UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de programas en COBOL para la mejora de reducción de tiempo y eficiencia energética

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero Mecánico

P R E S E N T A

Luis Antonio García Venado

ASESOR DE INFORME

M. en I. Mariano García Del Gállego

Ciudad Universitaria, Cd. Mx., 2025

PROTESTA UNIVERSITARIA DE INTEGRIDAD Y HONESTIDAD ACADÉMICA Y PROFESIONAL
(Titulación con trabajo escrito)

De conformidad con lo dispuesto en los artículos 87, fracción V, del Estatuto General, 68, primer párrafo, del Reglamento General de Estudios Universitarios y 26, fracción I, y 35 del Reglamento General de Exámenes, me comprometo en todo tiempo a honrar a la institución y a cumplir con los principios establecidos en el Código de Ética de la Universidad Nacional Autónoma de México, especialmente con los de integridad y honestidad académica.

De acuerdo con lo anterior, manifiesto que el trabajo escrito titulado DESARROLLO DE PROGRAMAS EN COBOL PARA LA MEJORA DE REDUCCION DE TIEMPO Y EFICIENCIA ENERGETICA que presenté para obtener el titulo de INGENIERO MECÁNICO es original, de mi autoría y lo realicé con el rigor metodológico exigido por mi Entidad Académica, citando las fuentes de ideas, textos, imágenes, gráficos u otro tipo de obras empleadas para su desarrollo.

En consecuencia, acepto que la falta de cumplimiento de las disposiciones reglamentarias y normativas de la Universidad, en particular las ya referidas en el Código de Ética, llevará a la nulidad de los actos de carácter académico administrativo del proceso de titulación.

LUIS ANTONIO GARCIA VENADO
Número de cuenta: 314300250

ÍNDICE

1. Introducción

En este informe compartí mi experiencia y actividades como desarrollador COBOL en INTEGRAP, donde estuve a cargo del análisis, modificación y mantenimiento de programas que operaban en un entorno mainframe, además, trabajé en conjunto con un equipo de desarrollo con el que asumí la responsabilidad de mantener y mejorar diversos procesos dentro del aplicativo.

Para el desarrollo de estos programas se involucraron diferentes factores, como la comprensión de los requerimientos del sistema, la organización clara del código, la selección de consultas y rutinas eficientes, y la interacción con bases de datos. A partir de estas consideraciones, se plantearon y ejecutaron distintas maneras de optimizar todas las partes involucradas, lo que permitió crear un proceso más eficiente y, de manera implícita, redujo el consumo de energía.

A pesar de que en este tipo de procesos de mantenimiento el consumo de energía no se consideraba directamente ni se traducía en un beneficio inmediato, reconocí su importancia al implementar mejoras que redujeron el uso de recursos computacionales, con el fin de que estas prácticas pudieran aplicarse en otras instancias y sirvieran para concientizar sobre la relevancia del factor energético en la eficiencia de los sistemas.

2. Objetivo

El objetivo de este trabajo fue evaluar la contribución de las mejoras a la reducción del impacto ambiental, tomando en cuenta las acciones que llevé a cabo para optimizar el desempeño de los programas en los que participé, lo que permitió resaltar cómo a partir de un análisis técnico, donde apliqué mi experiencia profesional para abordar problemáticas, se lograron identificar oportunidades de mejora. Se describió el proceso seguido desde la detección de necesidades hasta la implementación de estrategias de optimización, mostrando los

resultados logrados en tres aspectos clave: la reducción de los tiempos de ejecución, la disminución en el tamaño del código y la eficiencia energética.

Asimismo, planteé la necesidad de medir y evidenciar cómo las acciones realizadas durante el mantenimiento y optimización de los programas influyeron en la reducción de emisiones de carbono, proporcionando un indicador tangible del efecto ambiental de las mejoras implementadas. Este objetivo permitió combinar la eficiencia técnica con la sostenibilidad, otorgando un sentido integral al trabajo realizado.

3. Antecedentes

3.1 Contexto de ineficiencia en sistemas COBOL (orientado a rutinas y programas ineficientes e innecesarios por consumo de recursos)

Los procesos y sistemas implementados en COBOL, acrónimo de Common Business-Oriented Language, han sido ampliamente utilizados y son fundamentales para los sectores financieros, salud y empresariales[1]. Desde su diseño inicial, se enfocó en la portabilidad y la capacidad de procesar grandes volúmenes de datos comerciales de manera muy eficiente. Esto permitió abrirle camino a sectores como la banca, instituciones de salud y operaciones empresariales.

> A medida que surgen nuevas tecnologías y captan la atención pública, con la misma rapidez suelen quedar obsoletas frente a desarrollos aún más recientes. Sin embargo, hoy, como en cada década desde 1960, las computadoras mainframe y el estilo de cómputo que representan siguen dominando el panorama de la informática empresarial a gran escala.[2]

---

[1](IBM, s. f.).

[2] Página 4 (Redbooks, 2006)

Su sintaxis cercana al lenguaje natural en inglés y su robustez facilitaron su uso en la informática, para tareas administrativas y comerciales. Gracias a ello, se consolidó como el estándar para el procesamiento de datos comerciales durante varias décadas.

El sistema operativo que integraban estas computadoras de IBM era z/OS, el cual estaba diseñado para la gestión de operaciones con alta demanda de recursos, incluyendo funciones avanzadas para la gestión de memoria, ejecución de procesos de entradas y salidas (I/O). Sin embargo, solo se aprovechaba al máximo si el código subyacente estaba bien optimizado o si las rutinas utilizaban estructuras de control y acceso a los datos de manera eficiente.

Con el paso del tiempo, muchos sistemas desarrollados en COBOL fueron cambiando y creciendo sin que existiera una estrategia clara para optimizarlos. Esto llevó a que se acumulara código que ya no aportaba valor, pero que seguía presente. Cada vez que surgía una nueva necesidad operativa, se hacían ajustes sobre la marcha, lo que, sumado a la evolución constante de herramientas y al aumento del volumen de datos, volvía más complicado darles mantenimiento.

En ese contexto, los mainframes que se encargaban de ejecutar estos programas continuaron trabajando con aplicaciones que no habían sido actualizadas en años. Dicha falta de actualización no solo ralentizaba procesos importantes, sino que también incrementaba el uso de recursos y energía. En consecuencia, se elevaron los costos operativos y se redujo la eficiencia general del sistema.

3.2 Necesidad de optimizar tiempos y recursos

Al ser un lenguaje de programación de finales de la década de 1950, las aplicaciones COBOL fueron actualizadas para incorporar nuevas herramientas, como Db2, CICS, MVS, con el fin de que se adecuaran a las demandas de la operatividad, algo que no sucedió a nivel de los programas en la institución en la cual laboré. Las modificaciones se realizaron, en muchos casos, sin una planificación estructurada o un planteamiento claro, lo que se convirtió en un

factor crítico que afectaba directamente el rendimiento general del sistema, dando como resultado la existencia de acumulación o repetición de código a estructuras antiguas.

Este desajuste entre la evolución del hardware, software del mainframe y el código COBOL existente generó un entorno donde el procesamiento de datos era más lento. La falta de optimización, procesamiento de datos y reutilización de código, provocaban incertidumbre y cuestionamientos sobre si el hecho de ser un lenguaje tan antiguo lo convertía en obsoleto.

La reutilización de código, que es una práctica común y entendible debido a la existencia de programas antiguos que se reutilizan, en muchos casos no se llevó a cabo de forma ordenada. Con frecuencia se mantenían fragmentos del código original y se añadían nuevas instrucciones sobre programas ya modificados en múltiples ocasiones, lo que generaba estructuras poco claras y difíciles de mantener. Además, durante la ejecución se procesaban líneas innecesarias de código.

4. Definición del problema o contexto de la participación profesional

Durante mi experiencia profesional en el desarrollo y mantenimiento de entornos del lenguaje, observé que en su gran mayoría estos habían acumulado ineficiencias a lo largo de los años debido a múltiples motivos; la falta de optimización en el código, desactualización, longevidad de los programas, accesos innecesarios a disco y código redundante, lo que generaba un uso excesivo de CPU y por consecuencia un uso ineficiente de recursos.

Como parte de mis responsabilidades en la empresa en que laboré, me correspondía analizar, modificar y reestructurar código COBOL. Muchas veces, el principal objetivo no era hacerlo más eficiente o reducir los tiempos de ejecución, sino garantizar que los programas funcionaran correctamente de acuerdo con las necesidades que me eran solicitadas. En resumen, solo debía cumplir con los requerimientos funcionales; la optimización del código no era un requisito obligatorio.

Se me asignaron varios mantenimientos, en los cuales pude notar que en algunos programas se repetían rutinas de acceso a tablas. Esto era completamente normal debido a que muchos programas utilizaban información específica de alguna tabla; sin embargo, lo que no era habitual era su redundancia. En un entorno mainframe era habitual que los procesos sean largos debido al alto volumen de registros a procesar, hablamos de millones de registros, y a la complejidad de las operaciones involucradas.

En una de las tareas de mantenimiento que realicé, surgieron oportunidades para mejorar el rendimiento del código. Al revisar accesos a base de datos, analicé junto con mi equipo de trabajo, que ciertas rutinas se repetían en varios programas, circunstancia que prolongaba los tiempos de ejecución y aumentaba el uso de recursos del mainframe. Se decidió, en conjunto, implementar ajustes para optimizar la lógica del código y reducir el número de accesos a disco.

