Database

Prisma schema, models, seed data

- Database — Schema & Models
- Database Schema Reference

Database — Schema & Models

Bilko Database — Prisma + PostgreSQL

BookStack — Provjeri PRVO

Prije traženja bilo čega — provjeri BookStack (https://docs.basicconsulting.no). Centralna baza znanja za tools, skills, hooks, agents, rules, projekte, klijente, dokumentaciju. Ako odgovor postoji tamo — NE TRAŽI dalje.

Schema Location

`prisma/schema.prisma` — 15 models, fully defined

Database Models (15)

Core:

- Organization — Multi-tenant root (baseCurrency, country, language)
- User — RBAC (owner, admin, accountant, viewer)

Chart of Accounts:

- AccountType — Asset, Liability, Equity, Revenue, Expense
- Account — Hierarchical CoA with parent-child relations

Contacts:

- Contact — Customers, vendors, or both (type enum)

Invoicing:

- Invoice — Sales invoices with multi-currency support
- InvoiceItem — Line items with tax rates

Expenses:

- Expense — Purchase tracking with approval workflow

Transactions:

- Transaction — Double-entry ledger (debit + credit accounts)

Banking:

- BankAccount — Bank account metadata
- BankTransaction — Bank statement imports for reconciliation

Multi-Currency:

- Currency — Currency definitions (EUR, RSD, BAM, HRK, etc.)
- ExchangeRate — Historical exchange rates by date

Audit:

- LoggedAction — Immutable audit trail (APPEND-ONLY)
- SchemaVersion — Migration tracking

Key Design Decisions

1. NUMERIC(19,4) for Money

NEVER use float or JavaScript number for currency.

- Prisma type: `Decimal` (maps to PostgreSQL NUMERIC)
- Precision: 19 digits total, 4 decimal places
- Range: -999,999,999,999,999.9999 to +999,999,999,999,999.9999

2. Double-Entry Bookkeeping

Every financial event creates a `Transaction` with:

- `debitAccountId` — Account to debit
- `creditAccountId` — Account to credit
- `amount` — MUST be equal for both sides
- Balance = sum(debits) - sum(credits) per account

3. Multi-Currency with Rate Locking

- `Invoice.exchangeRate` — Locked at invoice date
- `Transaction.exchangeRate` — Locked at transaction date
- `baseAmount` — Amount converted to org's baseCurrency
- **NEVER recalculate** historical transactions with current rates

4. Immutable Audit Trail

`LoggedAction` table:

- **APPEND-ONLY** — NEVER delete or update
- Captures: table name, user ID, action (INSERT/UPDATE/DELETE), old/new values
- Used for: compliance, debugging, rollback simulation

5. Transaction Locking

- `Transaction.locked` — Once true, record is immutable
- Locked transactions cannot be edited or deleted
- Used for: end-of-period close, tax reporting

6. Organization-Scoped Multi-Tenancy

- Every record has `organizationId` foreign key
- Queries MUST filter by org (enforced in API middleware)
- No cross-org data access

7. UUID Primary Keys

- All IDs: `uuid_generate_v4()` (PostgreSQL function)
- NEVER use auto-increment for business data
- Portable across systems, no collisions

Migration Rules

1. **Never edit existing migrations** — Always create new ones
2. **Test migrations on copy** — Never run on production first
3. **Backward compatible** — Additive changes only
4. **Data migrations separate** — Use Prisma seed or custom scripts
5. **Rollback plan** — Document how to undo breaking changes

Naming Conventions

- **DB columns:** snake_case (via `@map`)
- **Prisma fields:** camelCase
- **Indexes:** `idx_{table}_{column(s)}`
- **Foreign keys:** Auto-generated by Prisma

Indexes

Defined for:

- All foreign keys (automatic)
- Common query patterns (org + date, org + status)
- Unique constraints (org + code, org + invoice number)

Enums

- UserRole: owner, admin, accountant, viewer
- NormalBalance: debit, credit
- ContactType: customer, vendor, both
- InvoiceStatus: draft, sent, viewed, paid, overdue, cancelled
- ExpenseStatus: pending, approved, paid, rejected
- AuditAction: INSERT, UPDATE, DELETE

Development Commands

```
# Generate Prisma Client
npx prisma generate

# Create migration
npx prisma migrate dev --name migration_name

# Apply migrations (production)
npx prisma migrate deploy

# Reset database (dev only)
npx prisma migrate reset

# Open Prisma Studio
npx prisma studio
```

Critical Rules

1. **NUMERIC for money** — NEVER float
2. **Double-entry enforced** — Every transaction has debit + credit
3. **Exchange rates locked** — At transaction date, NEVER recalculate
4. **Audit is append-only** — NEVER delete LoggedAction records
5. **UUID everywhere** — NEVER expose auto-increment IDs

Database Schema Reference

Bilko Database Schema

> **Status:** IMPLEMENTED (Prisma schema exists) **Location:** `/Users/makinja/ALAI/products/Bilko/packages/database/prisma/schema.prisma`
> **Database:** PostgreSQL 14+ **ORM:** Prisma 5.x **Last updated:** 2026-02-20

Purpose

This document describes the complete database schema for Bilko. The schema is IMPLEMENTED in Prisma and ready for migration. This doc explains the relationships, constraints, and design decisions.

