [16].
- **granted_at** – Timestamp permission was granted.
- **granted_by** – References `users(id)` of the admin who granted the permission [17].
- *Unique index on* `(user_id, permission)` *to prevent duplicates* [18].

Note: The `users` table covers all basic personal and account info. Employees can log in via Google (using `google_id` and matching email) or with `username/password`. Roles and permissions support admin controls (e.g. only admins can manage users) [19] [20]. Session tokens (with JWT) enable authentication persistence [21].

2. Google Calendar Integration (Scheduling)

We create a separate `google` **schema** to organize Google Calendar data. This includes tables for events, employee availability, and event assignments. These tables enable viewing all work calendars side-by-side and assigning events to employees.

google.calendar_events (Company Calendar Events) [22] [23]

- **id** – Primary key (serial).
- **google_event_id** – Google Calendar Event ID (varchar, unique), used to avoid duplicates and sync with Google [24].
- **google_calendar_id** – ID of the Google Calendar the event belongs to (varchar) – for example, a specific service calendar or technician's calendar [25].
- **title** – Event title or summary (varchar) [26].
- **description** – Event description/details (text).
- **location** – Event location (text), e.g. customer address or job site [26].
- **start_time**, **end_time** – Event start and end datetime (timestamp) [27].
- **all_day** – Boolean, true if event is an all-day event (no specific time) [28].
- **assigned** – *Deprecated in favor of* `work_assignments` *table.* (If each event could only have one technician, we could alternatively use a field like `assigned_to_employee_id` here. In our design, we use a separate assignments table to allow flexibility).
- **created_at** – Timestamp when the event was first pulled/created in our DB.
- **updated_at** – Timestamp when last updated in our DB.
- **last_synced** – Timestamp of last sync with Google (to track freshness).

(Indexes: index on `google_event_id` to quickly find events by external ID [29]; index on `start_time` to query events by date/time range.)*

google.work_assignments (Event ↔ Employee Assignments) [30] [31]

This table links calendar events to the employees assigned to work on them. It supports drag-and-drop assigning of technicians to events and tracking job status.

- **id** – Primary key (serial).
- **calendar_event_id** – References `google.calendar_events(id)` – the event/job to be done [32].
- **employee_id** – References `users.id` (or `employees.id`) of the technician assigned [33].
- **assigned_by** – References `users.id` of the admin/manager who assigned the task.
- **assigned_at** – Timestamp when the assignment was made (default `NOW()`) [34].
- **sequence_order** – Integer ordering for multiple jobs in a day (e.g., if an employee has 3 assignments, they can be ordered 1, 2, 3) [35].
- **status** – Status of the assignment (enum: e.g. `'pending'`, `'assigned'`, `'in_progress'`, `'completed'`, `'cancelled'`) [36]. Default `'assigned'` when first created.
- **started_at**, **completed_at** – Timestamps when the technician started and finished this job (for tracking progress) [37].
- **admin_notes** – Notes from the admin about this assignment (text), e.g. special instructions [38].
- **employee_notes** – Notes from the employee after completion (text), e.g. work performed or issues [38].
- **created_at** – Timestamp when the assignment record was created.
- **updated_at** – Timestamp when last updated (e.g. status change or notes added).

(Indexes: index on `calendar_event_id` (to quickly find assignments for an event) [39], and on `employee_id` (to find all assignments for a given employee) [39].)*

google.employee_availability (Availability/Time Off) [40]

This table optionally tracks employee availability or time-off entries. It can store when an employee is not available (e.g., on leave or PTO), which helps in scheduling and seeing free/busy times.

- **id** – Primary key (serial).
- **employee_id** – References `users.id` of the employee [41].
- **date** – Date of availability setting (timestamp, typically date portion used) [42].
- **start_time**, **end_time** – Time window on that date (if partial day). For full-day off, could be whole day or marked via a flag.
- **available** – Boolean indicating if the employee *is available* during that period (true = available, false = unavailable) [43]. For example, an entry could mark someone unavailable for a certain date or time range.
- **reason** – Text reason for unavailability (e.g. "Vacation", "Sick", "Training") [43].
- **created_at**, **updated_at** – Timestamps for record creation and updates.