En el caso específico que se abordó en este informe, el problema se presentó en el proceso de pagos a beneficios externos. La tabla de consulta de datos para esta tarea era muy extensa, ya que contenía información detallada de cada beneficiario, los montos a pagar, las fechas de pago, datos confidenciales, administrativos y encriptados, alrededor de ciento veinticinco millones de registros. Debido a esto y al diseño de las consultas utilizadas, el tiempo de procesamiento cada vez se volvía mayor, siendo cada vez más evidente en la lectura y consulta de registros. Mientras menos retrasos y errores existían, mejor era el desempeño.

4.1. Identificación de tiempos de procesamiento elevados

Un análisis de primer acercamiento para este proceso de pagos en específico reveló que el tiempo de procesamiento estaba directamente relacionado con el volumen de datos manejado, la estructura de las consultas SQL implementadas, el orden en el que eran ejecutados en un JCL (Job Control Language) y la convivencia con otros sistemas dentro del mainframe. Dado que los recursos eran compartidos, este último factor fue descartado en el proceso de optimización, ya que se trataba de aplicativos independientes.

En los programas involucrados en el proceso, la ejecución no estaba optimizada. Muchos de estos, además de utilizar consultas y actualizaciones a bases de datos, también procesaban archivos de lectura. En numerosas ocasiones se realizaban múltiples descargas de una tabla de bases de datos con campos relevantes para el proceso que utilizaba cada programa. En vez de tener una descarga simplificada, se contaba con una descarga adaptada a los requerimientos específicos de cada programa. Esto se relacionaba directamente con la forma en que se planeaba cada mantenimiento, debido a que al crearse un nuevo programa este buscaba cubrir ciertas solicitudes del cliente, pero no se tomaba en cuenta una buena optimización.

Este tipo de comportamientos generaba cargas innecesarias al sistema, ya que cada descarga implicaba un proceso adicional. Además, los programas eran ejecutados en secuencia dentro de un JCL, por lo que el retraso en un programa afectaba directamente el inicio del siguiente, provocando un efecto en cadena que extendía el tiempo total de procesamiento.

Los comandos SQL más utilizados para consultar bases de datos en entornos empresariales eran: SELECT, que permite recuperar datos de tablas, y UPDATE, que permitía modificar registros existentes. En muchos casos, las consultas no estaban optimizadas, por consecuencia llevaba a la realización de barridos completos a la tabla, haciendo que Db2 leyera todos las columnas y registros, incluso cuando solo se necesitaban datos específicos. Debido a estas ineficiencias, fue necesario analizar el comportamiento de las consultas y proponer estrategias para optimizar el acceso y la manipulación de los datos en Db2.

4.2 Mi participación en la mejora del rendimiento

Cuando detecté el problema, decidí involucrarme de lleno en mejorar la forma en que los programas accedían a la base de datos Db2. Me enfoqué principalmente en revisar cómo estaban construidas las consultas SQL y en la optimización de los JCL, sobre todo en el uso de tablas, ya que ahí se encontraba gran parte del cuello de botella: tiempos de respuesta altos y un uso innecesario de recursos como la CPU.

Lo primero fue entender qué información necesitaba realmente cada proceso. Muchas de las consultas usaban SELECT *, lo cual obligaba al sistema a traer todos los datos de una tabla, aunque solo se utilizaran unos cuantos campos. Cambié esa práctica, haciendo que las consultas fueran más precisas. También ajusté las condiciones en las cláusulas WHERE para que filtraran mejor los datos desde el principio y así aprovechara al máximo los índices ya definidos en las tablas.

Además de modificar el código de los programas, uno de los componentes clave con los que trabajé fue el programa IKJEFT01. Este programa funcionaba como una especie de intermediario o puente para ejecutar comandos TSO o sentencias SQL dentro del entorno batch del mainframe; es decir, no era que el mainframe no pudiera ejecutar SQL, sino que no lo hacía de forma directa en un JCL como se haría desde una terminal SQL tradicional. En lugar de eso, se necesitaba un intérprete, y ahí es donde entraba ese programa. Para esta tarea, lo utilicé con el propósito de ejecutar scripts, que no eran otra cosa que un miembro en el que se definían las instrucciones que el IKJEFT01 debía interpretar. Se emplearon comandos SQL que me permitieron hacer la descarga de la información directamente desde la base de datos Db2 hacia un archivo plano. Esta forma de extracción fue fundamental para evitar múltiples accesos innecesarios a la base de datos durante el procesamiento posterior.

Una vez descargados los datos en el archivo, utilicé el programa SORT, una herramienta muy útil y clásica en el entorno mainframe, que me permitió organizar los registros según ciertos criterios que me fueron requeridos previamente, por ejemplo, fechas o claves específicas. La ventaja era que se podía ordenar no solo por una columna, sino por varias.

Lo que buscaba era simple pero fundamental; entregarle al programa un archivo limpio y ya ordenado, con el fin de que pudiera leerse de principio a fin sin complicaciones, esto ayudaba al procesamiento dentro del programa, debido que encontrar un registro era más rápido. Para explicarlo mejor, es como si se tuvieran un montón de papeles desordenados sobre una mesa y se buscara uno con una fecha específica, lo más probable es que se tenga que revisar uno por uno hasta dar con el correcto. Este tipo de búsqueda se conoce como secuencial, tarda más porque no sabes dónde estará.

Ahora, si esos mismos papeles estuviesen ordenados por fechas de menor a mayor, se puede aplicar una especie de atajo. Si buscas una fecha intermedia, vas directo a la mitad de la pila y revisas, si la fecha que buscas esta al final, te vas a la parte final y así sucesivamente, a este tipo de búsqueda más rápida y ordenada se le conoce como búsqueda logarítmica.

La única desventaja era que en un archivo secuencial los registros estaban organizados uno tras otro, en un orden especifico, y se leían de principio a fin sin saltarse ninguno. Por ello, no se podía realizar una búsqueda logarítmica, como en un archivo VSAM. Sin embargo, si un registro se encontraba hasta el final, se leía todo el archivo; pero si el registro estaba al inicio o en medio, el programa podía comenzar a encontrar registros válidos de manera rápida, procesarlos y, una vez que terminaba, detenía la lectura. Entonces, aunque no era una búsqueda logarítmica, terminaba siendo eficiente.

4.3. Alcances del proyecto de optimización

Aunque el mantenimiento comenzó enfocado en el proceso de pagos a beneficios externos, terminó extendiéndose más allá de su objetivo inicial. Las soluciones que implementé junto con mi equipo también resultaron útiles en otros programas que presentaban problemas similares.

Como parte de los alcances del proyecto, se establecieron prácticas de programación más eficientes en programas batch-Db2, que sirvieron como referencia para el desarrollo posterior. Estas prácticas quedaron documentadas y adoptadas por el equipo como guía para futuros mantenimientos y nuevas iniciativas de optimización.

5. Metodología utilizada

5.1 Técnicas de optimización que utilicé en el código del JCL

Para entender cómo optimicé el proceso de ejecución de los programas mediante la descarga de un único archivo para la tabla de esta tarea, es necesario conocer cómo funciona el JCL, el

cual consiste en una serie de instrucciones o pasos, denominados steps, y pueden tener desde de uno hasta varios. Todos los programas COBOL deben de ejecutarse mediante un JCL, los pasos son los siguientes:

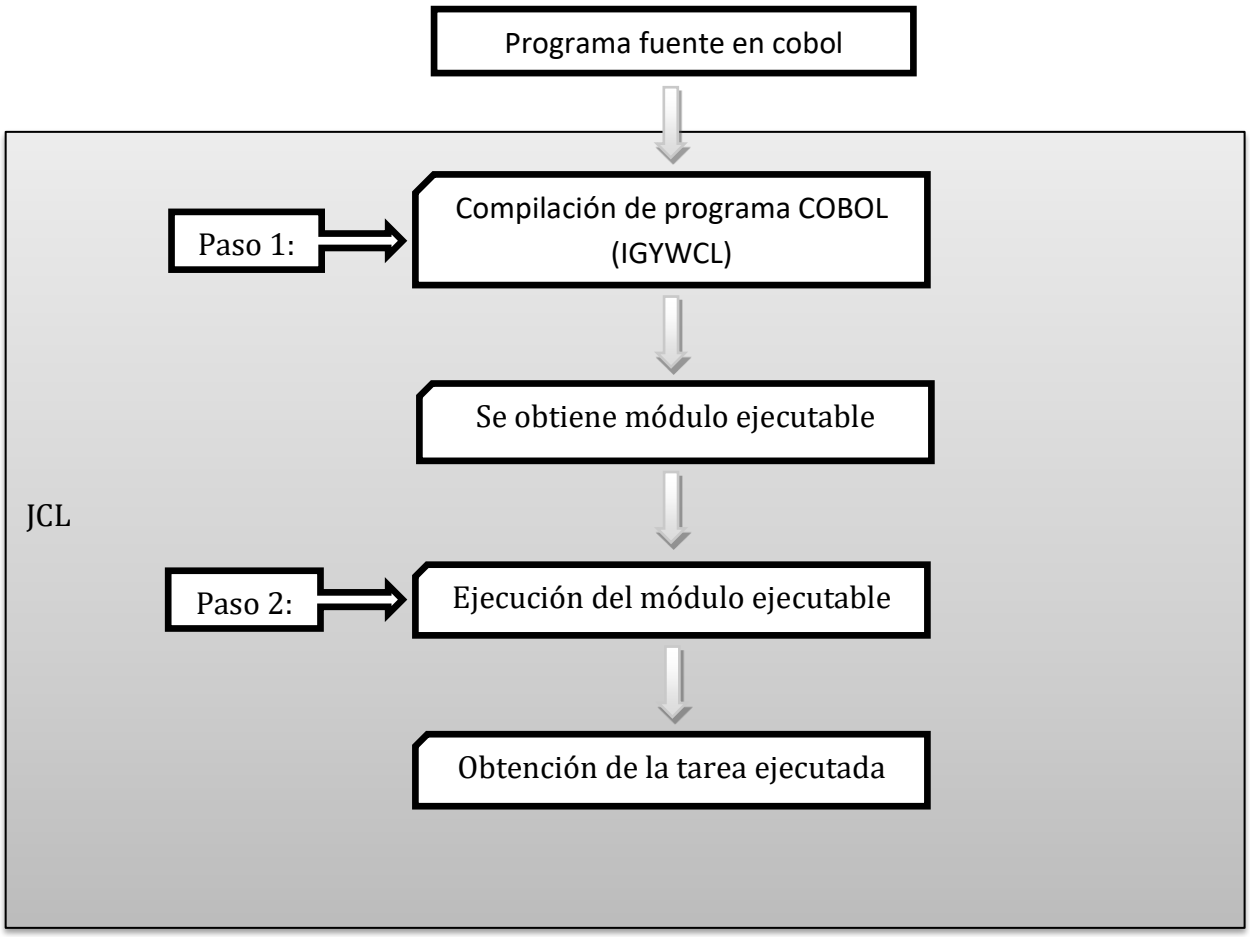


Figura 1. Modelo tradicional de ejecución de un programa batch por medio de un JCL.

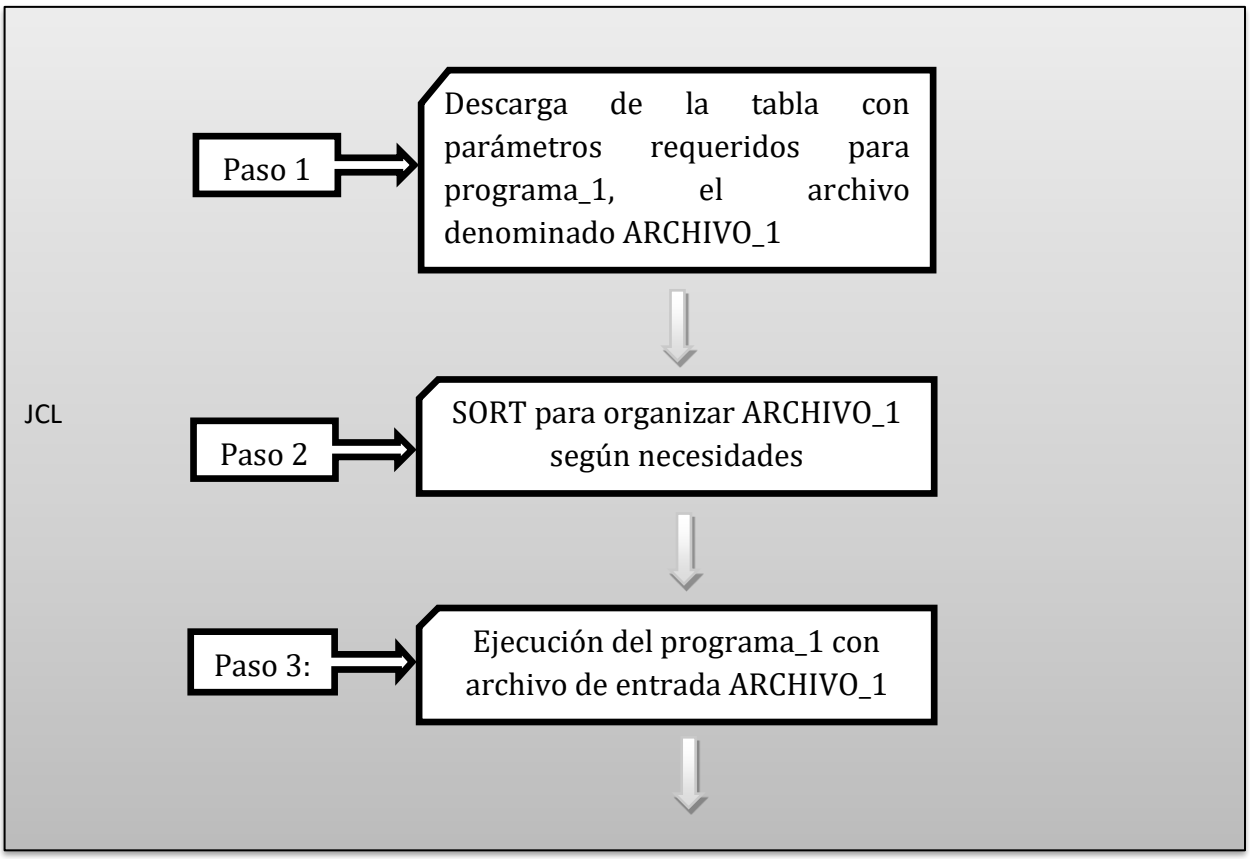
En la figura 1 se presentó un ejemplo común en el entorno batch. En el JCL se definen dos pasos, el primero compila un programa escrito en COBOL para generar su módulo ejecutable, y el segundo ejecuta dicho módulo una vez que ha sido creado con éxito. Esta forma de ejecutar es una práctica estándar en la automatización de procesos, donde es posible organizar toda la preparación y ejecución del programa sin intervención manual en cada etapa.

Es importante destacar que, aunque ambos pasos están contenidos dentro del mismo JCL, no pueden ejecutarse simultáneamente. Su naturaleza se los impide y obliga a que estos se

realicen uno tras otro, siguiendo un orden lógico, en algunos casos, uno de ellos puede omitirse si así lo indica la lógica del paso. Primero se debe asegurar que el programa haya sido compilado correctamente, sin errores, para que el módulo ejecutable esté disponible, de ahí se pasa al siguiente paso y así sucesivamente.

Este comportamiento secuencial además de garantizar la integridad del proceso, también permite una mayor localización en caso de errores, ya que cada paso deja registros de su ejecución. Fue en este punto donde encontré una de las aplicaciones directas a la resolución de conflictos, es decir, localizar y resolver los errores que llegan a surgir en estos entornos. Su flujo se volvía casi intuitivo, y entender la lógica detrás de este tipo de estructuras era clave para evitar problemas durante la ejecución de JCLs que incrementaban el número de pasos y se volvían más complejos.

A continuación, se explica la manera en la cual estaban organizados los pasos antes de que los optimizara:



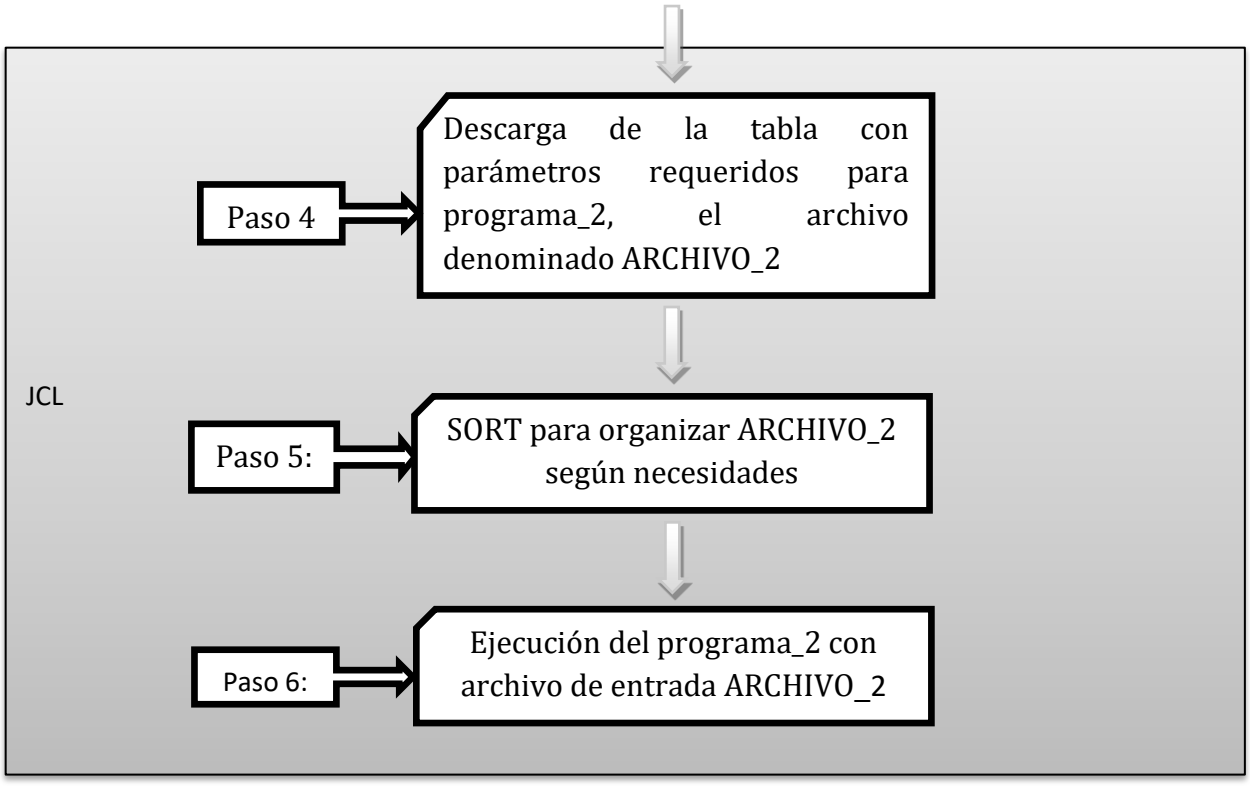
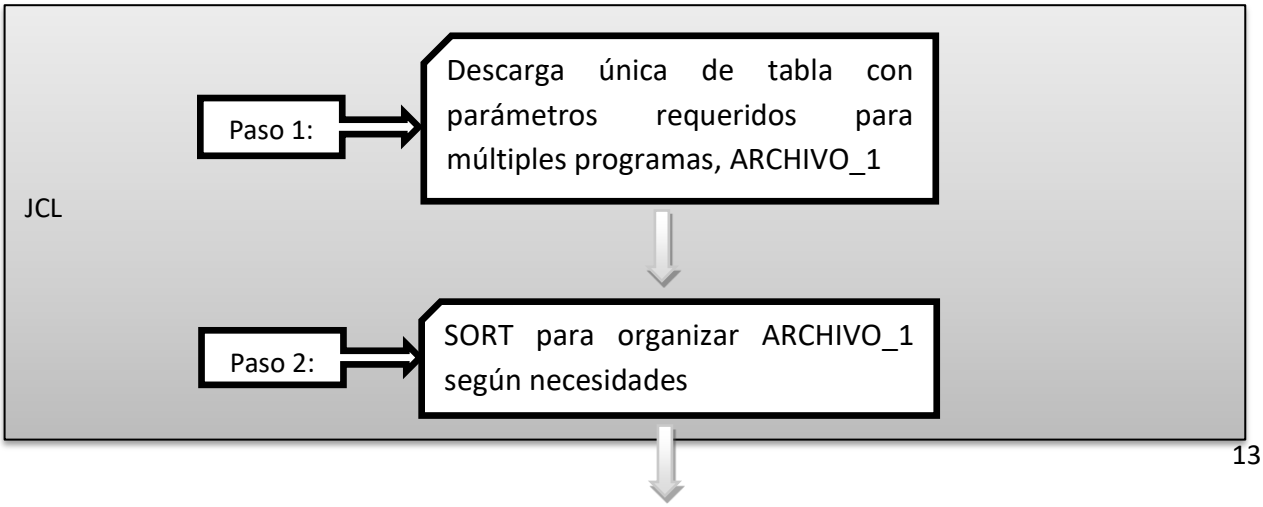


Figura 2. Flujo con procesamiento redundante por programa.

Antes de implementar las mejoras que realicé, como se muestra en la figura 2, el esquema de ejecución consistía en múltiples descargas desde la base de datos y aplicaciones de múltiples SORTs. Con el objetivo de hacer más eficiente la ejecución de estos pasos, reorganicé el flujo en el JCL de modo que la descarga de datos y su posterior ordenamiento se realizaran en dos únicos pasos. Esto con el fin de no ejecutar múltiples extracciones y procesos SORT individuales para cada rutina o subrutina, centralicé la descarga en un único paso que reuniera todas las columnas necesarias. A este archivo se le aplicó un SORT de ordenamiento con requisitos específicos. A continuación, se muestra como quedó el proceso.



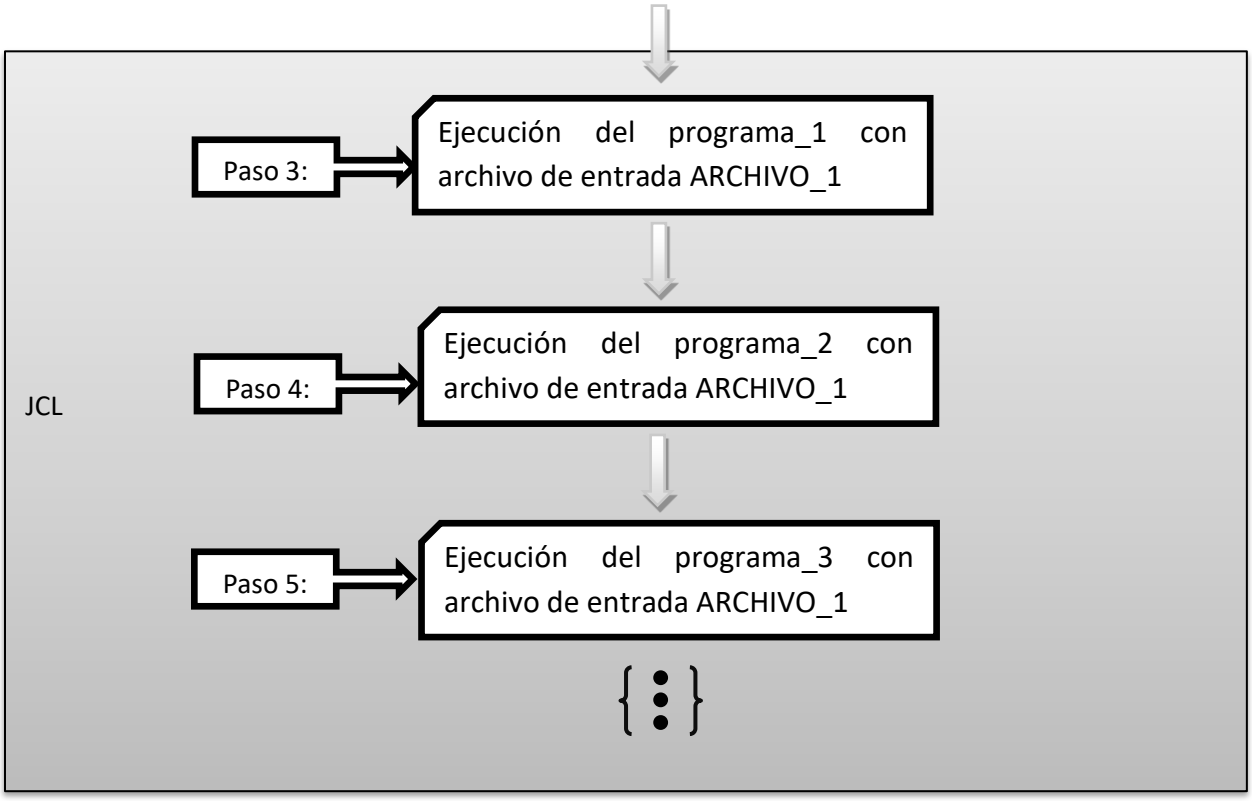


Figura 3. Flujo con procesamiento eficiente.

En la figura 3 se muestra de manera simplificada como realicé la modificación en el JCL. Esta solución permitió que, una vez generada, la descarga se pudiera utilizar de manera compartida por diversas rutinas o subrutinas, eliminando la redundancia y reduciendo significativamente el número de accesos al sistema de base de datos. Al tener un archivo previamente descargado, los programas ya no tenían que realizar búsquedas o descargas repetitivas o ajustes internos para encontrar la información que necesitaban.

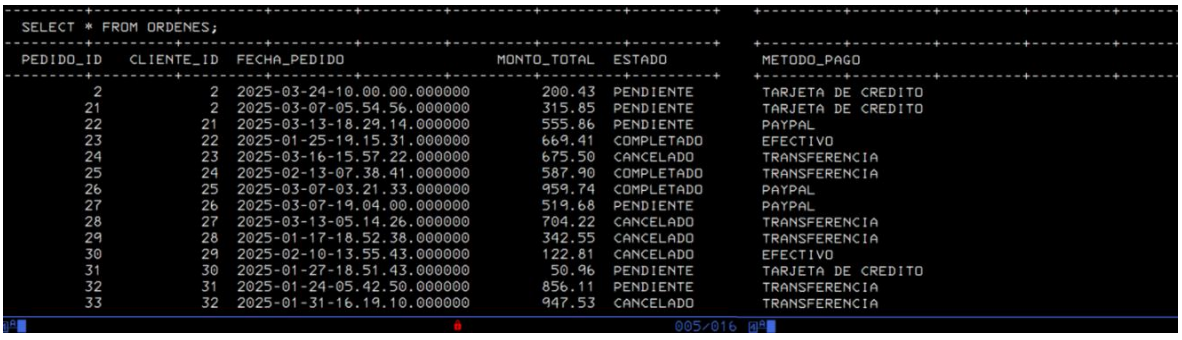
Una de las principales ventajas que trajo esta mejora fue resolver una limitación importante: en un entorno JCL, no era posible consultar la misma tabla varias veces de manera paralela, ya que una vez que una rutina accedía a ella, quedaba ocupada y bloqueada para los demás. Por esta razón, anteriormente era necesario realizar múltiples descargas para que cada programa tuviera su propia copia de los datos, lo cual obligaba a ejecutar los procesos de forma secuencial.

5.2 Técnicas de optimización que utilicé en el código embebido de Db2 para el programa COBOL

Una de las primeras acciones que llevé a cabo al empezar a mejorar el rendimiento del código embebido en Db2 fue revisar a fondo cada consulta SQL prestando atención a los SELECT. Aunque a simple vista pareciera algo básico, pequeños detalles en esas instrucciones pueden marcar diferencias importantes en términos de rendimiento.