Entity Relationship Overview

```
Organization (1) ──┬── (N) User
                   ├── (N) Account
                   ├── (N) Contact
                   ├── (N) Invoice
                   ├── (N) Expense
                   ├── (N) Transaction
                   └── (N) BankAccount

Contact (1) ──────┬── (N) Invoice
                  └── (N) Expense

Invoice (1) ──── (N) InvoiceItem

Account (1) ──────┬── (N) InvoiceItem
                  ├── (N) Expense
```

```
                ├── (N) BankAccount
                ├── (N) Transaction (debit)
                ├── (N) Transaction (credit)
                └── (N) Account (parent-child hierarchy)

BankAccount (1) ── (N) BankTransaction

Currency (1) ──┬── (N) ExchangeRate (base)
               └── (N) ExchangeRate (target)

User (1) ──┬── (N) Invoice (creator)
           ├── (N) Expense (creator)
           ├── (N) Expense (approver)
           ├── (N) Transaction (creator)
           └── (N) LoggedAction
```

Core Tables

1. Organization

Purpose: Multi-tenant root. Every business is one organization.

Column	Type	Constraints	Description
id	UUID	PK, default uuid_generate_v4()	Primary key
name	VARCHAR(255)	NOT NULL	Business name
registrationNumber	VARCHAR(50)	NULL	Company tax ID
vatNumber	VARCHAR(50)	NULL	VAT registration number
baseCurrency	CHAR(3)	NOT NULL, default 'EUR'	ISO 4217 currency code
country	CHAR(2)	NOT NULL	ISO 3166-1 alpha-2 country code
language	CHAR(2)	NOT NULL, default 'sr'	ISO 639-1 language code
fiscalYearStart	DATE	NOT NULL, default '2026-01-01'	Fiscal year start date
createdAt	TIMESTAMP	NOT NULL, default now()	Record creation timestamp
updatedAt	TIMESTAMP	NOT NULL, default now()	Last update timestamp

Indexes:

- Primary key: `id`

Business rules:

- baseCurrency determines default currency for all transactions
- country determines tax rules (Serbia 20%, BiH 17%, Croatia 25%)
- fiscalYearStart used for annual reports

---

2. User

Purpose: Users within an organization. Role-based access control.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL, CASCADE	Organization membership
email	VARCHAR(255)	UNIQUE, NOT NULL	Login email
passwordHash	VARCHAR(255)	NOT NULL	bcrypt hash (12 rounds)
fullName	VARCHAR(255)	NOT NULL	Display name
role	ENUM	NOT NULL	owner, admin, accountant, viewer
twoFactorEnabled	BOOLEAN	NOT NULL, default false	2FA status
twoFactorSecret	VARCHAR(255)	NULL	TOTP secret
lastLoginAt	TIMESTAMP	NULL	Last login timestamp
createdAt	TIMESTAMP	NOT NULL	Account creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Unique: `email`
- Foreign key: `organizationId` → Organization(id)
- Index: `idx_users_organization` on organizationId
- Index: `idx_users_email` on email

Enums:

```
enum UserRole {
  owner       // Full access, can delete org
  admin       // Can manage users and settings
  accountant  // Can create invoices/expenses
  viewer      // Read-only access
}
```

Business rules:

- One owner per organization (enforced in API)
- Cannot delete owner
- Password must be bcrypt hashed, NEVER plain text

Chart of Accounts

3. AccountType

Purpose: Defines account categories for double-entry bookkeeping.

Column	Type	Constraints	Description
id	INT	PK, AUTOINCREMENT	Primary key
name	VARCHAR(50)	UNIQUE, NOT NULL	Asset, Liability, Equity, Revenue, Expense
normalBalance	ENUM	NOT NULL	debit or credit
createdAt	TIMESTAMP	NOT NULL	Record creation

Enums:

```
enum NormalBalance {
  debit   // Asset, Expense accounts increase with debits
  credit  // Liability, Equity, Revenue accounts increase with credits
}
```

Seed data:

```
1 | Asset      | debit
2 | Liability  | credit
3 | Equity     | credit
```

```
4 | Revenue     | credit
5 | Expense     | debit
```

