73

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

"SISTEMA Y METODOLOGIA QUE GENERA EL DISEÑO FISICO Y MANTENIMIENTO DE BASES DE DATOS EN BANCOMER, S.A. PARA UNA MAQUINA IBM 9021-720 DB2DA (DB2 DESIGN AID)"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N :

GUSTAVO MEDEL OSORIO

JAVIER SOLANO TREJO

ASESOR DE TESIS: ING. ROBERTO REYES CHALICO

MEXICO, D. F. 1996

AGRADECIMIENTOS

Dedicada con todo mi amor a mis padres y hermanos.

También dedicada con cariño a todos mis amigos que me han ofrecido y ofrecen su amistad incondicional.

A los profesores quienes compartieron sus conocimientos y enseñanzas.

Y por último a la escuela por compartir sus espacios físicos y sus materiales que aunque escasos merecen respeto.

Desde esta página les envío un profundo agradecimiento.

"Culminamos algo que habíamos iniciado desde hace tiempo simplemente para darnos cuenta que apenas comenzamos y a lo que le podríamos llamar el inicio de un fin".

Javier

AGRADECIMIENTOS.

Gracias a Dios por mantener mi fe.

A mis padres:
Por su confianza y apoyo en todo momento.
Bases e inspiración en mis deseos de superación.
Gracias padre por darme la oportunidad de intentarlo.
Gracias madre por creer en mi.....
Con todo mi amor.

A mis hermanos :
Por su retroalimentación espiritual en mi ser.
Por ser parte de mi vida.
Gracias Bolivar por ser un pilar en la familia
Siempre viviran adentro de mi corazón.

Para mi tia y primas:
Gracias por estar siempre conmigo.

A mis amigos:
A todos mis amigos que de forma desinteresada me motivaron para la terminación de este trabajo.
Gracias Javier por tu paciencia y preserverancia, tu esfuerzo fue el motor para este largo viaje.

A la Facultad de Ingeniería y profesores:
Gracias por darme el espacio y tiempo para una formación.
Me siento previlegiado de haber sido parte de la universidad, siempre la recordare para volverlo a vivir.

En el camino de este trabajo nos encontramos con problemas, pero siempre encontramos una alternativa o solución, en un momento de desesperación alguien me dijo :
"Lo unico en la vida que no tiene solución es la muerte."
De lo anterior quiero expresar lo siguiente:
El hombre esta hecho para resolver problemas. No puede vivir sin ellos y necesita el reto que representa solucionarlos; cuando hay ausencia de éstos el hombre se los crea, ya que su mente humana está construida para darle solución a situaciones complicadas. Veamos a los problemas como retos y no como tragedias. Aún en medio de los problemas puede haber paz interior y es precisamente cuando estamos serenos que podemos resolverlos.

Gustavo

SISTEMA Y METODOLOGIA QUE GENERA EL DISEÑO FISICO Y MANTENIMIENTO DE BASES DE DATOS EN BANCOMER S.A. PARA UNA MAQUINA IBM 9021-720.

DB2DA (DB2 Design Aid)

TESIS

COMPLETA

INDICE GENERAL

INTRODUCCION

En la actualidad se ha incrementado de manera notoria la implementación de las bases de datos en los sistemas informáticos, teniendo como principal objetivo la automatización de las operaciones y manejo de información de una empresa corporativa, lo que nos ha llevado a poner más énfasis en el diseño de las bases de datos para que así se tengan datos más confiables, disponibles y consistentes, y con ello lograr proteger el activo más importante de la empresa que es la información. Por esto es que la presente tesis ofrece un diseño de un sistema integral automatizado que generará recomendaciones para el diseño físico y dará la frecuencia de ejecución de las utilerías de mantenimiento de una base de datos en DB2 para un alto rendimiento, proponiendo una metodología estándar para la elaboración del diseño físico de una base de datos relacional, lográndose con esto dar solución a problemas como: errores humanos en el cálculo de parámetros para el espacio físico, abortación de programas por falta de espacio, elevado número de cambios en la definición de objetos, etc.

Inicialmente se proporcionará un panorama general sobre la importancia y el objetivo de las bases de datos, presentando los diferentes tipos de modelos de bases de datos, mencionando sus características y haciendo énfasis en el modelo relacional. Se tratarán los fundamentos del diseño lógico referenciando aspectos como: modelo entidad relación, modelo relacional, restricciones de integridad y normalización. Se explicarán los componentes funcionales, características, utilerías y diccionario de datos de un Sistema Manejador de Base de Datos (DBMS); además indicando las funciones del administrador de base de datos (DBA) quién llevará el control central de dicho sistema. Este apartado mostrará de manera introductoria al diseño físico de las bases de datos.

Posteriormente se expondrá la situación actual de la organización de la empresa, así como la problemática detectada al liberar una aplicación de un ambiente de pruebas a un ambiente de producción teniendo su origen en un deficiente diseño físico. Se propondrá un sistema que automatizará la generación de parámetros del diseño físico de tablas e índices, además de recomendar las utilerías a usar y con que frecuencia ejecutarlas como una solución a la problemática identificada. También se mencionará una descripción funcional y objetivos del sistema propuesto, al igual que su justificación. Por último presentará la estimación de costos del proyecto, los requerimientos de hardware y software y el plan tentativo de actividades.

Se presentará el análisis del sistema enfocándose en el diseño físico en DB2, se analizarán tópicos tales como: los objetos de una base de datos que se usan para describir los datos, el método de acceso que se utiliza para la manipulación de la información, las consideraciones del diseño lógico y como se realiza su implemetación física, las decisiones de diseño que se deben tomar en cuenta para tener un buen performance, las utilerías que se aplican en el mantenimiento de las bases de datos tratando de explotar al máximo su rendimiento.

Finalmente se presenta el diseño estructurado del sistema iniciando con la carta de estructura primaria que es la representación gráfica a nivel cero de lo que constituirá el sistema. Se esquematizarán las tablas auxiliares que se manejarán dentro del sistema detallándose la descripción de sus campos y como se hace la conexión con el diccionario de datos de DB2. Se plantearán los diagramas de flujo de datos y el diseño modular estructurado a detalle, junto con las interfaces funcionales que se utilizarán para la manipulación de los archivos y la comunicación entre los módulos. Finalmente se presentará el pseudocódigo del sistema como el producto terminal del diseño.

1 GENERALIDADES

1.1 El enfoque de bases de datos.

1.2 Diseño lógico de una base de datos relacional.

1.3 Diseño físico de una base de datos relacional.

1.4 Sistemas manejadores de bases de datos (DBMS).

1.5 Administrador de bases de datos (DBA).

1.6 Diccionario de Datos.

1.7 Utilerías de un sistema manejador de bases de datos relacionales.

1.1 EL ENFOQUE DE BASES DE DATOS

En los últimos años ha aumentado considerablemente la implementación de las bases de datos en los sistemas de computación, su principal función es la automatización de las operaciones y manejo de la información de una empresa u organismo. Por lo tanto en la ciencia de la computación se ha abierto un campo de estudio a las Bases de Datos.

Una base de datos es una colección *integrada* de datos, la cual requiere tener una organización lógica y física en una forma especial. La organización física consiste en seleccionar los mecanismos de almacenamiento (estructuras, metódos de acceso, dispositivos, etc.) de los datos y la organización lógica se determina en la elección del modelo de datos (jerárquico, red y relacional).

Los modelos de datos más conocidos y aceptados son:

- Jerárquico 60'S
- Red 70'S
- Relacional 80'S

El enfoque Jerárquico:

- Se representan a los datos como estructuras de árbol.
- El árbol representa una jerarquía de registros de datos.
- Procesamiento Top-Down (De arriba hacia abajo).
- Términos jerárquicos:
  - SET (DATA SET): Registro.
  - PARENT: Registro Padre.
  - CHILD: Registro Hijo.
- Relaciones entre registros: 1 padre, múltiples hijos.
- Desventajas:
  - No modela con facilidad las relaciones N a N.
  - Anomalías de inserción.
  - Anomalías de borrado.
  - Anomalías de actualización.
  - Asimetría en la consulta.

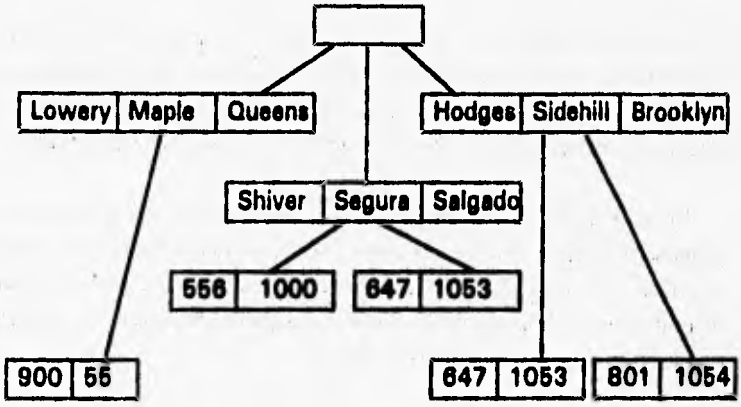


Figura 1.1. Ejemplo de un modelo jerárquico.

El enfoque de Red:

- Se representan los datos como registros ligados formando un conjunto de datos interconectados.
- Procesamiento Multidireccional, Navegacional.
- Términos de red:
  - Esquema
  - Subesquema.
- Cualquier tipo de relación entre registros puede ser modelada.
- Desventajas:
  - Es difícil definir nuevas relaciones.
  - Es difícil de mantener (cualquier cambio requiere un descarga de datos).
  - Elevado overhead (desperdicio de recursos).

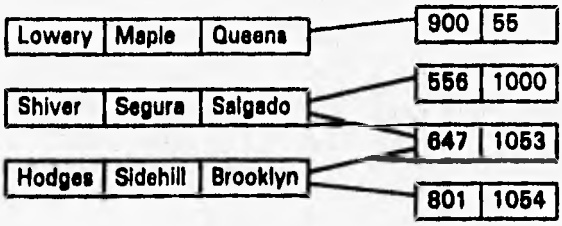


Figura 1.2. Ejemplo de un modelo de red.

El enfoque Relacional:

- Terminología:
  - Tabla:
    - Arreglo de dos dimensiones.
    - Consiste de renglones y columnas.
    - Colección de registros.
    - Columna: Contiene un mismo tipo de información.
    - Renglón: Contiene un mismo conjunto de información.
    - Campo: Contiene un valor en la intersección del renglón y la columna.

- Características:
  - La información de una tabla puede estar relacionada con la información de otra tabla.

Nombre	Calle	Ciudad	Número
Lowery	Maple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brooklyn	801
Hodges	Sidehill	Brooklyn	647

Número	Saldo
900	55
556	100000
647	105366
801	10533

Figura 1.3. Ejemplo de un modelo relacional.

El modelo relacional en la actualidad es el más aceptado e implementado en la mayoría de los Sistemas Manejadores de Bases de Datos (DBMS) disponibles en el mercado de la computación. Por lo anterior este trabajo tendrá como base el modelo relacional y por lo tanto su enfoque será hacia las bases de datos relacionales.

Es es necesario señalar la importancia del porque implantar las bases de datos relacionales en sistemas, ya que éstas presentan las siguientes características:

- La redundancia se puede controlar.
- La consistencia se puede mantener.
- Los datos se pueden integrar.
- Los datos se pueden compartir.
- Se pueden obligar los estándares.
- Se facilita el desarrollo de aplicaciones.
- Se uniformizan los controles de seguridad, restricción e integridad.
- Independencia entre los datos y los programas.
- Reducción en el mantenimiento a programas.

Algunos ejemplos que ilustran lo anterior son los siguientes: Cuando un usuario y programa autorizados requieran acceder a datos en un mismo tiempo; los datos pueden ser modificados por aquellos autorizados a hacerlo. Optimización del espacio de almacenamiento.

La implantación de una base de datos tiene su base fundamental en su diseño, éste se divide en etapas las cuales las estudiaremos mas adelante. El diseño de una base de datos se fundamenta principalmente en minimizar la cantidad de información *redundante* y tiempo de respuesta.

1.2 DISEÑO LOGICO DE UNA BASE DE DATOS RELACIONAL

En las aplicaciones de Bases de Datos se debe proporcionar a los usuarios una visión abstracta de los datos. Por lo tanto, la aplicación esconde ciertos detalles de cómo se almacenan y mantienen los datos. Se debe considerar, para que la aplicación sea manejable, la premisa de que los datos puedan extraerse en forma eficiente. Esta necesidad ha llevado al diseño de estructuras para la representación de datos.

Para describir una estructura de Datos es necesario definir el concepto de Modelo.

Un modelo de datos es un conjunto de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia. Para el diseño lógico de una Base de Datos Relacional se apoya en el modelo entidad-relación y el modelo relacional.

El modelo entidad-relación es un modelo lógico basado en objetos; se utilizan para describir datos en los niveles conceptual y de visión. Se basa en una percepción de un mundo real que consiste en una colección de objetos básicos llamados entidades y relaciones entre estos objetos. Una entidad es un objeto que es distinguible de otros objetos por medio de un conjunto específico de atributos. Una entidad puede ser un objeto tangible, por ejemplo un empleado, una pieza, un artículo, o un lugar. Pero también puede ser algo intangible tal como un suceso, un nombre de tarea, la cuenta de un cliente, o un concepto abstracto.

