Line-by-Line Code Explanation

File 1: parseExcelPrsMdms.js

Import Statements (Lines 1-3)

```javascript
import * as XLSX from 'xlsx';
import fs from 'fs';
import { query } from '../config/db';
```

- **Line 1**: Imports the entire XLSX library for reading Excel files. The `* as XLSX` syntax imports all exports from the xlsx package
- **Line 2**: Imports Node.js file system module for file operations (checking if file exists, reading files)
- **Line 3**: Imports the `query` function from your database configuration file to execute SQL queries

Main Function Declaration (Line 5)

```javascript
export async function parseAndInsertExcel(filePath: string) {
```

- `export`: Makes this function available for import in other files
- `async`: Declares this as an asynchronous function that can use `await` for promises
- `function parseAndInsertExcel`: Function name that describes its purpose
- `filePath: string`: TypeScript type annotation - parameter must be a string representing the Excel file path

Try Block Start & File Validation (Lines 6-13)

```javascript
try {
  console.log(`📖 Reading Excel file: ${filePath}`);

  // Check if file exists
  if (!fs.existsSync(filePath)) {
    throw new Error(`Excel file not found: ${filePath}`);
  }
```

- **Line 6**: Starts try-catch block for error handling
- **Line 7**: Logs the file path being processed using template literals (backticks)

- **Line 10**: `fs.existsSync()` synchronously checks if file exists at given path
- **Line 11**: If file doesn't exist, throws an error with descriptive message

Excel File Reading (Lines 14-18)

```javascript
const workbook = XLSX.read(fs.readFileSync(filePath), { type: 'buffer' });

// Debug: List all available sheets
console.log('📋 Available sheets in Excel file:', workbook.SheetNames);
```

- **Line 14**:
  - `fs.readFileSync(filePath)`: Reads entire file into memory synchronously
  - `XLSX.read()`: Parses the file buffer into a workbook object
  - `{ type: 'buffer' }`: Tells XLSX the input is a buffer (raw binary data)
- **Line 17**: Logs all sheet names found in the workbook for debugging

Transaction Start (Lines 20-22)

```javascript
// Start transaction
await query('BEGIN');
```

- **Line 21**: Starts a database transaction. All subsequent queries will be part of this transaction
- `await`: Waits for the query to complete before continuing

Data Cleanup (Lines 24-28)

```javascript
try {
  // Clear existing data
  console.log('🗑️ Clearing existing data...');
  await query('DELETE FROM prs');
  await query('DELETE FROM mdms');
```

- **Line 24**: Nested try block for transaction-specific error handling
- **Line 27-28**: Deletes all existing data from both tables before inserting new data

PRS Sheet Processing Setup (Lines 30-41)

```javascript
// PRS Sheet Processing
console.log('📊 Processing PRS sheet...');
const prsSheet = workbook.Sheets['PRS'];

if (!prsSheet) {
  throw new Error('PRS sheet not found in Excel file');
}

const PRS = XLSX.utils.sheet_to_json(prsSheet, { defval: null });
console.log(`Found ${PRS.length} rows in PRS sheet`);
```

- **Line 32**: Gets the 'PRS' sheet from the workbook object
- **Line 34-36**: Validates that the PRS sheet exists, throws error if not found
- **Line 38**:
  - `XLSX.utils.sheet_to_json()`: Converts sheet data to JavaScript array of objects
  - `{ defval: null }`: Sets empty cells to `null` instead of undefined
- **Line 39**: Logs how many rows were found

PRS Data Processing Loop (Lines 43-73)

```javascript
for (const [index, row] of PRS.entries()) {
  const {
    'S. No.': serialNo,
    'Coach Code': coachCode,
    'Composite Flag': compositeFlag,
    'Class': cls,
    'Berth Number': berthNumber,
    'Berth Type': berthType,
  } = row as Record<string, any>;
```

