USERS TABLE:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  password VARCHAR(255) NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Candidate Keys: `{id}, {email}`

Functional Dependencies:

- `id → name, email, password, createdAt`
- `email → id, name, password, createdAt`

Normalization Check for Users Table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- Candidate keys are id and email and all other attributes are fully dependent on candidate keys.

Check for 3NF:

- Non prime elements are directly dependent on candidate key and there is no transitivity for non prime elements.

```
CREATE TABLE posts (
   id INT AUTO_INCREMENT PRIMARY KEY,
   content TEXT NOT NULL,
   userId INT NOT NULL,
   createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    photo VARCHAR(255),
    video VARCHAR(255),
    article VARCHAR(255),
    event VARCHAR(255),
    FOREIGN KEY (userId) REFERENCES users(id) ON DELETE CASCADE
);
```

Candidate Keys: `{id}`

Functional Dependencies:

- `id → content, userId, createdAt, photo, video, article, event`

Normalization Check for Posts Table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- Candidate key is id and all other attributes are fully dependent on candidate keys.

Check for 3NF:

- Non prime elements are directly dependent on candidate key and there is no transitivity for non prime elements.

```
CREATE TABLE connections (
    id INT AUTO_INCREMENT PRIMARY KEY,
    sender_id INT NOT NULL,
    receiver_id INT NOT NULL,
    status ENUM('pending', 'accepted', 'rejected') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (sender_id) REFERENCES users(id),
    FOREIGN KEY (receiver_id) REFERENCES users(id)
);
```

Candidate Keys: `{id}`

Functional Dependencies:

- `id → sender_id, receiver_id, status, created_at`

Normalization Check for Posts Table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- Candidate key is id and all other attributes are fully dependent on candidate keys.

Check for 3NF:

- Non prime elements are directly dependent on candidate key and there is no transitivity for non prime elements.

```
CREATE TABLE user_connections (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   connection_id INT NOT NULL,
   status ENUM('pending', 'accepted', 'rejected') DEFAULT 'pending',
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (user_id) REFERENCES users(id),
   FOREIGN KEY (connection_id) REFERENCES users(id)
);
```

Candidate Keys: `{id}`

Functional Dependencies:

- `id → user_id, connection_id, status,created_at`

Normalization Check for Posts Table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- Candidate key is id and all other attributes are fully dependent on candidate keys.

Check for 3NF:

- Non prime elements are directly dependent on candidate key and there is no transitivity for non prime elements.

FEEDBACK TABLE:

```
use defaultdb;

CREATE TABLE feedbacks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  userId INT NOT NULL,
  rating INT NOT NULL,
  message TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (userId) REFERENCES users(id) ON DELETE CASCADE
);
select * from feedbacks;
```

Candidate Keys: `{id}`

The primary key is a single column (id).

Functional Dependency

id → userId, rating, message, createdAt

Normalization Check for feedback Table

1.) **Check for 1NF:**(First Normal Form)

- The table has no repeating values for each row and column.

2.) **Check for 2NF:**(Second Normal Form)

- The table follows 2NF because it is in 1NF, and there is no partial dependency (since 'id'is the only candidate key, and all non-key attributes fully depend on it).

3.) **Check for 3NF:** (Third Normal Form)

- The table follows 3NF because it is in 2NF, and there are no transitive dependencies (all non-key attributes depend directly on the primary key ‘id’, not on other non-key attributes).

```
create table tasks(
  task_id INT AUTO_INCREMENT PRIMARY KEY,
  client_id INT NOT NULL,
  task_name varchar(255) NOT NULL,
  task_date DATE,
  task_time TIME,
  remainder BOOLEAN,
  FOREIGN KEY (client_id) REFERENCES users(id)
);
```

```
ALTER TABLE tasks ADD COLUMN status ENUM('pending', 'completed') DEFAULT
'pending';
```

Candidate Keys : {task_id} , {client_id , task_date , task_time}

Functional Dependencies:

- task_id → client_id , task_name , task_date , task_time , remainder , status
- client_id, task_date, task_time → task_name, remainder, status

Normalization Check for Users Table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- All **non-prime attributes** task_name, remainder, status are fully dependent on the entire candidate keys task_id and {client_id, task_date, task_time}.

Check for 3NF:

- All non-prime attributes depend only on the candidate keys, and there is no transitive dependency.

AI RESUME BUILDER

```
CREATE TABLE resume (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   title VARCHAR(255) NOT NULL,
   profile VARCHAR(255),
   profileText TEXT,
   phone VARCHAR(20),
   email VARCHAR(255),
   linkedin VARCHAR(255),
   github VARCHAR(255),
   address TEXT
);
select * from resume;
```

Candidate key : {id} { email }

Functional Dependencies :

- `id → name, title, profile, profileText, phone, email, linkedin, github, address`
- `email → name, title, profile, profileText, phone, linkedin, github, address`

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF :

- The primary key is `id`, which is a **single attribute**.
- Since there are no **partial dependencies** (i.e., no attribute is dependent on a part of a composite key), the table is in **2NF**.

Check for 3NF :

- There are no **transitive dependencies** (no attribute depends on a non-key attribute).
- All attributes directly depend on the primary key `id`.

