CacheNCarry 🧳

Where "left behind" meets `LEFT JOIN`.

CacheNCarry is an end-of-semester storage management system designed to track and manage student belongings (cloakroom luggage + a mattress). The system ensures easy check-in/check-out processes, reduces human error, and provides traceability for checked-in bags and misplaced mattresses.

🎣 Flow

1. Student Registration:

At the end of each semester, a `student` registers their `belonging(s)`:

- Belongings can be of two types: `luggage` or `mattress`.
- Each item is assigned a unique `UUID`.
- A QR code is generated for each item and must be printed and physically attached to the respective belonging by the student.

2. Item Check-In:

At the warehouse:

- A `staff` member scans a QR code on the student's phone to begin a `session`, associating the session with that student, and logging check-in time.
- The QR codes on each belonging are scanned to link them to the student and register them into the system.
- The staff ends the session once all items have been scanned.

3. Storage Period:

- Belongings remain in the warehouse during the student's absence.
- Mattresses are handled separately: at the start of the semester, they are moved directly to student rooms by university workers.

4. Item Retrieval (Check-Out):

Upon the student's return:

- The student visits the warehouse.
- Staff scan the student's QR code to open a check-out session.
- For each item:
  - **Luggage**: The system validates whether the scanned luggage belongs to the student in session. If ownership does not match, the item cannot be checked out.
  - **Mattress**: Since the mattress is already placed in the student's room, the staff scan both the student and the mattress QR code to validate ownership.

- The staff ends the session once all items have been scanned.

5. Misplacement Handling:

If a scanned mattress does not belong to the student:

- An `incident` is opened.
- Because all mattresses are uniquely tagged and linked to their rightful owners, it can be easily determined:
    - Who the mattress actually belongs to.
    - Who mistakenly received another student's mattress.
- This allows staff to resolve such incidents efficiently without manual investigation.
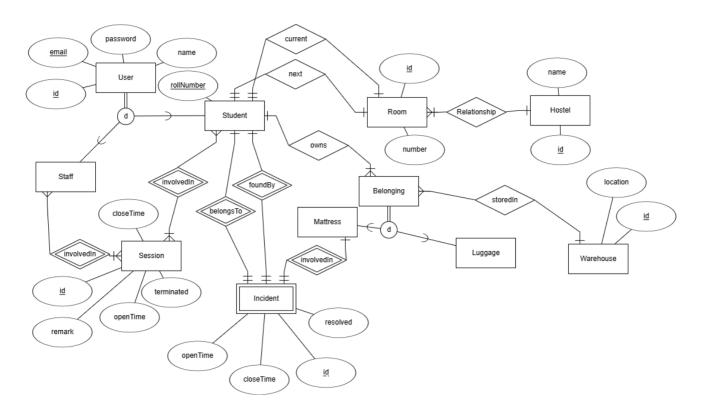

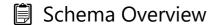
🛠 Tech stack

- **Frontend & API**: SvelteKit + TailwindCSS

- **Database**: MySQL

- **DB Connector**: Prisma

- **Containerization**: Docker

✏️ Entity Relationship Diagram

📋 Schema Overview

Entities

user

All students and staff register as `users`.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for user
name	VARCHAR(191)		user's real name
email	VARCHAR(191)	UNIQUE	unique organization email id
password	VARCHAR(191)	UNIQUE	hashed password

staff

A `staff` member oversees sessions and incidents.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY, FOREIGN KEY (`user.id`)	UUID identification for staff corressponding to user

student

A `student` can register their items and check them in with QR codes attached to them.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY, FOREIGN KEY (`user.id`)	UUID identification for student corressponding to user
roll_number	VARCHAR(12)	UNIQUE	unique roll number
current_room_id	VARCHAR(191)	FOREIGN KEY (`room.id`)	UUID identification of current occupancy
next_room_id	VARCHAR(191)	FOREIGN KEY (`room.id`), NULLABLE	UUID identification of next occupancy

hostel

A `hostel` is a place of residence containing rooms.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	UUID identification for hostel
name	VARCHAR(191)	PRIMARY KEY	name of the hostel

room

Represents a `room` within a hostel.

