

เอกสารรายงาน
โครงงานระบบ Document

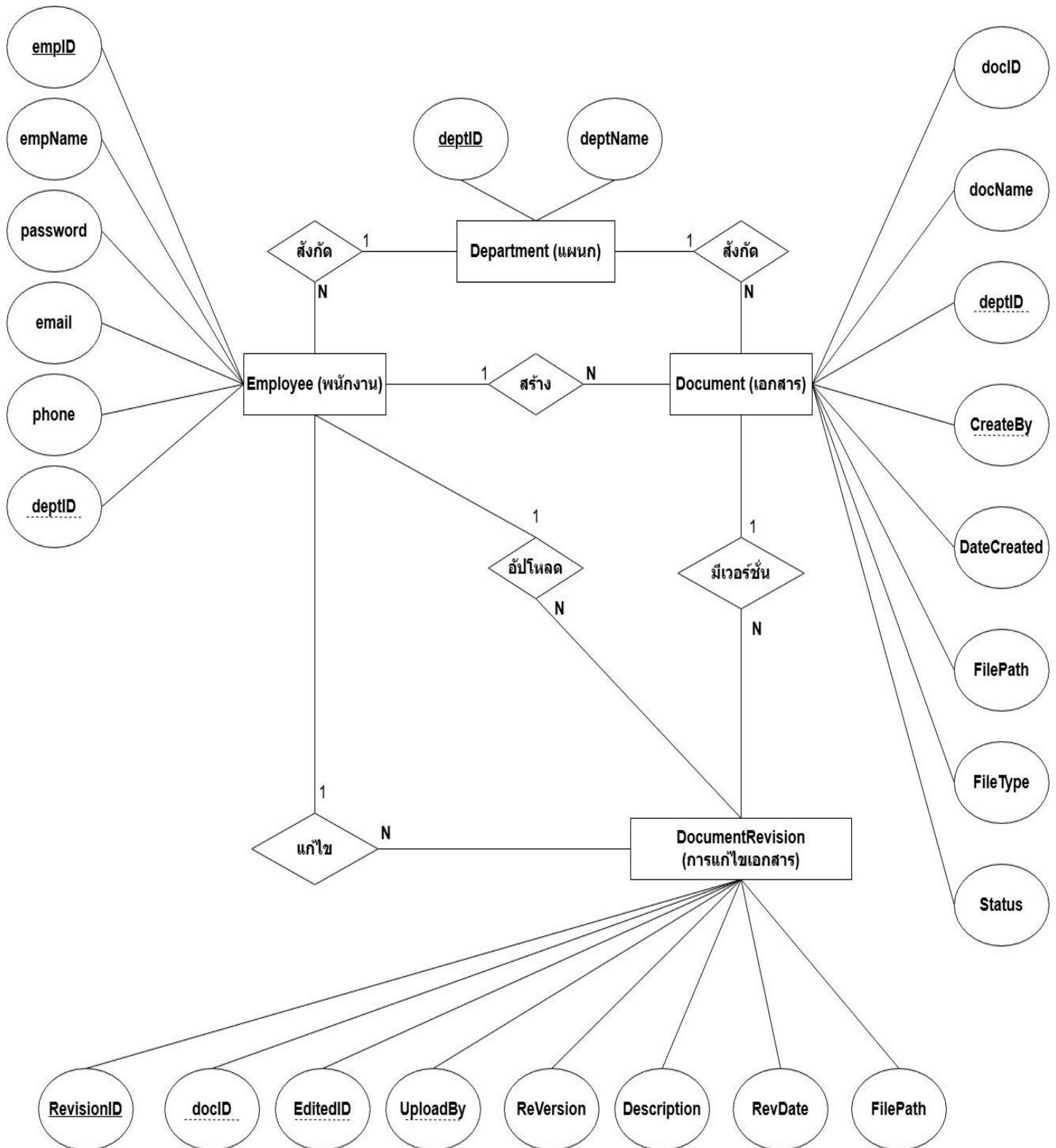
จัดทำโดย

1. 66049472 วรากร มาตุเรศ
2. 66028215 ธัญพิสิษฐ์ บัวบุตร
3. 66055274 ปวีศร เชื้อหมอ
4. 66070075 จิตาภา บำรุงนา