Uno de los cambios más evidentes, pero también más efectivos, fue evitar el uso del SELECT con columnas innecesarias. Más allá de evitar el clásico SELECT *, lo que realmente retrasaba las ejecuciones era que muchas de esas sentencias traían columnas que no eran necesarias para la lógica del programa. Es decir, aunque el SELECT ya listaba campos específicos, a veces incluía datos que ni siquiera se usaban en ninguna parte del procesamiento.

A continuación, se realizó un ejemplo de mi autoría que tuvo como objetivo ejemplificar la recuperación innecesaria de datos en una consulta de SQL, y sirvió como caso explicativo para extrapolar al que se realizó en INTEGRAP. Se trató de un SELECT * de una tabla de nombre ORDENES que contenía las columnas PEDIDO_ID, CLIENTE_ID, FECHA_PEDIDO, MONTO_TOTAL, METODO_PAGO, una vez que se realizó la consulta, se obtuvo lo siguiente:



```
SELECT * FROM ORDENES;

PEDIDO_ID   CLIENTE_ID  FECHA_PEDIDO                MONTO_TOTAL  ESTADO       METODO_PAGO
---------+-----------+--------------------------+------------+------------+----------------
         2           2  2025-03-24-10.00.00.000000       200.43  PENDIENTE    TARJETA DE CREDITO
        21           2  2025-03-07-05.54.56.000000       315.85  PENDIENTE    TARJETA DE CREDITO
        22          21  2025-03-13-18.29.14.000000       555.86  PENDIENTE    PAYPAL
        23          22  2025-01-25-19.15.31.000000       669.41  COMPLETADO   EFECTIVO
        24          23  2025-03-16-15.57.22.000000       675.50  CANCELADO    TRANSFERENCIA
        25          24  2025-02-13-07.38.41.000000       587.90  COMPLETADO   TRANSFERENCIA
        26          25  2025-03-07-03.21.33.000000       959.74  COMPLETADO   PAYPAL
        27          26  2025-03-07-19.04.00.000000       519.68  PENDIENTE    PAYPAL
        28          27  2025-03-13-05.14.26.000000       704.22  CANCELADO    TRANSFERENCIA
        29          28  2025-01-17-18.52.38.000000       342.55  CANCELADO    TRANSFERENCIA
        30          29  2025-02-10-13.55.43.000000       122.81  CANCELADO    EFECTIVO
        31          30  2025-01-27-18.51.43.000000        50.96  PENDIENTE    TARJETA DE CREDITO
        32          31  2025-01-24-05.42.50.000000       856.11  PENDIENTE    TRANSFERENCIA
        33          32  2025-01-31-16.19.10.000000       947.53  CANCELADO    TRANSFERENCIA
```

Figura 4. Ejecución de una sentencia SELECT sobre la tabla ORDENES en Db2.

El uso de columnas innecesarias dentro de un SELECT puede parecer algo relativo, como se exhibe en la figura 5, ya que depende del propósito específico de cada rutina y de los datos que se manejan en ese momento. No siempre se accede a las mismas tablas ni se requieren los

mismos campos. Sin embargo, es posible identificar este patrón cuando se recuperan más columnas de las que realmente se utilizan en el programa. A continuación, presento un ejemplo que ilustra cómo puede verse este tipo de situación en la práctica, se solicitan todos los campos de la tabla, PEDIDO_ID, CLIENTE_ID, FECHA_PEDIDO, MONTO_TOTAL, METODO_PAGO y se introdujeron en variables temporales del programa esto con el fin de ser procesadas posteriormente.

La instrucción también incorporó una condición en el WHERE, filtrando los valores contenidos en la variable WS-PEDIDO-ID, La cual corresponde a la llave primaria, decisión que no fue arbitraria, ya que la consulta permite un acceso más eficiente a los registros, se realizó la buscada únicamente en esa columna, en lugar de revisar todas, reduciendo considerablemente el uso de recursos. Al enfocar la consulta en este campo en específico, el sistema pudo resolverla rápidamente utilizando los índices ya definidos para esa llave. Por otro lado, se implementó un control mediante el SQLCODE para verificar el resultado de la operación: si todo se ejecuta correctamente (valor de cero), el programa continuaba, si ocurre algún problema se lanzaba un mensaje de error y se derivaba a una rutina específica para su control de errores.

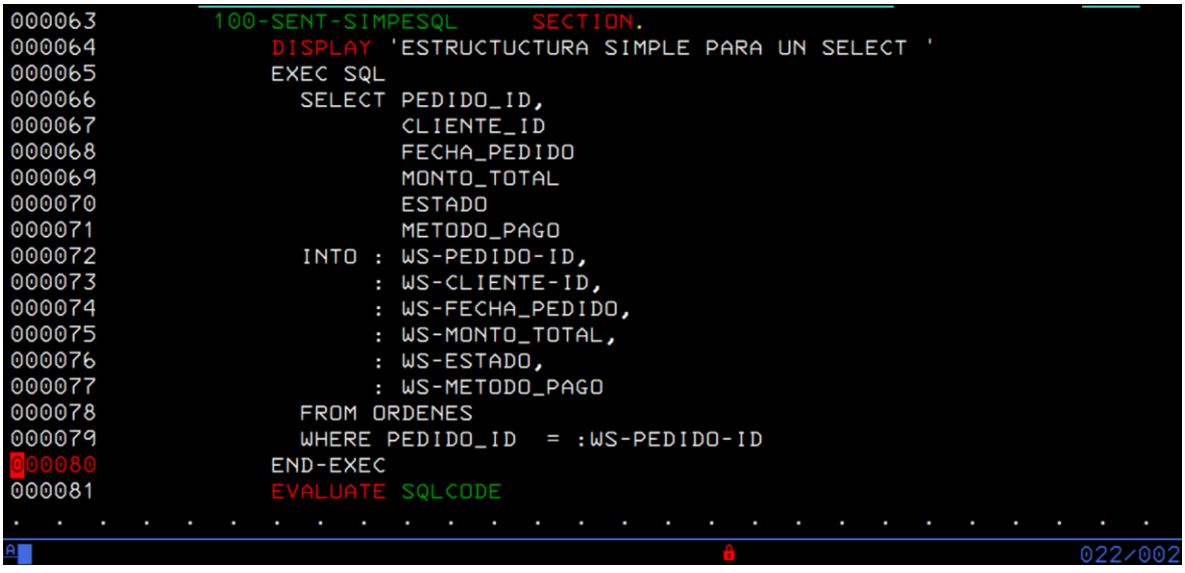


Figura 5. Consulta SQL embebida previa a la optimización en un programa COBOL

Después de que identifiqué que varias columnas recuperadas inicialmente no eran utilizadas en el procesamiento del programa, decidí replantear la estructura del SELECT. En lugar de traer todos los campos que podrían estar disponibles, enfoqué la consulta para qué recuperara únicamente aquellos datos que eran necesarios dentro de la lógica del programa o de las subrutinas llamadas por él; estos resultaron ser: MONTO_TOTAL y METODO_PAGO, como se muestra en la figura 6.

```
000073         100-SENT-SIMPESQL      SECTION.
000074             DISPLAY 'ESTRUCTUCTURA SIMPLE PARA UN SELECT '
000075             EXEC SQL
000076               SELECT MONTO_TOTAL
000077                      METODO_PAGO
000078               INTO : WS-MONTO-TOTAL,
000079                    : WS-METODO-PAGO
000080               FROM ORDENES
000081               WHERE PEDIDO_ID  = :WS-PEDIDO-ID
000082             END-EXEC
000083             EVALUATE SQLCODE
000084               WHEN  0
000085                DISPLAY 'SE HA REALIZADO EL SELECT SIN ERR :' SQLCODE
000086               WHEN  OTHER
000087                  DISPLAY 'ERR  PROBL. EN CONSULTA      '
000088                PERFORM 100-SENT-SIMPESQL-ERR
000089             END-EVALUATE
000090             EXIT.
```

Figura 6. Consulta SQL embebida posterior a la optimización en un programa COBOL

Al reducir la cantidad de columnas solicitadas, disminuyó la cantidad de datos transferidos desde Db2, lo cual también redujo el uso del búfer. En el contexto de la entrada/salida, un búfer es una memoria intermedia ubicada en la controladora del dispositivo que permite almacenar datos temporalmente durante la transferencia entre el dispositivo periférico y el sistema principal[3].

Se mantuvo el uso de la condición WHERE sobre PEDIDO_ID, que funcionaba como llave primaria, para este ejemplo solo se utilizó una sola variable, pero podía estar compuesta por más. Esto garantizó que la consulta se realizara de forma directa y precisa, accediendo al registro especifico sin tener que recorrer múltiples filas.

---

[3] Basado en la sección 1.2.3, página 10 (Silberschatz, Galvin & Gagne, 2018).

No solo optimicé el flujo de lectura de datos filtrados y la utilización el SELECT adecuadamente. También trabajé en las sentencias UPDATE que muchas veces pasaban desapercibidas en cuanto a su impacto en el rendimiento. Aquí me aseguré de que el WHERE estuviera bien acotado, utilizando llaves primarias o columnas con índices definidos. Actualizar sin filtrar adecuadamente no solo pone en riesgo la integridad de los datos, sino también podía bloquear registros innecesarios y generar esperas en otros procesos que accedían a la misma tabla.