Field	Type	Constraints	Description
`id`	VARCHAR(191)	PRIMARY KEY	`UUID` identification for a room
`number`	VARCHAR(6)	UNIQUE per hostel (`hostel_id`, `number`)	hostel room number
`hostel_id`	VARCHAR(191)	FOREIGN KEY (`hostel.id`)	`UUID` identification for the corressponding hostel

warehouse

A `warehouse` is a physical storage unit for student belingings.

Field	Type	Constraints	Description
`id`	VARCHAR(191)	PRIMARY KEY	`UUID` identification for a warehouse
`location`	VARCHAR(191)		information about general location

belonging

A `belonging` is a general entity for storable items.

Field	Type	Constraints	Description
`id`	INTEGER	PRIMARY KEY	`UUID` identification for each belonging
`description`	VARCHAR(191)	NULLABLE	general description of the item
`is_checked_in`	BOOLEAN	DEFAULT FALSE	indication of whether the item is checked in
`checked_in_at`	DATETIME	NULLABLE	timestamp for when the item was checked in
`checked_out_at`	DATETIME	NULLABLE	timestamp for when the item was checked out
`student_id`	VARCHAR(191)	FOREIGN KEY (`student.id`)	`UUID` identification for the owning student
`warehouse_id`	VARCHAR(191)	FOREIGN KEY (`warehouse.id`), NULLABLE	`UUID` identification for the storage warehouse

luggage

Student `luggage` that can be checked in and out in direct sessions.

Field	Type	Constraints	Description

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY, FOREIGN KEY (`belonging.id`)	`UUID` identification for each bag

mattress

A `mattress` can be stored over the semester break, and gets transported to a student's room before they arrive. An ownership discrepancy leads to an incident.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY, FOREIGN KEY (`belonging.id`)	`UUID` identification for a mattress

session

Represents a warehouse interaction involving a student and staff member.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	`UUID` identification for each session
remark	VARCHAR(191)	NULLABLE	Optional notes about the session
open_time	DATETIME	DEFAULT `NOW()`	timestamp for when the session started.
close_time	DATETIME	NULLABLE	timestamp for when the session ended.
terminated	BOOLEAN	DEFAULT FALSE	indication of whether the session was manually terminated.
staff_id	VARCHAR(191)	FOREIGN KEY (`staff.id`)	`UUID` identification for the staff member managing the session
student_id	VARCHAR(191)	FOREIGN KEY (`student.id`), UNIQUE per staff (`staff_id`, `student_id`)	`UUID` identification for the student who owns the belongings

incident

Represents misplacement or ownership conflict involving mattresses.

Field	Type	Constraints	Description
id	VARCHAR(191)	PRIMARY KEY	`UUID` identification for each incident.

Field	Type	Constraints	Description
`mattress_id`	VARCHAR(191)	FOREIGN KEY (`mattress.id`)	UUID identification for the mattress involved in the incident.
`found_by`	VARCHAR(191)	FOREIGN KEY (`student.id`)	UUID identification for the student who discovered the misplaced mattress.
`belongs_to`	VARCHAR(191)	FOREIGN KEY (`student.id`)	UUID identification for the student to whom the mattress actually belongs.
`resolved`	BOOLEAN	DEFAULT FALSE	indication of whether the incident has been resolved.
`open_time`	DATETIME	DEFAULT `NOW()`	timestamp for when the incident was reported.
`close_time`	DATETIME	NULLABLE	timestamp for when the incident was resolved.