---

4. Account

Purpose: Chart of Accounts. Hierarchical GL accounts.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
code	VARCHAR(10)	NOT NULL	Account code (e.g., "1000", "4000")
name	VARCHAR(255)	NOT NULL	Account name (e.g., "Cash", "Revenue")
accountTypeId	INT	FK → AccountType, NOT NULL	Account category
currencyCode	CHAR(3)	NOT NULL, default 'EUR'	Account currency
parentAccountId	UUID	FK → Account, NULL	Parent account (for sub-accounts)
isActive	BOOLEAN	NOT NULL, default true	Active status
createdAt	TIMESTAMP	NOT NULL	Record creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Unique: `(organizationId, code)`
- Index: `idx_accounts_organization` on organizationId
- Index: `idx_accounts_type` on accountTypeId

Business rules:

- Code MUST be unique within organization
- Cannot delete account with transactions
- Parent-child hierarchy for sub-accounts (e.g., 1000 → 1001, 1002)

Contacts (Customers & Vendors)

5. Contact

Purpose: Customers (invoice recipients) and vendors (expense payees).

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
type	ENUM	NOT NULL	customer, vendor, both
name	VARCHAR(255)	NOT NULL	Contact name
email	VARCHAR(255)	NULL	Email address
phone	VARCHAR(50)	NULL	Phone number
registrationNumber	VARCHAR(50)	NULL	Company registration number
vatNumber	VARCHAR(50)	NULL	VAT number
addressLine1	VARCHAR(255)	NULL	Street address
addressLine2	VARCHAR(255)	NULL	Apt/suite
city	VARCHAR(100)	NULL	City
postalCode	VARCHAR(20)	NULL	Postal/ZIP code
country	CHAR(2)	NULL	ISO 3166-1 alpha-2
currencyCode	CHAR(3)	NOT NULL, default 'EUR'	Preferred currency
paymentTerms	INT	NOT NULL, default 30	Payment terms in days
notes	TEXT	NULL	Free-text notes
isActive	BOOLEAN	NOT NULL, default true	Active status
createdAt	TIMESTAMP	NOT NULL	Record creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Index: `idx_contacts_organization` on organizationId
- Index: `idx_contacts_type` on type

Enums:

```
enum ContactType {
  customer  // Invoice recipient
```

```
  vendor    // Expense payee
  both      // Can be both customer and vendor
}
```

Business rules:

- Soft delete (isActive = false) if has invoices/expenses
- currencyCode determines default invoice/expense currency

Invoicing

6. Invoice

Purpose: Sales invoices (outgoing). Revenue recognition.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
customerId	UUID	FK → Contact, NOT NULL	Invoice recipient
invoiceNumber	VARCHAR(50)	NOT NULL	Auto-generated (e.g., INV-2026-001)
invoiceDate	DATE	NOT NULL	Invoice issue date
dueDate	DATE	NOT NULL	Payment due date
currencyCode	CHAR(3)	NOT NULL	Invoice currency
exchangeRate	DECIMAL(12,6)	NOT NULL, default 1.0	Exchange rate at invoiceDate
subtotal	DECIMAL(19,4)	NOT NULL	Sum of line totals (before tax)
taxAmount	DECIMAL(19,4)	NOT NULL, default 0	Total VAT/tax
discountAmount	DECIMAL(19,4)	NOT NULL, default 0	Total discount
totalAmount	DECIMAL(19,4)	NOT NULL	subtotal + taxAmount - discountAmount
baseAmount	DECIMAL(19,4)	NOT NULL	Converted to org baseCurrency
status	ENUM	NOT NULL, default 'draft'	Invoice status
sentAt	TIMESTAMP	NULL	When invoice was sent

Column	Type	Constraints	Description
viewedAt	TIMESTAMP	NULL	When customer viewed (email tracking)
paidAt	TIMESTAMP	NULL	When marked as paid
notes	TEXT	NULL	Internal notes
terms	TEXT	NULL	Payment terms text
pdfUrl	VARCHAR(500)	NULL	Cloudflare R2 URL
createdBy	UUID	FK → User, NULL	Creator user
createdAt	TIMESTAMP	NOT NULL	Record creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Unique: `(organizationId, invoiceNumber)`
- Index: `idx_invoices_organization` on organizationId
- Index: `idx_invoices_customer` on customerId
- Index: `idx_invoices_status` on status
- Index: `idx_invoices_due_date` on dueDate
- Composite: `idx_invoices_org_status_date` on (organizationId, status, invoiceDate)

Enums:

```
enum InvoiceStatus {
  draft      // Being edited
  sent       // Sent to customer
  viewed     // Customer viewed email
  paid       // Payment received
  overdue    // Past dueDate and unpaid
  cancelled  // Voided
}
```

Business rules:

- invoiceNumber auto-generated on first save
- exchangeRate locked at invoiceDate (NEVER recalculate)
- baseAmount = totalAmount * exchangeRate
- Cannot edit invoice unless status = draft
- When status changes to 'paid', create Transaction (debit BankAccount, credit AccountsReceivable)