Usage: The `calendar_events` table will store all events from Google Calendars (e.g., each service calendar) [44]. Managers can assign these events to employees via `work_assignments`. Employees will see their assigned jobs (e.g., in a “My Work Today”

view) and can update notes or status [45]. The `employee_availability` table, combined with calendar events, helps determine open time slots or track leaves, so schedulers can avoid assigning jobs when someone is off.

(We also have specialized tables for Google integration features, if needed: e.g., a `google.event_notes` table to store multiple threaded notes per event, or `google.task_checklists` for job checklists [46] [47]. These are optional enhancements to sync notes/checklists into the Google Calendar event descriptions.)

3. QuickBooks Integration (Accounting Data)

All QuickBooks data is organized under a separate `quickbooks` **schema** for clarity [48]. We mirror key QuickBooks Online objects for **Customers**, **Items**, **Estimates**, and **Invoices**, including line items for Estimates and Invoices. We also have tables for OAuth tokens and company info. These tables will be kept in sync with QuickBooks via webhooks and periodic sync jobs [49] [50].

quickbooks.companies (Connected QBO Company Files) [51]

- **id** – Primary key (varchar(50)). This is the QuickBooks **Company ID (Realm ID)** for the connected company file [52].
- **name** – Company name (e.g., "Marin Pest Control QBO") [52].
- **created_at**, **updated_at**, **last_updated** – Timestamps for creation, last update, and last sync update [53].

(This table allows for multiple QuickBooks companies if needed. Typically, there will be one entry for our company's QuickBooks Online realm.)

quickbooks.tokens (OAuth2 Tokens) [54] [55]

- **id** – Primary key (bigint, can auto-increment).
- **company_id** – References `quickbooks.companies(id)` – which company this token is for [56].
- **access_token**, **refresh_token** – OAuth 2.0 tokens for QuickBooks API access (text) [57].
- **token_type** – Token type (usually `"Bearer"`) [58].
- **scope** – OAuth scope granted (text) [59].
- **expires_at** – When the access token expires (timestamp) [60].
- **refresh_token_expires_at** – When the refresh token expires (timestamp) [61].
- **realm_id** – The Intuit realm ID (company ID) associated with the token (varchar) [62]. *Typically this matches `company_id`, stored separately for convenience.*
- **base_url** – API base URL for this realm (e.g., sandbox vs production endpoint) [63].
- **is_active** – Boolean, marks the currently active token (in case multiple tokens are stored over time) [64].
- **created_at**, **updated_at**, **last_updated** – Timestamps for record creation, last update (e.g. when token was refreshed), and last sync time [65].
  (Indexes on realm_id, company_id, is_active, expires_at help quickly find the valid token) [66].

  Usage: This table stores the latest QuickBooks OAuth tokens. The system will update/insert here whenever tokens are refreshed (using UPSERT on `realm_id`) [67] [68]. A scheduled job runs every 50 minutes to refresh the token before expiration [69].

quickbooks.customers (Customer List) [70] [71]

- **id** – Primary key (varchar(50)). The QuickBooks Customer ID.
- **name** – Full name or display name of the customer (varchar) [72].
- **company_name** – Company name (if the customer is a business) [73].
- **display_name** – Display name (as shown in QBO) [74].
- **primary_email_addr** – Email address of the customer (varchar) [75].
- **primary_phone**, **alternate_phone**, **mobile**, **fax** – Various contact phone numbers (varchar) [76].
- **website** – Website URL of the customer (if any).
- **bill_addr_ / *ship_addr_*** – **Billing** and **Shipping address** fields (line1, city, state, postal_code, country). We store the customer's address details to use in invoices or scheduling. *(These fields correspond to the customer's address info in QuickBooks.)*
- **taxable** – Boolean, whether the customer is taxable (for sales tax) [77].
- **balance** – Current open balance of the customer in QuickBooks (double precision) [78].
- **notes** – Customer notes (text) [78].
- **active** – Boolean, whether the customer is active in QBO (not deleted/inactive) [76].
- **company_id** – QuickBooks Company/Realm this customer belongs to (varchar, references `companies.id`).
- **created_at**, **updated_at**, **last_updated** – Timestamps for creation, last update, and last sync from QBO [79].