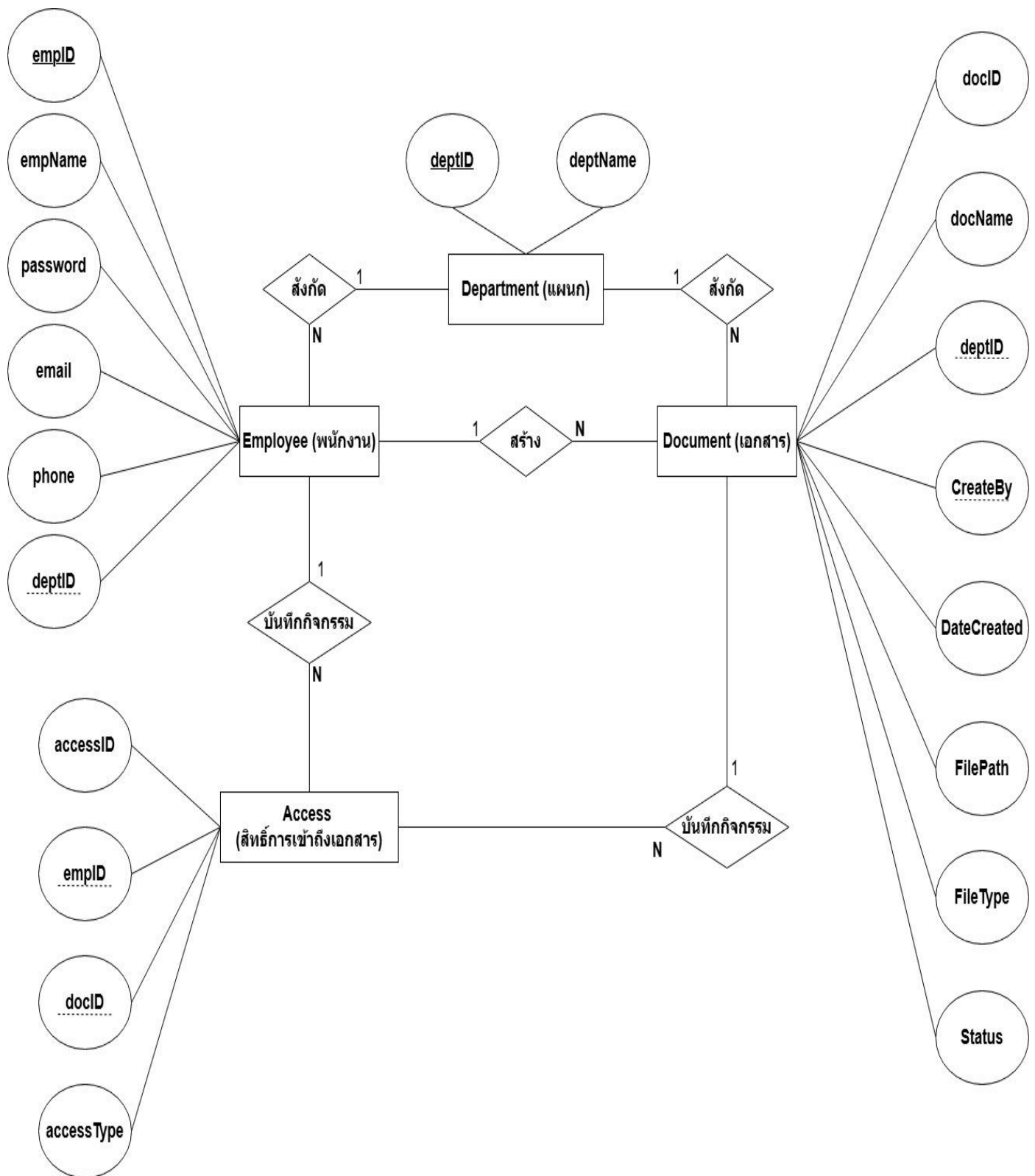
รายงานฉบับนี้เป็นส่วนหนึ่งของการศึกษา
วิชา CSI206 ฐานข้อมูลและการจัดการข้อมูลขนาดใหญ่
ภาคเรียนที่ 2 ปีการศึกษา 2567
มหาวิทยาลัยศรีปทุม กรุงเทพมหานคร