Por ejemplo, en lugar de hacer un UPDATE general que recorriera toda la tabla, utilicé condiciones definidas, y si era posible, aproveché los índices existentes para que la búsqueda previa al UPDATE fuera más rápida. Procuré evitar hacer múltiples UPDATE sobre el mismo conjunto de datos dentro del mismo programa. En su lugar, estructuré la lógica para hacer una sola modificación precisa.

5.3 Evaluación de impacto

A lo largo del desarrollo de estas mejoras comprobé cómo las decisiones simples podían generar cambios razonablemente mejores en el comportamiento de los programas y, sobre todo, a nivel general. Al revisar y ajustar las sentencias SQL en COBOL, me di cuenta de que muchas de ellas podían ser más específicas. Al limitar las columnas seleccionadas a solo las necesarias, no solo se redujo la cantidad de datos procesados, sino que también mejoró el tiempo de respuesta en cada consulta. Esta reducción tuvo un impacto directo, especialmente en trabajos por lotes (batch-Db2), donde se procesaban grandes volúmenes de información.

Implementar estas prácticas de forma organizada y general me permitió tener un mayor control sobre los requerimientos específicos de los programas, especialmente aquellos que involucraban sobrecarga. De manera, el resultado fue efectivo: menos datos en tránsito, menos recursos ocupados y, por tanto, más eficiencia en la ejecución.

Enfoqué esta evaluación desde una perspectiva técnica y práctica, pues no se trataba únicamente de comprobar que el SELECT dejara de traer columnas, que la descarga del

archivo funcionara o que el acceso a tablas estuviera mejor indexado. Lo importante era identificar cuánto se reducía la carga operativa, qué tanto mejoraba el tiempo de respuesta, y cómo eso afectaba el desempeño general del entorno de ejecución.

Además, esta evaluación me permitió descubrir nuevos espacios de mejora. Por ejemplo, al consolidar las descargas de datos en un solo proceso compartido, no solo se eliminó la redundancia en la ejecución de JCLs, sino que se abrió la posibilidad de ejecutar múltiples programas en paralelo, una condición que no era viable bajo la estructura anterior. Este tipo de efectos colaterales positivos no siempre fueron evidentes desde el principio y solo salían a la luz cuando se observa el sistema ya funcionando bajo las nuevas condiciones.

6. Resultados

6.1. Reducción de tiempos de ejecución

Como parte del análisis posterior a la implementación de técnicas de optimización, decidí realizar una comparación directa entre la versión original del programa (encargado de ejecutar actualizaciones sobre una tabla a partir de datos leídos) y su versión optimizada. El programa también incluía sentencias SELECT hacia la misma tabla, tomando como principal entrada los datos contenidos en un archivo previamente descargado. En ambos casos, el objetivo era identificar oportunidades de mejora tanto en tiempo de ejecución como en eficiencia de acceso a la base de datos.

Para llevar a cabo esta evaluación, utilicé los registros del JES2 JOB LOG, que simplemente es un registro de trabajos[4] como evidencia para medir los tiempos exactos de inicio y finalización de ambos procesos. Esta evaluación me permitió cuantificar de forma objetiva el impacto de los cambios realizados.

---

[4] (IBM, s.f.).

En la figura 7, correspondiente a la versión original del programa, el tiempo total de ejecución fue de aproximadamente una hora con treinta y ocho minutos en completarse. Aunque este valor, visto de forma aislada, podía parecer aceptable en ciertas circunstancias, la ausencia de una referencia previa hacía difícil dimensionar su verdadera carga.

Fue hasta revisar la figura 8, correspondiente a la versión optimizada, que se pudo evidenciar una mejora significativa: el mismo programa, tras ser reestructurado, finalizó su ejecución en tan solo cuarenta y seis minutos.

La tabla siguiente resume los datos y evidencia la mejora en el tiempo de ejecución:

Versión del programa	**Hora de inicio**	**Hora de finalización**	**Tiempo total (hh:mm:ss)**	**Tiempo Reducido(hh:mm)**
Antes de la optimización	10:46:13	12:24:17	01:38:04	-
Después de la optimización	12:48:45	13:34:56	00:46:11	51:53

Tabla 1. Resumen de tiempos antes y después de la optimización del programa.

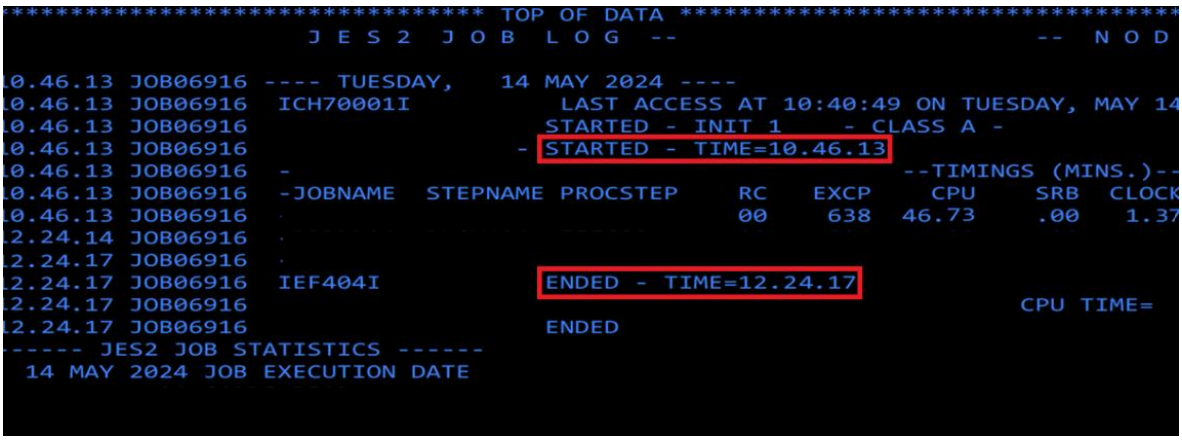


Figura 7. Captura del JES2 JOB LOG ejecución original del programa COBOL sin optimizaciones

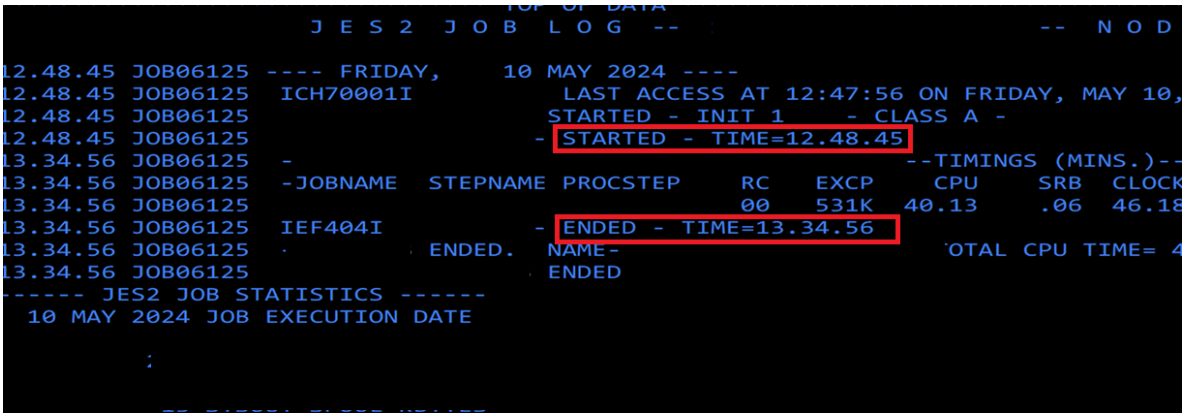


Figura 8. Captura del JES2 JOB LOG ejecución optimizada del programa COBOL.

Esta diferencia no fue producto de un único cambio, sino del conjunto de ajustes aplicados a nivel estructural y lógico. Entre los principales factores que contribuyeron a esta mejora se encontraba la reducción del número de columnas recuperadas en las sentencias SELECT, limitando la operación a los datos estrictamente necesarios. También fue determinante el uso de llaves primarias en las cláusulas WHERE, lo que permitió a Db2 ejecutar búsquedas más eficientes, aprovechando los índices ya definidos sobre la tabla. Adicionalmente, se eliminaron ciclos innecesarios de lectura y validación dentro del código COBOL, reduciendo las líneas de código del programa, además de la descarga única del archivo de la tabla.

6.2. Disminución del consumo de energía

Decidí realizar una estimación técnica del impacto energético derivado de la reducción de ejecución. Para este análisis, tomé como referencia el consumo eléctrico del IBM z15 modelo 8561, que representaba uno de los mainframes más modernos y robustos utilizados en entornos empresariales de alta demanda. Según el IBM z15 (8561) Technical Guide, en su Tabla 11-4, este equipo podía alcanzar un consumo máximo de 26.0 kilovatios (kW) bajo una configuración completa de 10 PCIe+ I/O drawers[5]. Esta cifra representa un escenario de carga

[5] Basado en la Tabla 11-4, página 474 (IBM, 2020).

máxima y fue seleccionada como base para el cálculo por tratarse de una estimación conservadora que asegura no subvalorar el uso real de energía durante ejecuciones intensivas.

Configuración (CPC/I/O Drawers)	**Consumo estimado (kW)**	**Condición de carga**
1 / 4	15.3	Carga típica
1 / 6	20.1	Carga alta
1 / 10	26.0	Carga máxima (utilizada para análisis)

Tabla 2. Consumo energético según configuración del IBM z15 (modelo 8561).