(Indexes: e.g., index on `name` and `primary_email_addr` for quick lookup [80].)*

quickbooks.items (Products/Services Items) [81] [82]

- **id** – Primary key (varchar(50)). The QuickBooks Item (Product/Service) ID.
- **name** – Name of the item (varchar) [83].
- **sku** – SKU or short code (varchar) [84].
- **description** – Description of the item (text) [85].
- **active** – Boolean, whether the item is active (available for use) [86].
- **unit_price** – Standard unit price or rate (double precision) [87].
- **sales_price** – Sales price if distinct (double).
- **qty_on_hand** – Quantity on hand (for inventory items, if applicable) [88] [89].
- **type** – Item type (e.g. `Inventory`, `Service`, `Non-inventory`) [88] [90].
- **taxable** – Boolean, whether the item is taxable [91].
- **income_account_ref**, **expense_account_ref**, **asset_account_ref** – References (IDs or names) to the associated income/expense/asset accounts in QuickBooks (for accounting purposes) [92].
- **company_id** – QuickBooks Company/Realm this item belongs to (varchar, references `companies.id`).
- **created_at**, **updated_at**, **last_updated** – Timestamps for creation, last update, and last sync.

(The `items` table ensures we have all data needed to recreate invoice line items: price, taxability, etc. Account reference fields allow mapping items to accounts if needed for two-way sync.)

quickbooks.invoices (Customer Invoices)

- **id** – Primary key (BIGINT). The QuickBooks Invoice ID.
- **doc_number** – Invoice number (as shown in QuickBooks) [93].

- **txn_date** – Invoice date (date of transaction) [93].
- **due_date** – Due date for payment [94].
- **total_amt** – Total amount of the invoice (double) [95].
- **balance** – Remaining balance due (if partial payments made) (double) [96].
- **customer_ref_id**, **customer_ref_name** – Reference to the customer (ID and name) for convenience. *Also,* `customer_id` *can be stored (same as customer_ref_id) to join with* `quickbooks.customers` [97].
- **email_status**, **print_status** – Status of delivery: e.g. whether invoice was emailed or printed (values like `NotSent`, `EmailSent`, etc.).
- **billing_address** and **shipping_address** – The billing and shipping address used on the invoice (line1, city, state, postal_code, country). These are copied from the customer at time of invoice creation (snapshot for record).
- **status** – Invoice status (e.g. `Open`, `Paid`, `Overdue` – derived from balance and due date; QuickBooks may not explicitly provide a status field, but we can infer or store if needed).
- **sync_token** – QuickBooks sync token for concurrency control [98].
- **metadata_create_time**, **metadata_last_updated_time** – Timestamps from QuickBooks indicating when the invoice was created and last updated in QBO [99].
- **last_updated** – Timestamp of last sync/update in our DB (for tracking stale data).
- **company_id** – QuickBooks Company/Realm ID (to scope the invoice to a company).
- **created_at**, **updated_at** – Timestamps for record creation and last update in our DB.

(We use a BIGINT for invoice IDs because QuickBooks invoice IDs are numeric. They often can be large numeric strings, so ensure the type can hold the values. Alternatively, use varchar if QuickBooks IDs may include non-numeric characters.)

quickbooks.invoices_line_items (Line Items on Invoices) [100] [101]

Each invoice can have multiple line items (products/services billed on that invoice). We store them in a separate table linked to the invoice.

- **id** – Primary key (serial). (Alternatively, could use QuickBooks Line Id if provided, but QBO line items might not have a stable ID, so we use our own ID.) [102]
- **invoice_id** – References `quickbooks.invoices(id)` (on delete cascade, so deleting an invoice removes its lines) [103].
- **line_num** – Line position number on the invoice (integer) [104].
- **detail_type** – QuickBooks detail type of the line (e.g. `SalesItemLineDetail`, `SubTotalLineDetail`, etc.) [105].
- **item_ref_id**, **item_ref_name** – The item/service being charged on this line (references QuickBooks Item: ID and name) [106]. Storing both ID and name simplifies reporting without join, and preserves the name even if item is later renamed.
- **description** – Line item description (text) [107]. May override the default item description.
- **unit_price** – Unit price for this line (double) [108].
- **qty** – Quantity for this line (double) [109].
- **amount** – Line total amount = unit_price * qty (double) [109].
- **tax_code_ref_id**, **tax_code_ref_name** – Tax code applied to this line, if any (ID and name) [110].
- **class_ref_id**, **class_ref_name** – Class tracking info for this line, if used (ID and name) [111].
- **last_updated** – Timestamp when this line was last updated/synced [112].