- **Line 43**:
  - `PRS.entries()`: Returns iterator with [index, value] pairs
  - `for...of`: Loops through each entry
  - Destructures to get both index (position) and row (data)
- **Lines 44-51**:
  - **Destructuring assignment**: Extracts specific properties from the row object
  - Maps Excel column headers to JavaScript variable names
  - `'S. No.': serialNo`: Excel column "S. No." becomes variable `serialNo`

- `as Record<string, any>`: TypeScript type assertion - tells compiler this is an object with string keys and any values

Data Validation (Lines 53-57)

```javascript
// Skip rows with missing essential data
if (!serialNo || !coachCode) {
  console.warn(`⚠️ Skipping PRS row ${index + 1}: Missing serial_no or coach_code`);
  continue;
}
```

- **Line 54**: Checks if essential fields are missing (falsy values)
- **Line 55**: Logs warning message with row number (index + 1 because arrays are 0-based)
- **Line 56**: `continue` skips rest of loop iteration for this row

Database Insertion (Lines 59-69)

```javascript
await query(
  `INSERT INTO prs (serial_no, coach_code, composite_flag, class, berth_number, berth_type)
   VALUES ($1, $2, $3, $4, $5, $6)`,
  [
    Number(serialNo),
    coachCode,
    parseBoolean(compositeFlag),
    cls,
    berthNumber ? Number(berthNumber) : null,
    berthType,
  ]
);
```

- **Lines 59-61**: SQL INSERT statement with parameterized placeholders ($1, $2, etc.)
- **Lines 62-68**: Parameter array that replaces the placeholders:
  - `Number(serialNo)`: Converts string to number
  - `coachCode`: Used as-is (string)
  - `parseBoolean(compositeFlag)`: Calls custom function to convert to boolean
  - `cls`: Used as-is (string)
  - `berthNumber ? Number(berthNumber) : null`: Ternary operator - converts to number if exists, otherwise null

- `berthType`: Used as-is (string)

Progress Logging (Lines 71-73)

```javascript
if ((index + 1) % 100 === 0) {
  console.log(`✅ Inserted ${index + 1} PRS records...`);
}
```

- **Line 71**: Modulo operator `%` - logs progress every 100 records
- **Line 72**: Shows current progress count

MDMS Sheet Processing (Lines 76-87)

```javascript
// MDMS Sheet Processing
console.log('📊 Processing MDMS sheet...');
const mdmsSheet = workbook.Sheets['MDMS'];

if (!mdmsSheet) {
  throw new Error('MDMS sheet not found in Excel file');
}

const MDMS = XLSX.utils.sheet_to_json(mdmsSheet, { defval: null });
console.log(`Found ${MDMS.length} rows in MDMS sheet`);
```

- **Similar structure to PRS processing**
- Gets MDMS sheet, validates existence, converts to JSON array

MDMS Data Processing Loop (Lines 89-102)

```javascript
for (const [index, row] of MDMS.entries()) {
  const {
    'S. No.': serialNo,
    'layout_variant_no': layoutVariantNo,
    'composite_flag': compositeFlag,
    'coach_class_first': coachClassFirst,
    'coach_class_second': coachClassSecond,
    'prs_coach_code': prsCoachCode,
    'coach_class': coachClass,
    'berth_no': berthNo,
    'berth_qualifier': berthQualifier,
  } = row as Record<string, any>;
```

- **Same loop structure as PRS**
- **Different destructuring**: Maps MDMS-specific column headers to variables

MDMS Data Validation (Lines 104-108)

```javascript
// Skip rows with missing essential data
if (!serialNo) {
  console.warn(`⚠️ Skipping MDMS row ${index + 1}: Missing serial_no`);
  continue;
}
```

- **Simpler validation**: Only checks for serial number (less strict than PRS)

MDMS Database Insertion (Lines 110-122)

```javascript
await query(
  `INSERT INTO mdms (
    serial_no, layout_variant_no, composite_flag, coach_class_first,
    coach_class_second, prs_coach_code, coach_class, berth_no, berth_qualifier
  ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)`,
  [
    Number(serialNo),
    layoutVariantNo,
    parseBoolean(compositeFlag),
    coachClassFirst,
    coachClassSecond,
    prsCoachCode,
    coachClass,
    berthNo ? Number(berthNo) : null,
    berthQualifier,
  ]
);
```