Cuando hablamos de información podemos referirla a tres diferentes campos, que tendemos, a veces confusamente a saltar de uno a otro sin advertencia previa. El primero de estos campos es el del mundo real, en el que existen entidades y éstas exhiben ciertas propiedades. El segundo es el dominio de las ideas y la información existente de las personas y los programadores. Es decir hablamos de los atributos de las entidades y nos referimos a estos atributos simbólicamente, haciéndolo en nuestra lengua común, o utilizando un lenguaje de programación. Por lo tanto asignamos valores a los atributos. El tercer campo es el de los datos, en el usamos conjuntos de caracteres o bytes (strings) para codificar items de información

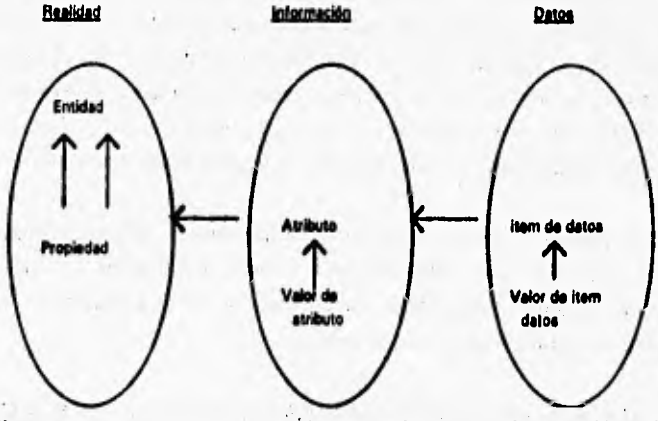


Figura 1.4. Tres campos. Lo que realmente se almacena es el valor del item de datos; éste debe estar asociado con un atributo en particular de una entidad dada.

Además de entidades y relaciones, el modelo entidad-relación representa ciertas restricciones a las que deben ajustarse los contenidos de una base de datos. Una restricción importante es la de las ***cardinalidades de asignación,*** que expresan el número de entidades con las que puede asociarse otra entidad mediante un conjunto de relaciones. Las cardinalidades de asignación son más útiles al describir conjuntos binarios de relaciones, aunque ocasionalmente contribuyen a la descripción de conjuntos de relaciones que implican más de dos conjuntos de entidades. Para ilustrar lo anterior, considérese un conjunto binario de relaciones R entre los conjuntos de entidades A y B, la cardinalidad de asignación debe ser una de las siguientes:

- Una a una. Una entidad en A esta asociada a lo sumo con una entidad en B, y una entidad en B está asociada a lo sumo con una entidad en A.

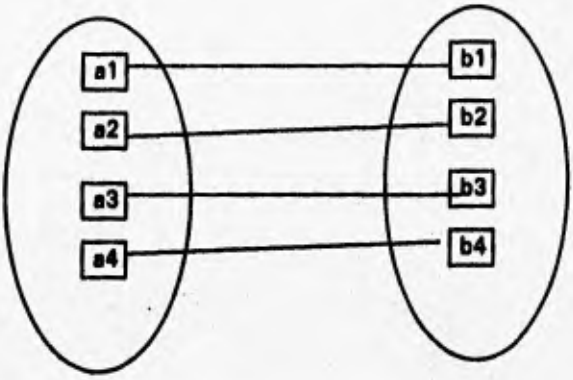


Figura 1.5. Relación una a una.

- Una a muchas. Una entidad en A está asociada con un número cualquiera de entidades en B. Una entidad en B, sin embargo, puede estar asociada a lo sumo con una entidad en A.

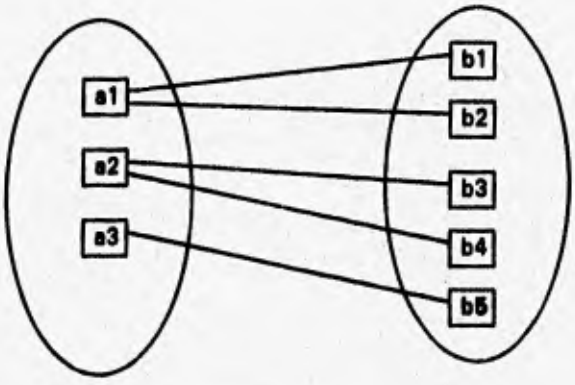


Figura 1.6. Relación una a muchas.

- Muchas a una. Una entidad en A está asociada a lo sumo con una entidad en B. Una entidad en B, sin embargo, puede estar asociada con un número cualquiera de entidades en A.

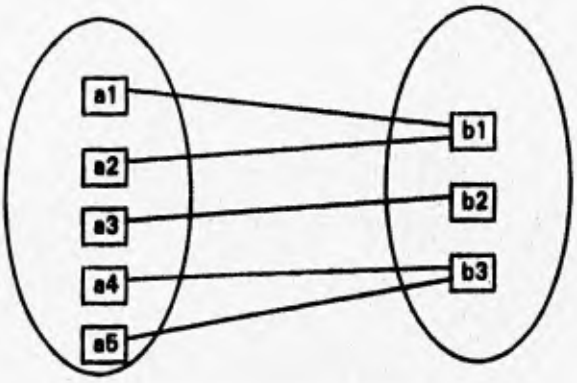


Figura 1.7. Relación muchas a una.

- Muchas a muchas. Una entidad en A está asociada con un número cualquiera de entidades en B, y una entidad en B está asociada con un número cualquiera de entidades en A.

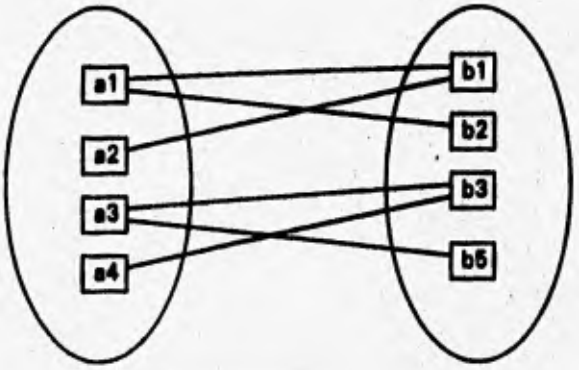


Figura 1.8. Relación muchas a muchas.

Es importante poder especificar cómo se distinguen las entidades y las relaciones. Por definición, las entidades individuales y las relaciones son distintas, pero, desde el enfoque de una base de datos, la diferencia entre ellas debe expresarse en términos de sus atributos. El concepto de superclave nos permite hacer tales distinciones. ***Una superclave*** es un conjunto de uno o más atributos que, consideramos conjuntamente, nos permiten identificar de forma única a una entidad en el conjunto de entidades. El concepto de superclave no es suficiente en la perspectiva de una base de datos, ya que una superclave puede contener atributos, es decir, si K es una superclave, entonces también lo será cualquier superconjunto de K. Frecuentemente estamos interesados en superclaves para las cuales ningún subconjunto propio es superclave. Dichas superclaves mínimas se llaman ***claves candidatas.***

Finalmente se utiliza el término ***clave primaria*** para denotar una clave candidata que elige el diseñador de la base de datos como el medio principal de indentificar entidades dentro de un conjunto de entidades.

La estructura lógica global de una base de datos puede expresarse gráficamente por medio de un ***diagrama entidad-relación*** que consta de los siguientes componentes:

- Rectángulos, que representan conjuntos de entidades.
- Elipses, que representan atributos.
- Rombos, que representan relaciones entre conjuntos de entidades.
- Líneas, que conectan atributos a conjuntos de entidades y conjuntos de entidades a relaciones.

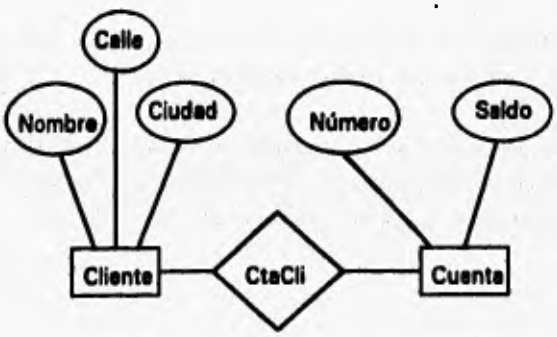


Figura 1.9. Ejemplo del modelo entidad relación.

El modelo relacional es un modelo lógico basado en registros que se usa para describir datos en los modelos conceptual y físico. A diferencia de los modelos de datos basados en objetos, se utilizan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a un nivel más alto de la implementación. El modelo relacional representa los datos y las relaciones entre los datos mediante una colección de tablas, cada una de las cuales tiene un número de columnas con nombres únicos. (Véase la figura 1.3)

Cuando hablamos de una base de datos debemos diferenciar entre el esquema o el diseño lógico, y ***una instancia de la base de datos***, que son datos en un instante de tiempo.

El concepto de ***esquema de una relación*** corresponde a la noción de la definición de tipo en los lenguajes de programación. Una variable de un tipo dado tiene un valor determinado en un instante de tiempo dado. Es conveniente dar un nombre al esquema de una relación, así como damos nombres a las definiciones de tipo en los lenguajes de programación. En general el esquema de una relación es una lista de atributos y sus correspondientes dominios.

Esquema-depósito=(nombre-sucursal, número-cuenta, nombre-cliente, saldo)

(nombre-sucursal: cadena, número-cuenta: entero,
nombre-cliente: cadena, saldo: entero)

Figura 1.10. Ejemplo de un esquema de una relación.

Las definiciones de superclave, clave candidata y clave primaria explicadas anteriormente también pueden aplicarse al modelo relacional.

Una vez que los datos se han almacenado en la base de datos, existe un lenguaje estándar en los DBMS, llamado ***SQL (Structured Query Language- Lenguaje de consulta estructurado)***, que permite el acceso y operaciones con los datos.

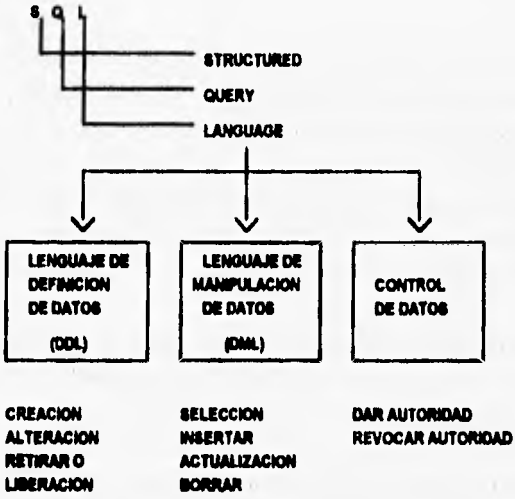


Figura 1.11. Lenguaje de Consulta Estructurado.

Los lenguajes de consulta (SQL) tienen su origen, en el *álgebra relacional y el cálculo relacional*, basados en la teoría de conjuntos, éstos son lenguajes de consulta concisos formales, que son inadecuados para usuarios casuales de un sistema de base de datos.

Dentro del esquema de la Base de Datos ***las restricciones de integridad*** proporcionan un medio de asegurar que los cambios que se hacen en la Base de Datos por usuarios autorizados no resultan en una pérdida de *consistencia* de los datos. Es decir, las restricciones de integridad protegen las Bases de Datos contra daños accidentales. Por ejemplo para el modelo entidad-relación dos formas de restricción de *integridad* son las siguientes:

- Declaraciones de clave. La estipulación de que ciertos atributos forman una clave candidata (llave primaria) para un conjunto de entidades dado. El conjunto de inserciones y actualizaciones permitidas están limitadas a aquellas que no crean dos entidades con el mismo valor en una clave candidata.

- Forma de una relación. Muchos a muchos, uno a muchos, uno a uno. Una relación uno a uno o uno a muchos restringe el conjunto de relaciones permitidas entre entidades de una colección de conjuntos de entidades.

Las restricciones de dominio de los datos especifican el conjunto de posibles valores que puedan asociarse a un atributo. Tales restricciones también pueden prohibir el uso de valores nulos para determinados atributos.

Con frecuencia queremos asegurar que un valor que aparece en una relación para un conjunto de atributos dado también aparece para un cierto conjunto de atributos en otra relación, a esto se le llama ***integridad referencial.***

Las dependencias funcionales son una generalización de las dependencias de clave. Exigen que el valor para un cierto conjunto de atributos determine en forma única el valor para otro conjunto de atributos.

Una vez que se tiene el esquema relacional, se aplica ***el proceso de normalización.***

Una relación no normalizada es una relación que contiene uno o más grupos repetitivos (Ver Figura 1.12). Como resultado hay varios valores en la intersección de ciertos renglones y columnas. Una de las desventajas de las relaciones no normalizadas es que contienen datos redundantes. Si no se actualizan todas las ocurrencias los datos estarían *inconsistentes.*

Matrícula	Nombre	Carrera	Curso	Título	Prof	Cub	Calif
38214	Pérez	IS	IS350	B de D	Ortiz	B104	10
38214	Pérez	IS	IS465	AN SIS	Flores	B213	8
69173	López	LC	IS465	AN SIS	Flores	B213	10
69173	López	LC	LC300	ADMON	Gómez	D317	9
69173	López	LC	LC440	SIS OP	Flores	B213	8

Figura 1.12. Relación no normalizada BOL-CALIF.

