**Database Schema Design**

Contents

[1. Users & Authentication 2](#_Toc195030578)

[1.1 Users Table 2](#_Toc195030579)

[1.2 Api Tokens Table 2](#_Toc195030580)

[2. Devices & Sensors 3](#_Toc195030581)

[2.1 Devices Table 3](#_Toc195030582)

[2.2 Sensor Streams Table 4](#_Toc195030583)

[3. Time-Series Data 5](#_Toc195030584)

[3.1 Sensor Data Table 5](#_Toc195030585)

[4. Analytics Results 6](#_Toc195030586)

[4.1 Correlation Table 6](#_Toc195030587)

[4.2 Anomalies Table 6](#_Toc195030588)

[5. File Management 7](#_Toc195030589)

[5.1 Uploaded Files Table 7](#_Toc195030590)

[6. Chat System 8](#_Toc195030591)

[6.1 Chat Sessions Table 8](#_Toc195030592)

[6.2 Chat Messages 9](#_Toc195030593)

# 1. Users & Authentication

## 1.1 Users Table

The users table serves as the central repository for user account information and authentication credentials, utilizing a UUID user\_id as its primary key. This table connects to multiple other tables through one-to-many relationships, including devices (tracking owned IoT hardware), api\_tokens (managing authentication tokens), uploaded\_files (recording user file transfers), and chat\_sessions (storing support interactions). As the core identity management component, it enforces security through encrypted password storage and multi-factor authentication fields while maintaining audit trails via timestamps for critical events like account creation and last login.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| user\_id | UUID | Unique identifier for the user | PRIMARY KEY |
| email | VARCHAR(255) | User's email address | UNIQUE, NOT NULL |
| password\_hash | VARCHAR(255) | Encrypted password | bcrypt hashing, NOT NULL |
| mfa\_secret | VARCHAR(32) | Secret key for multi-factor authentication (MFA) | NULL if MFA not enabled |
| last\_login | TIMESTAMP | Timestamp of the last successful login | - |
| failed\_attempts | INT | Count of consecutive failed login attempts | DEFAULT 0 |
| account\_locked | BOOLEAN | Indicates if the account is locked | DEFAULT FALSE |
| created\_at | TIMESTAMP | Account creation timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- User accounts and authentication credentials  CREATE TABLE users (  user\_id UUID PRIMARY KEY,  email VARCHAR(255) UNIQUE NOT NULL,  password\_hash VARCHAR(255) NOT NULL, -- BCrypt encrypted password  mfa\_secret VARCHAR(32), -- TOTP secret for 2FA  last\_login TIMESTAMP, -- Timestamp of last successful login  failed\_attempts INT DEFAULT 0, -- Consecutive failed login attempts  account\_locked BOOLEAN DEFAULT FALSE, -- Account lock status  created\_at TIMESTAMP DEFAULT NOW() -- Account creation time  ); |

## 1.2 Api Tokens Table

The api\_tokens table manages JSON Web Token authentication for API access, with token\_id as its primary key and a foreign key user\_id linking to the users table. It stores hashed tokens with scoped permissions and expiration dates, enabling secure service-to-service communication while allowing token revocation through deletion. This design directly supports the system's secure login requirements while providing granular access control for different API endpoints.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| token\_id | UUID | Unique token identifier | PRIMARY KEY |
| user\_id | UUID | Associated user | FOREIGN KEY (users.user\_id), ON DELETE CASCADE |
| token\_hash | VARCHAR(512) | Hashed JWT token value | NOT NULL |
| scopes | TEXT[] | List of permissions (e.g., ["read:data", "write:data"]) | - |
| expires\_at | TIMESTAMP | Token expiration timestamp | NOT NULL |
| created\_at | TIMESTAMP | Token creation timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- API access tokens for authenticated requests  CREATE TABLE api\_tokens (  token\_id UUID PRIMARY KEY,  user\_id UUID REFERENCES users(user\_id) ON DELETE CASCADE,  token\_hash VARCHAR(512) NOT NULL, -- Hashed JWT token  scopes TEXT[] NOT NULL, -- Array of permission scopes  expires\_at TIMESTAMP NOT NULL, -- Token expiration time  created\_at TIMESTAMP DEFAULT NOW(), -- Token issuance time  CONSTRAINT valid\_scopes CHECK (scopes <> '{}')  ); |

