

Developer Documentation

Billing & Service Contract Management System

Audience: a developer maintaining or extending this codebase. This document explains **how the system works, which tables each process touches, and where to make changes safely.**

1. Technology & Conventions

- **Language:** plain procedural PHP 8 (no framework).
- **Database:** MySQL/MariaDB, accessed exclusively through PDO with prepared statements
(config/db.php → db()), PDO::ATTR_ERRMODE = EXCEPTION , PDO::ATTR_EMULATE_PREPARES = false .
- **Frontend:** Bootstrap 5 + jQuery + Select2, all loaded from CDN; no build step, no bundler.
- **Auth:** native PHP sessions (includes/auth.php), bcrypt password hashes.
- **Every page** is a standalone .php file that require_once s the same four files in the same order: php
require_once __DIR__ . '/../config/constants.php'; require_once __DIR__ .
'/../config/db.php'; require_once __DIR__ . '/../includes/functions.php'; require_once
__DIR__ . '/../includes/auth.php'; requireLogin(); // or requireRole([...]) Keep this order
when adding new pages — constants.php defines BASE_URL which auth.php 's redirects depend
on.

2. Folder Structure

config/	db.php (PDO connection), constants.php (BASE_URL, upload paths)
includes/	auth.php, functions.php, header.php, sidebar.php, footer.php
sql/	schema.sql – full schema, re-runnable on a fresh database
auth/	login.php, logout.php
concern/	index, create, edit, view, ajax_search
contract/	index, create, view, edit (AMC/MMC setup)
bill/	index, create, view, split, generate_recurring
payment/	create
adjustment/	create (bonus/waiver)
report/	index + 6 report pages
assets/	css/, js/, uploads/ (concern attachments, bill attachments)
Documentations/	this folder

There is no MVC layer, router, or ORM — each file handles its own GET/POST, talks to the database directly, and renders its own HTML inline. This was a deliberate choice: the project started from an empty folder with no pre-existing framework to match, and a flat per-page structure keeps each process traceable in one file.

3. Database Schema & Relationships

All tables are defined in `sql/schema.sql` . Entity relationship summary:

```

users ————— (created_by / approved_by / etc. on most tables)
      |
concern —┬─ concern_billing_info (1:1, latest row wins)
          ├── concern_attachment (1:many, history retained)
          ├── billing_contract (1:many)
          └─ bill_master (1:many, direct bills)

billing_contract —┬─ bill_master (1:many, one row per billing_period)
                  └─ contract_date_history (1:many)

bill_master —┬─ bill_master (self-FK: master_bill_id – one level of split)
              ├── bill_payment (1:many)
              └─ bill_adjustment (1:many)

payment_account — bill_payment (1:many)

doc_sequence    (no FK – pure counter table, keyed by doc_type + year)

```

Table reference

Table	Key columns	Purpose
users	role ENUM(admin, billing_manager, billing_user, accounts_user)	Login + RBAC
doc_sequence	PK (doc_type , year), last_number	Source of truth for Bill/Receipt/Contract numbering
concern	concern_name , status	Customer master
concern_billing_info	concern_id FK, address/nature/contact fields	Extended concern info (one current row per concern)
concern_attachment	concern_id FK, file_path , original_name	Uploaded files, never overwritten
billing_contract	concern_id FK, bill_type (AMC/MMC/SERVICE), effective_from/to , frequency , billing_day , billing_amount , status	Recurring billing configuration
contract_date_history	contract_id FK, old_effective_to , new_effective_to	Audit trail for date extensions
bill_master	bill_no (unique), concern_id FK, bill_type , contract_id FK (nullable), master_bill_id FK (nullable, self), effective_date , billing_period , bill_amount , status	Every bill of every type — the central table
payment_account	name , type (CASH/BANK)	Minimal placeholder for real chart-of-accounts
bill_payment	receipt_no (unique), bill_id FK, amount , status (ACTIVE/REVERSED)	Every payment
bill_adjustment	bill_id FK, adjustment_type , amount , reason , status (APPLIED/REJECTED)	Every bonus/discount/waiver

Important uniqueness constraints: - bill_master.bill_no — UNIQUE - bill_payment.receipt_no — UNIQUE - bill_master (contract_id, billing_period) — UNIQUE. This is the hard backstop against duplicate recurring bills (MySQL treats NULL as distinct from NULL, so this does not restrict non-contract bills, which always have contract_id IS NULL).