A partir de esta referencia, procedí a calcular el consumo energético estimado del proceso antes y después de la optimización, utilizando la fórmula básica:

- Cálculo del consumo energético por ejecución:

$$\text{Consumo energetico(kWh)} = \text{Potencia (kW)} \times \text{Tiempo (h)}$$

Cálculo del consumo energético por ejecución:

La versión original requería 1 hora y 38 minutos con 4 segundos minutos de CPU, equivalente a 1.634 horas. Por lo tanto, el consumo estimado fue:

- Consumo antes de la optimización:

$$\text{Consumo energetico}_{antes}\text{(kWh)} = 26.0\text{(kW)} \times 1.633\text{(h)} = 42.5\ \text{kWh}$$

La versión optimizada se ejecutó en 46 minutos (0.767 horas), lo que resultó en un consumo de:

- Consumo después de la optimización:

$$\text{Consumo energetico}_{despues}\text{(kWh)} = 26.0\text{(kW)} \times 0.767\text{(h)} = 19.94\ \text{kWh}$$

Lo anterior representó una reducción neta de:

$\Delta\text{Consumo(kWh)} = \text{ Consumo energetico}_{antes} - \text{ Consumo energetico}_{despues} = 42.5\text{kWh} - 19.94\ \text{kWh} = 22.6\ \text{kWh}$

Con base en estos resultados, el ahorro energético por cada ejecución del proceso optimizado fue de 22.6 kWh, lo que representa una disminución considerable del uso de recursos computacionales.

Para convertir esta reducción en términos de emisiones de carbono, se utilizó el factor de emisión oficial de la SEMARNAT para el año 2023, el cual es de 0.438 toneladas de CO_2 equivalente por megawatt-hora (tCO_2e/MWh), equivalente a 0.438 $kgCO_2e/kWh$[6].

Cálculo del impacto ambiental por ejecución:

- Emisión antes de la optimización:

$\text{Emisiones}_{antes}(kgCO_2) = 42.5\ \text{kWh} \times 0.438\ kgCO_2e/\text{kWh} = 18.62\ kgCO_2$

- Emisión después de la optimización:

$\text{Emisiones}_{despues}(kgCO_2) = 19.94\ \text{kWh} \times 0.438\ kgCO_2e/\text{kWh} = 8.73\ kgCO_2$

- Reducción de emisiones por ejecución:

$\text{Emisiones evitadas}\ (kgCO_2) = 18.62\ kgCO_2 - 8.73\ kgCO_2 = 9.89\ kgCO_2$

Este resultado evidenció una reducción tangible, no solo en términos de eficiencia computacional, sino también en cuanto al impacto ambiental. Aunque la cifra pueda parecer modesta a pequeña escala, su proyección sobre entornos productivos donde el proceso se ejecuta varias veces al día o como parte de flujos por lotes puede representar un aporte significativo a las políticas institucionales de eficiencia energética y sostenibilidad.

---

[6] (SEMARNAT, 2023)

El siguiente cuadro sintetizó estos valores y permitió visualizar de forma clara la magnitud de la mejora lograda a través del proceso de optimización técnica.

Parámetro	**Valor estimado**
Potencia base utilizada para el cálculo	26.0 kW
Tiempo de ejecución original	1.633(h) (1h 38min)
Tiempo de ejecución optimizada	0.767 h (46 min)
Energía consumida antes	42.5 kWh
Energía consumida después	19.94 kWh
Reducción de energía por ejecución	22.6 kWh
Factor de emisión (CO_2 por kWh)	0.438 kg CO_2/kWh
Emisiones de CO_2 antes	18.62 kg CO_2
Emisiones de CO_2 después	8.73 kg CO_2
Reducción de emisiones por ejecución	9.89 kg CO_2

Tabla 3. Estimación energética y ambiental por corrida del programa

6.3. Equivalencias del ahorro energético y su impacto en actividades domésticas

La reducción de 9.89 kilogramos de dióxido de carbono (CO_2) obtenida por cada ejecución optimizada del programa puede parecer una cifra abstracta; sin embargo, al compararla con actividades cotidianas, se apreció su impacto ambiental. Por ejemplo, cada ejecución optimizada fue comparable con las emisiones generadas por un vehículo ligero común en México. En este contexto, se tomó como referencia el Nissan Versa, por ser uno de los modelos más vendidos en el país y ampliamente utilizado en servicios de transporte urbano y plataformas digitales. Según el Catálogo de Rendimientos de Combustible en Vehículos Ligeros

de Venta en México 2023, publicado por la Comisión Nacional para el Uso Eficiente de la Energía (CONUEE), el Versa genera aproximadamente 123 gramos de CO_2 por kilómetro recorrido en condiciones combinadas[7], cifra que se obtuvo mediante un promedio ponderado de los rendimientos de ciudad y carretera del automóvil.

- Cálculo de la equivalencia de emisiones en kilómetros recorridos:

$$\text{Kilometros recorridos} = 9.89 \text{ kg } CO^2 \div 0.123 \text{ kg } CO^2/\text{km} = 79.9 \text{ km}$$

Así, la reducción alcanzada equivale a las emisiones generadas al recorrer cerca de 80 kilómetros con este vehículo, lo que permite visualizar de forma más clara el beneficio ambiental logrado mediante la optimización del programa.

Otra aplicación fue en el consumo energético de una vivienda, de acuerdo con el Programa Nacional para el Aprovechamiento Sustentable de la Energía 2020–2024, publicado por la Secretaría de Energía (SENER), en 2018 el consumo residencial anual ascendió a poco más de 63.3 mil millones de kWh, distribuido entre aproximadamente 34.4 millones de hogares electrificados. Este dato permitió estimar que cada vivienda consumía en promedio 1,863 kWh al año[8], lo que equivalió a cerca de 5.04 kWh diarios.

Para la obtención del consumo diario se realizó lo siguiente:

$$1{,}838 \text{ kWh/año} \div 365 \text{ días} = 5.04 \text{ kWh/día por hogar}$$

Bajo esta referencia, la energía ahorrada en una sola ejecución del programa (22.6 kWh) equivalió a más de cuatro días completos de consumo eléctrico en un hogar mexicano promedio:

---

[7] Basado en la tabla "Consumo de combustible y emisiones de CO_2" en la página 13 (CONUEE, 2023)

[8] Basado en el cálculo del consumo eléctrico promedio por hogar (SENER, 2018)

Cálculo de equivalencia en días:

$22.6\ \text{kWh} \div 5.04\ \text{kWh/día} = 4.48\ \text{días}$

La última aplicación considerada fue en las lavadoras. Existen dos tipos, manual y semiautomática. En la NOM-005-ENER-2016, observamos que los valores de consumo energético anual estaban claramente delimitados como máximos permisibles. Esta normativa específica que, según el tipo y capacidad de carga, el consumo puede variar entre 17 y 144 kWh por año[9]. Para obtener una estimación balanceada que sirviera de referencia en esta comparación, se seleccionaron cuatro modelos con distintas capacidades: dos manuales y dos semiautomáticas, una de baja carga y otra de carga alta por tipo. Esta elección buscó representar de manera justa el espectro general de consumo reportado en la norma, sin sesgos hacia extremos muy bajos o elevados.

De acuerdo con la tabla, los valores promedio utilizados fueron:

- Lavadora manual de 6.0 a <10.0 kg: 22 kWh/año
- Lavadora manual de ≥10.0 kg: 34 kWh/año
- Lavadora semiautomática de 6.0 a <10.0 kg: 22 kWh/año
- Lavadora semiautomática de ≥10.0 kg: 34 kWh/año

Con base en estos datos se obtuvo un promedio general representativo del consumo anual:

- Cálculo del promedio anual:

$(22 + 34 + 22 + 34)\ \text{kWh} \div 4 = 28\ \text{kWh/año}$

Para contextualizar este consumo en términos cotidianos, se estimó el consumo diario dividiendo el valor anual entre 365 días:

---

[9] Cálculo basado en los valores máximos permisibles de consumo anual establecidos por la NOM-005-ENER-2016 para lavadoras de ropa manuales y semiautomáticas. (SENER, 2016).

- Cálculo diario estimado:

28 kWh/año ÷ 365 días ≈ 0.0767 kWh/día por lavadora

No obstante, debido a que el uso real no es diario, sino por ciclo de lavado, se estimó que un hogar promedio realiza 2 ciclos por semana, es decir, 104 ciclos al año. Esto nos permite calcular el consumo por ciclo de lavado:

- Cálculo por ciclo de lavado:

28 kWh/año ÷ 104 ciclos ≈ 0.27 kWh/ciclo

Con este valor estimado, es posible evaluar cuántos ciclos de lavado podrían realizarse con el ahorro energético obtenido a través de la optimización del programa COBOL previamente analizado, que fue de 22.6 kWh por ejecución:

- Cálculo de ciclos de lavado posibles:

22.6 kWh ÷ 0.27 kWh/ciclo ≈ 83 ciclos de lavado

En consecuencia, la energía ahorrada al ejecutar un programa optimizado en un entorno mainframe puede ser suficiente para realizar alrededor de 83 ciclos de lavado, lo cual equivale al consumo energético de más de 9 meses de esta actividad en un hogar mexicano promedio, si se realizan dos ciclos por semana. Tal comparación permite visualizar de manera tangible el impacto positivo que puede tener una optimización técnica sobre el consumo de recursos en el ámbito doméstico.