---

7. InvoiceItem

Purpose: Line items on invoices.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
invoiceId	UUID	FK → Invoice, CASCADE, NOT NULL	Parent invoice
lineNumber	INT	NOT NULL	Line order (1, 2, 3...)
description	VARCHAR(500)	NOT NULL	Item description
quantity	DECIMAL(10,2)	NOT NULL	Quantity sold
unitPrice	DECIMAL(19,4)	NOT NULL	Price per unit
taxRate	DECIMAL(5,2)	NOT NULL, default 0	VAT rate (20 = 20%)
lineTotal	DECIMAL(19,4)	NOT NULL	quantity * unitPrice
accountId	UUID	FK → Account, NULL	Revenue account
createdAt	TIMESTAMP	NOT NULL	Record creation

Indexes:

- Primary key: `id`
- Index: `idx_invoice_items_invoice` on invoiceId

Business rules:

- lineTotal = quantity * unitPrice (calculated before save)
- Tax amount = lineTotal * (taxRate / 100)
- accountId determines which revenue account is credited

Expenses

8. Expense

Purpose: Purchase tracking (incoming). Expense recognition.

Column	Type	Constraints	Description
id	UUID	PK	Primary key

Column	Type	Constraints	Description
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
vendorId	UUID	FK → Contact, NULL	Vendor (optional)
expenseNumber	VARCHAR(50)	NOT NULL	Auto-generated (e.g., EXP-2026-001)
expenseDate	DATE	NOT NULL	Expense date
currencyCode	CHAR(3)	NOT NULL	Expense currency
exchangeRate	DECIMAL(12,6)	NOT NULL, default 1.0	Exchange rate at expenseDate
amount	DECIMAL(19,4)	NOT NULL	Total expense amount
baseAmount	DECIMAL(19,4)	NOT NULL	Converted to org baseCurrency
taxAmount	DECIMAL(19,4)	NOT NULL, default 0	VAT amount
category	VARCHAR(100)	NOT NULL	Expense category
paymentMethod	VARCHAR(50)	NULL	cash, card, bank_transfer, etc.
accountId	UUID	FK → Account, NULL	Expense account
description	TEXT	NULL	Expense description
receiptUrl	VARCHAR(500)	NULL	Cloudflare R2 URL
status	ENUM	NOT NULL, default 'pending'	Approval status
approvedBy	UUID	FK → User, NULL	Approver user
approvedAt	TIMESTAMP	NULL	Approval timestamp
paidAt	TIMESTAMP	NULL	Payment timestamp
createdBy	UUID	FK → User, NULL	Creator user
createdAt	TIMESTAMP	NOT NULL	Record creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Unique: `(organizationId, expenseNumber)`
- Index: `idx_expenses_organization` on organizationId
- Index: `idx_expenses_vendor` on vendorId
- Index: `idx_expenses_category` on category
- Index: `idx_expenses_date` on expenseDate
- Composite: `idx_expenses_org_date_category` on (organizationId, expenseDate, category)

Enums:

```
enum ExpenseStatus {
  pending   // Awaiting approval
  approved  // Approved, ready to pay
  paid      // Payment made
  rejected  // Approval denied
}
```

Business rules:

- expenseNumber auto-generated
- exchangeRate locked at expenseDate
- baseAmount = amount * exchangeRate
- When status → approved, create Transaction (debit ExpenseAccount, credit AccountsPayable)
- When status → paid, create Transaction (debit AccountsPayable, credit BankAccount)

Transactions (Double-Entry Ledger)

9. Transaction

Purpose: General ledger transactions. Every financial event creates a transaction.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
transactionDate	DATE	NOT NULL	Transaction date
description	VARCHAR(255)	NOT NULL	Transaction description
debitAccountId	UUID	FK → Account, NOT NULL	Account to debit
creditAccountId	UUID	FK → Account, NOT NULL	Account to credit
amount	DECIMAL(19,4)	NOT NULL	Transaction amount
currencyCode	CHAR(3)	NOT NULL	Transaction currency
exchangeRate	DECIMAL(12,6)	NOT NULL, default 1.0	Exchange rate at transactionDate
baseAmount	DECIMAL(19,4)	NOT NULL	Converted to org baseCurrency

Column	Type	Constraints	Description
referenceType	VARCHAR(50)	NULL	invoice, expense, payment, manual
referenceId	UUID	NULL	Invoice/Expense ID
locked	BOOLEAN	NOT NULL, default false	Immutable if true
lockedAt	TIMESTAMP	NULL	When locked
reconciled	BOOLEAN	NOT NULL, default false	Matched to bank transaction
reconciledAt	TIMESTAMP	NULL	When reconciled
notes	TEXT	NULL	Free-text notes
createdBy	UUID	FK → User, NULL	Creator user
createdAt	TIMESTAMP	NOT NULL	Record creation

