LLDD BE - Job 3 ImportImpactCompetitor

SBP Mall - ระบบประกันรายได้ | Low Level Design Document

แนวทางการใช้เอกสาร

หัวข้อ	คำอธิบาย
วัตถุประสงค์	ใช้เป็นรายละเอียดระดับพัฒนาสำหรับออกแบบ ลงมือพัฒนา ตรวจทาน และทดสอบขอบเขตที่ระบุในฉบับนี้
ลำดับการอ่าน	เริ่มจาก Overview และ Scope จากนั้นตรวจ Field/Validation, Implementation, Contract, Processing Flow, Acceptance Criteria และ Test Checklist ตามลำดับ
ผลลัพธ์ที่คาดหวัง	ผู้อ่านสามารถระบุ input, ขั้นตอนประมวลผล, output, เงื่อนไขผิดพลาด และหลักฐานการทดสอบได้จากเนื้อหาในฉบับเดียว
Java เดิม	ภาคผนวกท้ายฉบับระบุ source file, ช่วงบรรทัด และ code Java เดิมสำหรับตรวจเทียบ behavior ก่อนย้ายระบบ

คำว่า Input, Progress และ Output ในเอกสารนี้หมายถึงข้อมูลตั้งต้น ลำดับการทำงาน และผลลัพธ์ที่ตรวจสอบได้ของขอบเขตที่กำลังพัฒนา

1. Overview

รายการ	รายละเอียด
Track	BE
Estimate	13 ชั่วโมง = implementation 10 + unit test 3 (30%)
Owner	Aphiwit <Bank> Khammoon
Target repository	`SBP/srm-sps-spsap-store-backend` (NestJS + TypeORM · schema `sps_store`) — batch runner ฝั่ง backend ไม่ผ่าน BFF · cron/พารามิเตอร์อยู่ใน backend config (env/config file)
Objective	นำเข้าร้านคู่แข่งจาก ALLMAP: นำข้อมูลร้านคู่แข่งรายงวดจากวิว COMPETITOR_IMPACT_VIEW ของ ALLMAP (SQL Server GSMALLMAP — ระบบภายนอก คงกลไกเดิม) เข้า fgi_impact_competitors ทีละ 10,000 แถว กันซ้ำระดับงวด (ถ้างวดมีข้อมูลแล้วจะข้ามทั้งงวด ไม่มี upsert)

Common contract reference: ทุกหัวข้อ API/FE ต้องยึด LLDD-BE-API-Common-Contracts.pdf และ LLDD-FE-Integration-Contracts.pdf สำหรับ error/auth/format/pagination/action/RBAC ก่อนลงรายละเอียดเฉพาะหน้าหรือเฉพาะ endpoint

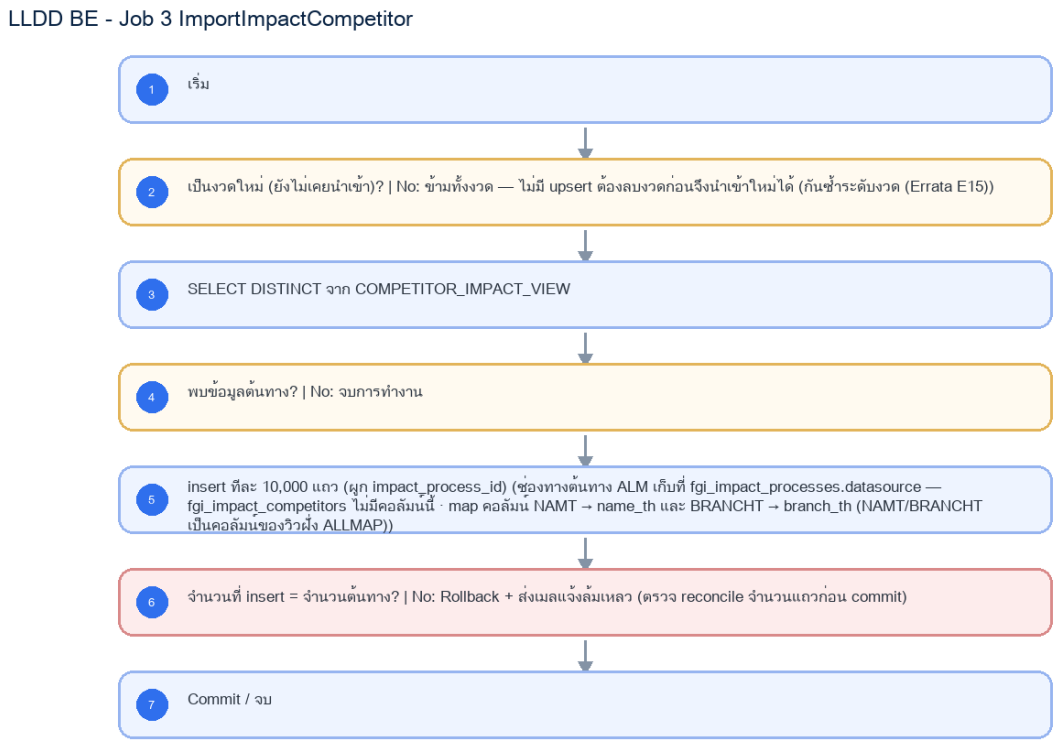
2. Screen / Functional Scope

- Main class/script: th.co.gosoft.fgi.main.ImportImpactCompetitor / /appstore/SPS/FGI/schedule/FGI_ImportCompetitor.sh
- Phase: A
- Output: fgi_impact_competitors
- Estimate: 10 ชั่วโมง
- พารามิเตอร์/cron อ่านจาก backend config (config file/env) — ไม่มีตาราง job_configs และไม่มีหน้าจอควบคุม (หน้า Flow Batch Job ในกลุ่มเมนู Flow เหลือแค่ Flowchart + Database ที่ใช้ · 2026-08-06)
- Runbook, rerun rule, risk และ history ตามเอกสาร Batch v4.0 · ผลการรันเขียน application log แบบ structured

3. Screenshot Reference

ไม่มีภาพหน้าจอสำหรับหัวข้อนี้ — เป็นเอกสารฝั่ง Backend/Batch ที่ไม่มี UI (ภาพหน้าจอทั้งหมดอยู่ในเอกสารชุด FE)