1. E-R Diagram

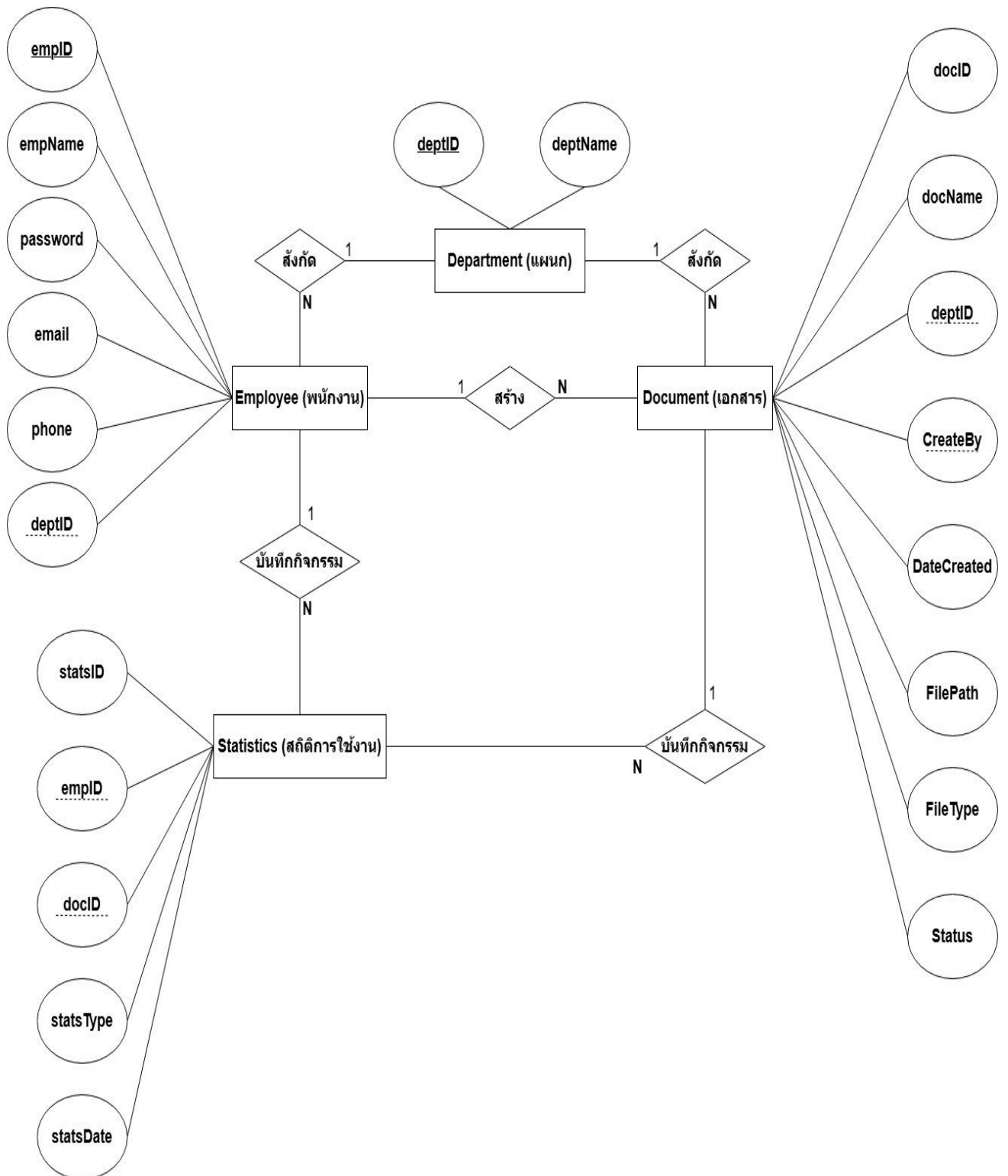
ภาพที่ 1.1 E-R Diagram การจัดการเอกสาร



ภาพที่ 1.2 E-R Diagram การจัดการสิทธิการเข้าถึง



ภาพที่ 1.3 E-R Diagram การเก็บสถิติการใช้งาน



2. Table

2.1 Employee (พนักงาน)

Name	Description	Data Type	Length	Key	Constraint	Reference
empID	รหัสพนักงาน	INT	10	PK	AUTO_INCREMENT, NOT NULL, UNIQUE	-
empName	ชื่อพนักงาน	VARCHAR	50	-	NOT NULL	-
password	รหัสผ่าน	VARCHAR	255	-	NOT NULL	-
email	อีเมล	VARCHAR	100		NOT NULL, UNIQUE	
phone	เบอร์โทรศัพท์	VARCHAR	100	-	NOT NULL, UNIQUE	-
deptID	รหัสแผนกที่สังกัด	INT	10	FK	NOT NULL	Department.deptID

2.2 Department (แผนก)

Name	Description	Data Type	Length	Key	Constraint	Reference
deptID	รหัสแผนก	INT		PK	NOT NULL, AUTO_INCREMENT	-
deptName	ชื่อแผนก	VARCHAR		-	NOT NULL	-

2.3 Document (เอกสาร)

Name	Description	Data Type	Length	Key	Constraint	Reference
docID	รหัสเอกสาร	INT	10	PK	NOT NULL, AUTO_INCREMENT	-
docName	ชื่อเอกสาร	VARCHAR	255	-	NOT NULL	-