4. Core Logic (all in includes/functions.php)

This file is the single source of truth for every derived number in the system. **If you need to change how outstanding or status is calculated, change it here — nowhere else recomputes it independently.**

```
getBillFinancials($pdo, $bill)
```

Reads: `bill_master` (children sum), `bill_payment` (active sum), `bill_adjustment` (applied sum). Returns `amount`, `childTotal`, `paidTotal`, `adjTotal`, `base` (= `amount - childTotal`), `outstanding` (= `base - paidTotal - adjTotal`).

This formula is why splitting never double-counts: a fully-split master's `base` is 0, so it stops contributing to outstanding — only its children (which have no children of their own, so their own `base` = their own `amount`) continue to carry the receivable.

`computeBillStatus($bill, $fin) / recalcAndSaveBillStatus($pdo, $billId)`

Pure function → stored status (see SRS §7 for the exact decision table). `recalcAndSaveBillStatus` writes the result back to `bill_master.status` and, if this bill is a child, also recomputes its master. Today that upward recompute is a no-op in the payment/adjustment case (the master's own `base / paidTotal / adjTotal` don't depend on a child's payments) — it only actually changes anything when a child bill is created (handled explicitly by `bill/split.php` calling it on the parent) or if a child is ever cancelled in the future (`childTotal` excludes cancelled children, so cancelling one would change the master's `base`). The recursion is kept so that future cancellation support (see §9) works correctly without having to remember to add it then. Called after every mutation: bill creation, split, payment, adjustment.

`generateDocNo($pdo, $docType, $prefix)`

Reads/writes `doc_sequence` using `SELECT ... FOR UPDATE` to serialize concurrent allocations. **Must be called from inside a transaction already opened by the caller** — it does not open its own (an earlier version did, which broke with "There is already an active transaction" the moment it was called from code that had already started one; see the history in the chat for that bug and fix).

`generateChildBillNo($pdo, $masterBillId, $masterBillNo)`

Counts existing children of `masterBillId` in `bill_master` and returns `{masterBillNo}-{next 2-digit sequence}`.

`contractDuePeriods($contract, DateTime $today)`

Pure function (no DB access) — walks month-by-month (MONTHLY) or year-by-year (YEARLY) from `effective_from` to `effective_to`, returning every period whose billing date is `<= $today`. Passing a far-future `$today` (as `contract/view.php` and the Recurring Billing report do) effectively returns *all* periods through `effective_to`, which is how "Next Due Period" can show a period that hasn't technically arrived yet.

`recalcContractStatus($pdo, $contractId)`

Flips `billing_contract.status` from `Active` to `Expired` once `effective_to` is in the past. Called on contract view and at the top of every generation run.

`generateRecurringBills($pdo, $actingUserId)`

The recurring billing engine — see §6 below for the full process.

5. Process Walkthroughs (with tables touched)

5.1 Create a License/Service bill — `bill/create.php`

Step	Tables touched
1. Validate <code>concern_id</code> , <code>effective_date</code> , amount	<code>concern</code> (read, to show selected name)
2. Open transaction	—
3. <code>generateDocNo('BILL','BILL')</code>	<code>doc_sequence</code> (read+write)
4. Save uploaded attachment to disk	filesystem only
5. <code>INSERT INTO bill_master</code> (<code>bill_type</code> LICENSE/SERVICE, <code>contract_id</code> NULL, <code>master_bill_id</code> NULL)	<code>bill_master</code> (write)
6. Commit, redirect to <code>bill/view.php</code>	—

5.2 Create an AMC/MMC contract — `contract/create.php`

Step	Tables touched
1. Validate dates, <code>billing_day</code> (MMC only), amount	—
2. Open transaction	—
3. <code>generateDocNo('CON','CON')</code>	<code>doc_sequence</code>
4. <code>INSERT INTO billing_contract</code> (status defaults to Active)	<code>billing_contract</code>
5. Commit, redirect to <code>contract/view.php</code>	—

No bills are created here — generation is a separate, explicit step (§5.3/§6), by design, so contract setup never silently creates financial records.

