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.

I also wanted to learn more and pay a closer attention to security and performance.

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

**Disclaimer about LLM usage**

I made use of GPT in:

* Research: clarification, explanation, validation, and refinement
* Learning: explain concept, best practice, SQL examples
* Documentation: heavy use to structure and refine text and ideas

For the code itself, I intentionally minimized LLM usage, opting instead to refresh my SQL knowledge and explore the new concepts head on.

**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**

### 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 | |

### 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 |

### 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 |

### 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 |

### 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')) |

### 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 |

### 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 |

### 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