# 2. Devices & Sensors

## 2.1 Devices Table

The devices table registers IoT hardware with a UUID device\_id primary key and a user\_id foreign key connecting to its owner in the users table. It captures critical metadata including device type (temperature/pressure/motion sensors), geographic location, and registration timestamp. As the foundational layer for data source management, it enables device-specific analysis and permissions control while supporting one-to-many relationships with sensor\_streams that describe individual measurement channels from each device.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| device\_id | UUID | Unique device identifier | PRIMARY KEY |
| user\_id | UUID | Owner of the device | FOREIGN KEY (users.user\_id), ON DELETE CASCADE |
| name | VARCHAR(100) | Human-readable device name | NOT NULL |
| type | VARCHAR(50) | Device type | CHECK: temperature, pressure, motion |
| location | GEOGRAPHY | Geographic coordinates (optional) | POINT geometry |
| registered\_at | TIMESTAMP | Device registration timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- Registered IoT devices  CREATE TABLE devices (  device\_id UUID PRIMARY KEY,  user\_id UUID REFERENCES users(user\_id) ON DELETE CASCADE,  name VARCHAR(100) NOT NULL, -- Human-readable device name  type VARCHAR(50) NOT NULL CHECK (type IN ('temperature', 'pressure', 'motion')),  location GEOGRAPHY(POINT, 4326), -- GPS coordinates (WGS84)  registered\_at TIMESTAMP DEFAULT NOW() -- Device registration time  ); |

## 2.2 Sensor Streams Table

The sensor\_streams table defines individual data collection channels from IoT devices, using stream\_id as its primary key and a device\_id foreign key linking to its parent hardware. It stores configuration details like metric names and sampling rates, serving as the organizational backbone for time-series data in sensor\_data while maintaining relationships with both correlation analyses and anomaly detection results. This structure allows flexible management of heterogeneous sensor types under unified data models.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| stream\_id | UUID | Unique sensor data stream identifier | PRIMARY KEY |
| device\_id | UUID | Associated device | FOREIGN KEY (devices.device\_id), ON DELETE CASCADE |
| metric\_name | VARCHAR(50) | Name of the measured metric (e.g., "cpu\_temp") | NOT NULL |
| sampling\_rate | INT | Data collection frequency in seconds | NOT NULL, >0 |
| created\_at | TIMESTAMP | Stream creation timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- Individual sensor data streams  CREATE TABLE sensor\_streams (  stream\_id UUID PRIMARY KEY,  device\_id UUID REFERENCES devices(device\_id) ON DELETE CASCADE,  metric\_name VARCHAR(50) NOT NULL, -- Measurement type (e.g., 'temperature')  sampling\_rate INT NOT NULL CHECK (sampling\_rate > 0), -- Samples per second  created\_at TIMESTAMP DEFAULT NOW() -- Stream creation time  ); |

# 3. Time-Series Data

## 3.1 Sensor Data Table

The sensor data table stores timestamped measurements as a time-series hypertable optimized through TimescaleDB partitioning, using a composite primary key of timestamp and stream\_id with a foreign key to sensor\_streams. It contains both raw sensor values and precomputed normalized values, enabling efficient temporal queries for real-time dashboards and historical analysis. The table's partitioning strategy and indexing significantly enhance performance for common access patterns like time-range queries and stream-specific data retrieval.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| timestamp | TIMESTAMPTZ | Precise timestamp with timezone | NOT NULL, indexed |
| stream\_id | UUID | Associated data stream | FOREIGN KEY (sensor\_streams.stream\_id), ON DELETE CASCADE |
| value | DOUBLE | Raw sensor value | NOT NULL |
| normalized\_value | DOUBLE | Z-score normalized value | Optional precomputed value |

|  |
| --- |
| SQL for Postgresql |
| -- Enable TimescaleDB extension for time-series optimization  CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;  -- Time-series sensor measurements  SELECT create\_hypertable(  'sensor\_data',  'timestamp',  chunk\_time\_interval => INTERVAL '7 days',  if\_not\_exists => TRUE  );  CREATE TABLE sensor\_data (  timestamp TIMESTAMPTZ NOT NULL, -- Precise measurement time  stream\_id UUID REFERENCES sensor\_streams(stream\_id) ON DELETE CASCADE,  value DOUBLE PRECISION NOT NULL, -- Raw sensor reading  normalized\_value DOUBLE PRECISION, -- Standardized value (Z-score)  PRIMARY KEY (timestamp, stream\_id)  );  -- Optimize for time-range queries  CREATE INDEX idx\_sensor\_data\_stream\_time ON sensor\_data (stream\_id, timestamp DESC); |