4. Implementation Flow Diagram (Reference)



รูปที่ 1: Implementation flow reference: LLDD BE - Job 3 ImportImpactCompetitor

5. Field, Format, and Validation

Field / UI	Format	Validation	Behavior
กำหนดการรัน (Cron)	0 07 7 * *	แก้ไขได้	ใช้สคริปต์ /appstore/SPS/FGI/schedule/FGI_ImportCompetitor.sh; Operations ตรวจ deployment path และ owner permission ก่อนขึ้น production
Argument (งวด)	2569|06	แก้ไขได้	รูปแบบ YYYY|MM · □□ ปีเป็น พ.ศ. ตามวิว ALLMAP — ขัดกับกติกา ค.ศ. ทั้งระบบ ต้องยืนยันกับเจ้าของ ALLMAP
Chunk Size	10000	แก้ไขได้	จำนวนแถวต่อรอบ insert
Source View	COMPETITOR_IMPACT_VIEW	ค่าคงที่/แก้ผ่านหน้าจอไม่ได้	SELECT DISTINCT / map คอลัมน์ NAMT -> NAME_TH, BRANCHT -> BRANCH_TH

5.9 Input / Progress / Output Contract

Stage	Contract for implementation
Input	Period year/month and competitor impact data from ALLMAP COMPETITOR_IMPACT_VIEW.
Progress	validate period, skip when period already exists, query competitor view, insert in chunks inside a transaction, send status mail.
Output	FGI_IMPACT_COMPETITOR rows for the target period; run status is success/no-data/failed with inserted-count reconciliation.

5.90 Job 3 Execution Stages

validate period, skip when period already exists, query competitor view, insert in chunks inside a transaction, send status mail.

Order	Service step	Repository	Output / failure contract
1	loadCompetitorPeriod	impactCompetitorRepository	คืน metrics และ throw typed error; transaction/rerun ใช้ contract ด้านล่าง
2	deduplicateCompetitors	impactCompetitorRepository	คืน metrics และ throw typed error; transaction/rerun ใช้ contract ด้านล่าง
3	upsertCompetitors	impactCompetitorRepository	คืน metrics และ throw typed error; transaction/rerun ใช้ contract ด้านล่าง
4	reconcileCompetitorCount	impactCompetitorRepository	คืน metrics และ throw typed error; transaction/rerun ใช้ contract ด้านล่าง

5.91 Job 3 Run Evidence

Evidence	Job-specific value	Acceptance
Input identity	Period year/month and competitor impact data from ALLMAP COMPETITOR_IMPACT_VIEW.	snapshot input file/business key/period in run record
Output identity	FGI_IMPACT_COMPETITOR rows for the target period; run status is success/no-data/failed with inserted-count reconciliation.	reconcile input, success, reject and skipped counts
Dedup proof	UNIQUE(impact_process_id, competitor_code, period_key); source row ซ้ำในไฟล์/วิวต้อง deduplicate ก่อน upsert	rerun fixture produces no duplicate target business key
Transaction proof	validate งวดก่อนอ่าน; upsert ทีละ chunk และ commit หลัง reconcile จำนวน input/success/reject ของ chunk ตรงกัน	injected failure leaves no partial committed state outside documented boundary
Security proof	ALLMAP datasource ใช้ secretRef และ TLS verify-full; จำกัด DB user เป็น SELECT เฉพาะ source view	config/log/error contains no plaintext secret

5.92 Legacy Java Source Reference

Legacy file	Line range	Responsibility to carry forward
fcsJar/src/th/co/gosoft/fgi/main/ImportImpactCompetitor.java	16-48	Legacy main entrypoint and notification wrapper.
fcsJar/src/th/co/gosoft/fgi/controller/ImportController.java	483-598	Validate params, skip duplicates, query source, chunk insert competitors.
fcsJar/src/th/co/gosoft/fgi/dao/jdbc/ImportJdbc.java	200-241	Count existing period, query COMPETITOR_IMPACT_VIEW, insert FGI_IMPACT_COMPETITOR.

Line ranges refer to the legacy Java implementation under /Users/bank_mac/gosoft/java/SBP/fcsJar. Use these ranges to preserve business behavior while implementing the target Node job.

5.93 Target Repository and SQL Contract

Contract	Target implementation
Repository	impactCompetitorRepository
Idempotency / dedup	UNIQUE(impact_process_id, competitor_code, period_key); source row ซ้ำในไฟล์/วิวต้อง deduplicate ก่อน upsert
Transaction boundary	validate งวดก่อนอ่าน; upsert ทีละ chunk และ commit หลัง reconcile จำนวน input/success/reject ของ chunk ตรงกัน
Security	ALLMAP datasource ใช้ secretRef และ TLS verify-full; จำกัด DB user เป็น SELECT เฉพาะ source view

Input / candidate query

```
SELECT impact_process_id, competitor_code, name_th, branch_th, opened_date, closed_date, period_key
FROM allmap_competitor_impact_view
WHERE period_key = :period_key;
```

Write / upsert query

```
INSERT INTO fgi_impact_competitors
   (impact_process_id, competitor_code, name_th, branch_th, opened_date, closed_date, period_key, updated_at)
VALUES (:impact_process_id, :competitor_code, :name_th, :branch_th, :opened_date, :closed_date, :period_key, CURRENT_TIMESTAMP)
ON CONFLICT (impact_process_id, competitor_code, period_key)
DO UPDATE SET name_th = EXCLUDED.name_th,
              branch_th = EXCLUDED.branch_th,
              opened_date = EXCLUDED.opened_date,
              closed_date = EXCLUDED.closed_date,
              updated_at = CURRENT_TIMESTAMP;
```

5.94 Target Node Implementation

โครงสร้างนี้ระบุ service/repository เฉพาะงานและต้อง implement ตาม SQL, transaction, idempotency และ security contract ด้านบน โดยทุกขั้นต้องคืน metrics สำหรับ reconcile และ run history

```
export async function runLlddBeJob3Importimpactcompetitor(ctx, services) {
  const run = await services.jobRuns.acquire({
    jobNo: "3", period: ctx.period, triggeredBy: ctx.triggeredBy
  });

  try {
    ctx = { ...ctx, runId: run.id, repository: services.impactCompetitorRepository };
    const step1 = await services.loadCompetitorPeriod(ctx, undefined);
    const step2 = await services.deduplicateCompetitors(ctx, step1);
    const step3 = await services.upsertCompetitors(ctx, step2);
    const step4 = await services.reconcileCompetitorCount(ctx, step3);
    const result = step4;
    await services.jobRuns.finish(run.id, "SUCCESS", result.metrics);
    return { runId: run.id, status: "SUCCESS", ...result };
  } catch (error) {
    await services.jobRuns.finish(run.id, "FAILED", {
      errorCode: error.code ?? "JOB_FAILED",
      errorMessage: error.message
    });
    throw error;
  }
}
```

