Fixture Dashboard Database Schema

The **Fixture Dashboard** manages GPU test fixtures, their health, alarms, test results, and related workflows. This schema ensures all entities are properly connected, so engineers and operators can track hardware, tests, and issues in a reliable and scalable way.

This script creates all required tables for the Fixture Dashboard system.
The tables are ordered in a **dependency-safe sequence** (parents first, children later), so you can run the entire script **from top to bottom without encountering foreign key errors**.

Why we need this schema

- **Users** → Defines who interacts with the system (operators, engineers, super users).
- **Fixture Health, Alarms, and Events** → Monitor real-time status and raise alerts when issues occur.
- **Tickets & Comments** → Manage troubleshooting and collaboration between teams.

This schema is the backbone of the Fixture Dashboard — without it, the application would not be able to enforce relationships, preserve history, or provide meaningful insights.
This schema represents the **base model** of the Fixture Dashboard database.
As the system grows, you may need to **alter tables for optimization, performance improvements, or new requirements**. Examples include adding indexes, refining constraints, or extending tables with additional columns. The current design ensures a strong foundation that can be safely built upon.

Future Enhancements

While this schema provides a solid starting point, future improvements may include:

- **Indexing strategies** for faster query performance on frequently accessed columns.
- **Partitioning or sharding** for large-scale test logs or event data.
- **Materialized views** to support reporting and analytics.
- **Additional relationships** if new hardware components or workflow stages are introduced.
- **Audit logging and versioning** for compliance and traceability.

Instructions

1. **Connect to your PostgreSQL database**

    - Use `psql`, pgAdmin, or DBeaver.
    - Ensure you're connected to the right database (or create one first with `CREATE DATABASE fixture_dashboard;`).

2. **Prepare your environment**
    - Confirm your user has privileges to create tables and sequences.
    - If re-running the script, reset the schema:

      ```
      DROP SCHEMA public CASCADE;
      CREATE SCHEMA public;
      ```
3. **Run the script top-to-bottom**
    - Copy and paste the entire script.
    - Tables are already ordered **dependency-safe** (parents first, children later), so all foreign keys resolve correctly.
    - No need to run one table at a time.

4. **Verify the schema**
    - List all tables:

      ```
      \dt
      ```
    - Inspect details of a table:

      ```
      \d+ table_name
      ```
5. **Next steps**
    - Seed base data: insert into `users`, `fixtures`, `racks` first.
    - Then add dependent records like `fixture_parts`, `gpus`, `fixture_events`, etc.
    - Stored procedures (if provided) should be used for inserts to enforce consistency.

Result:
After running this script, you will achieve:

- A complete schema for your **Fixture Dashboard system**.
- All **foreign keys are resolved automatically** — no dependency errors.
- A **normalized structure** that avoids redundant data and enforces consistency.
- A schema that supports **future expansion** — new fixtures, GPUs, stations, tickets, and logs can be added seamlessly.

User Table:

```
-- 1. Users
-- Stores system users. Tracks their role (super_user, operator, engineer)
```

```
-- and is referenced by many other tables (fixtures, tickets, comments, etc.).
CREATE TABLE users (
   user_id SERIAL PRIMARY KEY,
   username VARCHAR(100) NOT NULL UNIQUE,
   role VARCHAR(50) NOT NULL CHECK (role IN ('super_user','operator','engineer')),
   created_at TIMESTAMPTZ DEFAULT now(),
   updated_at TIMESTAMPTZ DEFAULT now()
);
```
Insert Data:
```
INSERT INTO users (username, role) VALUES
('Thay', 'super_user'),
('bob', 'operator'),
('charlie', 'engineer');
```

Fixtures Table:
```
-- 2. Fixtures
-- Represents each test fixture in the lab (e.g., refurbish, sort).
-- Tracks its networking details, who created it, and its rack.
CREATE TABLE fixtures (
   fixture_id SERIAL PRIMARY KEY,
   fixture_name VARCHAR(100) NOT NULL UNIQUE,
   fixture_type VARCHAR(50) NOT NULL CHECK (fixture_type IN ('refurbish','sort')),
   ip_address INET NOT NULL,
   mac_address VARCHAR(50) NOT NULL,
   created_by INT REFERENCES users(user_id),
   created_at TIMESTAMPTZ DEFAULT now(),
   updated_at TIMESTAMPTZ DEFAULT now()
);
```
Insert Data:
```
INSERT INTO fixtures (fixture_name, fixture_type, ip_address, mac_address, created_by)
VALUES
('NV-NCT20 -02', 'refurbish', '192.168.1.10', 'AA:BB:CC:DD:EE:01', 1),
('NV-NCT19 -03', 'sort', '192.168.1.11', 'AA:BB:CC:DD:EE:02', 2);
```

