# Creating Database Importer with NodeJS + AWS S3 + AWS Lambda + PostgreSQL RDS

Database importer is an alternative to import data between two databases. I think this method is more safety. The cons is performance may slower than using DMS (Database Migration Service).

# Database Structure

This database helps AWS Lambda to index importing tasks.

# Tables

## Date Indexing Table

```
CREATE TABLE indices.s3_date_hns (
	id bigserial NOT NULL,
	date_hns timestamp NULL,
	is_imported bool DEFAULT false NULL,
	started_at timestamp NULL,
	finished_at timestamp NULL,
	skip_flag bool DEFAULT false NULL,
	CONSTRAINT s3_date_hns_pk PRIMARY KEY (id)
);
```

## Execution Lock Table

```
CREATE TABLE indices.s3_execution (
	is_executing bool DEFAULT false NULL,
	started_at timestamp NULL,
	finished_at timestamp NULL,
	last_executed_date_hns timestamp NULL,
	id serial4 NOT NULL,
	last_error varchar(255) NULL,
	CONSTRAINT s3_execution_pk PRIMARY KEY (id)
);
```

## CSV File Importing Index Table

```
CREATE TABLE indices.s3_file_importing (
	id int4 DEFAULT nextval('indices.s3_importing_id_seq'::regclass) NOT NULL,
	date_hns timestamp NULL,
	imported_at timestamp NULL,
```

```
filename varchar(255) NULL,
is_successful bool DEFAULT true NULL,
CONSTRAINT s3_importing_pk PRIMARY KEY (id)
);
```

# Sequences

## Date Indexing Sequence

```
CREATE SEQUENCE indices.s3_date_hns_id_seq
INCREMENT BY 1
MINVALUE 1
MAXVALUE 9223372036854775807
START 1
CACHE 1
NO CYCLE;
```

## Execution Lock Sequence

```
CREATE SEQUENCE indices.s3_execution_id_seq
INCREMENT BY 1
MINVALUE 1
MAXVALUE 2147483647
START 1
CACHE 1
NO CYCLE;
```

## CSV File Importing Sequence

```
CREATE SEQUENCE indices.s3_importing_id_seq
INCREMENT BY 1
MINVALUE 1
MAXVALUE 2147483647
START 1
CACHE 1
NO CYCLE;
```

# Creating Global Environment Variables

Global environment variables are varible that is accessible globally. Usually it's declared in file with name **.env**.

```
# PostgreSQL
PGHOST="Your DB IP address"
PGPASSWORD="Your DB password"
PGDATABASE="You DB name"
PGUSER="Your DB username"

# CSV files on S3 bucket
S3_CSV_BUCKET_URI="s3://<Bucket name>/<Folder name>"
S3_CREDENTIAL_FILE="/var/task/prd_s3.cfg"
S3_CSV_FOLDER="/tmp/store"

# Logging file to upload into S3 bucket
LAMBDA_FOLDER="/tmp/log"
LAMBDA_FILE="export_logs.log"

# Connection
TIMEZONE="<Your timezone>"
MAX_CONCURRENT_REQUESTS=10
SERVER_ID_LIMIT=5
AFTER_FAILED_IGNORING_MINUTES=10
```

## PostgreSQL DB Variables

Make sure you have already set up AWS Lambda function to the RDS permission with IAM.

1. `PGHOST` - PostgreSQL RDS host IP address.
2. `PGPASSWORD` - PostgreSQL RDS database password.
3. `PGDATABASE` - PostgreSQL RDS database name.
4. `PGUSER` - PostgreSQL RDS database username.

## CSV Files on S3 Bucket Variables

Make sure you have already set up AWS Lambda function to the AWS S3 permission with IAM.

1. `S3_CSV_BUCKET_URI` - The S3 bucket's URI.
2. `S3_CREDENTIAL_FILE` - The credential configuration file.
3. `S3_CSV_FOLDER` - The folder where the downloaded CSV files will be stored.

# Connection Variables

1. `TIMEZONE` - Your local timezone.
2. `MAX_CONCURRENT_REQUESTS` - Maximum concurrent request for PostgreSQL connection.
3. `SERVER_ID_LIMIT` - Server limited count.
4. `AFTER_FAILED_IGNORING_MINUTES` - When invocation failed, how long the lock will be ignored in minutes.

# Credential Configuration File

Accessing AWS S3 bucket needs permission using credential. The credential will be stored into a file named **prd_s3.cfg**.

```
[default]
access_key = <Access key>
secret_key = <Secret key>
```

# Handler Function

Handler function is a function that will be invoked by AWS Lambda when the trigger is activated.

```
/**
 * The main function
 * @param {object} event Event from system
 * @returns
 */
export const handler = async (event) => {
    // Return the output
    return { statusCode: 200, body: 'Success' };
};
```

# Loading Global Variables

```
// Load environment variables
try {
    dotenv.config({ path: '/var/task/.env' });
} catch (err) {
    console.error('Error loading .env file:', err);
    return { statusCode: 500 };
}
```

# Creating the CSV File Storage

```
// Setup logging before executing the rest of the code
if (!fs.existsSync(process.env.S3_CSV_FOLDER)) {
    fs.mkdirSync(process.env.S3_CSV_FOLDER, { recursive: true });
}
```

# Checking Execution Lock

```
// Check execution
const executingQueryChk = `SELECT is_executing, started_at, finished_at
    FROM indices.s3_execution
    WHERE is_executing = true
    LIMIT 1`;
const resultEx = await execQuery(executingQueryChk);
if (resultEx.rows.length > 0) {
    const startedAt = convertDateStringToDate(resultEx.rows[0]['started_at']);
    const finishedAt = convertDateStringToDate(resultEx.rows[0]['finished_at'])
    const currentTime = new Date();
    // Convert ms to minutes
    const timeDifferenceInMinutes = Math.floor((currentTime - startedAt) / (1000 * 60));
    if (startedAt > finishedAt && Math.abs(timeDifferenceInMinutes) <
process.env.AFTER_FAILED_IGNORING_MINUTES) {
        return processEnding(false);
    }
}
```

# Finding Import Date

```
// Find importing date
const findDateQuery = `SELECT date_hns
    FROM indices.s3_date_hns
    WHERE is_imported = false
    ORDER BY date_hns ASC
    LIMIT 1`;
const result = await execQuery(findDateQuery);
const now = result.rows[0]['date_hns'];
```

# Updating Execution Info

```
// Update execution info
const updateStartQuery = `UPDATE indices.s3_execution
    SET is_executing = true, started_at = $1, last_executed_date_hns = $2, last_error = null`;
await execQuery(updateStartQuery, [
    getTimestamp(),
    formatDateForPostgres(now),
```

```
]);
```

# Updating Import Date Info

```
// Update importing date
const updateStartedAtQuery = `UPDATE indices.s3_date_hns
    SET is_imported = false, started_at = $1
    WHERE date_hns = $2`;
await execQuery(updateStartedAtQuery, [
    getTimestamp(),
    formatDateForPostgres(now),
]);
```