La normalización es un proceso de depuración de las relaciones y especificaciones de los archivos que forman al esquema, de tal manera que se garantice que las relaciones entre las diferentes entidades sean eficaces y eficientes, de acuerdo a la realidad, permitiendo esto responder a cualquier consulta a la Base de Datos. Con la normalización se analiza paso a paso las asociaciones entre los datos, eliminando todos los grupos repetitivos de la relación, obteniendo un conjunto de relaciones en primera forma normal (1NF). Se eliminan las dependencias funcionales parciales, para obtener

relaciones en segunda forma normal (2NF). Finalmente, se eliminan las dependencias transitivas, creando relaciones en tercera forma normal (3NF).

Una relación normalizada es una relación que contiene sólo valores elementales o simples en la intersección de cada renglón o columna. Así, una relación normalizada no tiene grupos repetitivos. En la Figura 1.12 se puede observar que la relación BOL-CALIF contiene grupos repetitivos, debido a que la información que pertenece al curso IS465 está en varias partes.

Para normalizar una relación que contiene un sólo grupo repetitivo, se elimina el grupo repetitivo y se forman dos nuevas relaciones. Por lo tanto, una relación está en primera forma normal si no tiene grupos repetitivos. En la Figura 1.13 se puede observar que la relación ESTUDIANTE-CURSO está en primera forma normal y presenta dependencias parciales, la razón es que varios atributos no llave dependen de CURSO y no de la llave primaria compuesta (MATRICULA + CURSO).

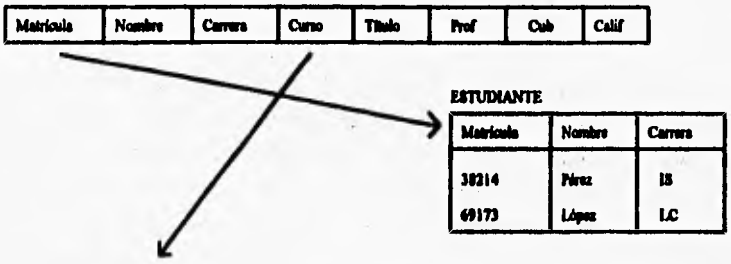


Figura 1.13. Descomposición de la relación BOL-CAL en ESTUDIANTE y ESTUDIANTE-CURSO.

Una relación en primera forma normal, puede presentar redundancias si existen dependencias parciales. Para eliminar las anomalías de la primera forma normal se deben eliminar las dependencias parciales. Una relación está en segunda forma normal si está en primera forma normal y se han elimininado las dependencias parciales. En la figura 1.14

la relación CURSO-PROF se encuentra en segunda forma normal y presenta dependencias transitivas, ésto es porque el atributo CUB depende del atributo PROF.

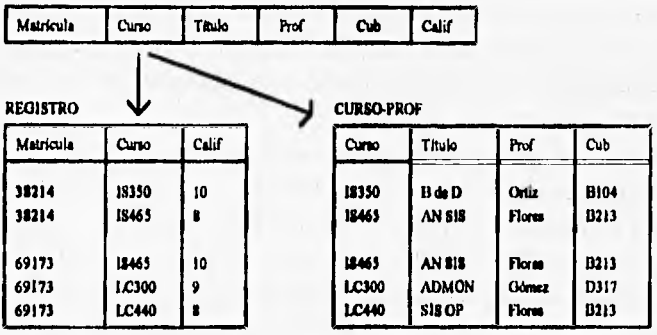


Figura 1.14. Descomposición de ESTUDIANTE-CURSO en REGISTRO y CURSO-PROF.

Una relación en segunda forma normal puede tener anomalías de inserción, actualización y eliminación si presenta dependencias transitivas. Para eliminar las anomalías de la segunda forma normal es necesario dar otro paso de normalización. Este paso convierte una relación en tercera forma normal eliminando dependencias transitivas. En la Figura 1.15 las relaciones CURSO y PROFESOR se encuentran en tercera forma normal.

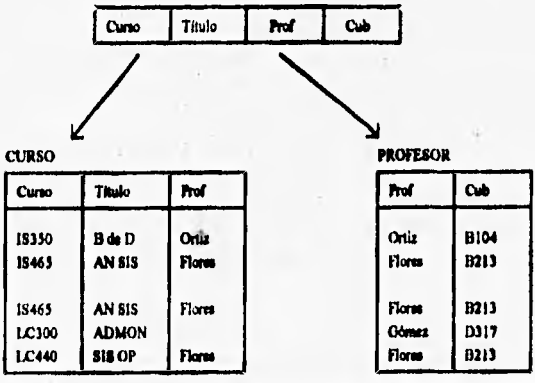


Figura 1.15. Descomposición de CURSO-PROF en CURSO y PROFESOR.

El proceso de normalizacion en diseño lógico establece una contrucción de una base integrada de datos a traves de su análisis.

Otro concepto que resulta decisivo en el diseño de una base de datos es ***la independecia de los datos***. Este consiste en la capacidad para utilizar la base de datos sin conocer los detalles de representacion. Las razones que justifican la independencia de los datos son:

- Permitir al administrador o programador hacer cambios de contenido, localización, representación y organización en la base de datos sin necesidad de volver a escribir los programas de aplicación que utilizan la base de datos.
- Permitir al proveedor de equipo de procesamiento de datos y de software introducir nuevas tecnologías sin que se tenga que reprogramar la aplicación del cliente.
- Facilitar el compartimiento de datos al permitir que parezca que los mismos datos están organizados de manera diferente para los diversos programas de aplicación.
- Proporcionar la centralización de control que necesita el administrador de la base de datos para garantizar seguridad e integridad de la base de datos.

Se observan dos etapas de la independencia de los datos en el diseño, la independencia lógica y fisica. (véase figura 1.16). El proceso de diseño de la base de datos comienza con los requerimientos conceptuales de varios usuarios. También se puede especificar los requerimientos conceptuales de algunas aplicaciones que no se iniciarán en un futuro cercano. Estos requerimientos de los usuarios individuales está integrada en un solo enfoque llamado modelo conceptual. Este representa las entidades y sus relaciones, nos da la capacidad de visualizar todas las entidades de datos y sus interelaciones, sin tener que ocuparnos de su almacenamineto fisico.

El modelo conceptual que los usuarios necesitan para el futuro cercano se traduce entonces a un modelo de datos compatible con el sistema de manejo de las bases de datos elegido. Es posible que las relaciones entre entidades tal como quedaron reflejadas en el modelo conceptual, no sean practicables con el paquete DBMS seleccionado. En tales casos se deberán hacer modificaciones al modelo conceptual que reflejen estas limitaciones. La versión del modelo conceptual que puede adaptarse al DBMS se llama modelo lógico. A los usuarios se les entrega subconjuntos de este modelo lógico. Estos subconjuntos, llamados modelos externos son los puntos de vista que los usuarios obtienen basados en el modelo lógico. Los requerimientos conceptuales son los enfoques que los usuarios deseaban inicialmente y constituyen la base sobre la que se desarrollo el modelo conceptual.

El modelo lógico se transfiere a un almacenamiento físico. El modelo físico que toma en cuenta la distribución de los datos, los métodos de acceso y las técnicas de clasificación se le conoce también como modelo interno.

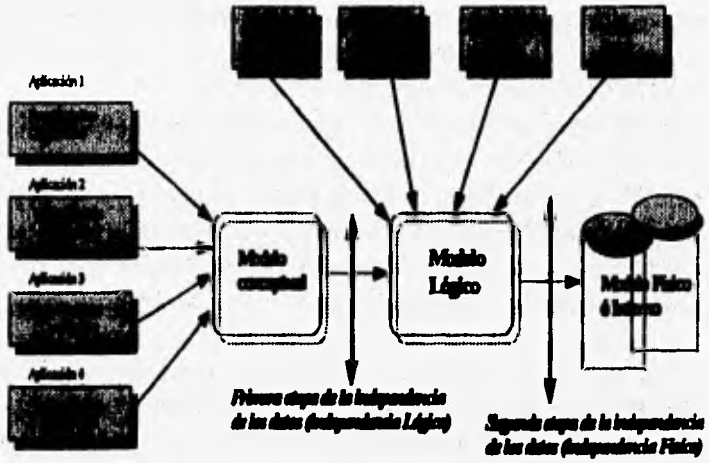


Figura 1.1.16 Los Elementos de la independencia de los datos.

1.- Independencia Lógica
2.- Independencia Física.

Modelo conceptual. Los requerimientos conceptuales de usuarios individuales son integrados en una vista sencilla común llamado "Modelo conceptual".

Modelo Lógico. La versión del modelo conceptual que puede ser presentada para el sistema manejador de base de datos es llamado "Modelo Lógico"

Modelo Interno. El modelo que toma dentro de sus consideraciones la distribución de los datos, los métodos de acceso y técnicas de indexación es llamado un modelo interno (Lo mencionaremos como "modelo Físico")

1.3 DISEÑO FISICO DE UNA BASE DE DATOS RELACIONAL

Uno de los principales factores que afectan el funcionamiento de los programas que interactúan con la base de datos, es la manera en la cual los datos se almacenan y se accesan. Un sistema de base de datos generalizado, utiliza algunos métodos de acceso del modelo interno junto con los métodos de acceso especializados disponibles a través del modelo externo. El modelo interno en la figura 1.16 es el modelo físico y el modelo externo es el punto de vista de un usuario. Algunos métodos de acceso del modelo físico son idénticos a los métodos de acceso proporcionados por el sistema operativo. En la mayoría de los DBMS el diseñador de la base de datos tiene la flexibilidad de elegir uno o una combinación de varios métodos de acceso disponibles. Para diseñar una base de datos que obtenga un almacenamiento y acceso eficientes, el diseñador debe estar familiarizado tanto con los métodos de acceso del modelo interno (modelo físico) como con los del modelo externo (puntos de vista del usuario).

El modelo físico es una estructura de la base de datos que se ha de almacenar en dispositivos físicos. Dado que un gran porcentaje de bases de datos se utilizan en un medio en línea, se tienen que estar muy relacionado con el funcionamiento "real". Es absolutamente necesario evaluar las características de funcionamiento del modelo físico antes de implantar la base de datos. La predicción del funcionamiento es un análisis multivariable, en el cual la principal variable es el modelo físico. Se debería efectuar un análisis cuantitativo del modelo físico con frecuencias promedio de ocurrencias de grupos de datos, con las estimaciones esperadas de espacio auxiliar y con estimaciones esperadas de tiempo para la recuperación y el mantenimiento de los datos.

La estimación del funcionamiento en la fase de diseño debe tener un amplio apoyo de la administración, debido a que una vez que la base de datos se ha instalado, es extremadamente difícil rediseñarla. Un diseñador de bases de datos generalmente trata de optimizar el modelo físico, en cuanto a consideraciones de espacio y tiempo. Se realizan afinaciones para obtener el equilibrio entre espacio y tiempo en la mayoría de los DBMS, es decir, se pueden eliminar varias entradas/salidas si se mantienen datos redundantes o si estos datos no se mantienen se puede salvar espacio, pero esto costaría más tiempo. Sin embargo, uno no se debe dejar llevar por las optimizaciones de espacio y tiempo y olvidar lo concerniente a los requerimientos comerciales.

Es posible que desde el punto de vista comercial, sea necesario tener múltiples puntos de entrada a una base de datos o accesar un tipo de registro particular con múltiples claves. Para proporcionar este tipo de acceso, puede ser necesario invertir un

archivo en esas claves, haciendo, por lo tanto, algunos gastos de espacio y/o tiempo. Si es así, ése es el precio que se tiene que pagar para satisfacer este requerimiento en particular.

Se requiere que el diseñador del modelo físico de una base de datos sea diestro en estas tres importantes áreas:

- **Conocimiento de las funciones del DBMS**. En esta área el diseñador del modelo físico tiene que ser experto en el conocimiento del DBMS, sea éste DB2, ORACLE, INFORMIX, por nombrar algunos; en otras palabras, el diseñador tiene que saber la forma en que el DBMS efectúa sus funciones específicas.
- **Comprensión de las características de los dipositivos de acceso directo**. Esta área involucra las características de los dispositivos de acceso directo. Al diseñar el modelo físico de una base de datos, necesitamos preocuparnos de los aspectos físicos de la base de datos, es decir, la disposición del registro en disco, tamaños de los bloques, tamaños de los buffers y características de entrada y salida, por nombrar algunos ejemplos. Si estos aspectos no se consideran en el modelo físico, la base de datos puede fracasar como consecuencia de un funcionamiento deficiente.
- **Conocimiento de las aplicaciones**. El diseñador tiene que saber las relaciones de los campos de datos y de las entidades referenciados por las aplicaciones, es decir, los modelos externos. También debe conocer las transacciones de gran volumen y los requerimientos de tiempo de respuesta para un medio en línea. Es necesario conocer los programas principales por lotes (batch) y los requerimientos de tiempo de respuesta crítica en un ambiente por lotes (batch).

La evaluación del modelo físico antes de la implantación de la base de datos y la elaboración de cualquier código de aplicación, constituye un método deseable de comparación de alternativas de diseño. Dos estimaciones que juegan un papel importante en el proceso de evaluación son espacio y tiempo.