Indexes:

- Primary key: `id`
- Index: `idx_transactions_organization` on organizationId
- Index: `idx_transactions_date` on transactionDate
- Index: `idx_transactions_debit` on debitAccountId
- Index: `idx_transactions_credit` on creditAccountId
- Index: `idx_transactions_reference` on (referenceType, referenceId)
- Composite: `idx_transactions_org_date` on (organizationId, transactionDate)

Business rules:

- **DEBITS = CREDITS** — Every transaction has exactly one debit and one credit
- debitAccountId ≠ creditAccountId (enforced in API)
- Cannot edit if locked = true
- Cannot delete if reconciled = true
- exchangeRate locked at transactionDate
- baseAmount = amount * exchangeRate

Common transaction patterns:

1. **Invoice created (draft → sent):**
   - Debit: Accounts Receivable (Asset)
   - Credit: Revenue (Revenue)
2. **Invoice paid:**
   - Debit: Bank Account (Asset)
   - Credit: Accounts Receivable (Asset)
3. **Expense approved:**
   - Debit: Expense Account (Expense)
   - Credit: Accounts Payable (Liability)

4. **Expense paid:**
   - Debit: Accounts Payable (Liability)
   - Credit: Bank Account (Asset)

Banking & Reconciliation

10. BankAccount

Purpose: Bank account metadata.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
organizationId	UUID	FK → Organization, NOT NULL	Organization scope
accountId	UUID	FK → Account, NOT NULL	GL account (must be Asset)
bankName	VARCHAR(255)	NOT NULL	Bank name
accountNumber	VARCHAR(50)	NULL	Account number
iban	VARCHAR(50)	NULL	IBAN
currencyCode	CHAR(3)	NOT NULL, default 'EUR'	Account currency
currentBalance	DECIMAL(19,4)	NOT NULL, default 0	Current balance
isActive	BOOLEAN	NOT NULL, default true	Active status
createdAt	TIMESTAMP	NOT NULL	Record creation
updatedAt	TIMESTAMP	NOT NULL	Last update

Indexes:

- Primary key: `id`
- Index: `idx_bank_accounts_organization` on organizationId

Business rules:

- accountId MUST be Asset type account
- currentBalance updated when transactions created
- Soft delete (isActive = false)

---

11. BankTransaction

Purpose: Bank statement imports. For reconciliation.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
bankAccountId	UUID	FK → BankAccount, CASCADE, NOT NULL	Parent bank account
transactionDate	DATE	NOT NULL	Transaction date
amount	DECIMAL(19,4)	NOT NULL	Positive = credit, negative = debit
description	VARCHAR(500)	NULL	Bank description
reference	VARCHAR(255)	NULL	Reference number
reconciled	BOOLEAN	NOT NULL, default false	Matched to GL transaction
matchedTransactionId	UUID	NULL	GL transaction ID
createdAt	TIMESTAMP	NOT NULL	Record creation

Indexes:

- Primary key: `id`
- Index: `idx_bank_transactions_account` on bankAccountId
- Index: `idx_bank_transactions_date` on transactionDate

Business rules:

- Imported from CSV bank statements
- Matched to GL transactions via reconciliation workflow
- reconciled = true when matched

Multi-Currency

12. Currency

Purpose: Supported currencies.

Column	Type	Constraints	Description
code	CHAR(3)	PK	ISO 4217 currency code
name	VARCHAR(100)	NOT NULL	Currency name
symbol	VARCHAR(10)	NULL	Currency symbol

Column	Type	Constraints	Description
decimalPlaces	SMALLINT	NOT NULL, default 2	Decimal precision
isActive	BOOLEAN	NOT NULL, default true	Active status
createdAt	TIMESTAMP	NOT NULL	Record creation

Seed data:

```
EUR | Euro          | €    | 2
RSD | Serbian Dinar | din. | 2
BAM | Bosnian Mark  | KM   | 2
HRK | Croatian Kuna | kn   | 2
USD | US Dollar     | $    | 2
```

---

13. ExchangeRate

Purpose: Historical exchange rates.

Column	Type	Constraints	Description
id	UUID	PK	Primary key
baseCurrency	CHAR(3)	FK → Currency, NOT NULL	From currency
targetCurrency	CHAR(3)	FK → Currency, NOT NULL	To currency
rate	DECIMAL(12,6)	NOT NULL	Exchange rate
effectiveDate	DATE	NOT NULL	Rate effective date
source	VARCHAR(50)	NULL	ECB, fixer.io, manual
lastUpdated	TIMESTAMP	NOT NULL	Last update timestamp

Indexes:

- Primary key: `id`
- Unique: `(baseCurrency, targetCurrency, effectiveDate)`
- Index: `idx_exchange_rates_date` on effectiveDate
- Index: `idx_exchange_rates_pair` on (baseCurrency, targetCurrency)

Business rules:

- Rates fetched daily from ECB or fixer.io API
- When creating transaction, rate is locked at transaction date
- If no rate for exact date, use nearest available (warn in logs)

Audit Trail

14. LoggedAction

Purpose: Immutable audit log. Captures all INSERT/UPDATE/DELETE.

