CIS 3530: Database Programming Project

Company Portal Web Application

Stack: Python (Flask), PostgreSQL, `psycopg`
Database: `company_v3.02.sql` (as provided)

1. Purpose

The goal of this project is to build a small, data-driven web application using Flask and raw SQL. You will demonstrate core database programming skills: schema understanding, writing complex multi-table queries (joins, aggregations), implementing safe CRUD (Create, Read, Update, Delete) operations, ensuring data integrity, and handling database constraints gracefully within a web application.

Marking

The project is worth 12 out of 15 marks, with an option to earn up to 4 more marks as a bonus. Details on how to earn bonus marks are provided at the end.

Group Member Contributions: Completing all the required tasks perfectly and ensuring that all group members can demonstrate and modify the project during the evaluation session are necessary to award full marks to the entire group.

2. Setup & Submission

Your project must be reproducible.

1. **`team_setup.sql`:** Include a single SQL script named `team_setup.sql`. This file must contain all SQL objects you add, including the `app_user` table, any views, and your two required indexes.
2. **`README.md`:** Include a `README.md` with:
   - Exact, step-by-step commands to set up and run your project (see example).
   - A section justifying your two chosen indexes (which page/query they help).
3. **Example `README.md` Run Steps:**

```
# 1. Setup environment (example)
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# 2. Setup database (example)
createdb my_company_db
export DATABASE_URL="postgresql://user:pass@localhost/my_company_db"

# 3. Load schema and your additions
% --> RECOMMENDATION 1 (continued): Update file version here too
psql -d $DATABASE_URL -f company_v3.02.sql
psql -d $DATABASE_URL -f team_setup.sql

# 4. Run the app
flask run
```

3. Technical Requirements (Hard Rules)

1. **No Base Schema Changes:** You may not alter or drop the existing base tables, columns, or constraints from the `company_v3.02.sql` file.
2. **Embedded SQL Only:** All database access must use the `psycopg` (or `psycopg2`) library. No ORMs or query builders.
3. **SQL Injection Prevention:** All dynamic values in SQL queries must use parameterized queries (e.g., `cursor.execute("... WHERE name = %s", (user_name,))`).
4. **Safe Sorting (Whitelist):** Whitelist any dynamic `ORDER BY` keys before inserting them into SQL.
5. **Immutable Primary Keys:** Users must not be able to edit an employee's `Ssn`.
6. **User-Friendly Errors:** Catch exceptions, log details server-side, and show friendly messages to users.
7. **Write Your Own Queries:** You must not query the pre-existing views (`AllProjectsWithHeadcount`, `CurrentEmployeeAssignments`) from the setup file. Your application must generate all aggregates and joins using its own embedded SQL.

4. Required Pages and Features (A1–A6)

A1. Authentication

Implement a full authentication system. All other pages must be protected and require a user to be logged in.

- **Database:** Create an `app_user` table in `team_setup.sql`. It must store at least a `username` and a `password_hash`. Authentication table example to add a minimal user info for the app:

```
CREATE TABLE app_user (
  id SERIAL PRIMARY KEY,
  username TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role TEXT NOT NULL CHECK (role IN ('admin','viewer'))
);
```

  Note: RBAC is optional (see Bonus). If not implemented, you may store a fixed role or ignore it.

- **Features:**
  - A login page that validates a user.
  - Passwords must be securely hashed (e.g., using `werkzeug.security`) and stored in `password_hash`.
  - A logout feature that clears the user's session.

A2. Home – Employee Overview

Display a searchable, sortable list of all employees.

- **Required Columns (must show all 5):**
  1. Employee Full Name (e.g., "John B. Smith").
  2. Department Name (from `Department`).

3. Number of Dependents (from `Dependent`).
4. Number of Projects (from `Works_On`).
5. Total Hours (from `Works_On`).

- **Key Requirement:** Employees with 0 dependents or 0 project assignments must still be listed, showing 0 for counts/sums (use `LEFT JOIN` and `COALESCE`).
- **Filters:** Department dropdown; case-insensitive partial match on employee name.
- **Sorting (Whitelist):** Allow sorting by `name` (ASC/DESC) or `total_hours` (ASC/DESC).

A3. Projects – Portfolio Summary

Display a sortable list of all projects.

- **Required Columns (must show all 4):**
  1. Project Name.
  2. Owning Department Name (from `Department`).
  3. Headcount (COUNT of distinct employees).
  4. Total Assigned Hours (SUM of hours).
- **Key Requirement:** Projects with 0 employees must still be listed with 0 headcount and 0 total hours.
- **Links:** Each project name links to its Project Details page (A4).
- **Sorting (Whitelist):** Allow sorting by `headcount` or `total_hours` (ASC/DESC).