- **Multi-line SQL**: More columns than PRS table
- **9 parameters**: $1 through $9 corresponding to array elements
- **Mixed data types**: Numbers, strings, booleans, and nullable numbers

Transaction Completion (Lines 131-140)

```javascript
// Commit transaction
await query('COMMIT');
console.log('🎉 PRS and MDMS data inserted successfully.');

// Print summary
const prsCount = await query('SELECT COUNT(*) FROM prs');
const mdmsCount = await query('SELECT COUNT(*) FROM mdms');
console.log(`📈 Summary: ${prsCount.rows[0].count} PRS records, ${mdmsCount.rows[0].count} MDM
```

- **Line 132**: Commits the transaction (makes all changes permanent)
- **Lines 135-136**: Query databases to get final record counts
- **Line 137**:
  - `prsCount.rows[0].count`: Database query results are in `.rows` array, first row, `count` column
  - Template literal to show summary

Error Handling (Lines 142-145)

```javascript
} catch (error) {
  // Rollback transaction on error
  await query('ROLLBACK');
  throw error;
}
```

- **Nested catch block**: Handles errors within the transaction
- **Line 144**: `ROLLBACK` undoes all changes made during the transaction
- **Line 145**: Re-throws the error so outer catch can handle it

Outer Error Handling (Lines 147-151)

```javascript
} catch (error) {
  console.error('❌ Failed to parse and insert Excel data:', error);
  throw error;
}
```

- **Outer catch block**: Handles any errors from the entire function
- **Line 148**: Logs the error with descriptive message
- **Line 149**: Re-throws error so calling code can handle it

Helper Function (Lines 154-162)

```javascript
function parseBoolean(val: any): boolean {
  if (typeof val === 'boolean') return val;
  if (typeof val === 'string') {
    const trimmed = val.trim().toLowerCase();
    return trimmed === 'y' || trimmed === 'yes' || trimmed === 'true' || trimmed === '1';
  }
  if (typeof val === 'number') return val === 1;
  return false;
}
```

- **Line 154**: Function that converts various formats to boolean
- **Line 155**: If already boolean, return as-is
- **Lines 156-159**: For strings:
  - `trim()`: Removes whitespace

- `toLowerCase()`: Converts to lowercase
- Checks multiple true values: 'y', 'yes', 'true', '1'

- **Line 160**: For numbers, only 1 is considered true
- **Line 161**: Default return false for any other type

File 2: seedDummyDataPrsMdms.js

Environment Configuration (Lines 1-2)

```javascript
import dotenv from 'dotenv';
dotenv.config();
```

- **Line 1**: Imports the dotenv package for environment variable management
- **Line 2**: Loads variables from .env file into `process.env`

Import Statements (Lines 4-7)

```javascript
import { initDb } from '../config/db.js';
import { parseAndInsertExcel } from './parseExcelPrsMdms.js';
import path from 'path';
import { fileURLToPath } from 'url';
import fs from 'fs';
```

- **Line 4**: Imports database initialization function
- **Line 5**: Imports the Excel parsing function we just explained
- **Line 6**: Node.js path utilities for file path manipulation
- **Line 7**: ES modules utility to work with file URLs
- **Line 8**: File system module for file operations

ES Module Directory Setup (Lines 10-12)

```javascript
// Get current directory for ES modules
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
```

- **Line 11**:
  - `import.meta.url`: Current module's URL in ES modules

- `fileURLToPath()`: Converts file URL to regular file path

- **Line 12**:
  - `path.dirname()`: Gets directory part of the file path
  - Recreates `__dirname` which doesn't exist in ES modules

IIFE (Immediately Invoked Function Expression) Start (Line 14)

```javascript
(async () => {
```