7. Conclusiones y recomendaciones para trabajos futuros

A lo largo del desarrollo de este proyecto o mantenimiento durante mi estancia laboral, pude constatar la relevancia de aplicar conocimientos técnicos en contextos reales donde la eficiencia, el rendimiento y la sostenibilidad son cada vez más valorados. La experiencia que adquirí como desarrollador COBOL en un entorno productivo me permitió no solo fortalecer

habilidades de programación y análisis, sino también comprender de manera más profunda la importancia de optimizar los procesos informáticos en instituciones de gran escala.

Uno de los principales logros fue la mejora en el tiempo de ejecución de distintos programas, a través de la revisión detallada del código, la simplificación de sentencias SQL y la correcta utilización de índices. Este tipo de ajustes, aunque en apariencia menores, tuvo un impacto directo y medible en el desempeño del sistema, que a pesar de solo ejemplificar un programa y tomar como consumo total el mainframe totalmente encendido, y no solo la parte correspondida al procesamiento de datos. En algunos casos, los tiempos de ejecución se redujeron de manera significativa, traduciéndose en una liberación más eficiente de recursos y una disminución en el uso energético del equipo. Lo anterior se demostró en los resultados, esta mejora permitió generar un entorno más sostenible y ágil, en línea con los objetivos planteados respecto a la eficiencia energética y optimización de recursos.

Desde un enfoque de ingeniería, este proyecto evidenció que la disciplina no se limita al diseño de objetos físicos o al cálculo estructural, sino que se extendió también al análisis lógico, la mejora de sistemas complejos y la responsabilidad ambiental. En ese sentido, el trabajo realizado representó un ejercicio completo de aplicación del pensamiento ingenieril: identificar un problema, analizar sus causas, proponer soluciones viables y medir sus efectos. El impacto positivo sobre el rendimiento del sistema informático no solo fortaleció mis competencias técnicas, sino que también reforzó la noción de que todo esfuerzo por mejorar un proceso repercute, directa o indirectamente, en el bienestar de los usuarios finales.

Con respecto a la formación académica, en específico en las áreas de oportunidad que podrían fortalecer el plan de estudios de la carrera de ingeniería mecánica, considero que debería incluirse de manera obligatoria el desarrollo de habilidades blandas, como el liderazgo técnico. Estas permitirían aprender a comunicar de manera efectiva en qué consiste nuestro trabajo y respaldarlo no solo con resultados, sino también mediante una adecuada exposición. Estas disciplinas son fundamentales para transmitir ideas técnicas con claridad, especialmente cuando se trabaja con técnicos de distintas áreas. Además, esta experiencia me

permitió fortalecer habilidades de liderazgo y comunicación al colaborar con diversos equipos y documentar soluciones para futuras referencias.

Relacionados con la parte de habilidades técnicas, la lógica de programación, estructuras de bases de datos y eficiencia energética en sistemas digitales podría aportar a los futuros ingenieros una visión más amplia y adaptada a las necesidades actuales. Hoy en día, el cruce entre la mecánica y la informática es cada vez más evidente, sobre todo en campos como la automatización, el análisis de datos o la sostenibilidad energética. Por ello, brindar a los estudiantes herramientas básicas como algunos lenguajes de programación o incluso conceptos de ahorro energético no solo ampliaría el campo de acción profesional del egresado, sino que también contribuiría a desarrollar un perfil más versátil, con mayor capacidad de adaptación frente a los desafíos tecnológicos y ambientales que plantea la industria moderna.

En resumen, este trabajo fue una gran oportunidad para comprobar, con resultados claros, cómo los conocimientos aprendidos en la carrera pueden aplicarse de forma útil en el trabajo real. Más allá del código o las cifras de consumo, esta experiencia me ayudó a crecer profesionalmente y confirmó mi interés por seguir mejorando y aplicando la ingeniería de manera responsable.

8. Glosario

Programa Batch: Programa diseñado para ejecutarse de manera automática y secuencial dentro de un JCL, sin intervención del usuario durante su ejecución, normalmente para procesar grandes volúmenes de datos en entornos mainframe.

Buffer (Búfer): Área de memoria intermedia utilizada para almacenar datos temporalmente durante operaciones de entrada/salida, optimizando la transferencia de información.

CICS (Customer Information Control System): Monitor de transacciones de IBM que permite la ejecución de aplicaciones interactivas en mainframes.

COBOL (Common Business-Oriented Language): Lenguaje de programación orientado a negocios, diseñado en la década de 1960 para el procesamiento de datos empresariales.

CONUEE: Comisión Nacional para el Uso Eficiente de la Energía

CPU (Central Processing Unit): Unidad central de procesamiento que ejecuta instrucciones y determina la capacidad de cómputo de un sistema.

Db2: Sistema de gestión de bases de datos relacional desarrollado por IBM, especializado en ambientes empresariales de alta demanda.

I/O (Input/Output): Operaciones de entrada y salida que permiten la interacción de un sistema con dispositivos externos o internos.

IGYWCL: Procedimiento catalogado en JCL utilizado para compilar programas COBOL en mainframes IBM.

Índice (en bases de datos): Estructura auxiliar que permite acceder de manera más rápida a los registros de una tabla.

IKJEFT01: Programa de utilería de IBM z/OS que ejecuta comandos TSO y SQL en entornos batch.

JCL (Job Control Language): Lenguaje de control utilizado en z/OS para definir, organizar y ejecutar trabajos en mainframes IBM.

JES2 JOB LOG: Registro del sistema en z/OS que documenta la ejecución de trabajos batch, incluyendo inicio, fin y uso de recursos.

Llave primaria: Atributo que identifica de manera única a cada registro dentro de una tabla de base de datos.

Mainframe: Computadora central de alto rendimiento utilizada para procesar grandes volúmenes de datos con alta fiabilidad.

Módulo ejecutable: Archivo generado a partir de la compilación de un programa fuente, listo para su ejecución.

MVS (Multiple Virtual Storage): Sistema operativo de IBM que antecedió al z/OS y permitió la gestión de múltiples espacios de memoria virtual.

PCIe+ I/O Drawer: Módulo de expansión para mainframes IBM que permite instalar tarjetas PCIe adicionales para aumentar la capacidad y el rendimiento de entrada/salida del sistema.

SELECT: Sentencia SQL que permite recuperar información de una o varias tablas en una base de datos.

SENER: Secretaría de Energía

SEMARNAT: Secretaría de Medio Ambiente y Recursos Naturales

SORT: Utilidad clásica de mainframe que organiza, filtra o combina archivos secuenciales de gran tamaño.

SQL (Structured Query Language): Lenguaje estándar para la definición, manipulación y consulta de datos en bases de datos relacionales.

SQLCODE: Variable en Db2 que guarda el estado de la última operación SQL ejecutada.

Step cards: Instrucciones dentro de un JCL que definen pasos específicos para compilar o ejecutar programas batch.

TSO (Time Sharing Option): Interfaz interactiva de z/OS que permite ejecutar comandos y programas en un entorno compartido.

UPDATE: Sentencia SQL que permite modificar registros existentes en una tabla.

VSAM (Virtual Storage Access Method): Método de acceso a datos en mainframes IBM utilizado para organizar y administrar archivos de gran tamaño.

WHERE: Cláusula de SQL que filtra registros de acuerdo con condiciones específicas.

z/OS: Sistema operativo de IBM diseñado para mainframes, orientado a la gestión de grandes volúmenes de datos y procesos críticos.

9. Bibliografía

1. IBM. (s. f.). *¿Qué es COBOL?*. Obtenido de https://www.ibm.com/es-es/topics/cobol

2. IBM Redbooks. (2006). *Introduction to the New Mainframe: z/OS*. Obtenido de https://www.redbooks.ibm.com/abstracts/sg246366.html

3. Silberschatz, A., Galvin, P. B., & Gagne, G. (2006). *Fundamentos de sistemas operativos* (7.ª ed.). Wiley.

4. IBM. (s.f). *Communication through job log*. Obtenido de https://www.ibm.com/docs/en/zos/2.4.0?topic=communication-through-job-log

5. IBM Corporation. (2020). *IBM z15 (8561) Technical Guide* (Redbook SG24-8890-00). International Business Machines. https://www.redbooks.ibm.com/redbooks/pdfs/sg248851.pdf

6. Secretaría de Medio Ambiente y Recursos Naturales (SEMARNAT). (2023). *Aviso del Factor de Emisión del Sistema Eléctrico Nacional 2023*. Dirección General de Políticas para el Cambio Climático. https://www.gob.mx/cms/uploads/attachment/file/895937/Aviso_FE-SEN23.pdf

7. Comisión Nacional para el Uso Eficiente de la Energía (CONUEE). (2023). *Catálogo de Rendimientos de Combustible en Vehículos Ligeros de Venta en México 2023*. Secretaría de Energía. https://www.gob.mx/cms/uploads/attachment/file/913595/Cat_logo_de_Rendimientos_2023_mayo2024.pdf

8. Secretaría de Energía (SENER). (2018). Programa Nacional para el Aprovechamiento Sustentable de la Energía 2020–2024. https://sidof.segob.gob.mx/notas/docFuente/5679748

9. Secretaría de Energía (SENER). (2016). NORMA Oficial Mexicana NOM-005-ENER-2016, Eficiencia energética de lavadoras de ropa electrodomésticas. Límites, método de prueba y etiquetado. Diario Oficial de la Federación. https://www.dof.gob.mx/normasOficiales/6241/sener2a11_C/sener2a11_C.html