👉 Constraints

Table	Foreign Key	References	On Delete	On Update
`staff`	`id`	`user(id)`	CASCADE	CASCADE
`student`	`id`	`user(id)`	CASCADE	CASCADE
`student`	`current_room_id`	`room(id)`	RESTRICT	CASCADE
`student`	`next_room_id`	`room(id)`	SET NULL	CASCADE
`room`	`hostel_id`	`hostel(id)`	CASCADE	CASCADE
`belonging`	`student_id`	`student(id)`	RESTRICT	CASCADE
`belonging`	`warehouse_id`	`warehouse(id)`	SET NULL	CASCADE
`luggage`	`id`	`belonging(id)`	CASCADE	CASCADE
`mattress`	`id`	`belonging(id)`	CASCADE	CASCADE
`session`	`staff_id`	`staff(id)`	RESTRICT	CASCADE
`session`	`student_id`	`student(id)`	RESTRICT	CASCADE
`incident`	`mattress_id`	`mattress(id)`	RESTRICT	CASCADE
`incident`	`found_by`	`student(id)`	RESTRICT	CASCADE
`incident`	`belongs_to`	`student(id)`	RESTRICT	CASCADE

🎯 Database Normalization

The following is a concise normalization analysis for each table:

`user`

```
CREATE TABLE `user` (
    `id` VARCHAR(191) NOT NULL,
    `name` VARCHAR(191) NOT NULL,
    `email` VARCHAR(191) NOT NULL,
    `password` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `user_email_key`(`email`),
    PRIMARY KEY (`id`)
)
```

- **1NF**: All attributes atomic
- **2NF**: All non-key attributes depend on full `PRIMARY KEY` (`id`)
- **3NF**: No transitive dependencies

staff

```
CREATE TABLE `staff` (
    `id` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
)
```

- **1NF**: Minimal table structure
- **2NF**: Inherits `PRIMARY KEY` from `user` (no partial dependencies)
- **3NF**: No non-key dependencies

student

```
CREATE TABLE `student` (
    `id` VARCHAR(191) NOT NULL,
    `rollNumber` VARCHAR(12) NOT NULL,
    `current_room_id` VARCHAR(191) NOT NULL,
    `next_room_id` VARCHAR(191) NULL,

    UNIQUE INDEX `student_rollNumber_key`(`rollNumber`),
    UNIQUE INDEX `student_current_room_id_key`(`current_room_id`),
    UNIQUE INDEX `student_next_room_id_key`(`next_room_id`),
    PRIMARY KEY (`id`)
)
```

- **1NF**: Atomic values
- **2NF**: `roll_number` depends on full `PRIMARY KEY`
- **3NF**: `room` references are ID-only (no transitive dependencies)

hostel

```
CREATE TABLE `hostel` (
    `id` VARCHAR(191) NOT NULL,
    `name` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
)
```

- **1NF**: Simple structure
- **2NF**: Single attribute depends on `PRIMARY KEY`
- **3NF**: No non-key dependencies

room

```
CREATE TABLE `room` (
    `id` VARCHAR(191) NOT NULL,
    `number` VARCHAR(6) NOT NULL,
    `hostel_id` VARCHAR(191) NOT NULL,

    UNIQUE INDEX `room_hostel_id_number_key`(`hostel_id`, `number`),
    PRIMARY KEY (`id`)
)
```

- **1NF**: Proper identifiers
- **2NF**: Composite unique key (`hostel_id` + `number`) depends on `PRIMARY KEY`
- **3NF**: No derived attributes

warehouse

```
CREATE TABLE `warehouse` (
    `id` VARCHAR(191) NOT NULL,
    `location` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
)
```

- **1NF**: Basic table
- **2NF**: Single non-key attribute
- **3NF**: No dependencies

belonging

```
CREATE TABLE `belonging` (
    `id` VARCHAR(191) NOT NULL,
    `description` VARCHAR(191) NULL,
    `is_checked_in` BOOLEAN NOT NULL DEFAULT false,
```

```
        `student_id` VARCHAR(191) NOT NULL,
        `warehouse_id` VARCHAR(191) NULL,
        `checked_in_at` DATETIME(3) NULL,
        `checked_out_at` DATETIME(3) NULL,

        PRIMARY KEY (`id`)
    )
```