(The line items table ensures that invoice details (which items, how many, price, etc.) are captured for two-way syncing. If an invoice is updated (e.g., a line edited), the webhook or sync process will update the corresponding line record.)

quickbooks.estimates (Customer Estimates/Quotes) [113] [99]

- **id** – Primary key (BIGINT). The QuickBooks Estimate ID.
- **doc_number** – Estimate number (if used) [114].
- **txn_date** – Estimate creation date [115].
- **expiration_date** – Estimate expiry date (if set) [116].
- **total_amt** – Total amount of the estimate (double) [117].
- **status** – Status of the estimate (e.g., `Pending`, `Accepted`, `Closed`). *(QuickBooks provides an* `Estimate.Status` *field to indicate if the estimate has been accepted or closed.)*
- **customer_ref_id**, **customer_ref_name** – Reference to the customer for this estimate (ID and name).
- **email_status**, **print_status** – Whether the estimate was emailed/printed (text statuses) [118].
- **billing_address** and **shipping_address** – Billing/Shipping address used on the estimate (same breakdown of line1, city, state, etc.).
- **sync_token** – QuickBooks sync token for concurrency [98].
- **sparse** – Boolean flag from QuickBooks indicating if the object is sparse (partially fetched/updated) [119].
- **metadata_create_time**, **metadata_last_updated_time** – QuickBooks timestamps for creation and last update of this estimate [99].
- **last_updated** – Timestamp of last sync to our DB [120].
- **company_id** – QuickBooks Company/Realm ID this estimate belongs to.
- **created_at**, **updated_at** – Timestamps for creation and last update in our DB.

quickbooks.estimates_line_items (Line Items on Estimates) [121] [122]

Similar to invoice line items, each estimate's line items are stored here.

- **id** – Primary key (serial).
- **estimate_id** – References `quickbooks.estimates(id)` (cascade on delete) [123].
- **line_num** – Line position number (integer) [124].
- **detail_type** – Detail type of the line (text, e.g. `SalesItemLineDetail`) [125].
- **item_ref_id**, **item_ref_name** – Item/service on this line (ID and name) [126].
- **description** – Description of the line item (text) [127].
- **unit_price** – Unit price (double) [128].
- **qty** – Quantity (double) [129].
- **amount** – Line total amount (double) [130].
- **tax_code_ref_id**, **tax_code_ref_name** – Tax code for this line (if any) [131].
- **class_ref_id**, **class_ref_name** – Class for this line (if any) [132].
- **last_updated** – Timestamp of last sync/update of this line [133].

Sync Strategy: When QuickBooks sends webhooks for created/updated objects, the backend will **upsert** records into these tables. For example, on an **Invoice** webhook, we update or insert into `quickbooks.invoices`, and similarly handle each line in `invoices_line_items` [134] [135]. We also run a periodic full sync (e.g., hourly) to reconcile

any missed changes [136] [137]. The schema above stores all key fields required to reconstruct invoices, estimates, and customers on both sides, ensuring a robust two-way integration.

(Other QuickBooks objects like Payments or Vendors are not included since the focus is on Customers, Items, Estimates, and Invoices as requested.)

4. Time Clock (Employee Punch In/Out)

For tracking work hours, we introduce a **time clock** table where employees clock in/out, including lunch breaks and administrative approvals. This table ties in with the user (employee) records and will be used for payroll calculations.

time_clock_entries (Employee Time Entries)