6. Button / User Action Mapping

Action	Trigger	API / Service	Expected Result
รันตามตารางเวลา	CRON	scheduler → runner (job 3)	อ่าน cron/พารามิเตอร์จาก backend config
รันนอกรอบ (manual/rerun)	CLI	CLI/ops runbook → runner (job 3)	guard ไม่ให้รันซ้อนด้วย distributed lock
แก้พารามิเตอร์/เปิด-ปิด job	CONFIG	แก้ backend config แล้ว deploy	ไม่มี endpoint และไม่มีหน้าจอควบคุม — หน้า Flow Batch Job เป็น reference อย่างเดียว (2026-08-06)

Action	Trigger	API / Service	Expected Result
ตรวจผลการรัน	LOG	application log (structured)	ไม่มีตาราง job_run_histories แล้ว · ไฟล์/ACK ดูที่ interface_transactions

7. API Contract

เอกสารฉบับนี้ไม่มี endpoint ของตัวเอง — เป็นสัญญา/งานภายในที่เอกสารอื่นเรียกใช้ (ดูขอบเขตใน 5.90 Endpoint Implementation Contract) · รายการ endpoint ทั้ง 29 เส้นของ SBPGI อยู่ที่ LLDD-API.pdf และ `API design`

8. Reference DB Mapping (No Database Page Work)

ส่วนนี้เป็นข้อมูลอ้างอิงสำหรับการ implement API/Job เท่านั้น ไม่ใช่งานสร้างหน้า Database, ไม่ใช่งานออกแบบ DB page และไม่ถูกนับเป็น deliverable แยกของ FE/BE

Table / Object	R/W	Usage
fgi_impact_competitors	W	insert รายงวด (งวดล่าสุดต่อร้าน) ดึงจาก ALLMAP · ช่องทางต้นทาง ALM เก็บที่ fgi_impact_processes.datasource

9. Skeleton Code (Batch Job 3)

9.1 ผังไฟล์ที่ต้องสร้าง (Job 3)

โครงไฟล์ของ Job 3 (th.co.gosoft.fgi.main.ImportImpactCompetitor เดิม) วางใต้ `src/batch/sbpgi/` ของ store-backend โดยใช้ convention เดียวกับ module ธุรกิจอื่น: inject custom provider `DATA_SOURCE` แล้วยิง raw SQL, repository ประกาศเป็น factory provider ที่ใช้ token string, entity อยู่ใน `src/entitys/`

หมายเหตุสำคัญ — `src/batch/*` ทั้งชุดเป็นของใหม่ที่ยังไม่มีใน store-backend: ปัจจุบัน repo ไม่มีโฟลเดอร์ `src/batch` เลย และแม้จะติดตั้ง `@nestjs/schedule` ไว้แล้วก็ยังไม่มี `@Cron`/`@Interval` แม้แต่จุดเดียว ดังนั้น `runner.ts` / `scheduler.ts` / `cli.js` / `job-failure.notifier.ts` คือ งานตั้งต้นของ Phase แรก ที่ต้องสร้างเองทั้งหมด พร้อม register `ScheduleModule.forRoot()` ใน `app.module.ts` — ไม่ใช่ของเดิมที่ reuse ได้

Path	หน้าที่
src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.job.ts	คลาส `ImportImpactCompetitorJob` — `run(ctx)` เรียงตาม flow ของ Job 3 ทีละขั้น, ครอบ transaction, จบด้วย structured log
src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.service.ts	คลาส `ImportImpactCompetitorService` — logic ต่อขั้น (อ่าน/parse/คำนวณ/เขียน) + repository token ที่ inject จาก `DATA_SOURCE`
src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.config.ts	คลาส `SbpgiJob3Config` (แบบเดียวกับ `src/config/app.config.ts` — โปรเจกต์นี้ไม่ใช้ `registerAs`) — cron และพารามิเตอร์ทั้ง 4 ตัวของ Job 3 อ่านจาก env/config file (ไม่มีตาราง job_configs)
src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.module.ts	NestJS module ผูก job + service + repository provider (factory token string) เข้ากับ `DatabaseModule`
src/batch/runner.ts	ตัวรันกลาง: resolve job ตาม jobNo, กันรันซ้อนด้วย advisory lock, จับ error → แจ้งเตือน, เขียน structured log สรุป (ใช้ร่วมทั้ง 11 job)
src/batch/scheduler.ts	ลงทะเบียน cron จาก config (`SBPGI_JOB3_CRON` = `0 07 7 * *`) และรองรับสั่งรันนอกรอบผ่าน CLI/runbook
src/batch/job-failure.notifier.ts	ส่งอีเมลแจ้งผู้ดูแลเมื่อ job ล้มเหลว ผ่าน `EmailLibService` ของ `@gosoft-sbp/email-lib` (log ลง `email_sent` ให้อัตโนมัติ)