Las estimaciones de espacio, es un punto del diseño donde al diseñador le conciernen:

- Los cálculos precisos del espacio del dispositivo de almacenamiento de acceso directo
- La longitud de los registros físicos de la base de datos en bytes.
- La volatibilidad, expandibilidad y la disponibilidad de los datos.
- La distribución de espacio. en los medios de almacenamiento.
- El tamaño de la base de datos

Las estimaciones de tiempo, se enfocan al problema que tienen la mayoria de los DBMS, una de las actividades que más consumen tiempo en la ejecución en los accesos a la base de datos es la de entrada/salida fisica. Al analizar los requerimientos de tiempo de las aplicaciones de la base de datos, se deben estudiar los principales programas de proceso en lotes (batch), el alto volumen de transacciones y las transacciones con caracteristicas criticas de tiempo de respuesta. El objetivo del análisis es estimar el tiempo de promedio de CPU y el tiempo de entrada/salida requerido para procesar una transacción o para ejecutar un acceso especifico desde un programa, esto nos ayudará a localizar las transacciones que usan tiempo de CPU y E/S en mayor medida y proporcionará una comparación de las alternativas de diseño para un mejor funcionamiento. Una de estas alternativas es la indexación, es decir, el uso de índices para optimizar los accesos a los datos con previo análisis de que implica el uso de estas estructuras considerando el espacio y tiempo.

El conjunto de criterios que determina el diseño fisico es diferente del que determina el diseño lógico. El diseño fisico está principalmente basado en la necesidad de asegurar el buen rendimiento operativo, adecuar los tiempos de respuesta y en la minimización de los costos de CPU y E/S.

1.4 SISTEMAS MANEJADORES DE BASES DE DATOS (DBMS)

Es un sistema cuyo propósito global es mantener la información y hacer que esta información esté disponible cuando y donde se le demande. Consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. La colección de datos, normalmente denominada base de datos, contiene información acerca de una empresa determinada. También se considera como un sistema de software usado psra manejar y mantener datos de una o múltiples aplicaciones al mismo tiempo para diferentes propósitos, independientemente de la clase de dispositivos de almacenamiento o métodos de acceso.

Componentes funcionales de un DBMS (Data Base Management System):

- Gestor de archivos: Asigna el espacio en la memoria del disco y la estructura de datos que se usa para representar información almacenada en el disco.
- Gestor de registros intermedios buffer: Transfiere la información entre la memoria del disco y la memoria principal.
- Analizador sintáctico de consultas: Traduce sentencias de un lenguaje de consulta a un lenguaje de nivel más bajo.
- Selector de estrategias: Intenta transformar la solicitud del usuario en una forma equivalente pero más eficiente, encontrando así una estrategia adecuada para ejecutar la consulta.
- Gestor de autorización e integridad: Prueba la satisfacción de las restricciones de integridad y comprueba que el usuario está autorizado para acceder a los datos.
- Gestor de recuperaciones: Asegura que la base de datos permanezca en un estado consistente (correcto) a pesar de que ocurran fallos en el sistema.
- Controlador de concurrencia: Asegura que las interacciones concurrentes con la base de datos se lleven a cabo sin conflictos entre ellas.

Además se requieren varias estructuras de datos como parte de la implementación física, entre las que se encuentran:

- Archivos de datos. Almacena la base de datos.
- Diccionario de datos. Almacena información acerca de la estructura de la base de datos y la información de autorización.
- Indices. Permiten tener acceso rápido a datos que tienen determinados valores.
- Datos estadísticos. Almacena información acerca de los datos en la base de datos. Esta información la utiliza el selector de estrategias.

Tareas de un DBMS:

- Mantener la consistencia.
- Resolver los problemas de concurrencia.
- Promover una interfaz universal a los datos.
- Regular el acceso a los datos.

Un DBMS debe incorporar:

- Independencia de los programas de aplicación respecto a los cambios en la estructura de los datos.
- Soporte de lenguajes de programación.
- Programación de utilería para facilitar la creación, mantenimiento y reestructuración de las bases de datos.
- Facilidades para la reorganización de los datos.
- Habilidad para efectuar la seguridad de los datos e imponer límites de acceso a ellos.
- Capacidad de reinicio (RESTART) automático en caso de falla del sistema.
- Habilidad para recuperar las operaciones en forma manual con esfuerzo mínimo.
- Facilidad del sistema para la afinación del DBMS.

Características de un sistema manejador de bases de datos relacionales (RDBMS):

- Representación de datos a través de tablas.
- Desarrollo a través de herramientas de alta productividad.
- Capacidades relacionales completas.
- Flexibilidad :
  - En el mantenimiento de las estructuras.
  - En el mantenimiento de los datos.
  - En el tipo de consultas.
- Diccionario de datos integrado.

Ventajas de un RDBMS

- Simplicidad
  - Fácil de usar
  - Fácil de obtener respuestas.
  - Fácil de insertar y actualizar datos.
  - Fácil de cambiar la estructura de los datos.
  - La navegación es responsabilidad del DBMS, no del programador.
- Poder. Todas las consultas son posibles.

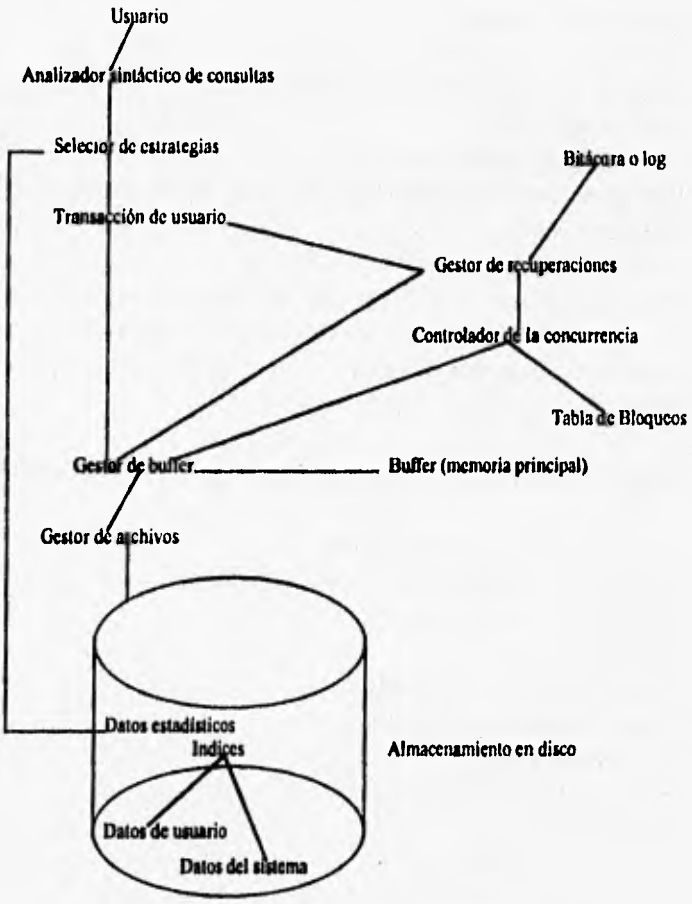


Figura 1.17. Estructura de un DBMS.

1.5 ADMINISTRADOR DE BASES DE DATOS (DBA)

Una de las razones principales para tener sistemas de gestión de bases de datos es tener control central de los datos y de los programas que accesan a esos datos. La persona que tiene dicho control central sobre el sistema se llama administrador de las base de datos (DBA - Data Base Administrator).

Un DBA debe poseer:

- Alto grado técnico.
- Habilidad para entender e interpretar los requerimientos de información de la empresa a nivel de alta administración.

Las funciones principales del DBA son las siguientes:

- Decidir el contenido de la base de datos.
- Decidir la estructura de almacenamiento y estrategias de acceso (modelado físico).
- Ser el enlace con los usuarios.
- Definir las validaciones de integridad y seguridad.
- Monitorear el rendimiento y actuar en consecuencia.
- Responder a requerimientos cambiantes.
- Uso de las rutinas de carga de archivos externos de la base de datos.
- Uso de las rutinas de respaldo (BACKUP) y restauración (RESTORE).
- Uso de las rutinas de reorganización, ya sea para obtener mayor rendimiento o recuperación de espacios.
- Uso de las rutinas estadísticas.
- Uso de las rutinas de análisis de rendimiento.
- Manejo del diccionario del sistema.

1.6 DICCIONARIO DE DATOS

Un sistema de base de datos relacional necesita almacenar información acerca de la estructura de la base de datos, y la información de autorización, como las restricciones de clave; es decir un sistema manejador de base de datos requiere mantener datos acerca de las relaciones. Esta información almacenada y mantenida se denomina diccionario de datos, o catálogo de datos. Las características principales del diccionario de datos son las siguientes:

- Es una ayuda para identificar y clasificar los datos almacenados en la base de datos.
- Consiste de archivos, registros y campos que contienen información descriptiva de los archivos, registros y campos de la base de datos.
- Define el tipo de valor que debe ir en cada campo.
- Al introducir nuevas aplicaciones al sistema y nuevos datos, el diseño de la base de datos se debe modificar para reflejar estas nuevas entradas. El diccionario de datos podrá ayudarnos a determinar el impacto de los cambios.
- Es una herramienta de documentación automática, auxilia al administrador de base de datos en la realización de sus funciones.

Entre los tipos de información que el sistema debe almacenar están:

- Los nombres de las relaciones.
- Los nombres de los atributos de cada relación.
- Los dominios de los atributos.
- Los nombres de las vistas definidas en la base de datos y la definición de esas vistas.
- Las restricciones de integridad de cada relación.
- Nombres de los usuarios autorizados.
- Información contable acerca de los Usuarios.
- Número de tuplas de cada relación.

El sistema también almacena información acerca de cada índice de cada una de las relaciones.

- Nombre del índice.
- Nombre de la relación que se indexa.
- Atributos sobre los que está el índice.
- Tipo de índice.

1.7 UTILERIAS DE UN SISTEMA MANEJADOR DE BASES DE DATOS RELACIONALES

Una ves que es evaluado el diseño fisico, se puede implantar. Al DBMS se le tiene que proporcionar las especificaciones de los métodos de acceso, así como varios detalles de recuperación, actualización, adición y supresión de datos. Dado que tareas diarias se efectuarán en la base de datos implantada y que existe el riesgo que el software como el hardware puedan fallar, es absolutamente necesario establecer procedimientos de recuperación y respaldo en cada ambiente de base de datos.

Un buen diseño fisico no puede proveer un funcionamiento bueno y consistente con datos volátiles, a pesar de las características de autoreorganización, de algunos DBMS, la mayoría de los DBMS proporcionan rutinas de reorganización, ya que el diseño fisico de una base de datos se debe evaluar pensando en una reorganización posible.

Una vez implantada la base de datos, uno de los principales problemas de la dirección es mantenerla disponible y a nivel de usuarios.Un DBMS deberá ser capaz de apoyar con herramientas que tengan una capacidad general y eficiente de dar mantenimiento a las estructuras de sus base de datos.

En el mantenimiento de una base de datos se especifican las siguientes operaciones:

Réplicas o respaldos de los datos. La creación de respaldos aumentan los costos de actualización y de almacenamiento, pero sus beneficios pueden sobrepasar estos costos especialmente en el caso de los datos que se leen con mucho más frecuencia de lo que se actualizan. Hay que considerar que el contar con respaldos aumenta la seguridad y confiabilidad para el caso de recuperación.

Reorganización de los datos. Cuando una base de datos agota el espacio extra, debido a inserciones, actualizaciones y a huecos dejados por el borrado de registros, puede resultar necesaria una reorganización de una base de datos. Por lo tanto, una reorganización optimiza el espacio asignado.

Recuperación a partir de un respaldo. Cuando un proceso de actualización termina anormalmente y no existe punto de reinicio se determina recuperar los datos como se encontraban en el punto de inicio, a través del último respaldo realizado.

El responsable de explotarlas o recomendarlas es el DBA, a través de una óptima programación, auxiliándose de un adecuado y actualizado diccionario de datos en el DBMS.

A estas herramientas es lo que comúnmente se les llama programas de utilerías de un DBMS. Además se debe establecer un período de ejecución de acuerdo a un análisis del uso (updates, consultas, borrado de datos) de la base de datos, para tener un alto porcentaje de confiabilidad y seguridad de los datos.

Esta situación ha creado una gran competencia en el mercado de software para proveer herramientas al DBA, es su responsabilidad explotarlas al máximo. El crecimiento de una Administración de base de datos es en relación hacia el volumen de datos de la empresa, viendo los datos como un activo de la organización, dicho crecimiento está enfocado a optimizar los recursos tanto en hardware (HW) como en software (SW).

La Dirección de Sistemas y Planeación BANCOMER S.A, cuenta con un ambiente de bases de datos para el desarrollo de sus sistemas, teniendo como sistema manejador de base de datos relacional el DB2, en este ambiente se apoyará esta metodolgía del diseño o modelo físico (DB2DA) para su desarrollo

2 VIABILIDAD DEL PROYECTO

2.1 Situación actual.

2.2 Identificación de la problemática.

2.3 Descripción funcional.

2.4 Objetivos de la metodología y sistema.

2.5 Justificación del sistema.

2.6 Estimación de tiempo y costos del proyecto.

2.7 Requerimientos de hardware y software (DB2).

2.1 SITUACION ACTUAL

La Dirección de Sistemas y Planeación de BANCOMER, S.A. cuenta con un ambiente de base de datos DB2 para el desarrollo de sistemas en sus aplicaciones, (Clientes, Cobranzas, Contabilidad, SAR etc.). Este se compone de 3 fases para ser liberado en un ambiente de producción, el cual se explica en el siguiente cuadro:

AMBIENTE	DBMS[i]	EQUIPO O MAQUINA
Desarrollo	DB2	S90C
Pruebas	DB2 , DB2T	S90C
Producción	DB2D, DB2Q	S90D

El proceso de liberación y mantenimiento de sistemas bajo DB2 al ambiente de producción abarca todas las aplicaciones que se encuentren bajo el manejador de datos DB2 y que se encuentren en la etapa de transición al ambiente de producción es decir que han cubierto las fases de desarrollo y pruebas del sistema.