# 4. Analytics Results

## 4.1 Correlation Table

The correlations table records statistical relationships between data streams with correlation\_id as its primary key, containing two foreign keys (stream\_a and stream\_b) that reference sensor\_streams. It stores analysis window timestamps, correlation coefficients, and algorithm identifiers, supporting the system's core functionality of relationship detection between IoT data streams. This structure enables historical tracking of evolving device interactions while allowing comparison of different correlation methods.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| correlation\_id | UUID | Unique correlation result ID | PRIMARY KEY |
| window\_start | TIMESTAMPTZ | Start time of the analysis window | NOT NULL |
| window\_end | TIMESTAMPTZ | End time of the analysis window | NOT NULL |
| stream\_a | UUID | First data stream | FOREIGN KEY (sensor\_streams.stream\_id) |
| stream\_b | UUID | Second data stream | FOREIGN KEY (sensor\_streams.stream\_id) |
| coefficient | DOUBLE | Correlation coefficient | CHECK: -1 ≤ coefficient ≤ 1 |
| algorithm | VARCHAR(50) | Algorithm used | NOT NULL |

|  |
| --- |
| SQL for Postgresql |
| -- Correlation analysis results  CREATE TABLE correlations (  correlation\_id UUID PRIMARY KEY,  window\_start TIMESTAMPTZ NOT NULL, -- Analysis window start  window\_end TIMESTAMPTZ NOT NULL, -- Analysis window end  stream\_a UUID REFERENCES sensor\_streams(stream\_id),  stream\_b UUID REFERENCES sensor\_streams(stream\_id),  coefficient DOUBLE PRECISION NOT NULL CHECK (coefficient BETWEEN -1 AND 1),  algorithm VARCHAR(50) NOT NULL, -- Correlation method used  CONSTRAINT valid\_stream\_pair CHECK (stream\_a <> stream\_b)  ); |

## 4.2 Anomalies Table

The anomalies table tracks detected abnormal patterns using an anomaly\_id primary key and a stream\_id foreign key linking to sensor\_streams. It records detection timestamps, anomaly types, confidence scores, and resolution statuses, forming the basis for security alerts and maintenance workflows. The table's design supports both real-time monitoring through timestamp indexing and historical analysis via its status tracking mechanism.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| anomaly\_id | UUID | Unique anomaly identifier | PRIMARY KEY |
| stream\_id | UUID | Associated data stream | FOREIGN KEY (sensor\_streams.stream\_id), ON DELETE CASCADE |
| detected\_at | TIMESTAMPTZ | Detection timestamp | NOT NULL |
| anomaly\_type | VARCHAR(50) | Type of anomaly (e.g., "spike", "dropout") | NOT NULL |
| raw\_value | DOUBLE | Original sensor value at detection time | NOT NULL |
| confidence\_score | DOUBLE | Detection confidence (0-1) | NOT NULL, CHECK: 0 ≤ score ≤ 1 |
| status | VARCHAR(20) | Status of the anomaly | CHECK: pending, confirmed, resolved; DEFAULT 'pending' |

|  |
| --- |
| SQL for Postgresql |
| -- Detected anomaly records  CREATE TABLE anomalies (  anomaly\_id UUID PRIMARY KEY,  stream\_id UUID REFERENCES sensor\_streams(stream\_id) ON DELETE CASCADE,  detected\_at TIMESTAMPTZ NOT NULL, -- Time of detection  anomaly\_type VARCHAR(50) NOT NULL, -- Classification of anomaly  raw\_value DOUBLE PRECISION NOT NULL, -- Original sensor value  confidence\_score DOUBLE PRECISION NOT NULL CHECK (confidence\_score BETWEEN 0 AND 1),  status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'confirmed', 'resolved'))  ); |