- **1NF**: Atomic fields
- **2NF**: All attributes depend on `PRIMARY KEY`
- **3NF**: `warehouse` reference is ID-only

luggage/mattress

```
    CREATE TABLE `luggage` (
        `id` VARCHAR(191) NOT NULL,

        PRIMARY KEY (`id`)
    )

    CREATE TABLE `mattress` (
        `id` VARCHAR(191) NOT NULL,

        PRIMARY KEY (`id`)
    )
```

- **1NF**: Inherited `PRIMARY KEY`
- **2NF**: No additional attributes
- **3NF**: Inherited normalization from `belonging`

session

```
    CREATE TABLE `session` (
        `id` VARCHAR(191) NOT NULL,
        `remark` VARCHAR(191) NULL,
        `open_time` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
        `close_time` DATETIME(3) NULL,
        `terminated` BOOLEAN NOT NULL DEFAULT false,
        `staff_id` VARCHAR(191) NOT NULL,
        `student_id` VARCHAR(191) NOT NULL,

        UNIQUE INDEX `session_staff_id_student_id_key`(`staff_id`, `student_id`),
        PRIMARY KEY (`id`)
    )
```

- **1NF**: Proper structure
- **2NF**: All attributes depend on `PRIMARY KEY`

- **3NF**: staff/student references are ID-only

incident

```
CREATE TABLE `incident` (
    `id` VARCHAR(191) NOT NULL,
    `found_by` VARCHAR(191) NOT NULL,
    `belongs_to` VARCHAR(191) NOT NULL,
    `mattress_id` VARCHAR(191) NOT NULL,
    `resolved` BOOLEAN NOT NULL DEFAULT false,
    `open_time` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
    `close_time` DATETIME(3) NULL,

    UNIQUE INDEX `incident_found_by_key`(`found_by`),
    UNIQUE INDEX `incident_belongs_to_key`(`belongs_to`),
    UNIQUE INDEX `incident_mattress_id_key`(`mattress_id`),
    PRIMARY KEY (`id`)
)
```

- **1NF**: Atomic values
- **2NF**: All fields relate to PRIMARY KEY
- **3NF**: mattress/student references are ID-only

🔫 Trigger Deep Dive

belonging_checkin_log

```
CREATE TRIGGER belonging_checkin_log
    BEFORE UPDATE ON belonging
    FOR EACH ROW
        BEGIN
            IF OLD.is_checked_in = FALSE AND NEW.is_checked_in = TRUE THEN
                SET NEW.checked_in_at = NOW();
            END IF;
        END;
```

Logic:

- WHEN: is_checked_in changes false → true
- WHAT: Sets checked_in_at to current timestamp
- WHY: Logging check-in request time

belonging_checkout_log

```
CREATE TRIGGER belonging_checkout_log
    BEFORE UPDATE ON belonging
```

```
    FOR EACH ROW
        BEGIN
            IF OLD.is_checked_in = TRUE AND NEW.is_checked_in = FALSE THEN
                SET NEW.checked_out_at = NOW();
            END IF;
        END;
```

Logic:

- WHEN: `is_checked_in` changes true → false
- WHAT: Sets `checked_out_at` to current timestamp
- WHY: Logging check-out request time

session_log

```
CREATE TRIGGER session_log
        BEFORE UPDATE ON session
        FOR EACH ROW
            BEGIN
                IF OLD.terminated = FALSE AND NEW.terminated = TRUE THEN
                    SET NEW.close_time = NOW();
                END IF;
            END;
```

Logic:

- WHEN: `terminated` changes false → true
- WHAT: Sets `close_time` to current timestamp
- WHY: Logging session close time

incident_log

```
CREATE TRIGGER incident_log
        BEFORE UPDATE ON incident
        FOR EACH ROW
            BEGIN
                IF OLD.resolved = FALSE AND NEW.resolved = TRUE THEN
                    SET NEW.close_time = NOW();
                END IF;
            END;
```

Logic:

- WHEN: `resolved` changes false → true
- WHAT: Sets `close_time` to current timestamp
- WHY: Logging incident close time