9.2 Config Schema ของ Job 3 (backend config / env)

cron ปัจจุบันของ Job 3 คือ `0 07 7 * *` (ทุกวันที่ 7 เวลา 07:00) — ประกาศเป็น `SBPGI_JOB3_CRON` และอ่านตอน bootstrap ของ `scheduler.ts`; ถ้า `enabled=false` scheduler ต้องไม่ลงทะเบียน cron ของ job นี้

```
// src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.config.ts
// convention จริงของ store-backend คือคลาส config (`src/config/app.config.ts` ที่ export ผ่าน
// `AppConfigModule` แบบ @Global แล้วอ่าน process.env ตรง ๆ) — โปรเจกต์นี้ **ไม่ได้ใช้ registerAs**
// แม้แต่จุดเดียว จึงประกาศเป็นคลาสให้รีวิว/ทดสอบเหมือน config ตัวอื่น
import { Injectable } from '@nestjs/common';

// TODO: Job 3 ไม่มีตาราง job_configs และไม่มี Job Admin API แล้ว (ตัดสินใจ 2026-08-06)
// TODO: ค่าทุกตัวอ่านจาก env/config file ของ backend เท่านั้น — เปลี่ยนค่า = แก้ config แล้ว deploy
export interface Job3Config {
  /** เปิด/ปิด job รอบถัดไปโดยไม่ต้อง deploy โค้ด */
  enabled: boolean;
  /** cron ของ job นี้ (อ่านตอน bootstrap ของ scheduler.ts) */
  cron: string;
  /** กำหนดการรัน (Cron) — ใช้สคริปต์ /appstore/SPS/FGI/schedule/FGI_ImportCompetitor.sh; Operations ตรวจ deployment path และ owner
permission ก่อนขึ้น production */
  cron: string;
  /** Argument (งวด) — รูปแบบ YYYY|MM · □□ ปีเป็น พ.ศ. ตามวิว ALLMAP — ขัดกับกติกา ค.ศ. ทั้งระบบ ต้องยืนยันกับเจ้าของ ALLMAP */
  argument: string;
  /** Chunk Size — จำนวนแถวต่อรอบ insert */
  chunkSize: number;
  /** Source View — SELECT DISTINCT / map คอลัมน์ NAMT -> NAME_TH, BRANCHT -> BRANCH_TH */
  sourceView: string;
  /** ผู้รับอีเมลเมื่อ job ล้มเหลว — เก็บเป็น string คั่น comma ให้ตรง signature ของ
     `EmailLibService.sendMail({ mailTo })` ที่รับ string ไม่ใช่ string[] */
  mailTo: string;
}

@Injectable()
export class SbpgiJob3Config implements Job3Config {
  // TODO: ยืนยันค่า default ทุกตัวกับ Ops ก่อนขึ้น production (ไม่มีหน้าจอแก้ค่าแล้ว)
  enabled = (process.env.SBPGI_JOB3_ENABLED ?? 'true') === 'true';
  cron = process.env.SBPGI_JOB3_CRON ?? '0 07 7 * *';
  cron = process.env.SBPGI_JOB3_CRON ?? '0 07 7 * *'; // TODO: แก้ผ่าน env/config file แล้ว deploy
  argument = process.env.SBPGI_JOB3_ARGUMENT ?? '2569|06'; // TODO: ปีเป็น พ.ศ. ตามวิว ALLMAP — ขัดกับกติกา ค.ศ. ทั้งระบบ
ต้องยืนยันกับเจ้าของ ALLMAP (□□)
  chunkSize = Number(process.env.SBPGI_JOB3_CHUNK_SIZE ?? 10000); // TODO: แก้ผ่าน env/config file แล้ว deploy
  sourceView = process.env.SBPGI_JOB3_SOURCE_VIEW ?? 'COMPETITOR_IMPACT_VIEW'; // TODO: ค่าคงที่ทางธุรกิจ —
เปลี่ยนต้องผ่านการอนุมัติ
  mailTo = process.env.SBPGI_JOB3_MAIL_TO ?? ''; // TODO: ผู้รับอีเมลแจ้ง error คั่นด้วย comma (เดิม: config mailTo / storeretention)
}

// TODO: เพิ่ม SbpgiJob3Config ใน providers/exports ของ AppConfigModule (@Global) เหมือน AppConfig
```

9.3 Job Class — `run(ctx)` ของ Job 3 ทีละขั้นตามผัง

9.3.1 สัญญาของชั้นกลาง (`runner.ts`) + โครง service ของ Job 3

job class อ้าง `JobRunContext` / `JobRunResult` / `JobState` / `JobFailedError` — ทั้งหมดนิยาม ครั้งเดียวใน `src/batch/runner.ts` (ไฟล์ร่วมของทุก job ให้ merge ไม่ใช่เขียนทับ) และ service ต้องมี method ครบตามตารางขั้นตอนด้านล่าง มิฉะนั้น job class จะเรียก method ที่ไม่มีอยู่

```
// src/batch/runner.ts — สัญญากลางของทุก job (ประกาศครั้งเดียว ใช้ร่วมทั้ง 10 ฉบับ)

export interface JobRunContext {
  jobNo: string;
  period: string; // YYYYMM ของงวดที่รัน
  triggeredBy: string; // 'CRON' | userId ที่สั่งรันนอกรอบ
  params?: Record<string, string>;
}

export interface JobRunResult {
  event: 'job.finish';
  jobNo: string;
  jobName: string;
  status: 'SUCCESS' | 'SKIPPED' | 'SKIPPED_LOCKED' | 'FAILED';
  period: string;
  output: string;
  read: number; written: number; skipped: number; rejected: number;
  durationMs: number;
}

/** counter + ค่าที่ทุกขั้นของ job ใช้ร่วมกัน (service เป็นผู้สร้างผ่าน createState) */
export interface JobState {
  period: string;
  read: number; written: number; skipped: number; rejected: number;
  // TODO: เพิ่ม field เฉพาะของ job นี้ (เช่น rows ที่อ่านมา, path ไฟล์ที่เขียน)
  [key: string]: unknown;
}

/** error ที่ทำให้ job จบเป็น FAILED และส่งอีเมลแจ้งผู้ดูแล */
export class JobFailedError extends Error {
  constructor(public readonly code: string, message: string) { super(message); }
```

```
}

/** ใช้ออกจาก transaction เมื่อสาขา NO บอกให้ข้ามงวด/เรคคอร์ด — runner สรุปเป็น SKIPPED ไม่ใช่ FAILED */
export class JobSkippedError extends Error {}
// ImportImpactCompetitorService — method ที่ job class เรียก (1 method ต่อ 1 ขั้นในตารางด้านบน)
import { Inject, Injectable } from '@nestjs/common';
import type { DataSource, EntityManager } from 'typeorm';
import type { JobRunContext, JobState } from '../../runner';
export type { JobState };

@Injectable()
export class ImportImpactCompetitorService {
  constructor(@Inject('DATA_SOURCE') private readonly dataSource: DataSource) {}

  createState(ctx: JobRunContext): JobState {
    return { period: ctx.period, read: 0, written: 0, skipped: 0, rejected: 0 };
  }

  // เป็นงวดใหม่ (ยังไม่เคยนำเข้า)?
  async check02ResolvePeriod(state: JobState): Promise<boolean> {
    return true; // TODO: เงื่อนไขจริงตามผัง
  }

  // SELECT DISTINCT จาก COMPETITOR_IMPACT_VIEW
  async step03Query(state: JobState, manager?: EntityManager): Promise<void> {
    // TODO: implement
  }

  // พบข้อมูลต้นทาง?
  async check04Condition(state: JobState): Promise<boolean> {
    return true; // TODO: เงื่อนไขจริงตามผัง
  }

  // insert ทีละ 10,000 แถว (ผูก impact_process_id)
  async step05Insert(state: JobState, manager?: EntityManager): Promise<void> {
    // TODO: implement
  }

  // จำนวนที่ insert = จำนวนต้นทาง?
  async check06Insert(state: JobState): Promise<boolean> {
    return true; // TODO: เงื่อนไขจริงตามผัง
  }

}
```