El área de Administración de Base de Datos revisa y aprueba el proceso de liberación de sistemas en DB2 de las aplicaciones a producción. Las funciones que realiza y que son de interés para este caso de estudio (diseño físico y mantenimiento de bases de datos) son las siguientes:

- Validación del modelo de datos.
- *Elaboración del diseño físico.*
- Revisión de los *procedimientos de mantenimiento, respaldo y recuperación* de las Bases de Datos.
- Validación de que exista disponibilidad del medio de almacenamiento.

[i] Identificador del sistema manejador de base de datos. (Systemid)

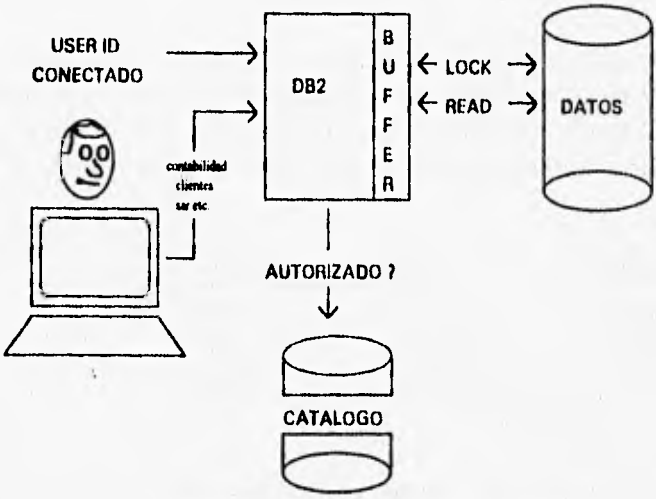


Figura 2.1. Aplicaciones en DB2.

Existen distintos procesos en la liberación y mantenimiento de un sistema en DB2, estos se componen de un conjunto de actividades donde algunas son comunes.

- Instalación de un nuevo sistema o nuevas tablas :
  1) El diseño físico.
  2) Codificación al lenguaje de definición de datos.
  3) Generación de objetos en DB2.
  4) Respaldo de datos.

- Modificación a la estructura física y lógica de tablas ya instaladas:
  1) El diseño físico.
  2) Codificación al lenguaje de definición de datos
  3) Respaldo de datos.
  4) Ejecución del ALTER (modificación).
  5) Reorganización de datos.
  6) Respaldo de datos.

- Instalación de índices adicionales a un sistema:
  1) El diseño físico.
  2) Codificacón al lenguaje de definición de datos.
  3) Generación de objetos en DB2.
  4) Reorganización de datos.

- Remplazo de una tabla mediante la operación DROP/CREATE
  1) El diseño físico.
  2) Codificación al lenguaje de definición de datos.
  3) Generación de objetos en DB2.
  4) Respaldo de datos.

El diseño físico resulta fundamental en todos los procesos porque en la calidad de su elaboración se apoyará el éxito de la codificación al lenguage de definición de datos (También conocido como DDL Data Definition Language), esta última tiene precedencia para la generación de objetos (tablas e índices) en DB2. La dependencia de actividades crean la necesidad y prerequisito de invertir más tiempo y análisis en la elaboración del diseño físico para minimizar los posibles impactos en la instalación y operación de los sistemas. El diseño físico tiene la precedencia de una revisión y aprobación del modelo de datos en los rubros tales como estándares de nomenglatura y normalización, actividades que están fuera del alcance de este proyecto, el enfoque y análisis que tendrá éste será hacia la actividad del diseño físico y recomendaciones para el mantenimiento de las bases de datos.

El diseño físico se apoya en el formato llamado D821[i] el cual consiste en información relevante de tablas e índices tal como el tipo de espacio para tablas e índices, longitud del registro de las tablas, cardinalidad de las tablas, tipo de actualización, crecimiento esperado, etc, dicha información es obtenida en actividades previamente realizadas[ii] y son base en las recomendaciones y cálculo de parámetros para el diseño físico.

El producto final del diseño físico se refleja en los formatos D822 y D824[iii]. El D822 contiene las recomendaciones acerca del mantenimiento de tablas e índices, información tal como frecuencia de respaldo, tipo de respaldo, frecuencia de reorganización y verificación de índices, etc. El D824 consiste en las recomendaciones relevantes del diseño físico para

[i] Ver en el apéndice formatos D821.

[ii] La obtención de esta información se estudiará en el capítulo 3.

[iii] Ver en el apéndice formatos D822 y D824.

tablas e índices, cantidad primaria y secundaria de espacio en disco, tipo de espacio (tablespace) para tablas e índices , grupo de almacenamiento (storage-group), etc.

En base a la información del formato D821 y reglas de cálculo para el espacio[1] en disco, se obtienen las recomendaciones especificadas en los formatos D822 y D824, tal información tiene el objetivo de dar a los usuarios un alto rendimiento a la operación y disponibilidad de las bases de datos DB2.

2.2 IDENTIFICACION DE LA PROBLEMATICA

El proceso de liberación a producción de un sistema con información en DB2, consiste en su implementación e instalación a través de 3 fases: desarrollo, pruebas y producción. Los problemas se identifican en los sistemas con mayor frecuencia durante la transición de fases y con un mayor impacto durante su operación y mantenimiento en producción.

La identificación de anomalías se ve más remarcada en la transición de pruebas a producción, debido principalmente a los factores de volumen de información que alcanza sus valores máximos al esperado, al espacio físico, tiempo de respuesta al accesar datos y la disponibilidad de la información que requiere un elevado nivel de servicio.

Dentro de esta metodología se identifican los siguientes problemas:

- Errores en el cálculo de espacio.
- Abortación de programas por espacio, debido al crecimiento no esperado de una base de datos de un ambiente a otro.
- Periodicidad no confiable de respaldos de las bases de datos para el caso de recuperación de los datos y definición.
- Periodicidad no confiable de reorganización y depuración de los datos.
- Tiempo elevado de E/S en la consulta de tablas de datos (falta de índices).
- Falta de documentación en los cambios realizados en la definición de tablas e índices.
- Elevado costo en el tiempo de liberación de un ambiente a otro.

[1] Las reglas de cálculo tienen su referencia en el manual de administración de base de datos DB2. Ver en el apéndice reglas de cálculo.

- Elevado costo en el tiempo de búsqueda de documentación en las definiciones generadas en el diseño físico.
- La aplicación de reglas de cálculo para obtener algunos parámetros en el diseño físico, representa un elevado gasto de tiempo y la posibilidad de errores humanos.

Este conjunto de problemas en los sistemas produce una inestabilidad de las aplicaciones en el ambiente de producción, teniendo como causa principal el inadecuado análisis para la obtención de parámetros en el diseño físico y la frecuencia programada no confiable para utilerías de respaldo y mantenimiento de las bases de datos.

2.3 DESCRIPCION FUNCIONAL

Esta metodología pretende ser una guía en la obtención del modelo físico aplicable a cualquier RDBMS (Relational Data Base Management System) en el mercado de software, además se construirá el diseño del sistema que automatizará la generación de los parámetros[1] de diseño físico de tablas e índices para DB2 como son:

- Tipo de espacio para la tabla (tablespace).
- Espacio primario y secundario.
- Espacio libre reservado.
- Tamaño de segmentos o cantidad de particiones de las tablas.
- Estado del espacio de tabla o índice (abierto o cerrado).
- Tamaño de subpáginas para índices.
- Niveles estimados para un índice.

Además de recomendar que utilerías de mantenimiento a las bases de datos tales como reorganización y respaldo de datos deben usar y con que frecuencia ejecutarlas. También generará los programas que ejecutarán las utilerías codificadas en JCL[1] (lenguaje de programación del sistema operativo MVS).

[1] Los parámetros del diseño físico se estudiarán en el capítulo 3.

[1] Ver el punto de requerimientos de hardware y software de este capítulo.

2.4 OBJETIVOS DE LA METODOLOGIA Y SISTEMA

- Reducir el tiempo de liberación a producción de un sistema a través de una metodología y la automatización.
- Aumentar la estabilidad de un sistema en producción a través de una adecuada definición de los parámetros en el diseño físico de las bases de datos, programación de utilerías de respaldo y recuperación.
- Documentar los estimados dados por el usuario, así como las definiciones generadas.

2.5 JUSTIFICACION DEL SISTEMA

Debido al crecimiento de los sistemas en las aplicaciones que corren bajo de DB2 y la importancia de la información, es necesario automatizar las actividades para el cálculo de parámetros de diseño físico de tablas DB2, ya que se realizan actualmente en forma manual y tediosa. Esto origina la generación de información poco confiable, por lo tanto se produce como consecuencia un retraso en su liberación a producción; además de no tener un mantenimiento adecuado a las bases de datos (respaldo y reorganización de datos). Por lo que es conveniente desarrollar un sistema que proporcione a la administración de base de datos la generación de los valores de los parámetros en forma confiable y rápida, en base a los estimados de volumen que el usuario espera. También proporcionará una recomendación de la frecuencia con que se deberán de ejecutar las utilerías de respaldo y reorganización de datos.

Beneficios que se obtendrán al desarrollar la metodología y el sistema:

- Se satisfacerán los requerimientos del usuario.
- Se tendrá un nivel elevado de seguridad sobre la información que será manejada.
- Disminuirá la tolerancia de errores.
- Se manejará la información con rapidez minimizando tiempos muertos.

2.6 ESTIMACION DE TIEMPO Y COSTOS DEL PROYECTO

Antes de establecer la planeación del proyecto es necesario obtener estimaciones del esfuerzo humano requerido, de la duración cronológica del proyecto y del coste.

Una de las técnicas más usada para la estimación de costos y la cual emplearemos en el proyecto, debido a que se acopla a nuestra situación actual, es la técnica de juicio experto.

El Area de Administración de Bases de Datos ha adquirido una experiencia eficiente, a través de las siguientes funciones a su cargo:

- Implementación, operación y mantenimiento de las bases de datos DB2.
- Manejo y conocimiento de las interfaces que realiza DB2 con los diferentes sistemas (software) que lo auxilian.
- Elaboración de programas de aplicación para manipular bases de datos DB2.

Considerando lo anterior se puede establecer una estimación de bajo costo, debido a que los recursos de cómputo y humano ya se tienen. Por lo tanto, tomando como base la experiencia con la que se cuenta, se estimaron los siguientes factores que repercuten en la planeación del proyecto:

- El número de líneas de código esperado es de 10,000 líneas.
- El número de personas involucradas en todo el desarrollo del proyecto esperado es de máximo 5 personas.
- La duración cronológica del proyecto esperado es de 6 meses.

El sistema se ha planeado desarrollar en el lenguaje de programación REXX.

Una vez que se estableció la estimación de valores, de los principales factores que interfieren con la planeación del proyecto, se tiene el siguiente plan tentativo de actividades para desarrollar el proyecto:

- Una vez que se planteó el problema se iniciará a identificar los pasos que constituyen la metodología y modelar la entidades que integrarán al sistema, para que así se proceda a establecer las relaciones entre cada una de ellas. Así como establecer las entradas y salidas de cada una de éstas.

- Ya que el problema esté bien definido, se pasará a la etapa del diseño que consta de la serie de pasos de la metodología y una arquitectura del software, detallando lo más que se pueda para que en la etapa de codificación se simplifique.
- Se elabora el pseudocódigo del sistema y se realizarán pruebas para comprobar que se cumplen los requerimientos preestablecidos.
- Una vez que el sistema supere las pruebas de escritorio establecidas en la fase de diseño se codificará e implantará.
- Se realizará un mantenimiento periódico o cuando el sistema lo requiera.

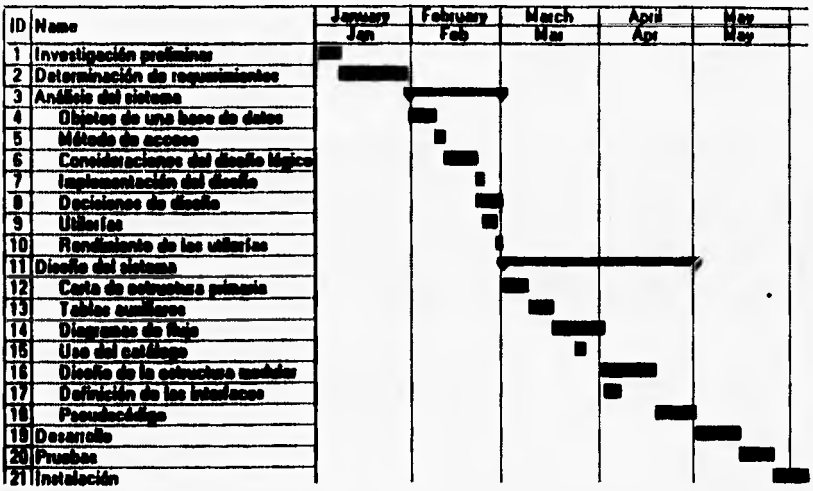


Figura 2.2. Lista de actividades graficadas en la carta de Gantt.

La figura 2.3 muestra la distribución de esfuerzo relativo esperado, durante el ciclo de vida del proyecto, tomando como referencia la técnica de juicio experto.