CreatedBy	รหัสพนักงาน ที่สร้าง	TEXT	-	-	NOT NULL	Employee .empID
deptID	รหัสแผนกของ เอกสาร	INT	10	FK	NOT NULL	Department .deptID
DateCreated	วันที่สร้างเอกสาร	TIME STAMP	-	-	DEFAULT CURRENT_TIME STAMP	-
FilePath	ที่อยู่ไฟล์เอกสาร	VARCHAR	255	-	NOT NULL	-
FileType	ประเภทไฟล์เอกสาร (PDF, DOCX, TXT, ฯลฯ)	VARCHAR	50	-	NOT NULL	-
Status	สถานะเอกสาร	ENUM	ACTIVE ARCHIVED DELETED	-	DEFAULT 'ACTIVE'	-

2.4 DocumentRevision (การแก้ไขเอกสาร)

Name	Description	Data Type	Length	Key	Constraint	Reference
RevisionID	รหัสการแก้ไข	INT	10	PK	NOT NULL, AUTO_INCREMENT	-
docID	เอกสารที่ถูกแก้ไข	INT	10	FK	NOT NULL	Department .deptID
EditedID	รหัสพนักงานที่แก้ไขเอกสาร	INT	10	FK	NOT NULL	Employee .empID
UploadBy	รหัสพนักงานที่อัปโหลดเวอร์ชันใหม่	INT	10	FK	NOT NULL	Employee .empID
Version	เวอร์ชันเอกสาร	INT	10	-	NOT NULL, AUTO_INCREMENT	-
Description	คำอธิบาย	TEXT	-	-	NOT NULL	-
RevDate	วันที่แก้ไขเอกสาร	TIME STAMP	-	-	DEFAULT CURRENT_ TIMESTAMP	-
FilePath	ที่อยู่ไฟล์เวอร์ชันนี้	VARCHAR	255	-	NOT NULL	-