# 5. File Management

## 5.1 Uploaded Files Table

The uploaded\_files table manages user data imports/exports with a file\_id primary key and user\_id foreign key connecting to users. It tracks file metadata including original names, storage paths, formats, and processing statuses, directly enabling the system's data upload/export functionality while maintaining user-specific file ownership and access control.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| file\_id | UUID | Unique file identifier | PRIMARY KEY |
| user\_id | UUID | Uploading user | FOREIGN KEY (users.user\_id), ON DELETE CASCADE |
| original\_name | VARCHAR(255) | Original filename | NOT NULL |
| storage\_path | VARCHAR(512) | File storage location | NOT NULL |
| format | VARCHAR(10) | File format | CHECK: csv, json, xlsx |
| status | VARCHAR(20) | Processing status | CHECK: uploading, processed, failed; DEFAULT 'uploading' |
| uploaded\_at | TIMESTAMP | Upload timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- User-uploaded files  CREATE TABLE uploaded\_files (  file\_id UUID PRIMARY KEY,  user\_id UUID REFERENCES users(user\_id) ON DELETE CASCADE,  original\_name VARCHAR(255) NOT NULL, -- Original filename  storage\_path VARCHAR(512) NOT NULL, -- Filesystem path  format VARCHAR(10) NOT NULL CHECK (format IN ('csv', 'json', 'xlsx')),  status VARCHAR(20) DEFAULT 'uploading' CHECK (status IN ('uploading', 'processed', 'failed')),  uploaded\_at TIMESTAMP DEFAULT NOW() -- Upload timestamp  ); |

# 6. Chat System

# 6.1 Chat Sessions Table

The chat\_sessions table organizes support interactions through a session\_id primary key and user\_id foreign key, recording start/end times and escalation levels. It connects to chat\_messages through a one-to-many relationship, providing structure for conversational histories while supporting both bot and human support tracking.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| session\_id | UUID | Unique chat session ID | PRIMARY KEY |
| user\_id | UUID | Initiating user | FOREIGN KEY (users.user\_id), ON DELETE SET NULL |
| started\_at | TIMESTAMP | Session start time | DEFAULT NOW() |
| ended\_at | TIMESTAMP | Session end time | - |
| escalation\_level | INT | Support escalation level | DEFAULT 0 (bot), 1 (human) |

|  |
| --- |
| SQL for Postgresql |
| -- Chatbot conversation sessions  CREATE TABLE chat\_sessions (  session\_id UUID PRIMARY KEY,  user\_id UUID REFERENCES users(user\_id) ON DELETE SET NULL,  started\_at TIMESTAMP DEFAULT NOW(), -- Conversation start time  ended\_at TIMESTAMP, -- Conversation end time  escalation\_level INT DEFAULT 0 CHECK (escalation\_level IN (0, 1)) -- 0=bot, 1=human  ); |

## 6.2 Chat Messages

Table

The chat\_messages table stores individual conversation entries with a message\_id primary key and session\_id foreign key linking to chat\_sessions. It preserves message content, sender identification (bot/human), and timestamps, enabling conversation analysis for both user support quality improvement and chatbot training purposes.

| Field Name | Data Type | Description | Constraints & Values |
| --- | --- | --- | --- |
| message\_id | UUID | Unique message ID | PRIMARY KEY |
| session\_id | UUID | Associated chat session | FOREIGN KEY (chat\_sessions.session\_id), ON DELETE CASCADE |
| content | TEXT | Message text content | NOT NULL |
| is\_bot | BOOLEAN | Whether the message is from the bot | NOT NULL |
| sent\_at | TIMESTAMP | Message timestamp | DEFAULT NOW() |

|  |
| --- |
| SQL for Postgresql |
| -- Individual chat messages  CREATE TABLE chat\_messages (  message\_id UUID PRIMARY KEY,  session\_id UUID REFERENCES chat\_sessions(session\_id) ON DELETE CASCADE,  content TEXT NOT NULL, -- Message text  is\_bot BOOLEAN NOT NULL, -- Whether message is from bot  sent\_at TIMESTAMP DEFAULT NOW() -- Message timestamp  );  -- Optimize for message retrieval by session  CREATE INDEX idx\_chat\_messages\_session ON chat\_messages (session\_id, sent\_at); |