A4. Project Details & Assignment "Upsert"

Show details for a single project (e.g., `/project/10`). Two parts:

- **Part 1: Display:** List all employees on this project with Full Name and Hours.
- **Part 2: Upsert Form:**
  - Inputs: Employee (dropdown of all employees), Hours (number).
  - **Logic:** If the employee is already on the project, the query must **add** the new hours to their existing total. If they are not yet on the project, it should insert them with the new hours.
  - Upsert must be atomic via `INSERT ... ON CONFLICT`:

```
-- Example pattern for adding hours
INSERT INTO Works_On (Essn, Pno, Hours)
VALUES (%s, %s, %s)
ON CONFLICT (Essn, Pno)
DO UPDATE SET Hours = Works_On.Hours + EXCLUDED.Hours;
```

A5. Employee Management (CRUD)

Provide CRUD for employees.

- **Add Employee:** Create a new employee. `Ssn` must be unique.

- **Edit Employee:** Allow updates to Address, Salary, and Department (`Dno`). Show `Ssn` but do not allow edits.
- **Delete Employee:**
  - The schema uses `ON DELETE RESTRICT`. If the employee has `Works_On` or `Dependent` rows, (or is a manager/supervisor) the DB will block deletion.
  - Catch this error and show a friendly message.
  - e.g., "Cannot delete employee: They are still assigned to projects, have dependents listed, or are a manager/supervisor."

A6. Managers Overview

Display a high-level overview of all departments.

- **Required Columns (must show all 4):**
  1. Department Name and Number.
  2. Manager's Full Name (from `Employee`).
  3. Department Employee Count (COUNT of employees in that department).
  4. Department Total Hours (SUM of hours worked on projects by all employees in that department).
- **Key Requirement:** Departments with 0 employees or 0 total hours must still be listed with 0s.
- If a department has no manager (`Mgr_ssn` is NULL), the manager's name should be displayed as 'N/A' or 'None'.

5. Indexes

Create at least two sensible indexes in `team_setup.sql`. In `README.md`, justify them by pointing to specific queries/pages. Sensible defaults if you are unsure:

- `CREATE INDEX idx_employee_name ON Employee (Lname, Fname);` speeds up name search on Home (A2).
- `CREATE INDEX idx_workson_pno ON Works_On (Pno);` speeds up project aggregates on Projects (A3) and Details (A4).

6. Bonus Features (up to +4 Marks)

The following optional features can earn up to four additional marks. These directly correspond to the bonus rows in the marking rubric.

- **Role Based Access Control (RBAC) (+2.0):** Implement page-level permissions so that:
  - Admin-only pages display Add/Edit/Delete options.
  - Viewer users see read-only versions of all pages.
  - Unauthenticated users are redirected to the login page and cannot access any protected routes.

- **CSV Export (+1.0):** Provide an "Export" button (on the Home or Projects page) that downloads the *currently filtered and sorted* list as a `.csv` file.
- **Excel Import (+1.0):** Implement a validated `.xlsx` upload feature that inserts new rows into a chosen table. Invalid or inconsistent rows must be rejected gracefully with clear error messages (no partial apply).

7. Marking Rubric (Total: 12 Base + 4 Bonus)

Base (12 Marks)	**What the TA checks**	**Pts**
Setup	`README.md` has clear run steps. `team_setup.sql` runs and creates the `app_user` table and 2 justified indexes.	1.0
SQL Quality	Code uses `psycopg` with parameterized queries for all user inputs. No ORM. No string concatenation for values. No use of pre-made views.	1.0
Auth (A1)	Login page works. Passwords are hashed. Session persists. Logout works. All pages (except login) are protected.	2.0
Home (A2)	Shows all 5 columns. Employees with 0 projects/dependents appear with 0. Department filter works. Name search works. Safe sorting by name/hours works.	2.0
Projects (A3)	Shows all 4 columns. Projects with 0 staff appear with 0. Links to details page work. Safe sorting by headcount/hours works.	1.0
Details & Upsert (A4)	Details page lists correct staff for that project. Upsert adds hours for existing staff; inserts row for new staff. Uses `ON CONFLICT`.	2.0
Employee CRUD (A5)	Add/Edit forms work. `Ssn` is not editable. Delete fails gracefully with a friendly message when blocked by FKs.	2.0
Managers (A6)	Shows all 4 columns. Departments with 0 employees or 0 hours appear with 0. Manager name is correct (handles `NULL`).	1.0
Base total		**12.0**

Bonus (up to +4)	**What the TA checks**	**Pts**
Role Based Access Control	Admin-only pages show Add/Edit/Delete; viewer sees read-only; unauthenticated users see only login	+2.0
CSV Export	"Export" downloads the currently filtered list as a `.csv`.	+1.0
Excel Import	Validated upload inserts rows; rejects bad inputs with clear errors (no partial apply)	+1.0

Partial credit policy: Partial work will receive partial marks. However, each partial implementation must complete a task fully. For example, if only authentication is implemented, then the user table, login, and logout must work together independently.