2.5 Access (สิทธิ์การเข้าถึง)

Name	Description	Data Type	Length	Key	Constraint	Reference
accessID	รหัสสิทธิ์	INT	10	PK	NOT NULL, AUTO_INCREMENT	-
emplID	รหัสพนักงาน ที่ได้รับสิทธิ์	INT	10	FK	NOT NULL	Employee .emplID
docID	รหัสเอกสารที่ ให้สิทธิ์เข้าถึง	INT	10	FK	NOT NULL	Department .deptID
accessType	ระดับสิทธิ์	INT	10	FK	NOT NULL	-

2.6 Statistics (สถิติการใช้งาน)

Name	Description	Data Type	Length	Key	Constraint	Reference
statsID	รหัสสถิติการ เปิดเอกสาร	INT	10	PK	NOT NULL, AUTO_INCREMENT	-
emplID	รหัสพนักงาน ที่เปิดเอกสาร	INT	10	FK	NOT NULL	Employee .emplID
docID	รหัสเอกสารที่ ถูกเปิด	INT	10	FK	NOT NULL	Department .deptID
statsType	ประเภทกิจกรรม ที่ทำ	ENUM	LOGIN LOGOUT VIEW EDIT	-	NOT NULL	-
statsDate	วันที่และเวลา ที่เปิดเอกสาร	TIME STAMP	-	-	NOT NULL	-

3. Data Dictionary

3.1 Employee (พนักงาน)

- empID : รหัสพนักงาน
- password : รหัสผ่าน
- empName : ชื่อพนักงาน
- email : อีเมล
- phone : เบอร์โทรศัพท์
- deptID (FK) : สังกัดแผนก

3.2 Department (แผนก)

- deptID (PK) : รหัสแผนก
- deptName : ชื่อแผนก

3.3 Document (เอกสาร)

- docID (PK) : รหัสเอกสาร
- docName : ชื่อเอกสาร
- CreatedBy (FK) : พนักงานที่สร้างเอกสาร
- DateCreated : วันที่สร้าง
- deptID (FK) : เอกสารนี้เป็นของแผนกไหน
- FilePath : ตำแหน่งไฟล์หรือ URL
- FileType : ประเภทไฟล์ (.pdf, .docx, .xlsx)
- Status : สถานะ (ACTIVE, ARCHIVED, DELETED)

3.4 DocumentRevision (การแก้ไขเอกสาร)

- RevisionID (PK) : รหัสการแก้ไข
- docID (FK) : รหัสเอกสารที่ถูกแก้ไข
- UploadBy (FK) : รหัสพนักงานที่แก้ไข
- Version : เวอร์ชันเอกสาร
- Description : คำอธิบาย
- RevDate : วันที่แก้ไขเอกสาร
- FilePath : ที่อยู่ไฟล์เอกสารเวอร์ชันนี้

3.5 AccessControl (สิทธิ์การเข้าถึง)

- accessID (PK) : รหัสสิทธิ์
- empID (FK) : พนักงานที่มีสิทธิ์
- DocumentID (FK) : เอกสารที่เข้าถึง
- accessType : ระดับสิทธิ์ (View, Edit, Delete)

3.6 Statistics (สถิติการใช้งานระบบ)

- statsID : รหัสสถิติการเปิดเอกสาร
- empID : รหัสพนักงานที่เปิดเอกสาร
- docID : รหัสเอกสารที่ถูกเปิด
- statsType : ประเภทกิจกรรมที่ทำ
- statsDate : วันที่และเวลาที่เปิดเอกสาร