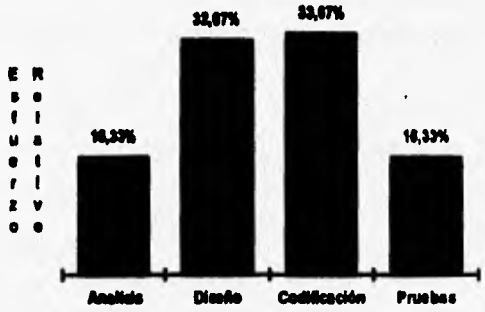


Figura 2.3. Distribución de esfuerzo relativo esperado.

2.7 REQUERIMIENTOS DE HARDWARE Y SOFTWARE (DB2)

El ambiente que permitirá implantar la metodología y al sistema DB2DA operar se va a generar considerando los recursos humanos de software y hardware, que existen en el Area de Administración de Base de Datos y que forma parte de la Dirección de Desarrollo de Sistemas 'A' de BANCOMER, S.A.

La implementación de DB2DA tendrá como principales requerimientos los siguientes:

- **Sistema operativo MVS:** MVS significa Multiple Virtual Storage, trabaja en arquitectura sistema 370 ó 390, sus principales características operativas son las siguientes:
  - Sistema Multiusuario.
  - Multiprogramación.
  - Soporta múltiples CPU's.
  - Utilización de Spool.
  - Soporta comunicaciones programa a programa (APPC).
  - Permite comunicaciones locales y remotas.
  - Soporta discos compartidos.
  - 2 Gigabytes de memoria virtual y real.
  - Maneja cadenas de datos de hasta 16 terabytes en memoria real.
  - Memoria expandida hasta 8 Gigabytes.
  - Posicionado fuertemente en seguridad, integridad y confiabilidad.

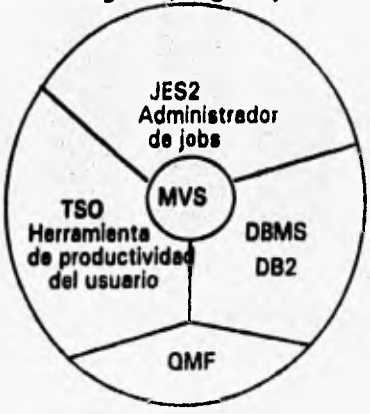


Figura 2.4. Entorno del MVS.

- **El procesador** del sistema 390, forma parte de la familia de procesadores **IBM ES/9000** (Enter System /9000) modelo 720. Las principales características son:
  - Permite una arquitectura común, arquitectura S/370 ó S/390.
  - Utiliza software de sistema común.
  - Permite aplicaciones comunes.
  - Utiliza canales de I/O (input/ output).
  - Soporta un ambiente operacional común.
  - Opciones de crecimiento gradual.
  - Administrador de multisistemas.
  - Soporta hasta 9,216 Megabytes de memoria central y expandida.
  - Facilidad de movimiento a nivel página (una pagina = 1024Kbytes).
  - Soporta hasta 256 canales de I/O.
  - Soporta el sistema operativo MVS.

- **EL subsistema JES** (Job Entry Subsystem) del MVS: JES adapta los requerimientos de un programa o tarea en ejecución comúnmente llamado "job" para el MVS y establece el tiempo de trabajo para ser procesado. JES permitirá que los programas generados por DB2DA puedan ser ejecutados a traves del MVS.

- **El lenjuage JCL** (Job Control Languaje): JCL es un lenguaje de control del MVS, es usado para describir al sistema los requerimientos para los programas, archivos, dispositivos, volúmenes, prioridades, etc. Los programas generados por DB2DA serán codificados en JCL.

- **El sistema manejador de bases de datos relacional DB2** de IBM opera como un subsistema formal del sistema operativo MVS. Los programas de aplicación de DB2 pueden correr en un ambiente batch y sus programas de utilerías mediante la herramienta TSO del MVS, también el MVS provee facilidades de conexión para el DB2 con otras herramientas de software (QMF, JES, REXX).

- **El producto QMF** (Query Management Facility) de DB2, es utilizado para consultas interactivas y en batch, además cuenta con la facilidad de generación de reportes. Esta diseñado para permitir a los usuarios un manejo amigable en el uso de la información que maneja DB2. Facilidades del QMF:
  - Accesar datos almacenados en las tablas de DB2.
  - Ejecutar cálculos sobre los datos de DB2.
  - Produce reportes en diferentes formatos.
  - Inserta , cambia o borra datos.
  - Comunicación con otros productos.

QMF proporciona un lenguaje de consulta para accesar los datos, llamado SQL. SQL es el mismo lenguaje usado en DB2. Los usuarios de QMF pueden exportar queries de SQL a data sets (archivos) de TSO.

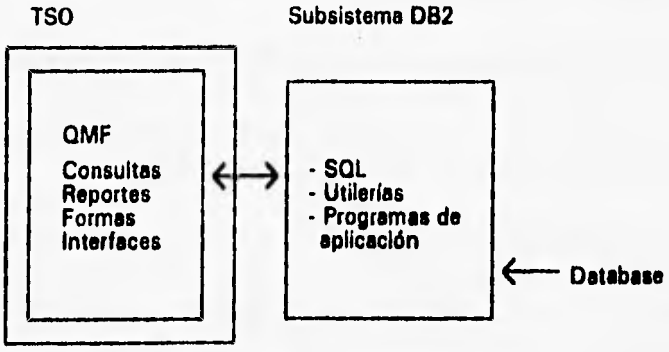


Figura 2.5. Relación de DB2 con QMF.

- **El software TSO/E** (Time Sharing Options Extensions) : TSO trabaja en el ambiente del sistema operativo MVS, es una herramienta de productividad, con la que se puede realizar las siguientes actividades:
  - Manipulan datos y archivos.
  - Monitorean varios ambientes operativos.
  - Procesan compiladores en ambientes batch y conversacionales.
  - Comunica con funciones de proceso jerárquico y relacional.
  - Administra la operación del sistema, su seguridad y se transfieren datos.
  - La herramienta TSO permite al MVS efectuar el trabajo interactivo, es decir, soporta a una amplia varidad de usuarios para realizar diferentes clases de trabajo.

- Permite al programador del sistema mantener al MVS y otros productos asociados para estar disponibles para su uso.
- Los programadores y analistas pueden editar, compilar, y probar programas de aplicación a través de TSO.
- A un usuario final le permite accesar bases de datos corporativas para tomar decisiones.

El TSO permitirá al DB2DA comunicarse con el software que utilizará como son:

- El sistema manejador de base de datos relacional DB2
- El sistema operativo MVS
- EL subsistema JES del MVS.
- La herramienta de DB2 QMF.
- EL intérprete del lenguage REXX.

- El producto **TSO/E REXX** (Restructured EXtended eXecutor), es un lenjuage de procedimientos e intérprete. Su implementación se realiza en un ambiente de TSO. REXX cuenta con un conjunto de interfaces con otros productos (QMF, DB2, TSO) para el diseño y desarrollo de aplicaciones. DB2DA se desarrollará en REXX debido a que cubre lss necesidades del sistema.

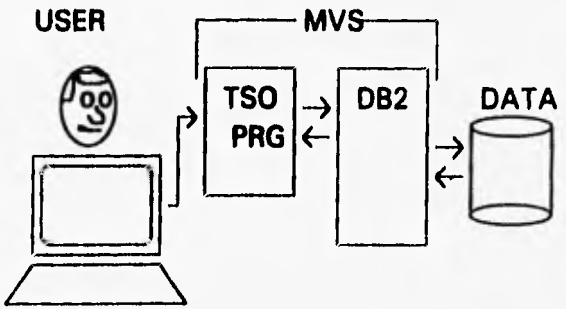


Figura 2.6. Software utilizado por el usuario para accesar información.

3 ANALISIS DE LA METODOLOGIA Y SISTEMA

3.1 Objetos de una base de datos.

3.2 Método de acceso para almacenamiento virtual (VSAM).

3.3 Organización y estructura de los data sets VSAM.

3.4 Consideraciones del diseño lógico.

3.5 Implementación del diseño en DB2.

3.6 Decisiones de diseño.

3.7 Descripción de las utilerías de DB2.

3.8 Rendimiento de las utilerías en el diseño físico.

3.1 OBJETOS DE UNA BASE DE DATOS EN DB2

En el diseño de una base datos es conveniente dividirlo en dos partes, en el diseño lógico y físico, ya que la parte lógica corresponde a la representación del mundo real, a través de esquemas y modelos de datos, sin que tenga una dependencia del sistema manejador de base de datos, y la parte física resulta de la implementación física del diseño lógico teniendo en cuenta ya las características del sistema manejador de base de datos específico.

Dada la problemática previamente mencionada es importante analizar las estructuras que intervienen en el diseño lógico y físico, siendo este último el más esencial, porque la mayor parte de los problemas se detectan en esta fase. Por lo tanto, es importante identificar la organización de los datos que maneja DB2.

DB2 organiza los datos a través de un sistema de entidades lógicas y físicas llamados *objetos*. Por ejemplo, las tablas e índices son objetos. Una database, la cual es una colección de tablas relacionadas, es también un objeto. Los objetos que describen una database en el transcurso que los usuarios y desarrolladores piensan acerca de estos, son llamados objetos lógicos. Aquellos que se refieren a la manera en que los datos son almacenados en el sistema son llamados objetos físicos. Entre los objetos más importantes del DB2 están:

- Tables (tablas)
- Indexes (Indices)
- Views (Vistas)
- Storage Structures (Estructuras de almacenamiento)
- Storage Groups (Grupos de almacenamiento)
- Databases (Base de Datos)
- Synonyms and Aliases (Alias y Sinónimos)

Una tabla es la unidad básica de DB2 para la definición de datos. Frecuentemente una tabla es referenciada como una tabla base para distinguirla de una view (vista), la cual es una tabla virtual derivada de una o más tablas bases. Los usuarios perciben las tablas, como un conjunto de valores arreglados en renglones y columnas. Una tabla con r renglones y c columnas contienen un total de r veces c valores: un valor por cada combinación de renglón y columna. Cada columna es un conjunto de r valores; cada

renglón es una secuencia de c valores tal que el n'esimo valor en la secuencia está en la n'esima columna de la tabla.

Cada tabla puede tener una o más columnas, pero el número de renglones puede ser cero. Una tabla con cero renglones es llamada tabla vacía.

Cada columna de la tabla tiene un tipo de dato. El tipo de dato de la columna es el tipo de dato de sus valores, el cual determina como esos valores están representados.

Una tabla representa una clase de entidad, cada renglón representa un miembro de la clase y cada columna representa un atributo de los miembros. La siguiente tabla ilustra estos conceptos.

EMPNO	WORKDEPT	HIREDATE	SALARY	BONUS
000010	A00	1965-01-01	52750.00	1000.00
000070	D21	1980-09-30	36170.00	0700.00
000160	D11	1977-10-11	22250.00	0400.00
000280	E11	1967-03-24	26550.00	0500.00

La tabla anterior muestra los valores en cuatro renglones y cinco columnas de una tabla llamada DSN8230.EMP. Para esta tabla, las entidades representadas son los empleados en una empresa. Cada renglón representa un empleado. Cada columna representa un atributo de empleado, como se muestra a continuación:

columna	*atributo*
EMPNO	Identificador del empleado
WORKDEPT	Identificación del departamento del empleado
HIREDATE	Fecha de contratación
SALARY	Salario del empleado
BONUS	Bonos del empleado

Un índice es un conjunto ordenado de apuntadores a los renglones de un tabla base, el DB2 construye está estructura y la mantiene automáticamente.

Una vista proporciona vistas particulares de tablas base. Las Views son tablas virtuales, compuestas por columnas y renglones de tablas base y otras vistas, pero sus datos no son almacenados separadamente. Proporciona una manera de ver los datos de una o más tablas. Dentro de su definición en términos de un query de SQL (select) y un nombre para ser asignado a la vista.

Un storage structure es un conjunto de uno o más datos que son tomados de las tablas o índices de DB2. Los conjuntos de datos son manejados por métodos de acceso. Un storage structure puede ser:

- **Un table space.** Todas las tablas están guardadas en table spaces. Un table space puede tomar una o más tablas.
- **Un index space.** Cada index space contiene sólo un índice. Para un índice dado, un index space está definido.

El mantenimiento para los conjuntos de datos de un storage structure puede ser dejado para DB2. Si es dejado para DB2, la storage structure tiene asociado un storage group. El storage group es esencialmente una lista de volúmenes DASD (direct access storage device) sobre los cuales DB2 puede asignar conjuntos de datos para los storage structures asociados. La asociación entre un storage structure y su strorage group es realizado, implícitamente o explícitamente, por la declaración que crea el storage structure.

Un Storage Group es un conjunto de volúmenes que DB2 utiliza para asignar espacio. Todos los volúmenes en el grupo deben ser del mismo tipo de dispositivo. El usuario pueden asignar espacio vía VSAM (Servicios de métodos de acceso).

Una Database de DB2 es un conjunto de table spaces e index spaces que están relacionadas. Las Databases son utilizadas primordialmente para la administración.

Los **sinónimos y alias** son nombres alternativos para las tablas y vistas.