Column	Type	Constraints	Description
eventId	BIGINT	PK, AUTOINCREMENT	Event ID
schemaName	TEXT	NOT NULL	Database schema (default: public)
tableName	TEXT	NOT NULL	Table name
userId	UUID	FK → User, NULL	User who performed action
actionTimestamp	TIMESTAMP	NOT NULL, default now()	When action occurred
action	ENUM	NOT NULL	INSERT, UPDATE, DELETE
rowData	JSONB	NULL	Full row data before change
changedFields	JSONB	NULL	Changed fields (UPDATE only)
queryText	TEXT	NULL	SQL query (if available)
clientIp	INET	NULL	Client IP address
applicationName	TEXT	NOT NULL, default 'fiken-clone-api'	Application identifier

Indexes:

- Primary key: `eventId`
- Index: `idx_logged_actions_timestamp` on actionTimestamp
- Index: `idx_logged_actions_table` on tableName
- Index: `idx_logged_actions_user` on userId

Enums:

```
enum AuditAction {
  INSERT
  UPDATE
  DELETE
}
```

Business rules:

- **APPEND-ONLY** — NEVER delete or update records
- Triggered via Prisma middleware (automatic)
- Used for: compliance, debugging, rollback simulation

Schema Version

15. SchemaVersion

Purpose: Migration tracking.

Column	Type	Constraints	Description
version	VARCHAR(20)	PK	Version string (e.g., "1.0.0")
appliedAt	TIMESTAMP	NOT NULL, default now()	Migration timestamp
description	TEXT	NULL	Migration description

Business rules:

- Updated by Prisma migrations
- Used to verify schema version matches application version

Data Types & Precision

NUMERIC(19,4) for ALL Money

CRITICAL: NEVER use `float`, `double`, or JavaScript `number` for currency.

- Precision: 19 digits total, 4 decimal places
- Range: -999,999,999,999,999.9999 to +999,999,999,999,999.9999
- Prisma type: `Decimal`
- PostgreSQL type: `NUMERIC(19,4)`

Why:

- JavaScript `number` has 53-bit precision (safe up to 2^53 - 1 = 9,007,199,254,740,991)
- Financial calculations require exact decimal precision
- Example: 0.1 + 0.2 = 0.30000000000000004 (float error)

Usage in code:

```
import { Decimal } from '@prisma/client/runtime'

const amount = new Decimal('125000.0000')
const taxRate = new Decimal('0.20')
const taxAmount = amount.times(taxRate)  // 25000.0000
```

Constraints Summary

Primary Keys

All tables use UUID primary keys (except AccountType uses INT auto-increment).

Foreign Keys

All foreign keys have `onDelete: Cascade` (deleting organization deletes all data).

Unique Constraints

- User.email
- Account.(organizationId, code)
- Invoice.(organizationId, invoiceNumber)
- Expense.(organizationId, expenseNumber)
- ExchangeRate.(baseCurrency, targetCurrency, effectiveDate)

Check Constraints

(Enforced in API layer, not database):

- amount > 0 for all financial amounts
- dueDate >= invoiceDate for invoices
- debitAccountId ≠ creditAccountId for transactions

Indexes Strategy

Query Patterns Optimized

1. **List by organization + filter:**
   - `(organizationId, status, date)` composite index on invoices
   - `(organizationId, category, date)` composite index on expenses
2. **Foreign key lookups:**
   - All foreign keys have indexes
3. **Date range queries:**
   - Dedicated indexes on `transactionDate`, `invoiceDate`, `expenseDate`, `dueDate`
4. **Reconciliation:**
   - Index on `(referenceType, referenceId)` for transaction lookups

Migration Commands

```
# Generate Prisma Client
npx prisma generate

# Create migration
npx prisma migrate dev --name migration_name

# Apply migrations (production)
npx prisma migrate deploy

# Reset database (dev only)
npx prisma migrate reset

# Seed initial data
npx prisma db seed
```

Seed Data

AccountType

```
INSERT INTO account_types (id, name, normal_balance) VALUES
(1, 'Asset', 'debit'),
(2, 'Liability', 'credit'),
(3, 'Equity', 'credit'),
(4, 'Revenue', 'credit'),
```

```
(5, 'Expense', 'debit');
```

