Timesheet DB Documentation

Raul-Mihai Rusu – *Data Engineer Dava.X Cluj*

Descriptions

This timesheets database is designed to support time tracking across the organization. Its primary used to store timesheets entries submitted by employes through internal applications.

Given the potential value of this data, it could play a key role in enhancing other processes such as billing, payroll, analytics, planning, and audits.

Since we are designing it from the ground up, rather than retrofitting a legacy system, we have more flexibility. We can structure the schema, access model, and integration points to accommodate modern data tools and practices. These include BI platforms, ETL pipelines, and compliance workflows.

The aim is to create a design to serve the core timesheet app while also supporting the downstream processes mentioned earlier. At the same time, we aim to minimize the potential performance issues, system overheads, and bottlenecks that these practices can sometimes introduce.

Research

Use Cases

Use Case	Details
Timesheet App	Who: Employees Goal: Submit weekly work logs Interaction: Web/Mobile interfaces via API calls to a backend
ETL Processes	Who: Data engineers Goal: Clean, transform, and store data for downstream system usage Interaction: Direct access to the database
Business Intelligence	Who: BI analysts, team leads, department heads Goal: Analytics and performance reports Interaction: Indirect access to the database via third-party platforms and dashboards
Workforce and Productivity Analysis	Who: HR department Goal: Monitor workforce utilization Interaction: Indirect - dashboards or integrations (e.g., Workday)
Billing and Finance	Who: Finance department Goal: Generate invoices based on billable time Interaction: Indirect - finance systems or exports
Audit and Compliance Checks	Who: Internal auditors, compliance officers Goal: Ensure time entries are accurate, legal, and policy-compliant Interaction: Indirect - dashboards; direct – provided SQL queries

Design considerations

- **API-Oriented Schema**
  Minimize joins during writes; enforce clear and traceable state transitions.
- **Normalized Structure with Strong Integrity**
  Use primary/foreign keys and constraints (NOT NULL, CHECK) to ensure data quality.
- **Descriptive and Verbose Fields**
  Favor explicit column names and meaningful statuses for downstream clarity.
- **Change Logging and Auditing**
  Record *who*, *when* for modifications.
- **Flexible Metadata Storage**
  Use JSON columns for optional or extensible fields without schema changes.

- **Materialized Views for Reporting**
  Pre-aggregate weekly/monthly summaries to reduce query load.
- **Role-Based Access Control (RBAC)**
  Implement granular roles for different use cases.
- **Personal Data Handling**
  Apply anonymization or pseudonymization where personal data is not essential.
- **Timestamps and Temporal Columns**
- **Indexing**
  Identify bottlenecks and costly queries -> optimize using indexes

Database Design

Overview and Scope

The purpose of this database is to serve as a learning playground where I can apply the information acquired during the RDBMS training.

The solution is intended to serve as a centralized data store for employee time tracking. Given the value of this information, I also want the design to be a foundation that can support broader organizational needs such as analytics, compliance, and ETL processes.

The database will be implemented using **SQL Server** and the notes will reflect that.

Supported Use Cases

The database is designed to accommodate for the following business scenarios:

- Timesheet Application
- ETL (Extract, Transform, Load) Processes
- Business Intelligence (BI)
- Audit and Compliance Checks

Out of Scope

- Workforce and Productivity Analysis
- Billing and Finance

Tables and Schema

1. EMPLOYEE

Stores all employee master data and relationships, including department, role, and reporting structure.

- Uses self-referencing FK for manager
- Unique email constraint enforces user identity
- FK to department and role ensure referential integrity
- Temporal fields for low audit needs, and basic ETL
- Allows for soft deletes

Field	Data Type	Constraints / Notes
employee_id	INT	PK, IDENTITY(1,1), NOT NULL
employee_name	NVARCHAR(100)	NOT NULL
employee_email	NVARCHAR(150)	NOT NULL, UNIQUE
department_id	INT	FK, NOT NULL
hire_date	DATE	NOT NULL
role_id	INT	FK, NOT NULL
experience_level	NVARCHAR(50)	NOT NULL
manager_id	INT	FK (self), NULLABLE
created_at	DATETIME2	NOT NULL, DEFAULT GETDATE()
updated_at	DATETIME2	NOT NULL, DEFAULT GETDATE()
is_active	BIT	NOT NULL, DEFAULT TRUE

2. DEPARTMENT

Reference table for departments, used for grouping and reporting employee affiliations.

- Unique department_name constraint prevents duplicates.

Field	Data Type	Constraints / Notes
department_id	INT	PK, IDENTITY(1,1), NOT NULL
department_name	NVARCHAR(100)	NOT NULL, UNIQUE

3. ROLES

Defines all roles available to employees within the organization.

- Unique role_name constraint for distinct roles.

Field	Data Type	Constraints / Notes
role_id	INT	PK, IDENTITY(1,1), NOT NULL
role_name	NVARCHAR(50)	NOT NULL, UNIQUE

4. CLIENT

Reference data for external clients; projects are associated with clients.

