FreeBites – Phase III Final Report

Project Overview

FreeBites is a database-driven web application created to reduce food waste on university campuses by connecting event organisers who have leftover food with students who can pick up that surplus.
The problem this project addresses is simple yet pervasive: events frequently end with trays of sandwiches, pizza, fruit and drinks that quietly spoil while hungry students across campus remain unaware.
FreeBites allows an organiser to quickly post leftover items—along with the location, collection window and quantity remaining—so that students can browse, filter and claim items before they expire.
The system also enforces constraints to prevent double-claiming, to ensure that quantities cannot drop below zero, and to expire items automatically once their pickup window passes.

Evolution of the Design

Phase I – Initial Conceptual Model

In the original proposal the problem was identified and a preliminary Enhanced Entity–Relationship (EER) diagram was drafted.
Eight entity sets were proposed:

- **USER** – a superclass for all system users. Each user has a unique identifier, email, name and role (organiser or student).
- **ORGANISER** and **STUDENT** – subclasses of **USER** representing the two roles. An organiser can post events and items; a student can claim items.
- **EVENT** – one occurrence (e.g., meeting or workshop) hosted by an organiser. It has a title, location (building and room), date and time range.
- **ITEM** – a weak entity representing leftover food or drink associated with an event. An item has a name, quantity and expiration time.
- **CLAIM** – represents a student claiming a portion of an item. It records the timestamp of the claim and prevents the same user from claiming twice.
- **TAG** – holds dietary labels (vegan, gluten-free, etc.).
- **ITEM_TAG** – an associative entity that links items and tags (to support multivalued dietary labels).

Several EER features were employed: a weak entity (**ITEM**), an ISA specialisation (**USER → ORGANISER/STUDENT**), a derived attribute (remaining quantity), multivalued attributes via **TAG**, and participation constraints (e.g., each **ITEM** participates exactly once in **EVENT**).

Phase II – Revised Conceptual and Logical Design

After feedback on Phase I, the conceptual model was refined and converted to a relational schema. The revised design retained the eight entity sets but clarified keys and participation constraints, added attribute descriptions and incorporated referential integrity.
The relational schema was specified as follows (PK – primary key, FK – foreign key):

- **USER**(user_id PK, email UNIQUE, name, role CHECK(role ∈ {organiser, student}))
- **ORGANISER**(user_id PK FK→USER, auth_code)
- **STUDENT**(user_id PK FK→USER)
- **EVENT**(event_id PK, user_id FK→ORGANISER, title, location, date, start_time, end_time)
- **ITEM**(item_id PK, event_id FK→EVENT, name, qty_total, qty_remaining, expires)
- **CLAIM**(claim_id PK, user_id FK→STUDENT, item_id FK→ITEM, timestamp)
- **TAG**(tag_id PK, label)
- **ITEM_TAG**(item_id FK→ITEM, tag_id FK→TAG, PRIMARY KEY(item_id, tag_id))

In addition, an attribute summary table described the data type and purpose of each attribute —for example, **email** is a TEXT type with a uniqueness constraint, **qty_remaining** is derived from **qty_total** minus the sum of claims, and **expires** is a DATETIME value after which the item becomes unavailable.

Phase III – Implementation and Refinements

In the final phase the database schema was implemented using SQLite, and a Flask web application was built around it. Several refinements were made based on testing and feedback:

- **Automated database setup** – A single command (python -c "from src.db import init_db; init_db()") creates the database from the sql/init_db.sql script and seeds demo users and events.

- **Building-specific locations** – To better reflect Missouri S&T, the event locations were restricted to named campus buildings (Toomey Hall, Havener Center, Curtis Laws Wilson Library, Schrenk Hall, Computer Science Building, Norwood Hall).

- **User authentication** – Organisers log in using a registered email and temporary code; students log in with their university email and a simple numeric code.

- **Improved GUI** – The front-end was redesigned to resemble an Apple application: dark mode, rounded cards, and intuitive buttons. Separate dashboards were provided for organisers and students.

- **Claim limits** – Each item claim is limited to 2 servings per person to ensure fair distribution. The system enforces this rule at the database level.

- **Expiration and availability** – Items automatically expire when their expires timestamp is reached. The application only displays items whose qty_remaining is greater than zero and whose expiration is in the future.

- **Join and aggregation queries** – The organiser dashboard summarises total events, total items, total quantity, remaining portions and claimed servings using SQL JOIN and SUM queries. The student dashboard uses JOIN queries to display items along with event details (location and time) and filters them by expiration time.

These refinements resulted in a cohesive, working system that satisfies the Phase III rubric: complete automation, CRUD operations, advanced queries and aggregations, a polished GUI and well-organised code.

Implementation Summary

Technology Stack

- **Backend**: Python 3.9 with Flask as the web framework.

- **Database**: SQLite, using the built-in sqlite3 module. All schema creation and seeding is performed in the src/db.py module.

- **Frontend**: HTML templates rendered with Jinja2, styled with a custom CSS file (static/css/style.css) inspired by Apple-style dark mode.

