Tables and Relationships:

1. **users**
   - **PK**: id
   - **Relationships**:
     - **One-to-Many with deleted_users**: deleted_users.user_id → users.id (tracks deleted user records).
     - **One-to-Many with user_department_affiliations**: user_department_affiliations.user_id → users.id (links users to departments).
     - **One-to-Many with files**: files.user_id → users.id (tracks file uploaders; nullable if user is deleted).
     - **One-to-Many with file_transfers (sender)**: file_transfers.sender_id → users.id (tracks file senders).
     - **One-to-Many with file_transfers (recipient)**: file_transfers.recipient_id → users.id (tracks file recipients; nullable if sent to department).
     - **One-to-Many with access_requests (requester)**: access_requests.requester_id → users.id (tracks access requesters).
     - **One-to-Many with access_requests (owner)**: access_requests.owner_id → users.id (tracks file owners).
     - **One-to-Many with notifications**: notifications.user_id → users.id (tracks notification recipients).
2. **deleted_users**
   - **PK**: id
   - **FK**: user_id → users.id
   - **Relationship**: One-to-One with users (each deleted user record corresponds to one user; CASCADE ensures deletion consistency).
3. **departments**
   - **PK**: id
   - **Relationships**:
     - **One-to-Many with sub_departments**: sub_departments.department_id → departments.id (sub-units belong to departments).
     - **One-to-Many with user_department_affiliations**: user_department_affiliations.department_id → departments.id (users affiliated with departments).
     - **One-to-Many with cabinets**: cabinets.department_id → departments.id (cabinets assigned to departments).
     - **One-to-Many with file_transfers**: file_transfers.department_id → departments.id (files sent to departments; nullable if sent to a user).
4. **sub_departments**
   - **PK**: id
   - **FK**: department_id → departments.id
   - **Relationships**:
     - **One-to-Many with user_department_affiliations**: user_department_affiliations.sub_department_id → sub_departments.id (users affiliated with sub-departments; nullable).
     - **One-to-Many with cabinets**: cabinets.sub_department_id → sub_departments.id (cabinets optionally tied to sub-departments; nullable).
5. **user_department_affiliations**
   - **PK**: (user_id, department_id)
   - **FK**:
     - user_id → users.id
     - department_id → departments.id
     - sub_department_id → sub_departments.id (nullable)
   - **Relationship**: Junction table resolving many-to-many between users and departments (one user can belong to many departments, one department can have many users).
6. **document_types**
   - **PK**: id
   - **Relationship**:
     - **One-to-Many with document_type_fields**: document_type_fields.document_type_id → document_types.id (fields defined per document type).
     - **One-to-Many with files**: files.document_type_id → document_types.id (files categorized by document type).
7. **document_type_fields**
   - **PK**: id
   - **FK**: document_type_id → document_types.id
   - **Relationship**: One-to-Many with document_types (each document type can have multiple fields).

8. **files**
    - **PK**: id
    - **FK**:
        - user_id → users.id (nullable)
        - document_type_id → document_types.id
    - **Relationships**:
        - **One-to-Many with file_metadata**: file_metadata.file_id → files.id (metadata entries per file).
        - **One-to-One with file_storage**: file_storage.file_id → files.id (one file per storage location).
        - **One-to-Many with file_transfers**: file_transfers.file_id → files.id (tracks file transfers).
        - **One-to-Many with access_requests**: access_requests.file_id → files.id (access requests per file).
        - **One-to-Many with notifications**: notifications.file_id → files.id (notifications tied to files; nullable).
9. **file_metadata**
    - **PK**: id
    - **FK**: file_id → files.id
    - **Relationship**: One-to-Many with files (each file can have multiple metadata entries).
10. **cabinets**
    - **PK**: id
    - **FK**:
        - department_id → departments.id
        - sub_department_id → sub_departments.id (nullable)
    - **Relationship**:
        - **One-to-Many with storage_locations**: storage_locations.cabinet_id → cabinets.id (multiple slots per cabinet).
11. **storage_locations**
    - **PK**: id
    - **FK**: cabinet_id → cabinets.id
    - **Relationship**:
        - **One-to-One with file_storage**: file_storage.storage_location_id → storage_locations.id (one storage location per file).
12. **file_storage**
    - **PK**: file_id
    - **FK**:
        - file_id → files.id
        - storage_location_id → storage_locations.id
    - **Relationship**: Junction table linking files and storage_locations (one-to-one mapping).
13. **file_transfers**
    - **PK**: id
    - **FK**:
        - file_id → files.id
        - sender_id → users.id
        - recipient_id → users.id (nullable)
        - department_id → departments.id (nullable)
    - **Relationship**: Tracks file transfers, connecting files, users (sender/recipient), and optionally departments.
14. **access_requests**
    - **PK**: id
    - **FK**:
        - requester_id → users.id
        - file_id → files.id
        - owner_id → users.id
    - **Relationship**: Connects users (requesters and owners) to files for access control.
15. **notifications**
    - **PK**: id
    - **FK**:
        - user_id → users.id
        - file_id → files.id (nullable)
    - **Relationship**: Links users to files for event notifications (e.g., uploads, transfers).

Connection Guide:

- **Primary Keys**: Use id (or composite keys like user_id, department_id) as the unique identifier for each table.
- **Foreign Keys**: Draw arrows from FKs to their corresponding PKs:
  - Solid lines for mandatory relationships (e.g., file_metadata.file_id → files.id).
  - Dashed lines for optional relationships (e.g., files.user_id → users.id).
- **Cardinality**:
  - One-to-One: files ↔ file_storage, storage_locations ↔ file_storage.
  - One-to-Many: Most relationships (e.g., users → files, files → file_metadata).
  - Junction Tables: user_department_affiliations resolves users ↔ departments.

How to Connect in ERD:

1. **Start with Core Tables**: Place users and files centrally as they anchor most relationships.
2. **Branch to Admin Tables**: Connect users to deleted_users, user_department_affiliations, departments, sub_departments, cabinets, storage_locations, document_types, and document_type_fields on the left/top.
3. **Branch to Client Tables**: Connect files to file_metadata, file_storage, file_transfers, access_requests, and notifications on the right/bottom.
4. **Link Junctions**: Position user_department_affiliations between users and departments, and file_storage between files and storage_locations.
5. **Ensure Clarity**: Use labels (e.g., "1:N", "1:1") and avoid crossing lines where possible.

This structure ensures all relationships are explicit, concise, and aligned with the system’s functionality, making it an ideal guide for diagramming the ERD in a research paper.