5.3 Recurring generation — `bill/generate_recurring.php` → `generateRecurringBills()`

Step	Tables touched
1. Load all <code>status = 'Active'</code> contracts	<code>billing_contract</code> (read)
2. For each: compute due periods (<code>contractDuePeriods</code> , pure function)	—
3. Load already-billed periods for this contract	<code>bill_master</code> (read, <code>WHERE contract_id = ?</code>)
4. For each due period not already billed: open transaction, <code>generateDocNo</code> , <code>INSERT INTO bill_master</code> (<code>contract_id</code> set, <code>billing_period</code> set)	<code>doc_sequence</code> , <code>bill_master</code>
5. <code>recalcContractStatus</code> per contract	<code>billing_contract</code> (write, only if expiring)

Dual entry point: web (`admin / billing_manager` , via `requireRole`) or CLI (`php bill/generate_recurring.php` , attributes bills to the first `admin` user found — see §7 for scheduling).

5.4 Split a bill — `bill/split.php`

Step	Tables touched
1. Open transaction, <code>SELECT ... FOR UPDATE</code> on the target bill (locks it against a concurrent split)	<code>bill_master</code> (read, locked)
2. Reject if it's already a child (<code>master_bill_id IS NOT NULL</code>), cancelled, or amount exceeds <code>getBillFinancials()</code> ['outstanding']	<code>bill_master</code> , <code>bill_payment</code> , <code>bill_adjustment</code> (all read, via <code>getBillFinancials</code>)
3. <code>generateChildBillNo</code>	<code>bill_master</code> (read, count children)
4. <code>INSERT INTO bill_master</code> (master_bill_id = parent's id, same concern/bill_type/effective_date)	<code>bill_master</code> (write)
5. <code>recalcAndSaveBillStatus</code> on the parent (may flip to "Fully Split")	<code>bill_master</code> (write)
6. Commit	—

5.5 Record a payment — `payment/create.php`

Step	Tables touched
1. Open transaction, <code>SELECT ... FOR UPDATE</code> on the bill	<code>bill_master</code> (read, locked)
2. Reject if cancelled, or amount exceeds <code>getBillFinancials()</code> ['outstanding']	<code>bill_master</code> , <code>bill_payment</code> , <code>bill_adjustment</code>
3. <code>generateDocNo('RCPT', 'RCPT')</code>	<code>doc_sequence</code>
4. <code>INSERT INTO bill_payment</code> (status ACTIVE)	<code>bill_payment</code>
5. <code>recalcAndSaveBillStatus</code> on the bill (and its master, if it's a child)	<code>bill_master</code>
6. Commit	—

5.6 Apply a bonus/adjustment — `adjustment/create.php`

Gated by `requireRole([ 'admin', 'billing_manager' ])` at the top of the file — this is checked **before** anything else runs, so a disallowed role gets HTTP 403 even if they call the URL directly with a crafted POST body.

Step	Tables touched
1. Open transaction, <code>SELECT ... FOR UPDATE</code> on the bill	<code>bill_master</code>
2. Reject if cancelled, reason empty, or amount exceeds outstanding	<code>bill_master</code> , <code>bill_payment</code> , <code>bill_adjustment</code>
3. <code>INSERT INTO bill_adjustment</code> (status APPLIED, approved_by = current user)	<code>bill_adjustment</code>
4. <code>recalcAndSaveBillStatus</code> — flips to "Bonus Closed" if this zeroes outstanding with <code>paidTotal</code> still 0, otherwise "Paid"/"Partially Paid" as usual	<code>bill_master</code>
5. Commit	—

5.7 Extend / edit a contract — `contract/edit.php`

Step	Tables touched
1. If <code>effective_to</code> is changing, <code>INSERT INTO contract_date_history</code> first (old + new value)	<code>contract_date_history</code>
2. <code>UPDATE billing_contract</code> (<code>effective_to</code> , <code>billing_day</code> , status, remarks)	<code>billing_contract</code>

Note: changing `billing_day` only affects **future** calls to `contractDuePeriods()` — rows already in `bill_master` are never touched, which is what makes "already-generated bills are unaffected" true.

6. The Recurring Billing Algorithm, In Detail

`contractDuePeriods()` is the heart of recurring billing. For **MMC** (monthly):

1. Start at the first day of `effective_from`'s month.
2. For each month up to `effective_to`: clamp the configured `billing_day` to that month's actual last day (so day 31 in February becomes the 28th/29th), forming the period's billing date.
3. Include the period if its billing date falls within `[effective_from, effective_to]` **and** is `<= $today`.
4. Advance one calendar month and repeat.