Currency

```
INSERT INTO currencies (code, name, symbol, decimal_places) VALUES
('EUR', 'Euro', '€', 2),
('RSD', 'Serbian Dinar', 'din.', 2),
('BAM', 'Bosnian Mark', 'KM', 2),
('HRK', 'Croatian Kuna', 'kn', 2),
('USD', 'US Dollar', '$', 2);
```

Entity Relationship Diagram

```
erDiagram
    ORGANIZATION ||--o{ USER : "members"
    ORGANIZATION ||--o{ CONTACT : "contacts"
    ORGANIZATION ||--o{ INVOICE : "invoices"
    ORGANIZATION ||--o{ EXPENSE : "expenses"
    ORGANIZATION ||--o{ TRANSACTION : "transactions"
    ORGANIZATION ||--o{ BANK_ACCOUNT : "bank accounts"
    ORGANIZATION ||--o{ ACCOUNT_TYPE : "account types"
    ORGANIZATION ||--o{ ACCOUNT : "chart of accounts"
    ORGANIZATION ||--o{ CURRENCY : "currencies"
    ORGANIZATION ||--o{ LOGGED_ACTION : "audit log"

    INVOICE ||--o{ INVOICE_ITEM : "line items"
    INVOICE }o--|| CONTACT : "billed to"
    INVOICE }o--|| USER : "created by"

    EXPENSE }o--o| CONTACT : "vendor"
    EXPENSE }o--|| ACCOUNT : "category account"

    ACCOUNT_TYPE ||--o{ ACCOUNT : "classifies"
    ACCOUNT ||--o{ TRANSACTION : "debited in"
    ACCOUNT ||--o{ TRANSACTION : "credited in"
    ACCOUNT |o--o| BANK_ACCOUNT : "linked GL"
```

```
    BANK_ACCOUNT ||--o{ BANK_TRANSACTION : "transactions"
    BANK_TRANSACTION }o--o| TRANSACTION : "reconciles"

    CURRENCY ||--o{ EXCHANGE_RATE : "base rates"

    LOGGED_ACTION }o--o| USER : "performed by"
```

Domain groupings:

- **Identity:** Organization, User
- **Financial Core:** Account, AccountType, Transaction
- **Invoicing:** Invoice, InvoiceItem, Contact
- **Expenses:** Expense
- **Banking:** BankAccount, BankTransaction
- **Compliance:** LoggedAction, SchemaVersion
- **Currency:** Currency, ExchangeRate

Migration Strategy

Prisma Migrate Workflow

Bilko uses **Prisma Migrate** for all schema changes. No manual SQL migrations.

Development workflow

```
# 1. Edit packages/database/prisma/schema.prisma
# 2. Generate and apply migration
npx prisma migrate dev --name describe_your_change

# 3. Regenerate Prisma Client
npx prisma generate

# 4. Seed if new lookup data needed
npx prisma db seed
```

Production deployment workflow

```
# Applied automatically during Railway deploy via package.json postinstall:
# "postinstall": "prisma migrate deploy && prisma generate"
```

```
# Manual production apply:
npx prisma migrate deploy

# Verify migration status:
npx prisma migrate status
```

Migration naming conventions

Type	Name format	Example
Add table	`add_{table_name}`	`add_webhook_subscriptions`
Add column	`add_{column}_to_{table}`	`add_sefId_to_invoices`
Remove column	`remove_{column}_from_{table}`	`remove_legacy_field_from_users`
Add index	`add_index_{table}_{columns}`	`add_index_invoices_status_dueDate`
Data fix	`fix_{description}`	`fix_applicationName_default`

Zero-Downtime Migration Patterns

For production migrations on live data:

Pattern	When to use	Example
Additive only	New nullable columns	Add `sefId String?` to Invoice
Expand-contract	Rename column	Add new column → backfill → drop old
Shadow table	Large table restructure	Create new table → migrate → swap
Index concurrently	Add index without lock	Manual SQL via `prisma db execute`

> **Rule:** Never drop columns in the same migration that removes their usage from code. Deploy code first (ignoring old column), then migrate.

Pre-Deploy Migration Checklist

Before deploying any migration to production:

- ☐ `prisma migrate status` shows no drift from dev
- ☐ Migration tested on a staging DB with production data volume
- ☐ Rollback plan documented (additive migrations are safe; destructive need manual rollback SQL)

☐ `LoggedAction.applicationName` default changed from `"fiken-clone-api"` to `"bilko-api"` (**pending — fix before first production deploy**)

☐ Backup taken before running destructive migrations

Known Pre-Deploy Fix Required

```
-- LoggedAction.applicationName has legacy default value in current schema
-- Run via: npx prisma db execute --file fix_applicationName.sql

ALTER TABLE "LoggedAction"
  ALTER COLUMN "applicationName" SET DEFAULT 'bilko-api';

UPDATE "LoggedAction"
  SET "applicationName" = 'bilko-api'
  WHERE "applicationName" = 'fiken-clone-api';
```