9.3.2 `run(ctx)` ของ Job 3

ทุกขั้นใน `run()` ตรงกับ flowchart ของ Job 3 หนึ่งต่อหนึ่ง (decision และ error path รวมอยู่ด้วย) — method ที่ต้อง implement ใน service ตามตารางนี้

ลำดับ	ชนิด	ขั้นตอนจากผัง	Method ที่ต้อง implement	เส้นทาง NO / error
1	start	เริ่ม	createState()	-
2	decision	เป็นงวดใหม่ (ยังไม่เคยนำเข้า)?	check02ResolvePeriod()	[branch] ข้ามทั้งงวด — ไม่มี upsert ต้องลบงวดก่อนจึงนำเข้าใหม่ได้
3	io	SELECT DISTINCT จาก COMPETITOR_IMPACT_VIEW	step03Query()	throw JobFailedError เมื่อทำไม่สำเร็จ
4	decision	พบข้อมูลต้นทาง?	check04Condition()	[end] จบการทำงาน
5	process	insert ทีละ 10,000 แถว (ผูก impact_process_id)	step05Insert()	throw JobFailedError เมื่อทำไม่สำเร็จ
6	decision	จำนวนที่ insert = จำนวนต้นทาง?	check06Insert()	[err] Rollback + ส่งเมลแจ้งล้มเหลว
7	end	Commit / จบ	summarize()	-

```
// src/batch/sbpgi/job-3-import-impact-competitor/job-3-import-impact-competitor.job.ts
import { Inject, Injectable, Logger } from '@nestjs/common';
import type { DataSource, EntityManager } from 'typeorm';
import { ImportImpactCompetitorService, type JobState } from './job-3-import-impact-competitor.service';
// 4 symbol นี้นิยามใน src/batch/runner.ts (ดูหัวข้อ 9.3.1)
import { JobFailedError, JobSkippedError, JobRunContext, JobRunResult } from '../../runner';

@Injectable()
```

```
export class ImportImpactCompetitorJob {
  static readonly jobNo = '3';
  private readonly logger = new Logger(ImportImpactCompetitorJob.name);

  constructor(
    // TODO: DATA_SOURCE = custom provider ที่ route SELECT/WITH ไป slave pool และ write ไป master
    @Inject('DATA_SOURCE') private readonly dataSource: DataSource,
    private readonly service: ImportImpactCompetitorService,
  ) {}

  async run(ctx: JobRunContext): Promise<JobRunResult> {
    const startedAt = Date.now();
    // TODO: state ถือ counter (read/written/skipped/rejected) และค่าจาก job3Config
    const state = this.service.createState(ctx);
    try {
      // ขั้นที่ 2 (decision): เป็นงวดใหม่ (ยังไม่เคยนำเข้า)? · TODO: กันซ้ำระดับงวด (Errata E15)
      const ok02 = await this.service.check02ResolvePeriod(state);
      if (!ok02) { // NO → ข้ามทั้งงวด — ไม่มี upsert ต้องลบงวดก่อนจึงนำเข้าใหม่ได้
        // TODO: เส้น NO ของขั้นนี้เป็น branch ระดับ record — ผังไม่ได้ระบุว่าหยุดหรือไปต่อ
        // ถ้าเป็น 'ข้ามรายการ' -> state.skipped += 1; แล้ว continue ในลูปของ record
        // ถ้าเป็น 'ตั้งค่าแล้วไปต่อ' -> เรียก service ตั้งค่าสถานะ แล้วเดินขั้นถัดไป (ห้าม return)
        // ถ้าเป็น 'คงสถานะเดิม/ไม่เปิดงาน' -> หยุดเฉพาะ record นี้ ห้ามไหลไปขั้นถัดไป
      }
      // ขั้นที่ 3: SELECT DISTINCT จาก COMPETITOR_IMPACT_VIEW
      await this.service.step03Query(state);
      // ขั้นที่ 4 (decision): พบข้อมูลต้นทาง?
      const ok04 = await this.service.check04Condition(state);
      if (!ok04) { // NO → จบการทำงาน
        return this.summarize(state, 'SKIPPED', startedAt);
      }
      // === transaction boundary === TODO: หนึ่ง transaction + savepoint (insert เป็น chunk ละ 10,000)
      await this.dataSource.transaction(async (manager: EntityManager) => {
        // ขั้นที่ 5: insert ทีละ 10,000 แถว (ผูก impact_process_id) · TODO: ช่องทางต้นทาง ALM เก็บที่ fgi_impact_processes.datasource —
fgi_impact_competitors ไม่มีคอลัมน์นี้ · map คอลัมน์ NAMT → name_th และ BRANCHT → branch_th (NAMT/BRANCHT เป็นคอลัมน์ของวิวฝั่ง
ALLMAP)
        await this.service.step05Insert(state, manager);
      });
      // ขั้นที่ 6 (decision): จำนวนที่ insert = จำนวนต้นทาง? · TODO: ตรวจ reconcile จำนวนแถวก่อน commit
      const ok06 = await this.service.check06Insert(state);
      if (!ok06) throw new JobFailedError('JOB3_STEP06', 'Rollback + ส่งเมลแจ้งล้มเหลว');
      return this.summarize(state, 'SUCCESS', startedAt);
    } catch (error) {
      // TODO: error path ของ Job 3 — กันซ้ำระดับงวดเท่านั้น — ไม่ใช่ upsert (E15)
      this.logger.error(JSON.stringify({ event: 'job.failed', jobNo: '3', period: ctx.period,
        triggeredBy: ctx.triggeredBy, durationMs: Date.now() - startedAt, error: (error as Error).message }));
      // TODO: แจ้งผู้ดูแลผาน JobFailureNotifier (หัวข้อ 9.6.1) — runner เป็นผู้เรียกให้
      throw error;
    }
  }

  private summarize(state: JobState, status: JobRunResult['status'], startedAt = Date.now()): JobRunResult {
    // TODO: structured log บรรทัดเดียวจบ — ไม่มีตาราง job_run_histories แล้ว (2026-08-06)
    const summary = {
      event: 'job.finish', jobNo: '3', jobName: 'ImportImpactCompetitor', status,
      period: state.period, output: 'fgi_impact_competitors',
      read: state.read, written: state.written, skipped: state.skipped,
      rejected: state.rejected, durationMs: Date.now() - startedAt,
    };
    this.logger.log(JSON.stringify(summary));
    return summary as JobRunResult;
  }
}
```