4. Source code

โค้ดนี้เป็นทูล API Express.js และ PostgreSQL โดยใช้ pool.query() ที่ข้อมูลจากรายละเอียด
6609315 6609315 6609315

```
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  port: 5432,
  database: 'mydb'
});

app.get('/api/files/:id', async (req, res) => {
  try {
    const result = await pool.query(
      `SELECT id, filename AS name, url AS fileurl, file_type AS type, department, document_date AS date, description, uploaded_by, uploaded_at FROM files WHERE id = $1`
    );
    res.json(result.rows);
  } catch (err) {
    console.error('Failed to fetch file:', err);
    res.status(500).json({ message: 'Failed to fetch file' });
  }
});
```

- กำหนด route ให้ API รองรับ GET ที่ /api/files/:id โดยใช้ async เพราะมันมี await เพื่อรอข้อมูลจากรายละเอียด
- ใช้ try เพื่อป้องกันข้อผิดพลาดที่อาจเกิดขึ้นขณะรันโค้ด
- ใช้ pool.query() เพื่อเรียกใช้ SQL เพื่อดึงข้อมูลจากรายละเอียด
- ส่งค่า result.rows ที่ client ในรูปแบบ JSON
- หากมีข้อผิดพลาดเกิดขึ้นใน catch block
- ส่งค่าข้อผิดพลาดที่ client ในรูปแบบ console
- ส่ง HTTP 500 (Internal Server Error) แทนที่ client พร้อมข้อความ 'Failed to fetch file'

```
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  port: 5432,
  database: 'mydb'
});

app.get('/api/files/:id', async (req, res) => {
  try {
    const result = await pool.query(
      `SELECT id, filename AS name, url AS fileurl, file_type AS type, department, document_date AS date, description, uploaded_by, uploaded_at FROM files WHERE id = $1`
    );
    res.json(result.rows);
  } catch (err) {
    console.error('Failed to fetch file:', err);
    res.status(500).json({ message: 'Failed to fetch file' });
  }
});
```

- กำหนด route /api/files/:id โดยใช้ HTTP GET | User Id เป็นพารามิเตอร์ที่ส่งมาเพื่อระบุไฟล์
- ใช้ pool.query() เพื่อเรียกใช้ SQL เพื่อดึงข้อมูลจากรายละเอียด
- ส่งค่า result.rows ที่ client ในรูปแบบ JSON
- หากมีข้อผิดพลาดเกิดขึ้นใน catch block
- ส่งค่าข้อผิดพลาดที่ client ในรูปแบบ console
- ส่ง HTTP 500 (Internal Server Error) แทนที่ client พร้อมข้อความ 'Failed to fetch file'

ภาพที่ 4.1 Select

66049472 วรากร นาคะวงศ์

```
const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  port: 5432,
  database: 'mydb'
});

app.put('/api/files/:id', async (req, res) => {
  try {
    const result = await pool.query(
      `UPDATE files
      SET filename = $1, file_type = $2, document_date = $3, department = $4
      WHERE id = $5`
    );
    res.json(result.rows);
  } catch (err) {
    console.error('Error updating file:', err);
    res.status(500).json({ message: 'Failed to update file' });
  }
});
```

app.put → กำหนดให้ API รองรับ HTTP PUT Method ซึ่งใช้สำหรับ อัปเดตข้อมูล
'/api/files/:id' → กำหนดเส้นทาง (route) ที่รับ id ของไฟล์เป็นพารามิเตอร์
async (req, res) → ใช้ async/await เพื่อจัดการกับฟังก์ชันที่คืนค่าเป็น asynchronous

const fileId = req.params.id; // id ของไฟล์ URL parameter ที่ส่งเข้ามา

const { name, type, date, department } = req.body;

const { name, type, date, department } = req.body;

name → ชื่อไฟล์ใหม่

type → ประเภทของไฟล์

date → วันที่ของเอกสาร

department → แผนกที่เกี่ยวข้อง

ใช้ try {} catch {} ... เพื่อป้องกันข้อผิดพลาดที่อาจเกิดขึ้น

await pool.query(...) → ใช้ await เพื่อรอให้คำสั่ง SQL ทำงานเสร็จ

คำสั่ง SQL UPDATE

UPDATE files → อัปเดตข้อมูลในตาราง files

SET filename = \$1, file_type = \$2, document_date = \$3, department = \$4 → กำหนดค่าต่างๆ ใหม่