Fixture Usage Table:
```
– 3.Usage Table
–The fixture_usage table records each test or activity performed on a specific fixture (a hardware
setup used to test GPUs).
CREATE TABLE fixture_usage (
    usage_id SERIAL PRIMARY KEY,
    fixture_id INT NOT NULL,
    FOREIGN KEY (fixture_id) REFERENCES fixtures(fixture_id) ON DELETE CASCADE,
    test_slot VARCHAR(32) CHECK (test_slot IN ('Left', 'Right')),
    test_station VARCHAR(256),
    test_type VARCHAR(32) CHECK (test_type IN ('Refurbish', 'Sort', 'NA')),
    gpu_part_number VARCHAR(32),
    gpu_serial_number VARCHAR(32),
    log_path VARCHAR(256),
    created_by VARCHAR(64),
    created_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

Insert Data:

```
INSERT INTO fixture_usage (fixture_id, test_slot, test_station, test_type, gpu_part_number,
gpu_serial_number, log_path, created_by
) VALUES
(1, 'Left', 'Station-A', 'Refurbish', 'PN-1040', 'SN-56789', '/logs/2025-10-13/test_001.txt',
'emilyr'),
(2, 'Right', 'Station-B', 'Sort', 'PN-2045', 'SN-12345', '/logs/2025-10-13/test_002.txt', 'emilyr'),
(1, 'Right', 'Station-A', 'NA', 'PN-3040', 'SN-67890', '/logs/2025-10-13/test_003.txt', 'johns'),
(3, 'Left', 'Station-C', 'Refurbish', 'PN-4050', 'SN-54321', '/logs/2025-10-13/test_004.txt',
'emilyr');
```

Fixture Alarms Table:

```
-- 4. Fixture Alarms
-- Tracks alarms raised by a fixture (server_down, freeze, hardware failure).
-- Includes severity, status, and who acknowledged/resolved the alarm.
CREATE TABLE fixture_alarms (
    alarm_id SERIAL PRIMARY KEY,
    fixture_id INT NOT NULL REFERENCES fixtures(fixture_id) ON DELETE CASCADE,
    alarm_type VARCHAR(50) NOT NULL CHECK (alarm_type IN
('server_down','freeze','hardware_failure','network_issue')),
    severity VARCHAR(20) NOT NULL CHECK (severity IN ('info','warning','critical')),
    status VARCHAR(20) NOT NULL CHECK (status IN ('active','acknowledged','resolved')),
    message TEXT,
    triggered_at TIMESTAMPTZ DEFAULT now(),
    acknowledged_by INT REFERENCES users(user_id),
    acknowledged_at TIMESTAMPTZ,
    resolved_at TIMESTAMPTZ
);
```

Insert Data:

```
INSERT INTO fixture_alarms (fixture_id, alarm_type, severity, status, message)
VALUES
(1, 'server_down', 'critical', 'active', 'NV-NCT19 - 03 is offline'),
(2, 'hardware_failure', 'warning', 'acknowledged', 'Need riser card replaced');
```

Fixture Events Table:

```
-- 5. Fixture Events
-- Generic event log for fixtures (e.g., alarm triggered, resolved, status update).
-- Stores flexible event details as JSON.
CREATE TABLE fixture_events (
    event_id SERIAL PRIMARY KEY,
    fixture_id INT NOT NULL REFERENCES fixtures(fixture_id) ON DELETE CASCADE,
    event_type VARCHAR(50) NOT NULL CHECK (event_type IN
('alarm_triggered','alarm_resolved','status_update','maintenance')),
    details JSONB,
    created_at TIMESTAMPTZ DEFAULT now()
);
```

Insert Data:

```
INSERT INTO fixture_events (fixture_id, event_type, details) VALUES
(1, 'alarm_triggered', '{"alarm":"server_down"}'),
(1, 'status_update', '{"status":"rebooted"}'),
(2, 'maintenance', '{"action":"riser card replaced"}');
```

Fixture Ticket Table:

```
-- 6. Fixture Tickets
-- Ticketing system for fixtures (e.g., "Replace riser card").
-- Linked to fixtures, users, and optionally alarms.
-- Tracks assignment, priority, due dates, and status.
CREATE TABLE fixture_tickets (
    ticket_id SERIAL PRIMARY KEY,
    fixture_id INT NOT NULL REFERENCES fixtures(fixture_id) ON DELETE CASCADE,
    created_by INT NOT NULL REFERENCES users(user_id),
    assigned_to INT REFERENCES users(user_id),
    alarm_id INT REFERENCES fixture_alarms(alarm_id),
    title VARCHAR(200) NOT NULL,
    description TEXT,
    priority VARCHAR(20) CHECK (priority IN ('low','medium','high','urgent')) DEFAULT
'medium',
    status VARCHAR(20) CHECK (status IN ('open','in_progress','closed')) DEFAULT 'open',
    due_date DATE,
    created_at TIMESTAMPTZ DEFAULT now(),
    updated_at TIMESTAMPTZ DEFAULT now(),
    closed_at TIMESTAMPTZ
);
```

Insert Data:

```
INSERT INTO fixture_tickets (fixture_id, created_by, assigned_to, title, description, priority,
status, due_date)
VALUES
(1, 1, 2, 'Replace riser card', 'Riser card failure detected in NV-NCT20- 01', 'high', 'open',
'2025-10-15'),
(2, 2, 3, 'Check GPU seating', 'Possible loose connection in NV-NCT19-04', 'medium',
'in_progress', '2025-10-20');
```

Ticket Comments Table:

```
-- 7. Ticket Comments
-- Stores threaded comments on fixture tickets.
-- Each comment is linked to the ticket and the user who wrote it.
CREATE TABLE ticket_comments (
    comment_id SERIAL PRIMARY KEY,
    ticket_id INT NOT NULL REFERENCES fixture_tickets(ticket_id) ON DELETE
CASCADE,
    user_id INT NOT NULL REFERENCES users(user_id),
    comment TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT now()
);
```

Insert Data:

```
INSERT INTO ticket_comments (ticket_id, user_id, comment) VALUES
(1, 2, 'Working on riser card issue'),
(1, 3, 'Replacement part ordered'),
(2, 1, 'Check scheduled for tomorrow');
```

Fixture Health Table:

```
-- 8. Fixture Health
-- Tracks high-level operational health of fixtures
-- (active, non_active, partial_active, need_maintenance).
CREATE TABLE fixture_health (
   health_id SERIAL PRIMARY KEY,
   fixture_id INT NOT NULL REFERENCES fixtures(fixture_id) ON DELETE CASCADE,
   operational_status VARCHAR(50) CHECK (operational_status IN
('active','non_active','partial_active','need_maintenance')) DEFAULT 'active',
   last_updated TIMESTAMPTZ DEFAULT now()
);
```

Insert Data:

```
INSERT INTO fixture_health (fixture_id, operational_status) VALUES
(1, 'active'),
(2, 'need_maintenance');
```

—---

Backend Set up - Instruction using Express JS and Node JS:

🔹 Step 1: Create a New Project

- Make a new folder for your backend:

```
 mkdir fixture-backend
cd fixture-backend
```

- Initialize Node.js project:

```
 npm init -y
```

🔹 Step 2: Install Required Packages

- We'll need Express and PostgreSQL client (pg):

```
npm install express pg cors dotenv
npm install --save-dev nodemon
```

- **express** → web framework
- **pg** → PostgreSQL client
- **cors** → allows frontend (React) to call backend
- **dotenv** → load environment variables (DB credentials, etc.)
- **nodemon** → auto-restarts server on file changes

🔹 Step 3: Project Structure

- Your backend folder should look like this:

```
fixture-backend/
|── node_modules/
|── server.js
|── db.js
|── .env
|── package.json
```

- Create the `.env` File
  1. Make sure you are in your **project root folder** (`dashboard-backend`) in terminal or file explorer.
  2. Create a new file called `.env`:

     Windows:
     Open Command Prompt in your project folder and type:

     ```
     type nul > .env
     ```

     Or, in File Explorer, right-click → New → Text Document → rename to `.env` (make sure there's no `.txt` at the end).

Mac/Linux:
Open terminal in project folder:

```
touch .env
```

- Open .env in a text editor (VS Code, Notepad, etc.) and add your PostgreSQL database info and server port like this:

```
DB_USER=postgres
DB_HOST=localhost
DB_NAME=dashboard_db
DB_PASSWORD=yourpassword
DB_PORT=5432
PORT=5000
```

- **DB_USER** → PostgreSQL username (default is usually postgres)
- **DB_HOST** → usually localhost if running locally
- **DB_NAME** → your database name (dashboard_db)
- **DB_PASSWORD** → your PostgreSQL password
- **DB_PORT** → default is 5432
- **PORT** → port your Express server will run on (5000 is common)

Important: Do NOT include .env in version control (Git) because it contains sensitive info.

🔹 Step 4: Setup Database Connection (`db.js`)

- Create a file called `db.js`:

```
const { Pool } = require('pg');
require('dotenv').config();
```

```
const pool = new Pool({
     user: process.env.DB_USER,
      host: process.env.DB_HOST,
      database: process.env.DB_NAME,
      password: process.env.DB_PASSWORD,
      port: process.env.DB_PORT,
});

module.exports = pool;
```

🔹 Step 5: Setup Express Server (`server.js`)

- Create `server.js`: (Copy and paste following, save & close the file)

```
const express = require('express');
const cors = require('cors');
const pool = require('./db');
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

// Test route
app.get('/', (req, res) => {
  res.send('Fixture Backend API is running');
});

// Example: Get all fixtures
app.get('/fixtures', async (req, res) => {
  try {
    const result = await pool.query('SELECT * FROM
fixtures');
    res.json(result.rows);
  } catch (err) {
    console.error(err.message);
    res.status(500).send('Server error');
  }
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on
http://localhost:${PORT}`);
});
```

🔹 Step 7: Update `package.json` for `nodemon`

- In `package.json`, add this to `scripts`:

  ```
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  }
  ```

- Now run:

  ```
  npm run dev
  ```

✅ Your backend should now be running on:
👉 http://localhost:5000

- Try hitting http://localhost:5000/ → should say *Fixture Backend API is running*
- Try hitting http://localhost:5000/fixtures → should return data from your `fixtures` table.

Local Setup Guide: React + Backend (fixtures.js)

This guide assumes:

- Backend routes are ready (Express + PostgreSQL).
- `fixtures.js` is part of your backend routes or a script.
- React frontend will connect to the backend on your local machine.

⚙️ 1. Prerequisites

✅ Common for Both (Mac & Windows)

Tool	Description	Verify
Node.js	JavaScript runtime	`node -v`
npm	Node package manager	`npm -v`
VS Code	Recommended editor	—

🖥️ Mac Users

```
# Install Homebrew if not installed
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.
sh)"

# Install Node.js
brew install node
```

🪟 Windows Users

1. Download **Node.js LTS** from https://nodejs.org
2. Check **Add to PATH** in installer.
3. Restart terminal (PowerShell / Git Bash).
4. Verify:

```
node -v
npm -v
```

📁 2. Install Backend Dependencies

Navigate to your backend folder (where `server.js` and `fixtures.js` are located):

```
cd path/to/backend
npm install
```

⚙️ 3. Run Backend Server (with fixtures.js)

If `fixtures.js` is **imported in your routes**, simply run the backend:

```
npm start
```

or, if using **nodemon**:

```
npx nodemon server.js
```

If `fixtures.js` is a **standalone script** (e.g., inserts test data), run:

```
node fixtures.js
```

Test it by opening your browser or terminal:

```
http://localhost:5000/api/fixtures
```

⚛️ 4. Set Up React Frontend Locally

If you don't have a React app yet:

Using Create React App

```
npx create-react-app client
cd client
```

Or Using Vite (faster)

```
npm create vite@latest client
# Select React → JavaScript or TypeScript
cd client
npm install
```

▶️ 5. Run React Development Server

```
npm run start       # CRA
# OR
npm run dev         # Vite
```

Default local ports:

- React: `http://localhost:3000` (CRA) or `http://localhost:5173` (Vite)
- Backend: `http://localhost:5000`

🔗 6. Connect React to Backend

Example API call in React (`src/App.js`):

```
useEffect(() => {
  fetch('http://localhost:5000/api/fixtures')
    .then(res => res.json())
    .then(data => setFixtures(data))
    .catch(err => console.error(err));
}, []);
```

🛡️ 7. Handle CORS in Backend

If frontend and backend run on different ports, add to `server.js`:

```
const cors = require('cors');
app.use(cors({ origin: 'http://localhost:3000' })); // adjust
port if using Vite
```

⚙️ 8. Environment Variables (Optional)

Create `client/.env`:

```
REACT_APP_API_URL=http://localhost:5000
```

Use in React:

```
fetch(`${process.env.REACT_APP_API_URL}/api/fixtures`)
```

🚀 9. Build Frontend (for Production)

```
cd client
```

```
npm run build
```

Then serve with backend:

```
const path = require('path');
app.use(express.static(path.join(__dirname, 'client/build')));

app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, 'client/build/index.html'));
});
```

✅ Quick Command Summary (Local Setup)

Task	Command
Install backend deps	`cd backend && npm install`
Run backend	`npm start` or `npx nodemon server.js`
Run fixtures.js (standalone)	`node fixtures.js`
Install frontend deps	`cd client && npm install`
Start frontend	`npm run start` (CRA) or `npm run dev` (Vite)
Build frontend	`npm run build`