- **Async arrow function**: Wraps the main logic
- **Parentheses**: Creates an IIFE that executes immediately
- `async`: Allows use of `await` in the function body

Main Try Block and Initialization (Lines 15-21)

```javascript
try {
  console.log('🚀 Starting PRS and MDMS data seeding process...');

  // Use relative path from project root or make it configurable
  const filePath = process.env.EXCEL_FILE_PATH ||
                   path.join(__dirname, '../public/excel/MDMS_PRS_Differences_BerthQualifier.xls
```

- **Line 16**: Logs start message with emoji for visual appeal
- **Lines 19-20**:
  - **Logical OR operator** `||`: If first value is falsy, use second value
  - `process.env.EXCEL_FILE_PATH`: Environment variable (if set)
  - `path.join()`: Safely combines path segments regardless of OS
  - `__dirname`: Current script directory
  - `'../public/excel/...'`: Relative path to Excel file

File Existence Validation (Lines 22-29)

```javascript
// Check if file exists before proceeding
if (!fs.existsSync(filePath)) {
  console.error('❌ Excel file not found at:', filePath);
  console.log('💡 Please ensure the Excel file exists at the specified path.');
  console.log('💡 You can also set EXCEL_FILE_PATH environment variable to specify a different
  process.exit(1);
}
```

- **Line 23**: Checks if file exists at calculated path
- **Lines 24-26**: User-friendly error messages with helpful suggestions
- **Line 27**:
  - `process.exit(1)`: Terminates the Node.js process
  - Exit code `1` indicates error (0 would indicate success)

Process Logging (Lines 31-33)

```javascript
console.log('📁 Using Excel file:', filePath);
console.log('🔧 Creating/updating database tables...');
```

- **Line 31**: Confirms which file will be processed
- **Line 32**: Indicates database setup is starting

Database Initialization (Lines 34-36)

```javascript
// Initialize database (create tables if they don't exist)
await initDb();
console.log('✅ Database tables ready.');
```

- **Line 35**:
  - `await initDb()`: Calls database initialization function
  - This likely creates tables if they don't exist
- **Line 36**: Confirms database is ready

Excel Processing (Lines 38-39)

```javascript
console.log('📊 Parsing Excel file and inserting data...');
await parseAndInsertExcel(filePath);
```

- **Line 38**: Logs start of Excel processing
- **Line 39**:
  - `await parseAndInsertExcel(filePath)`: Calls our main function
  - Waits for all data to be processed and inserted

Success Logging (Lines 41-43)

```javascript
console.log('🎉 Data seeding completed successfully!');
console.log('💾 PRS and MDMS data has been imported into the database.');
```

- **Line 41**: Success message with celebration emoji
- **Line 42**: Confirms what data was imported

Error Handling (Lines 45-49)

```javascript
} catch (error) {
    console.error('💥 Error during data seeding:', error.message);
    console.error('📋 Full error details:', error);
    process.exit(1);
}
```

- **Line 45**: Catches any errors from the entire process
- **Line 46**:
  - `error.message`: Human-readable error message
  - Logs with explosion emoji for visual impact
- **Line 47**:
  - `error`: Full error object with stack trace
  - Useful for debugging
- **Line 48**: Exits with error code 1

IIFE Closing (Line 50)

```
javascript
})();
```

- **Closing parentheses and immediate invocation**: Executes the async function immediately

Key Programming Concepts Used:

1. **Async/Await**: For handling asynchronous operations (database queries, file reading)
2. **Destructuring**: Extracting values from objects `const { prop } = object`
3. **Template Literals**: String interpolation with backticks
4. **Ternary Operators**: Conditional expressions `condition ? value1 : value2`
5. **Error Handling**: Try-catch blocks with transaction rollback
6. **ES Modules**: Modern import/export syntax
7. **Type Annotations**: TypeScript type hints
8. **Array Methods**: `.entries()` for index/value pairs
9. **Logical Operators**: `||` for default values, `&&` for conditional execution
10. **Environment Variables**: Configuration through `.env` files