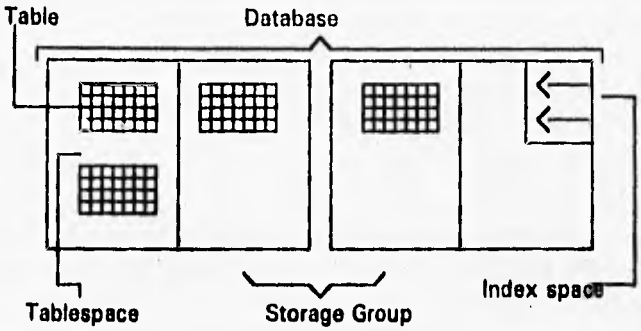


Figura 3.1. Objetos de una Base de Datos en DB2.

Los objetos, anteriormente descritos, participan en todas las fases del diseño de una database en DB2.

El diseño de una *database* tiene las siguientes fases en DB2 :

- Diseño Lógico:
  - Modelado de datos.
    Define las entidades, atributos y relaciones que son usadas para construir *tables (tablas)* DB2. El modelado de datos no será cubierto en este análisis, asumiremos que esta actividad ya ha sido concluida.
  - Diseño de *Tables*
  - Las consideraciones de la normalización de tablas necesitan ser siempre una parte del diseño. Generalmente es recomendable normalizar hasta la tercera forma normal.
  - La integridad referencial debe ser otra parte del diseño.
  - Selección de llaves para indexar.
  - Considerar el crecimiento de tablas para una operación extendida sobre datos.
- Diseño Físico:
  - Tipo de Table space usado
  - Selección de parámetros para la definición de objetos físicos (DDL).

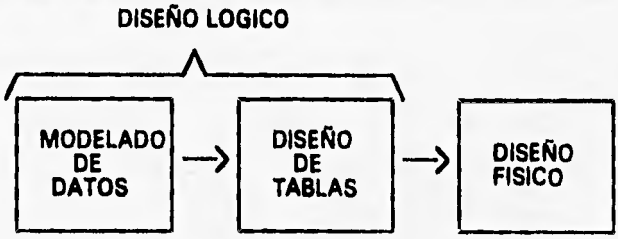


Figura 3.2. Fases del Diseño de Base de datos.

3.2 METODO DE ACCESO PARA ALMACENAMIENTO VIRTUAL (VSAM)

Dentro de la organización física del almacenamiento DB2 distribuye sus datos en las unidades de almacenamiento a través de un método de acceso llamado VSAM (Virtual Storage Access Method). Es por esto que se analizarán los mecanismos de acceso de almacenamiento que utiliza DB2.

VSAM es un método usado sólo por archivos de almacenamiento virtual, fue diseñado especialmente para proporcionar un acceso de alto rendimiento en sistemas de almacenamiento virtual. La organización de los archivos VSAM es secuencial indexada, es decir, obedece a la secuencia de la clave. El método usual de direccionamiento es el que se basa en el empleo de una tabla llamada *índice*. La entrada a la tabla es la clave del registro buscado; el resultado del examen de la tabla es la dirección relativa o la dirección verdadera del registro en el archivo.

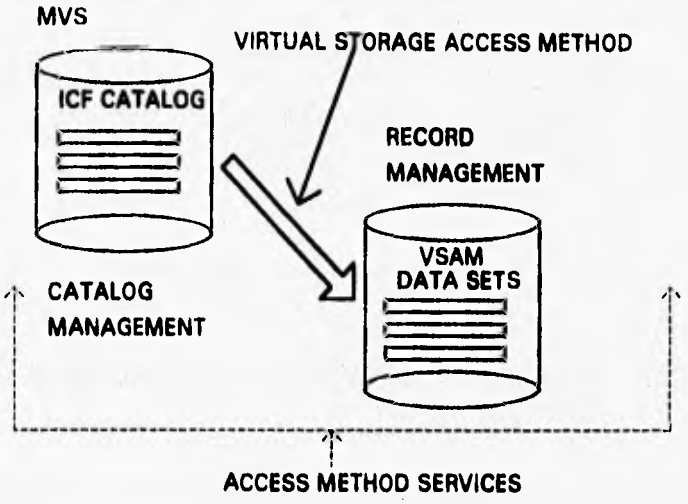
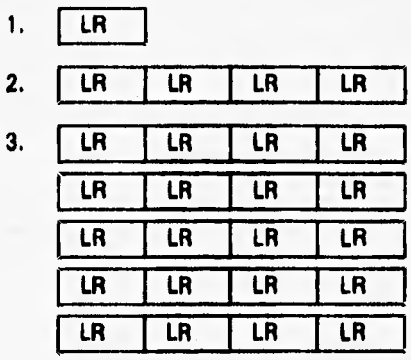


Figura 3.3. Los Data Sets (archivos) VSAM en el ambiente del sistema operativo MVS.

El MVS contiene un catálogo maestro administrador (Catalog Management), también llamado como ICF master Catalog (Integrated Catalog Facility) este cuenta con la información acerca de todos los data sets en el MVS. Un catálogo secundario (Record Management) almacena información de todos los data sets VSAM residentes en dispositivos DASD (Direct Access Storage Device) o discos. La herramienta de software Access Method Services (programa IDCAMS) de VSAM, provee rutinas para la ejecución de funciones tales como copias, borrados, alteraciones, etc, en el catálogo y los data seta.

3.3 ORGANIZACION Y ESTRUCTURA DE LOS DATA SETS VSAM

Construyendo bloques VSAM



Notas:
LR: Logical Record CI: Control Interval CA: Control Area

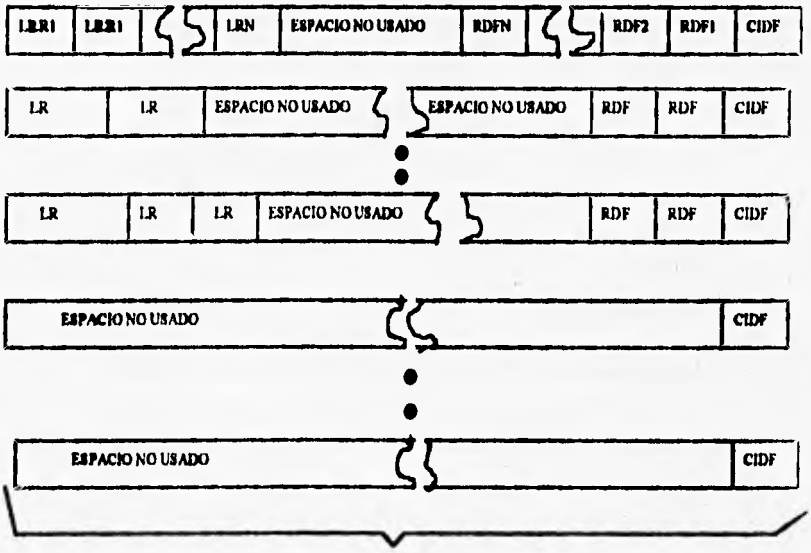
Un registro lógico (LR) es información acerca de un dato específico, un Intervalo de Control (CI) está compuesto de uno o más registros logicos (LR), una área de control la componen uno o más CI. Las CA y CI son usados en todos los tipos de data sets VSAM.

El formato de un intervalo de control tiene las siguientes características:

- Tamaño mínimo: 512 bytes
- Tamaño máximo: 32K

- Los registros lógicos (LR) son almacenados a la derecha del intervalo de control (CI).
- Existe un espacio no usado.
- Información de control: Campo de definición del registro (RDF, Record Definition Field); campo de definición del intervalo de control (CIDF, control Interval Definition Field).

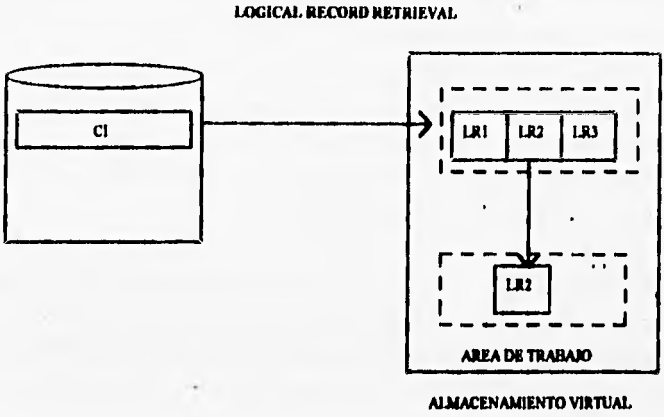
EL formato del área de control tiene las siguientes características:



FORMATO DE AREA DE CONTROL

- Tamaño mínimo: 1 track (1 track = 58K)
- Máximo tamaño: 1 cylinder (1 cylinder = 15 tracks)
- Seleccionado por el data set VSAM basándose en los parámetros de asignación de espacio.
- Todas las asignaciones son múltiples del tamaño de la área de control (CA).

El intervalo de control es la unidad de transmisión entre el DASD y el almacenamiento virtual o Address Space, dicho registro de transmisión se le denomina Logical Record Retrieval, la siguiente figura muestra lo anterior:



Existen 4 tipos de archivos VSAM , en el ambiente del sistema operativo MVS se les denomina comúnmente como Cluster VSAM o Data Set VSAM:

- Lineal Data Set (LDS)

Conjunto de datos secuenciado por clave. Cada registro tiene una clave o llave, cuando se realizan las operaciones de carga de registros o introducción de nuevos registros se realiza por la secuencia de la clave.

- Relative Record Data Set (RRDS)

Conjunto de datos de registro relativo. Los registros se cargan de acuerdo con un número de registro relativo. Si el VSAM asigna el número de registro relativo, se añaden nuevos registros al final del conjunto de datos. Si el programa de usuario maneja la numeración de los registros relativos, los registros nuevos pueden ser añadidos en secuencia de registro relativo. Los registros de un conjunto de datos de registro relativo, ocupan ranuras (Slots) de longitud fija. Las ranuras se numeran secuencialmente de 1 al número de conjunto de datos. El almacenamiento y recuperación de registros se realiza por número de ranura.

- Entry Sequenced Data Set (ESDS)

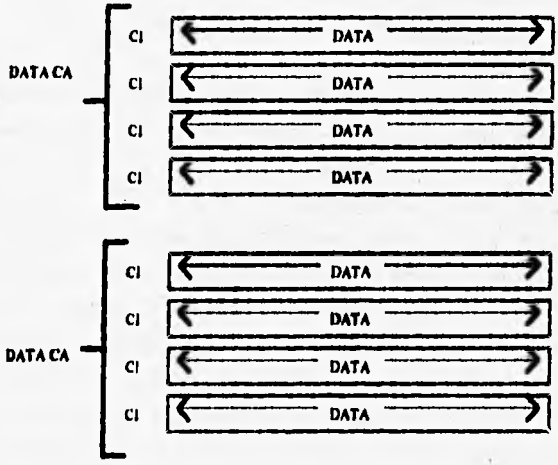
Conjunto de datos secuenciados por entrada. Los registros se cargan en el conjunto de datos de forma secuencial, igual que fueron introducidos. Los registros nuevos se añaden al final del conjunto de datos. Cuando el VSAM almacena un registro en un conjunto de datos secuenciado por entrada, devuelve al usuario la dirección del byte relativo del registro. Opcionalmente el programa de usuario puede crear su propio índice al archivo para permitir el acceso directo a los registros.

- Key Sequenced Data Set (KSDS)

Conjunto de datos secuenciados por llave o clave. Permite la construcción de un índice alternativo, para un conjunto de datos secuenciados por clave o por entrada. Los índices alternativos se utilizan para proporcionar modos múltiples de acceso a un sólo conjunto de datos, eliminando así la necesidad de mantener varias copias del mismo conjunto de datos para diferentes aplicaciones.

Cada cluster VSAM consiste de un componente de datos y para un KSDS, un componente índice. El nombre del cluster y de sus componentes son especificados cuando el data set es definido por Access Method Services. El sistema DB2 almacena los datos de usuarios en LDS VSAM, por lo tanto, enfocaremos nuestra atención en estos.

LINEAR DATA SET (LDS)



- Sólo tiene componente de datos.
- Tamaño del intervalo de control (CI) siempre será 4K (También se utlizan las unidades de almacenamiento llamadas páginas, 1página = 4K).
- No habra información de control en CI.
- El programa de aplicación debe bloquear/desbloquear los registro lógicos (LR).
- Disponible sólo con cátalogos ICF.

Para la asignación de espacio del data set VSAM se utilizan las unidades de almacenamiento Track y Cylinder en al siguiente expresión:

$$\left\{ \begin{matrix} \text{TRACKS} \\ \text{CYLINDERS} \end{matrix} \right\} \text{(primary [,secondary])}$$

Ejemplos: TRACKS(100,3)
CYLINDERS(10,1)
CYLINDERS(1)

- Mínima asignación es 1 track
- La asignacion primaria debe ser suficientemente grande para:
  -. Contener todos los registros de datos inicialmente cargados.
  -. Proveer espacio para el crecimiento.
- La asignación secundaria es opcional.

El tamaño en espacio de un Data Set VSAM requiere los siguientes cálculos:

Adicionalmente necesitamos tener o asumir la siguiente información:

- Tipo de DASD (dispositivo disponibles 3380 y 3390).
- Número total de registros que serán cargados.
- longitud (lenght) del registro.
- Tamaños de CI y CA (CA size = 1 Cylinder)

3.4 CONSIDERACIONES DEL DISEÑO LOGICO

Varios de los problemas que se presentan en la implementación física y operación de las bases de datos tienen su origen en el desarrollo del diseño lógico, y esto se debe a ineficiencias en el diseño lógico, por lo tanto se debe invertir un mayor análisis al desarrollo de éste con lo cual se logrará reducir tiempo y esfuerzo en la implementación física, además de tener el menor número de cambios en la definición de objetos ya instalados, en consecuencia ayudará a mantener una estabilidad en la operación de las bases de datos. Por lo anterior es conveniente hacer énfasis a ciertas consideraciones en el diseño lógico dado que se obtendrán beneficios que se reflejarán en el rendimiento e implementación física.