9.4 การกันรันซ้อนของ Job 3 (PostgreSQL advisory lock)

Job 3 มีข้อควรระวังจาก legacy: กันซ้ำระดับงวดเท่านั้น — ไม่ใช่ upsert (E15) — runner ล็อกด้วย `pg_try_advisory_lock` ก่อนเริ่มขั้นแรกเสมอ และรอบที่ล็อกไม่ได้ให้จบด้วยสถานะ SKIPPED_LOCKED (ไม่ใช่ FAILED)

```
// src/batch/runner.ts (ส่วนกันรันซ้อน)
import { Inject, Injectable, Logger } from '@nestjs/common';
import type { DataSource } from 'typeorm';

// TODO: ห้ามใช้แถวสถานะ RUNNING ในตารางเป็นตัวกัน (ไม่มีตาราง job_run_histories แล้ว)
// ใช้ PostgreSQL advisory lock ระดับ session แทน — ปลดอัตโนมัติเมื่อ connection หลุด
export const SBPGI_JOB_LOCK_CLASS_ID = 861000; // namespace ของระบบ SBPGI
export const JOB_LOCK_KEYS: Record<string, number> = { '3': 30 /* TODO: เพิ่มครบทั้ง 11 job */ };

@Injectable()
export class BatchRunner {
  private readonly logger = new Logger(BatchRunner.name);
  constructor(@Inject('DATA_SOURCE') private readonly dataSource: DataSource) {}
```

```
  async runExclusive<T>(jobNo: string, fn: () => Promise<T>): Promise<T | { status: 'SKIPPED_LOCKED' }> {
    // TODO: ต้องใช้ QueryRunner (connection เดียวบน master) — dataSource.query() ของโปรเจกต์นี้
    // route SQL ที่ขึ้นต้นด้วย SELECT ไป slave pool ทำให้ lock ไปตกที่ replica คนละ connection
    const runner = this.dataSource.createQueryRunner('master');
    await runner.connect();
    const objectId = JOB_LOCK_KEYS[jobNo];
    try {
      const [{ locked }] = await runner.query(
        'SELECT pg_try_advisory_lock($1, $2) AS locked',
        [SBPGI_JOB_LOCK_CLASS_ID, objectId],
      );
      if (!locked) {
        // TODO: รอบนี้ข้ามไปเฉย ๆ ไม่ถือเป็น error และไม่ต้องส่งอีเมล
        this.logger.warn(JSON.stringify({ event: 'job.skipped.locked', jobNo }));
        return { status: 'SKIPPED_LOCKED' };
      }
      return await fn();
    } finally {
      // TODO: ปลด lock ทุกกรณี แล้วคืน connection เข้า pool
      await runner.query('SELECT pg_advisory_unlock($1, $2)', [SBPGI_JOB_LOCK_CLASS_ID, objectId]);
      await runner.release();
    }
  }
}
```

9.5 Repository / SQL หลักของ Job 3

repository ของ Job 3 ประกาศเป็น factory provider (`{provide: 'IMPORT_IMPACT_COMPETITOR_REPOSITORY', useFactory: (ds) => ds.getRepository(Entity), inject: ['DATA_SOURCE']}`) แล้วยิง raw SQL ตามแบบ module ธุรกิจอื่นของ store-backend (schema `sps_store` มาจาก search_path)

ตาราง	R/W	การใช้งานตามผัง	หมายเหตุ target design
fgi_impact_competitors	W	insert รายงวด (งวดล่าสุดต่อร้าน) ดึงจาก ALLMAP · ช่องทางต้นทาง ALM เก็บที่ fgi_impact_processes.datasource	เขียน SQL ตรงผ่าน DATA_SOURCE

```
-- Job 3 ImportImpactCompetitor — query หลักที่ต้อง implement
-- TODO: ทุก statement รันผ่าน DATA_SOURCE (SELECT ไป slave, write ไป master) และ
-- write ทั้งหมดต้องอยู่ใน transaction เดียวกับที่ระบุใน 9.3

-- [W] fgi_impact_competitors : insert รายงวด (งวดล่าสุดต่อร้าน) ดึงจาก ALLMAP · ช่องทางต้นทาง ALM เก็บที่ fgi_impact_processes.datasource
-- TODO: เติมคอลัมน์ payload จริงจาก Database design
INSERT INTO fgi_impact_competitors
  (/* TODO: business key + payload + created_by, created_at */)
VALUES (/* TODO: bind params ตามลำดับคอลัมน์ด้านบน */)
ON CONFLICT (impact_process_id, competitor_code, period_key) -- unique key จริงตาม DDL ของ fgi_impact_competitors (ห้ามเดา)
DO UPDATE SET /* TODO: คอลัมน์ที่ยอมให้ทับ */
      updated_at = NOW(), updated_by = 'JOB3';
```

9.6 การแจ้งเตือนและการรันซ้ำของ Job 3

9.6.1 อีเมลแจ้งผู้ดูแลเมื่อ job ล้มเหลว

ใช้ `EmailLibService` จาก `@gosoft-sbp/email-lib` ตัวเดียวกับที่ระบบเดิมใช้ (inform-evaluate / external-audit / statement PTT) — ไม่สร้างกลไกส่งเมลใหม่

```
// src/batch/job-failure.notifier.ts
import { Injectable, Logger } from '@nestjs/common';
// ชื่อ method ของ lib ที่ store-backend เรียกจริงคือ `sendMail` (ไม่ใช่ sendEmail) และ
// `mailTo` / `mailCc` เป็น **string** คั่นด้วย comma — ดู evaluation-process.service.ts,
// external-audit.service.ts, statement.service.ts, inform-evaluate.service.ts, performance.service.ts
import { EmailLibService } from '@gosoft-sbp/email-lib';
import type { JobRunContext } from './runner';

@Injectable()
export class JobFailureNotifier {
  private readonly logger = new Logger(JobFailureNotifier.name);
  // TODO: ใช้ lib อีเมลของระบบเดิม — template อยู่ในตาราง email_template และ log ลง email_sent อัตโนมัติ
  // (ตั้งชื่อ property ว่า mailService ตาม call site เดิมทุกที่ใน store-backend)
  constructor(private readonly mailService: EmailLibService) {}

  async notifyFailure(jobNo: string, ctx: JobRunContext, error: Error): Promise<void> {
    // TODO: ผู้รับของ Job 3 เดิมคือ config mailTo / storeretention — ย้ายมาเป็น env SBPGI_JOB3_MAIL_TO
    const recipients = (process.env.SBPGI_JOB3_MAIL_TO ?? '').split(',').map((s) => s.trim()).filter(Boolean);
    if (!recipients.length) {
      this.logger.warn(JSON.stringify({ event: 'job.mail.skipped', jobNo, reason: 'NO_RECIPIENT' }));
      return;
```

```
    }
    try {
      await this.mailService.sendMail({
        // TODO: emailId = id ของ template EM-07 (แจ้ง error batch) ในตาราง email_template
        emailId: Number(process.env.SBPGI_JOB_FAIL_EMAIL_TEMPLATE_ID),
        mailTo: recipients.join(','), // signature รับ string ไม่ใช่ string[]
        mailCc: '',
        param: {
          jobNo, jobName: 'ImportImpactCompetitor',
          jobTitle: 'นำเข้าร้านคู่แข่งจาก ALLMAP',
          period: ctx.period, triggeredBy: ctx.triggeredBy,
          output: 'fgi_impact_competitors',
          errorMessage: error.message,
          rerunNote: 'ต้องลบข้อมูลงวดเองก่อน re-import แล้วตรวจจำนวนแถวเทียบต้นทาง',
        },
      });
    } catch (mailError) {
      // TODO: ส่งเมลไม่สำเร็จห้ามกลบ error เดิมของ job — log แล้วปล่อยผ่าน
      this.logger.error(JSON.stringify({ event: 'job.mail.failed', jobNo, error: (mailError as Error).message }));
    }
  }
}
```