CREATE TABLE skills (
id INT AUTO_INCREMENT PRIMARY KEY,
user_id INT,
skill VARCHAR(255) NOT NULL,
FOREIGN KEY (user_id) REFERENCES resume(id) ON DELETE CASCADE
);

Candidate key : id, {user_id, skill}

Functional Dependencies :

- `id → user_id, skill`
- `{user_id, skill} → id`

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- The composite key `{user_id, skill}` ensures there is **no partial dependency**.
- Each non-key attribute (`skill`) depends on the whole key `{user_id, skill}`, and not just on `user_id`.

Check for 3NF :

- There is **no transitive dependency**.
- `skill` depends only on `{user_id, skill}` and does not depend on any **non-primary key attribute**.

CREATE TABLE education (
id INT AUTO_INCREMENT PRIMARY KEY,
user_id INT,

```
  degree VARCHAR(255) NOT NULL,
  institution VARCHAR(255) NOT NULL,
  startYear INT NOT NULL,
  endYear INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES resume(id) ON DELETE CASCADE
);
```

Candidate key : `{id} {user_id, degree, institution, startYear, endYear }`

Functional Dependencies :

- `id → user_id, degree, institution, startYear, endYear`
- `{user_id, degree, institution, startYear, endYear} → id`

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF :

- The composite key `{user_id, degree, institution, startYear}` ensures that all attributes are **fully functionally dependent** on it.
- There is **no partial dependency**.

Check for 3NF :

- There is **no transitive dependency**.
- All attributes are **directly dependent** on the primary key `{id}`

```
CREATE TABLE experience (
  id INT AUTO_INCREMENT PRIMARY KEY,
  user_id INT,
  position VARCHAR(255) NOT NULL,
  company VARCHAR(255) NOT NULL,
  startMonth VARCHAR(20) NOT NULL,
  startYear INT NOT NULL,
  endMonth VARCHAR(20),
  endYear INT,
  description TEXT,
```

```
    FOREIGN KEY (user_id) REFERENCES resume(id) ON DELETE CASCADE
);
```

Primary key : id
Candidate key : {user_id, position, company, startMonth, startYear}

Functional Dependencies :

- `id → user_id, position, company, startMonth, startYear, endMonth, endYear, description`
- `{user_id, position, company, startMonth, startYear} → id`

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- The composite key `{user_id, position, company, startMonth, startYear}` ensures that no **attribute is dependent on just a part of the key**.
- `description`, `endMonth`, and `endYear` **fully depend** on `{user_id, position, company, startMonth, startYear}`.

Check for 3NF :

- No **transitive dependency** exists.
- Every attribute is dependent on `{id}` or `{user_id, position, company, startMonth, startYear}` directly.

```
CREATE TABLE certificates (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    certificate VARCHAR(255) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES resume(id) ON DELETE CASCADE
);
select * from certificates;
```

Primary key : id

Candidate key : {id} {user_id, certificate}

Functional Dependencies :

- id → user_id, certificate
- {user_id, certificate} → id

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF :

- Since {user_id, certificate} is a composite key, each attribute **fully depends on the entire key**.

Check for 3NF:

- All attributes are **directly dependent** on {id} or {user_id, certificate}.

```
CREATE TABLE languages (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT,
   language VARCHAR(255) NOT NULL,
   FOREIGN KEY (user_id) REFERENCES resume(id) ON DELETE CASCADE
);
select * from languages;
```

Primary key : id
Candidate key : {user_id, language}

Functional Dependencies :

- id → user_id, language
- user_id, language → id

Normalization :

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF

- `{user_id, language}` ensures **full dependency**.
- No **partial dependency**.

Check for 3NF

- No **transitive dependencies**.
- Every attribute is **directly dependent** on the primary key `{id}`.

CHAT LIST TABLE

```
CREATE TABLE chat_list (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT NOT NULL,
   contact_id INT NOT NULL,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   UNIQUE (user_id, contact_id),
   FOREIGN KEY (user_id) REFERENCES users(id),
   FOREIGN KEY (contact_id) REFERENCES users(id)
);
```

Attributes: id , user_id, contact_id, created_id
Candidate key: {id} , {user_id, contact_id}

Functional Dependencies (FDs):

1. (id) → user_id, contact_id, created_at
2. (user_id, contact_id) → id, created_at

Normalization Check for Chat_list table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- Candidate keys are (id) and (user_id, contact_id) and all other attributes are fully dependent on the candidate keys.

Check for 3NF:

- Non-prime attributes are directly dependent on the candidate keys, and there is no transitivity for non-prime attributes.

MESSAGES TABLE

```
CREATE TABLE IF NOT EXISTS messages (
  id INT AUTO_INCREMENT PRIMARY KEY,
  chat_id INT NOT NULL,
  senderId INT NOT NULL,
  recipientId INT NOT NULL,
  message TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (chat_id) REFERENCES chat_list(id) ON DELETE
CASCADE,
  FOREIGN KEY (senderId) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (recipientId) REFERENCES users(id) ON DELETE CASCADE
);
```

Attributes: (id, chat_id, senderId, recipientId, message, createdAt)
Candidate key: {id}

Functional Dependencies (FDs):

1. (id) → chat_id, senderId, recipientId, message, createdAt

Normalization Check for Messages table

Check for 1NF:

- The table has no repeating values for each row and column.

Check for 2NF:

- The candidate key is (id), and all other attributes are fully dependent on the candidate key.

Check for 3NF:

- Non-prime attributes are directly dependent on the candidate key (id), and there is no transitivity for non-prime attributes.