WHERE id = \$5 → อัปเดตเฉพาะไฟล์ที่มี id ตรงกับ id ที่ส่งมา

RETURNING ... → คืนค่าข้อมูลหลังจากอัปเดตเสร็จ

await pool.query(...) → ใช้ await เพื่อรอให้คำสั่ง SQL ทำงานเสร็จ

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

res.json(result.rows[0]);

ภาพที่ 4.2 Update

[illegible]

1) Simple API

API 9: Post /api/files/upload
API 9: Get /api/files/uploads/:single (file)

2) API 10

req. file - file name
req. body - {url, file name, description, date, description, uploads}

3) API 11

API 11: GET /api/files/uploads HTTP 400
API 11: GET /api/files/uploads/:single HTTP 200
API 11: GET /api/files/uploads/:single HTTP 200

4) API 12

API 12: GET /api/files/uploads/:single HTTP 200
API 12: GET /api/files/uploads/:single HTTP 200

ภาพที่ 4.3 Insert

66070075 ಸಿಎಂ ಪಾಠ್ಯ

HTTP DELETE METHOD

```
// delete file  
app.delete('/api/files/:id', async (req, res) => {  
    const fileId = req.params.id;  
    try {  
        const result = await pool.query('DELETE FROM files WHERE id = ? RETURNING *', [fileId]);  
        if (result.rows.length == 0) return res.status(404).send("File not found");  
  
        const filePath = path.join(__dirname, "uploads", path.basename(result.rows[0].url));  
        if (!fs.existsSync(filePath)) fs.unlinkSync(filePath);  
  
        res.json({ message: "File deleted", files: result.rows[0] });  
    } catch (err) {  
        console.error(err);  
        res.status(500).send("Failed to delete file");  
    }  
})
```

ಇಲ್ಲಿ ID ನ್ನು URL ನಿಗದಿಸಿ. *FileID + ನ್ನು

ಅನ್ವೇಷಣೆಯನ್ನು pool.query ಗೆ ಕೊಡುತ್ತೇವೆ.

ಈಗ Delete ಅನ್ವೇಷಣೆ ಬಿಟ್ಟು ಡಿಲೀಟ್ ಮಾಡೋಣ.

#1 ನ್ನು prepared statement ಜೊತೆ ಈ query ರೂಪದಲ್ಲಿ ಕೊಡುತ್ತೇವೆ.
(ಮುಂದೆ, ಈ ವಿಧಾನವನ್ನೇ ಬಳಸೋಣ)

#2 ... Return statementನಲ್ಲಿ ಹಾಗೆಯೇ ಮಾಡಿದೆ. res.status(404) ತಿಳಿಸುವುದು.

res.status(404) ಮತ್ತು result.rows[0].url ನ್ನು URL ನಿರ್ದೇಶನವಾಗಿಟ್ಟು

path.basname ಡಿಕ್ಕಿಯಾದರೆ path.join(... __dirname, 'uploads', 'report.pdf')

ಉದಾಹರಣೆಗೆ: existsSync ಇಲ್ಲದಿದ್ದರೆ unlinkSync ಎಂಬುದು

ಇದು ಖಾಲಿ ಆಗಿದ್ದರೆ message ವಿನಿಯೋಜನೆ. File: result.rows[0] ಇದನ್ನೇ ಬಳಸುತ್ತೇವೆ.

ಇಂತಿಹಂತದಲ್ಲಿ ಈ ಕಡೆ ಬಿಟ್ಟು ಇಂತಿಹಂತದಲ್ಲಿ

ภาพที่ 4.4 Delete