Enhanced Index Design

Index Inventory

All indexes defined in `packages/database/prisma/schema.prisma` and verified against query patterns:

Table	Index Columns	Type	Purpose
`Invoice`	`(organizationId, status, invoiceDate)`	Composite	List invoices with status filter
`Invoice`	`(organizationId, contactId)`	Composite	Invoices by customer
`Invoice`	`(organizationId, dueDate, status)`	Composite	Overdue invoice cron
`InvoiceItem`	`(invoiceId)`	FK	Load line items
`Expense`	`(organizationId, status, expenseDate)`	Composite	List expenses with filter
`Expense`	`(organizationId, categoryAccountId)`	Composite	Expense by category
`Transaction`	`(organizationId, transactionDate)`	Composite	Date range reports
`Transaction`	`(organizationId, debitAccountId)`	Composite	Account balance calc

Table	Index Columns	Type	Purpose
`Transaction`	`(organizationId, creditAccountId)`	Composite	Account balance calc
`Transaction`	`(referenceType, referenceId)`	Composite	Lookup by invoice/expense
`BankTransaction`	`(bankAccountId, transactionDate)`	Composite	Bank statement view
`BankTransaction`	`(organizationId, reconciled)`	Composite	Unreconciled transactions
`ExchangeRate`	`(baseCurrency, targetCurrency, effectiveDate)`	Unique	Daily rate lookup
`LoggedAction`	`(organizationId, tableName, createdAt)`	Composite	Audit trail queries
`LoggedAction`	`(organizationId, userId)`	Composite	User activity queries

Index Design Principles

1. **Org-first composite indexes**: Every query filters by `organizationId` first — it must be the leftmost column in all multi-column indexes.
2. **Status + date combos**: List endpoints combine status filter + date sort, so `(orgId, status, date)` triples cover both filter and ORDER BY.
3. **Foreign key indexes**: Prisma creates FK indexes automatically; verify with `\d tableName` in psql.
4. **Covering indexes**: For report queries that fetch only a few columns, consider partial indexes in Phase 2 (e.g., `WHERE status = 'paid'`).

Query Performance Targets

Query	Target	Notes
List invoices (1000 rows)	< 50ms	With org + status index
Invoice by ID	< 5ms	PK lookup
Dashboard metrics	< 300ms	7 parallel aggregations
P&L report (1 year)	< 500ms	Transaction table scan with date index
Audit trail query	< 100ms	LoggedAction composite index
VAT report (monthly)	< 200ms	Invoice + Expense aggregation

Audit Log Partitioning Strategy

Why Partition LoggedAction?

The `LoggedAction` table is append-only and retains data for **7 years** (financial compliance requirement). At 10 requests/minute per organization, a 100-org instance generates ~500K audit rows/month. After 3 years: ~18M rows.

Without partitioning: sequential scans on `LoggedAction` become slow. With partitioning: queries can prune partitions by year.

Partitioning Approach: Range by Year

```
-- Convert LoggedAction to range-partitioned table (PostgreSQL 11+)
-- Execute ONCE during Phase 2 setup, before data volume grows

CREATE TABLE "LoggedAction_partitioned"
  PARTITION OF "LoggedAction" (
    -- same columns
  ) PARTITION BY RANGE (EXTRACT(YEAR FROM "createdAt"));

-- Annual partitions
CREATE TABLE "LoggedAction_2026" PARTITION OF "LoggedAction_partitioned"
  FOR VALUES FROM (2026) TO (2027);

CREATE TABLE "LoggedAction_2027" PARTITION OF "LoggedAction_partitioned"
  FOR VALUES FROM (2027) TO (2028);
-- etc.
```

Prisma Compatibility

Prisma does not natively manage PostgreSQL table partitioning. Strategy:

1. Define base table in `schema.prisma` (no partitioning directive)
2. Apply partitioning via `prisma db execute` with raw SQL after initial migration
3. Maintain partition creation as a yearly operations task (or use `pg_partman` extension)

Archive Strategy (7-Year Retention)

```
Year 1-7:    Active partitions in PostgreSQL (Railway EU Frankfurt)
Year 7+:     Archive partition to cold storage (Cloudflare R2 Glacier tier)
             Delete from PostgreSQL
             Retain R2 archives for GDPR compliance period
```

Trigger: Automated yearly job checks `MIN(createdAt)` in oldest partition. If > 7 years: export to R2, drop partition.

MVP Approach (Phase 1)

For MVP: **no partitioning required**. Single table with composite index `(organizationId, tableName, createdAt)` sufficient for < 1M rows. Implement partitioning in Phase 2 before reaching 5M rows.

End of Database Schema