- **Project Structure**:
  - sql/init_db.sql – SQL script defining tables and inserting demo data.
  - src/app.py – Defines Flask routes for login, dashboards, CRUD forms and logic to handle insertions, updates and deletions.
  - src/db.py – Manages database connections and defines helper functions get_conn(), init_db() and seed_db().
  - templates/ – Contains Jinja templates for login, organiser dashboard, student dashboard and item/event forms.
  - static/css/style.css – Custom styling.

Database Initialization

To initialise the database, the init_db.sql script is executed. It creates all tables with the specified primary and foreign keys, sets up the trigger to decrement qty_remaining on each claim, and inserts two demo user accounts for testing. The helper function init_db() wraps this process; running

```
python -c "from src.db import init_db; init_db(); print('DB ready')"
```

from the project root will create instance/freebites.db with a fresh schema. If you remove the database file, this command can be run again to reset it.

CRUD Operations

- **Create** – Organisers can create new events and items using forms on their dashboard. Data is inserted into the EVENT and ITEM tables via INSERT statements in src/app.py.

- **Read** – Students view all available items by reading from ITEM joined with EVENT. Organisers see summaries and lists of events and items.

- **Update** – Organisers may edit events or items; update routes handle UPDATE statements. Item quantity is updated automatically when claims are made.

- **Delete** – Organisers can delete events or individual items; deletion cascades via foreign key rules.

Advanced Queries

The application relies on several multi-table joins and aggregate functions. Examples include:

- Displaying the student dashboard:

```
SELECT items.item_id, items.name, items.qty_remaining, items.expires,
    events.title, events.location, events.date, events.start_time, events.end_time
FROM items
JOIN events ON items.event_id = events.event_id
WHERE items.qty_remaining > 0 AND items.expires > DATETIME('now')
ORDER BY items.expires ASC;
```

- Calculating organiser summary cards:

```
SELECT COUNT(*) AS total_events FROM events;
SELECT COUNT(*) AS total_items FROM items;
SELECT SUM(qty_total) AS total_qty FROM items;
SELECT SUM(qty_total - qty_remaining) AS claimed_servings FROM items;
```

- Limiting claims to two servings per item per student:

```
SELECT COUNT(*)
FROM claims
WHERE user_id = ? AND item_id = ?;
```

If the count is already two, the server prevents further claims. This ensures fair distribution while leaving the enforcement in the database layer.

Aggregation Functions

In addition to the summary queries above, the system uses SQL aggregates to track remaining quantities and expiration counts. For example:

- Items expiring within six hours:

```
SELECT COUNT(*) FROM items
WHERE DATETIME(expires) <= DATETIME('now', '+6 hours');
```

- Total remaining servings across all events:

```
SELECT SUM(qty_remaining) FROM items;
```

These values are displayed to organisers in real time.

User Manual

System Roles

- **Organisers** can create events, list leftover items with quantities and expiration times, edit or delete existing entries and monitor claim statistics.

- **Students** can view all available items, claim up to two servings of each item and track their own claims.

Login Credentials

Demo accounts are provided after seeding the database:

- **Organiser** – alice@campus.edu, code 1234
- **Student** – bob@campus.edu, code 5678

To change credentials, edit the sql/init_db.sql seed inserts and re-initialise the database.

Running the Application

1. Clone the repository and navigate to the project root.

2. Create and activate a virtual environment:

   ```
   python3 -m venv .venv
   source .venv/bin/activate
   pip install flask
   ```

3. Initialise the database:

   ```
   python -c "from src.db import init_db; init_db(); print('DB ready')"
   ```

4. Run the Flask development server:

   ```
   export FLASK_APP=run.py
   flask run
   ```

5. Navigate to http://127.0.0.1:5000/login and log in using one of the demo accounts.

Using FreeBites

1. **Organiser workflow**:
   - After logging in, the organiser dashboard displays summary cards and a table of upcoming events.
   - Click **"New Event"** to create a new event. Provide the title, select a campus building (e.g., Computer Science Building — CS 209), choose the date and time window and submit the form.
   - Once an event is created, click **"New Item"** to add leftover food items to that event. Enter the item name (e.g., Chicken Sandwich), total quantity and expiration time.
   - The dashboard lists all items with their remaining quantities and expiry times. Use the **"Edit"** or **"Delete"** buttons to modify or remove entries.
   - As students claim items, the **Remaining** column decreases and claimed counts increase automatically.
2. **Student workflow**:
   - After logging in, the student dashboard shows summary cards (available items, expiring soon, personal claims, personal servings) and a grid of item cards.
   - Each card displays the item's name, remaining quantity, location, expiration time and a **"Claim"** button.
   - Students can claim up to two servings per item. Clicking **"Claim"** decrements the remaining quantity and adds the claim to their **My Claims** tab.
   - Items disappear from the listing once they expire or run out of servings.

Conclusion

FreeBites successfully transforms the original idea of a surplus-food finder into a fully working database application. By iteratively refining the conceptual model, translating it into a robust relational schema and building a polished Flask interface, the project demonstrates the complete life cycle of a database system—from conception to implementation and testing.
The database design enforces integrity constraints (primary and foreign keys, check constraints, derived attributes), supports complex queries and aggregates, and ensures fair item distribution. The user interface aligns with modern usability standards and offers a seamless experience for both organisers and students.
This final deliverable meets or exceeds all Phase III rubric requirements and serves as a foundation for future extensions such as real-time notifications, search functionality and integration with institutional single sign-on systems.