- **id** – Primary key (serial).
- **user_id** – References `users(id)` of the employee who is clocking time.
- **clock_in** – DateTime when the employee clocked in.
- **clock_out** – DateTime when the employee clocked out (NULL if not yet clocked out).
- **lunch_start** – DateTime when lunch break started (if taken).
- **lunch_end** – DateTime when lunch break ended.
- **total_hours** – Calculated total hours for this entry (optional, can be computed as `(clock_out - clock_in) - (lunch_end - lunch_start)`).
- **suspicious** – Boolean flag for irregular entries. Marked `true` if the entry is flagged for review (e.g., missing clock-out, long hours, or manual edits) that might indicate an error or misconduct.
- **employee_note** – A note left by the employee about this entry (text). For example, "Forgot to clock out, adjusted by manager" or "Left early for appointment."
- **last_edited_at** – Timestamp of the last manual edit to this entry (if an admin adjusted the times).
- **last_edited_by** – References `users(id)` who last edited the entry (typically a manager/admin).
- **approved_by_admin** – Boolean flag indicating an admin/supervisor has approved this entry for accuracy. (This might be set after reviewing a suspicious entry or at the end of the pay period.)
- **approved_by_payroll** – Boolean flag indicating the entry is approved for payroll processing. This could be a final approval by accounting/payroll staff.
- **created_at** – Timestamp when the entry was created (usually when clock-in occurred, default `NOW()`).
- **updated_at** – Timestamp when the entry was last updated (automatically updated on any change).

(Indexes: you may index on `user_id` and date (from `clock_in`) to quickly retrieve all entries for a user in a date range, and on `approved_by_payroll` to find unprocessed entries, etc.)*

Workflow: Employees will create an entry when they **clock in**, which records the `clock_in` time. That same entry is updated when they **clock out** (setting `clock_out`), and if they take lunch, `lunch_start`/`lunch_end` are recorded. If any entry is flagged as **"suspicious"** (for example, extremely long shift or missing clock-out), admins can review and edit the times; the system notes who edited (`last_edited_by`) and marks the entry for approval. Managers will then mark `approved_by_admin` once an entry is verified [138], and payroll can do a final approval before processing hours. The `hours_worked_this_week` and `...last_week` in

the `users` table can be updated based on these entries (for quick dashboard display), or computed via queries by summing the time intervals per week.

By organizing the database into these sections and tables, we ensure a robust structure:

- The **User Management tables** cover all employee info and login credentials in one place, with role-based access control and session tracking [139] [140].
- The **Google Calendar tables** allow storing all calendar events (with details like time, location) and linking them to employees for assignment [22] [141], as well as tracking availability to avoid scheduling conflicts.
- The **QuickBooks tables** capture all necessary fields for customers, items, estimates, and invoices (including line items) to facilitate a two-way sync with the accounting system [142] [135].
- The **Time Clock table** records precise work hours and statuses for each employee, feeding into both scheduling and payroll needs.

This NeonDB schema provides a single source of truth for the dashboard, ensuring each part of the system (scheduling, finance, user auth, and time tracking) has the data it needs, and all of it can be cross-referenced (e.g., linking calendar events to customers and employees, linking time entries to users and ultimately to payroll reports). It also leaves room for future growth, such as adding more QuickBooks entities or more Google Calendar features, without a major redesign. Each table can be extended or indexed as usage patterns demand, but this foundation covers the requirements **across the board** as requested. [49] [143]

---

[1] [3] [4] [7] [11] [12] [13] [14] [15] [19] [20] [21] [139] [140] USER-MANAGEMENT.md
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/docs/USER-MANAGEMENT.md

[2] [5] [8] [9] [10] employee_schema.sql
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/employee_schema.sql

[6] [16] [17] [18] 007_create_users_system.sql
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/backend/db/migrations/007_create_users_system.sql

[22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [141] calendar-schema.ts
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/backend/src/db/calendar-schema.ts

[44] [45] [46] [47] [138] [143] GOOGLE-CALENDAR-IMPLEMENTATION.md
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/docs/GOOGLE-CALENDAR-IMPLEMENTATION.md

[48] [52] [53] [67] [68] [69] quickbooks_drizzle_fixed.md
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/docs/quickbooks_drizzle_fixed.md

[49] [50] [51] [54] [55] [56] [57] [60] [62] [65] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [83] [84] [85] [86] [87] [88] [93] [94] [95] [96] [97] [134] [135] [136] [137] [142] quickbooks_backend_full.md
https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/docs/quickbooks_backend_full.md

58 59 61 63 64 66 003_add_missing_oauth_columns.sql

https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/backend/db/migrations/003_add_missing_oauth_columns.sql

82 89 90 91 92 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 005_add_missing_quickbooks_tables.sql

https://github.com/Sonni4154/dash3/blob/674febcbc95e3c27b1ef7fff660ef4123adb3701/backend/db/migrations/005_add_missing_quickbooks_tables.sql