9.6.2 Checklist การ rerun

- กติกา rerun ของ Job 3: ต้องลบข้อมูลงวดเองก่อน re-import แล้วตรวจจำนวนแถวเทียบต้นทาง
- ขอบเขต transaction ที่ต้องรักษาเมื่อรันซ้ำ: หนึ่ง transaction + savepoint (insert เป็น chunk ละ 10,000)
- ความเสี่ยงที่ต้องตรวจก่อน/หลังรันซ้ำ: กันซ้ำระดับงวดเท่านั้น — ไม่ใช่ upsert (E15)
- ตรวจว่ารอบก่อนหน้าไม่ได้ค้าง lock อยู่ (`SELECT * FROM pg_locks WHERE locktype = 'advisory'`) ก่อนสั่งรันนอกรอบ
- สั่งรันนอกรอบผ่าน CLI/runbook เท่านั้น (ไม่มีหน้าจอและไม่มี Job Admin API): `node dist/batch/cli.js --job=3 --period=<YYYYMM>`
- หลังรันซ้ำ ตรวจ output `fgi_impact_competitors` และ log บรรทัด `job.finish` ว่า read/written/skipped/rejected ตรงกับที่คาด
- ถ้ารอบก่อนล้มเหลวกลางทาง ตรวจ `interface_transactions` ของงวดนั้นว่ามีแถวค้างสถานะ READY/PENDING หรือไม่ ก่อนสั่งรันใหม่

10. Processing Flow

Step	Description
1	เริ่ม
2	เป็นงวดใหม่ (ยังไม่เคยนำเข้า)? | No: ข้ามทั้งงวด — ไม่มี upsert ต้องลบงวดก่อนจึงนำเข้าใหม่ได้ (กันซ้ำระดับงวด (Errata E15))
3	SELECT DISTINCT จาก COMPETITOR_IMPACT_VIEW
4	พบข้อมูลต้นทาง? | No: จบการทำงาน
5	insert ทีละ 10,000 แถว (ผูก impact_process_id) (ช่องทางต้นทาง ALM เก็บที่ fgi_impact_processes.datasource — fgi_impact_competitors ไม่มีคอลัมน์นี้ · map คอลัมน์ NAMT → name_th และ BRANCHT → branch_th (NAMT/BRANCHT เป็นคอลัมน์ของวิวฝั่ง ALLMAP))
6	จำนวนที่ insert = จำนวนต้นทาง? | No: Rollback + ส่งเมลแจ้งล้มเหลว (ตรวจ reconcile จำนวนแถวก่อน commit)
7	Commit / จบ

11. Acceptance Criteria

- พารามิเตอร์และ cron อ่านจาก backend config เท่านั้น — เปลี่ยนค่าโดย deploy config ไม่ใช่ผ่าน API/หน้าจอ
- การรันต้องตรวจ enabled flag ใน config และกันรันซ้อนด้วย distributed/advisory lock
- ทุกรอบต้องเขียน application log แบบ structured (เวลา/แถว/ไฟล์/ผล) และ error ต้องส่ง EM-07
- DB/table mapping ใช้เป็น reference สำหรับ implement Job เท่านั้น ไม่ใช่งานสร้างหน้า Database
- รองรับ rerun rule และ risk note ตาม runbook

12. Developer Test Checklist

No	Test
1	รันตามตารางเวลาแล้วผลถูกต้องบน fixture
2	รันนอกรอบผ่าน CLI ได้ผลเดียวกับ cron
3	สั่งรันซ้อนขณะกำลังรัน → runner ปฏิเสธ (lock ทำงาน)
4	แก้ config แล้ว deploy → รอบถัดไปใช้ค่าใหม่
5	job throw error → EM-07 ออก และ log มีบรรทัด error
6	ตรวจผลกระทบตารางตาม R/W mapping reference

13. Unit Test Scope

3 ชั่วโมง (30% ของ implementation 10 ชั่วโมง) · เครื่องมือ: Jest + mock repository/DataSource (ไม่ต่อ DB จริง)

หัวข้อนี้คือ unit test ที่ต้องเขียนคู่กับโค้ด — ต่างจาก *Developer Test Checklist* ซึ่งเป็น scenario ระดับ end-to-end/manual ที่ใช้ตอนตรวจรับ · รายการด้านล่าง derive จาก field/validation, acceptance criteria, endpoint และตารางที่เอกสารนี้เขียน

สิ่งที่ทดสอบ	ประเภท	เกณฑ์ผ่าน
business rule	logic	พารามิเตอร์และ cron อ่านจาก backend config เท่านั้น — เปลี่ยนค่าโดย deploy config ไม่ใช่ผ่าน API/หน้าจอ
business rule	logic	การรันต้องตรวจ enabled flag ใน config และกันรันซ้อนด้วย distributed/advisory lock
business rule	logic	ทุกรอบต้องเขียน application log แบบ structured (เวลา/แถว/ไฟล์/ผล) และ error ต้องส่ง EM-07
business rule	logic	DB/table mapping ใช้เป็น reference สำหรับ implement Job เท่านั้น ไม่ใช่งานสร้างหน้า Database
business rule	logic	รองรับ rerun rule และ risk note ตาม runbook
`fgi_impact_competitors`	transaction	จำลอง error กลางทาง แล้วยืนยันว่า rollback ครบ ไม่เหลือแถวค้าง (mock DataSource/QueryRunner)
runner	idempotency	รันซ้ำด้วย fixture เดิมต้องไม่เกิดแถวซ้ำ (ON CONFLICT / business unique key ทำงาน)
runner	lock	เรียกซ้อนขณะกำลังรัน ต้องถูกปฏิเสธด้วย advisory lock