- Unique client_name constraint to avoid duplicates

Field	Data Type	Constraints / Notes
client_id	INT	PK, IDENTITY(1,1), NOT NULL
client_name	NVARCHAR(150)	NOT NULL, UNIQUE

5. PROJECTS

Captures all active and historical projects, linked to clients.

- Foreign key to client enforces integrity.
- CHECK constraint on project_status restricts values.

Field	Data Type	Constraints / Notes
project_id	INT	PK, IDENTITY(1,1), NOT NULL
client_id	INT	FK, NOT NULL
project_name	NVARCHAR(200)	NOT NULL
project_status	NVARCHAR(20)	NOT NULL, CHECK (project_status IN ('active','ended','freeze'))

6. TIMESHEET

Tracks weekly or period-based work logs submitted by employees

- Metadata field allows for extensibility.
- CHECK constraint on status for valid workflow states.
- Timestamps track creation and audit changes.

Field	Data Type	Constraints / Notes
timesheet_id	INT	PK, IDENTITY(1,1), NOT NULL
employee_id	INT	FK, NOT NULL
start_date	DATE	NOT NULL
end_date	DATE	NOT NULL, CHECK (end_date >= start_date)
timesheet_status	NVARCHAR(20)	NOT NULL, CHECK (timesheet_status IN ('draft','submitted','approved','rejected','edited'))
created_at	DATETIME2	NOT NULL, DEFAULT GETDATE()
updated_at	DATETIME2	NOT NULL, DEFAULT GETDATE()
metadata	NVARCHAR(MAX) (JSON)	NULLABLE

7. TimeEntry

Granular records of work performed, hours logged, and project allocation.

- Foreign key links to both timesheet and project.
- CHECK constraint on hours prevents invalid entries.

Field	Data Type	Constraints / Notes
entry_id	INT	PK, IDENTITY(1,1), NOT NULL
timesheet_id	INT	FK, NOT NULL
project_id	INT	FK, NOT NULL
entry_date	DATE	NOT NULL, CHECK (entry_date >= start_date of linked timesheet)
hours_worked	DECIMAL(5,2)	NOT NULL, CHECK (hours_worked > 0 AND hours_worked <= 24)
entry_description	NVARCHAR(255)	NULLABLE
is_billable	BIT	NOT NULL

8. AuditLog

Captures all changes or actions performed on timesheets for compliance and auditing.

- References both the timesheet and the acting user.
- Supports full audit trail via before/after JSON.

Field	Data Type	Constraints / Notes
log_id	INT	PK, IDENTITY(1,1), NOT NULL
timesheet_id	INT	FK, NOT NULL
action_taken	NVARCHAR(30)	NOT NULL, CHECK (action_taken IN ('added', 'edit', 'add_timeentry', 'edit_timeentry', 'delete_timeentry', 'status_change', 'delete'))
action_date	DATETIME2	NOT NULL, DEFAULT GETDATE()
employee_id	INT	FK, NOT NULL (to Employee)
new_value	NVARCHAR(MAX) (JSON)	NULLABLE
new_value	NVARCHAR(MAX) (JSON)	NULLABLE

Notes:

- Temporal fields and soft deletes were considered only for `employee` table, although all table could benefit from them

- Although this data can be inferred from log files, I think the small overhead and query inefficiencies at the DB level is worth due to simplified ETL and BL workflows, data remains consistent and queryable, single source of truth regardless of how the data is modified

Triggers

1. Simple triggers to set `update_at` on update: `employee` table and `timesheet` table
2. Triggers to populate `auditlog` table when:
    - Insert -> new timesheets are added
    - Update -> state change on timesheet (submitted -> approved)
    - Update -> change other fields on timesheet
    - Insert/Update/Delete -> crud on timeentry

Notes:

- Triggers inserting into the audit log expect `SESSION_CONTEXT employee_id` to be set beforehand to identify the user acting on the timesheet (e.g., a manager approving it). Especially important for the backed app that interacts with the DB.
- When a timeentry is modified, the `updated_at` field of the parent timesheet should also be updated. However, this change introduced unpredictable behavior in the trigger chains, occasionally resulting in duplicate entries in the log table.

Views

Notes:

- In hindsight, the chosen view could have better supported the use cases and adhered more closely to security and anonymization goals defined by me, though I found it challenging to identify a better approach.

Roles and Permissions

1. **db_admin**
   Database administrators: comprehensive controls over the database.

2. **etl_engineer**
   User or processes responsible for ETL operations. Require broad access to read/write, create views, materialized views, read log files, create procedures to facilitate data transformation and movement.

3. **bi_analyst**
   BI analysts, tools, and dashboards. Querry and analyze data to generate reports and offer insights, without modifying the underlying data.

4. **timesheet_app_user**
   Indended for the end-user, to be more specific, API's that interact with the DB and expose the functionality to the end-user.
   Although not implemented for this use case row level security could be added, especially since `SESSION_CONTEXT` is required for correct audit logging.
   Insert, update and delete on relevant tables for business logic.

5. **auditor**
   Internal auditors or compliance officers. Review historical data and activity logs. Read-only access.