For **AMC** (yearly): same idea, but the "period" is the contract's own anniversary date, advancing one year at a time, with the period label being just the year.

`generateRecurringBills()` then, for each Active contract, diffs this due list against `billing_period` values already present in `bill_master` for that contract, and inserts exactly the missing ones. Three independent safety nets make this idempotent:

1. **Application-level diff** against existing `billing_period`s (cheap, handles the normal case).
2. **The unique index** on (`contract_id`, `billing_period`) — if two processes somehow raced past the application check, the second `INSERT` throws a duplicate-key exception, which is caught and silently skipped (anything *other* than that specific constraint name re-throws, so real errors are never swallowed).

3. **Per-insert transaction** — each bill is inserted in its own transaction, so one failure doesn't roll back bills already generated earlier in the same run.

7. Deployment Notes

- **Database:** run `sql/schema.sql` once against a MySQL/MariaDB server (`mysql -u root < sql/schema.sql`). It creates the database, all tables, seed `payment_account` rows (Cash/Bank), and a default `admin / admin123` user — **change this password in production**.
 - **Web server:** point an Apache (or any PHP 8) vhost's document root at the folder *containing* `Bill_management` , so the app is reachable at `/Bill_management/...` — this matches the hardcoded `BASE_URL` constant in `config/constants.php` . If you deploy at a different path, update that constant.
 - **DB credentials:** `config/db.php` — defaults to `root /no password` on `127.0.0.1` , matching XAMPP defaults. Update for your environment.
 - **Recurring billing automation:** schedule `php.exe <path>\bill\generate_recurring.php` via Windows Task Scheduler (daily is sufficient for both MMC and AMC cadences). The script detects CLI mode (`PHP_SAPI === 'cli'`) automatically and attributes generated bills to the first `admin` user it finds — do not rename/delete that user without updating this fallback.
 - **File uploads:** stored under `assets/uploads/concern/{concern_id}/` and `assets/uploads/bill/` . Ensure the web server user has write access to `assets/uploads/` .
-

8. Extending the System

- **Add a new bill type:** extend the `bill_type` ENUM in `bill_master` (and `billing_contract` if it recurs), add it to the dropdown(s) in `bill/create.php` or `contract/create.php` , and add any type-specific validation. `getBillFinancials` / `computeBillStatus` are already type-agnostic — no changes needed there.
 - **Add a new report:** copy the pattern in any `report/*.php` file — a filter form (`GET`) building a parameterized query, then a table. Register it in `report/index.php` 's `$reports` array.
 - **Add a new role or permission:** extend the `role` ENUM in `users` , and add/adjust `requireRole([...])` calls in `includes/auth.php` consumers. `canApplyBonus()` in `auth.php` is the one convenience helper for the most common check (`admin / billing_manager`) — add similar helpers rather than repeating role arrays everywhere if a new check becomes common.
 - **Real accounting/GL integration:** the current "ledger" (see `report/concern_statement.php`) is computed on the fly from `bill_master` / `bill_payment` / `bill_adjustment` — it does not post anywhere. If a real chart of accounts is introduced, the natural integration points are: (a) after a payment is saved (`payment/create.php` , step 4 in §5.5) and (b) after an adjustment is saved (`adjustment/create.php` , step 3 in §5.6) — post a journal entry there using the organization's real ledger IDs instead of the current placeholder `payment_account` table.
-

9. Known Simplifications (intentional, not oversights)

- Splitting is limited to one level (a child bill cannot be split again).
- Bonus/adjustment is applied directly by an authorized role, with no separate Pending→Approved workflow.
- No payment/adjustment reversal UI yet — `bill_payment.status` and `bill_adjustment.status` support `REVERSED` / `REJECTED` values in the schema for this, but no page currently sets them; `getBillFinancials` already excludes non- `ACTIVE` /non- `APPLIED` rows, so building the reversal UI later requires no data-model change.
- `concern_billing_info` keeps only the latest row per concern (edits overwrite); full history is only retained for attachments and contract date changes, not general concern-info edits.

Billing & Service Contract Management System — Developer Documentation