- ทุกเคสต้องรันได้โดยไม่ต่อ DB/บริการภายนอกจริง — mock ที่ขอบ repository/client เสมอ
- ข้อความไทยที่ยืนยันในเทสต้องเป็น verbatim ตาม ข้อกำหนดทางธุรกิจ ห้ามพิมพ์ใหม่
- เกณฑ์ผ่านของ CI: ทุกเคสในตารางนี้มี test จริงและผ่านทั้งหมด

ภาคผนวก: Java Source เดิม

Code ในส่วนนี้คัดจาก Java source เดิมโดยตรงและคงข้อความตามต้นฉบับ ใช้เลขบรรทัดที่ระบุหน้าทุก snippet เพื่อตรวจเทียบ business behavior, SQL, transaction และ error handling

J1. ImportImpactCompetitor.java — lines 16-27

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/main/ImportImpactCompetitor.java
Original lines	16-27
Responsibility	Legacy main entrypoint and notification wrapper.

```
public class ImportImpactCompetitor {

private static ApplicationContext context = new FileSystemXmlApplicationContext("config.xml");
private static ImportController importController = (ImportController) context.getBean("importController");
private static FgiConfig config = (FgiConfig) context.getBean("fgiConfig");

public static void main(String[] args) {
EmailTemplateInformStatusBean informBean = new EmailTemplateInformStatusBean();
try{
LogUtils.info(ImportImpactCompetitor.class,"Start => import FGI_IMPACT_COMPETITOR");
informBean.setStartDateTime(Calendar.getInstance(Locale.US).getTime());
informBean.setJobName(FgiConstant.JOB_NAME_IMPORT_IMPACT_COMPETITOR);
```

J1. ImportImpactCompetitor.java — lines 37-48

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/main/ImportImpactCompetitor.java
Original lines	37-48
Responsibility	Legacy main entrypoint and notification wrapper.

```
informBean.setErrorMsg(ex.toString());
informBean.setEndDateTime(Calendar.getInstance(Locale.US).getTime());

SendMailUtil.sendMailToAdmin(informBean, config, true);
} catch (Exception e){
LogUtils.error(ImportImpactCompetitor.class,e);

}
}
LogUtils.info(ImportImpactCompetitor.class,"End => import FGI_IMPACT_COMPETITOR ");
}
}
```

J2. ImportController.java — lines 483-494

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/controller/ImportController.java
Original lines	483-494
Responsibility	Validate params, skip duplicates, query source, chunk insert competitors.

```
public void importImpactCompetitor(String[] params,EmailTemplateInformStatusBean informBean) throws Exception{
String month = new String();
String year = new String();
String[] zoneArr = {};
String[] paramsArr = {};
boolean flagSendMail = true;

if(params.length > 0){

paramsArr = params[0].split("\\|",-1);

if(paramsArr.length == 2 && !StringUtils.isEmpty(paramsArr[0]) && !StringUtils.isEmpty(paramsArr[1])){
```

J2. ImportController.java — lines 587-598

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/controller/ImportController.java
Original lines	587-598
Responsibility	Validate params, skip duplicates, query source, chunk insert competitors.

```
});
if(totalRecordInsert == impactCompetitorList.size() && flagSendMail){ // insert == select
informBean.setStatus(FgiConstant.JOB_STATUS_SUCCESS);
informBean.setEndDateTime(Calendar.getInstance(Locale.US).getTime());
SendMailUtil.sendMailToAdmin(informBean, config, true);
}else if(totalRecordInsert != impactCompetitorList.size() && flagSendMail){
informBean.setStatus(FgiConstant.JOB_STATUS_FAIL);
informBean.setNote("totalRecordInsert != impactCompetitorList.size()");
informBean.setEndDateTime(Calendar.getInstance(Locale.US).getTime());
SendMailUtil.sendMailToAdmin(informBean, config, true);
}
}
```

J3. ImportJdbc.java — lines 200-211

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/dao/jdbc/ImportJdbc.java
Original lines	200-211
Responsibility	Count existing period, query COMPETITOR_IMPACT_VIEW, insert FGI_IMPACT_COMPETITOR.

```
public int countRowCompetitor(String year, String month,String[] zoneArr){
ArrayList<String> params = new ArrayList<String>();
String zoneStr = new String();
params.add(year);
params.add(month);
String sql = "select count(1) count_rec from fgi_impact_competitor where year = ? and month = ? ";

/*if(zoneArr != null && zoneArr.length > 0){
zoneStr = TextUtils.arrayElementToStringHasQuote(zoneArr);
sql += " AND ZONE_CODE in ("+zoneStr+")";
}*/
```

J3. ImportJdbc.java — lines 230-241

รายการ	รายละเอียด
Source file	fcsJar/src/th/co/gosoft/fgi/dao/jdbc/ImportJdbc.java
Original lines	230-241
Responsibility	Count existing period, query COMPETITOR_IMPACT_VIEW, insert FGI_IMPACT_COMPETITOR.

```
return ImpactCompetitorList;
}

public int[] insertImpactCompetitor(List<ImpactCompetitorBean> impactCompetitorList){
StringBuffer sql = new StringBuffer();
sql.append(" INSERT INTO FGI_IMPACT_COMPETITOR");
sql.append(" (STORECODE_I,COMPET_ID,NAME_TH,NAME,BRANCH_TH,ZONE_CODE,SUBZONE_CODE,OPEN_DATE,CLOSE_DATE,MONTH,YEAR,DATAS
OURCE,IMPACT_COMPETITOR_ID)");
sql.append(" VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,SEQ_FGI_IMPACT_COMPETITOR.NEXTVAL)");
int[] result = jdbcTemplate.batchUpdate(sql.toString(),new ImpactCompetitorBatch(impactCompetitorList));
LogUtils.info(getClass(),"insert FGI_IMPACT_COMPETITOR : " + result.length + " Record (s)");
return result;
}
```