El Diseño es un esfuerzo en equipo que involucra los siguientes elementos:

- Administradores de datos.
- Usuarios.
- Analistas de la empresa.
- Diseñadores de aplicación.
- Otros:
  - Capacidad de almacenamiento.
  - Performance.

El diseño lógico es un análisis y reglas que gobiernan los datos de la empresa. El diseño lógico está relacionado con la implementación física de la base de datos.

Para realizar un buen diseño lógico se recomienda llevar a cabo la siguiente secuencia:

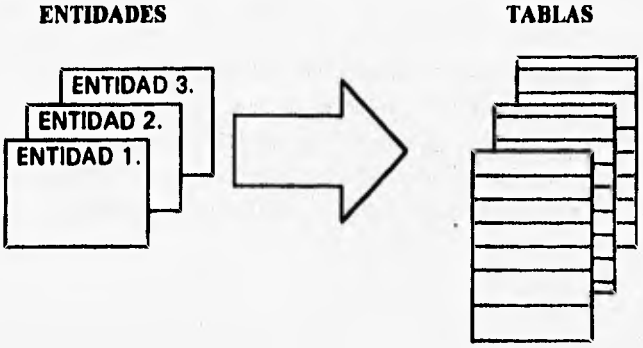
1. Decidir que datos registrar en la base de datos relacional.
   Antes de diseñar las tablas, deben identificarse las entidades y las relaciones. Una entidad es un objeto que es fundamental para la empresa. Generalmente la entidad es considerada a ser una persona, lugar, objetos físicos o eventos. Una relación indica como dos entidades se asocian una con la otra. Los atributos son piezas de información acerca de una entidad. En este paso el modelo entidad-relación nos ayuda a decidir que datos registrar en la base de datos.

MODELO ENTIDAD-RELACION

UN PREREQUISITO PARA EL DISEÑO DE BASE DE DATOS

2. **Definir las tablas para cada tipo de relación.**

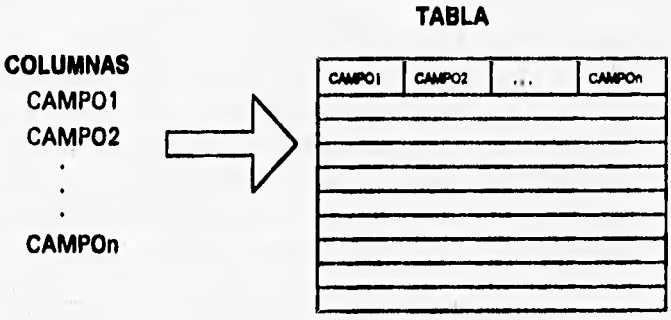
   Una vez que se ha obtenido el modelo entidad-relación se procede a definir el modelo relacional. En una base de datos relacional se pueden expresar varios tipos de relaciones, por lo que en este paso se obtienen las tablas para el tipo de relación que se maneje. Las tablas se obtiene directamente del conjunto de entidades, es decir se va a tener una tabla por cada conjunto de entidades. Cada tabla debe tener un nombre que debe ser único en la base de datos.



3. Proporcionar las definiciones de columnas para todas las tablas.
   En esta parte se va a tener una columna para cada elemento de dato necesitado.

   Definir una columna en una tabla DB2 consiste en:

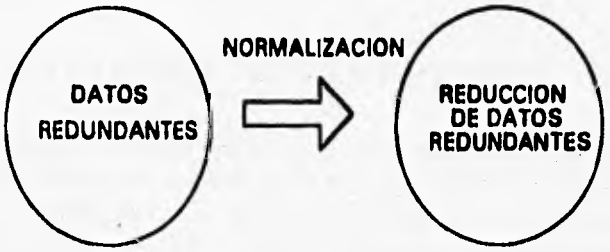
   - Seleccionar un nombre para la columna.
     Cada columna en una tabla debe tener un nombre que es único en la tabla.
   - Identificar el tipo de dato para la columna.
     A cada columna se le asigna un tipo específico y tamaño de dato.



4. Identificar una o más columnas como llaves primarias.
   Una tabla debe tener una columna o columnas que proporcionan un identificador único para los renglones de la tabla. Esta columna o conjunto de columnas es llamada llave primaria y no debe contener valores duplicados, es decir una o más columnas identifican una ocurrencia de la entidad específica.

5. Considerar la normalización de las tablas.

   La normallización es un proceso que permite la claridad de los datos de la empresa como sea posible. Para obtener la primera forma normal los grupos repetidos deben ser eliminados y todos los elementos deben ser atómicos. Esta forma normal es beneficiosa para determinar la llave correcta de una entidad. La segunda forma normal nos forza para evaluar todos los atributos definidos para una entidad. La segunda forma normal evita anomalías de update/delete (inconsistencia de datos). La tercera forma normal nos permite examinar los datos para eliminar otras inconsistencias. Un atributo no debe depender de otro atributo que no sea la llave primaria. Hay otras formas normales, pero en la práctica la mayoría de los administradores de datos paran en la tercera forma normal. Se debe hacer notar que esto no es una explicación rigurosa de la normalización.



6. Plan para el mantenimiento de la integridad referencial.

   La integridad referencial mantiene la validez de los datos aplicando las reglas de integridad durante el procesamiento, es decir, la integridad referencial es la aplicación de todas las restricciones o reglas de integridad. A través de DB2 es posible planear la aplicación de la integridad referencial aplicando las siguientes reglas:

   - Cada valor de llave primaria es única y no es nula.
   - Cada valor de llave foránea es igual a un valor de llave primaria.
   - Reglas de borrado:
     - Cascada. Cuando se borra un renglón de la tabla padre, todos los renglones relativos en la tabla dependiente también son borrados.
     - Restricción. No se pueden borrar todos los renglones de la tabla padre que tengan renglones dependientes.

- Set null. Cuando se borra un renglón, los valores correspondientes de la llave foránea en todos los valores correspondientes son puestos nulos.

INTEGRIDAD REFERENCIAL

La fase del diseño lógico requiere una participación activa por los Administradores de Bases de Datos (DBA's), ya que su buen diseño repercutirá en el diseño físico.

3.5 IMPLEMENTACION DEL DISEÑO EN DB2

El diseño físico es la integración del diseño lógico de las bases de datos dentro del SMBD DB2. Establece la distribución de las tablas e índices sobre los dispositivos de almacenamiento, también, determina como las bases de datos deberán ser manejadas durante el procesamiento de datos.

Después del diseño lógico se debe asignar el espacio físico para el almacenamiento de datos y la creación física de los objetos DB2, es decir, la etapa de la implementación física. Todos los objetos DB2 están almacenados en data sets (archivos) VSAM. Para la creación y manejo de estos data sets se utilizan las estructuras de almacenamiento llamadas storage groups DB2.

La herramienta auxiliar que se utiliza para crear e implementar las bases de datos relacionales es el lenguaje de definición de datos (DDL - Data Definition Language) que es parte del lenguaje SQL.

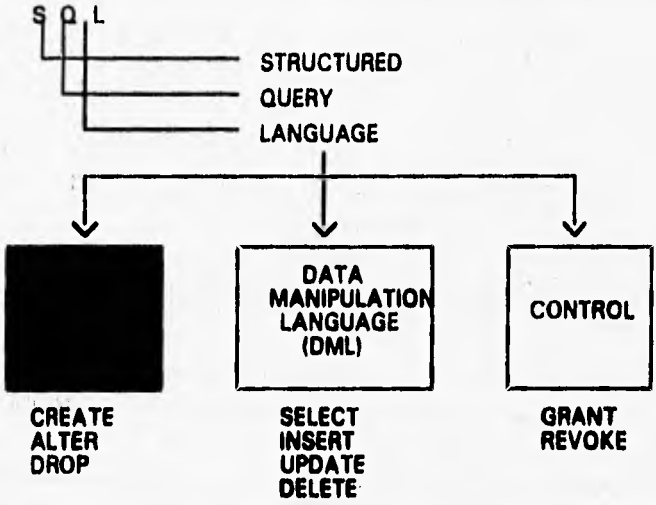


Figura 3.4. El DDL es un elemento del lenguaje SQL.

Las decisiones de diseño se realizan a través del DDL, es decir, utilizando las declaraciones CREATE, ALTER y DROP es posible definir, cambiar y remover los objetos:

- CREATE.
  Define un nuevo objeto.
- DROP.
  Remueve un objeto.
- ALTER.
  Cambia su descripción.

Nota: Para la manipulación de datos se usan las declaraciones SELECT, INSERT, UPDATE y DELETE y para el control de datos GRANT (para dar privilegios) y REVOKE (para revocarlos).

Todos los movimientos de objetos realizados con el DDL hacen que se actualice el Catálogo del DB2.

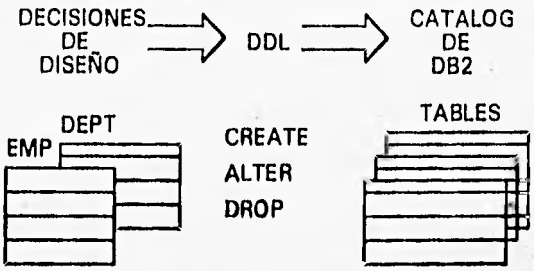


Figura 3.5. El DDL y el Catálogo del DB2

Cada una de las tablas que se crean con el DDL residen en un Table Space. Cada Table Space y su contenido pertenecen a una misma Database. Una Database es un agrupamiento de tablas y de los espacios que éstos ocupan.

Un Table Space se divide en bloques de 4k bytes llamados páginas. Una página es la unidad de Transmisión Input/Output. Una página puede contener uno o más renglones (máximo 127 especificado como límite en el manual de instalación). Para acomodar renglones más grandes de 4k, las páginas pueden ser de 32k.

Para establecer los nombres identificadores de los objetos físicos del DB2 se utilizan las siguientes reglas:

- Los nombres de la DATABASE y del STORAGE GROUP deben ser únicos dentro del sistema DB2. Y dentro de éstas los otros objetos indicados en la figura 4.4.3 sus nombres calificados deben ser únicos.
- El CREATOR es el dueño del objeto y esta identificado dentro del catálogo de DB2.
- Todos los nombres de los objetos deben de comenzar con un caracter alfabético seguido por cualquier caracter alfanumérico.

MAX 8 CARACTERES	MAX 18 CARACTERES
DATABASE	TABLE
TABLESPACE	VIEW
CREATOR	INDEX
STORAGE GROUP	COLUMN
	SYNONYM
	ALIAS

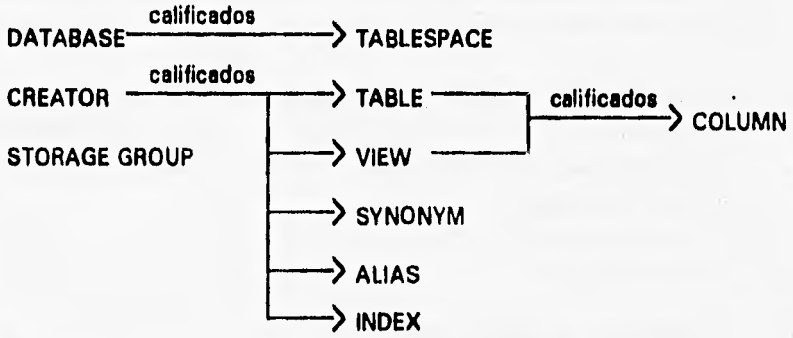


Figura 3.6. Diagrama de reglas para la asignación de nombres a los objetos del DB2.

Para crear los objetos DB2 se utiliza la declaración CREATE de SQL. Esta declaración tiene la siguiente forma:

CREATE {tipo de objeto} {nombre del objeto} {lista de opciones}

Creando Storage Groups DB2.

DB2 maneja los requerimientos de almacenamiento externo de un database DB2 usando storage groups.

DB2 realiza lo siguiente en el manejo de los requerimientos de almacenamiento externo:

- Cuando un table space es creado, DB2 define los data sets VSAM necesarios usando métodos de servicios de acceso VSAM.
- Cuando un table space es borrado, DB2 automáticamente borra los data sets asociados.

Después de que se crea un storage groups, DB2 almacena información referente a ésto en el catálogo DB2.

Para crear un storage group DB2 se debe utilizar la declaración CREATE STOGROUP de SQL. A esta declaración se le proporciona una lista de volúmenes que DB2 puede usar para definir los data sets físicos.

El máximo número de volúmenes por storage group es de 133 (Límite especificado en el manual de instalación). Todos los volúmenes de un storage group deben ser del mismo tipo de dispositivo.

Cuando DB2 busca la lista de volúmenes candidatos para el espacio disponible éstos son buscados en el orden en que los volúmenes fueron definidos o añadidos en el storage group.

EL VCAT es un usuario que está identificado en el catálogo VSAM ICF (Integrated Catalog Facility) el cual será el dueño de todos los datasets (archivos) que serán almacenados en este storage group. El nombre VCAT será el calificador de más alto nivel de tales datasets.

Se pueden retirar o añadir volúmenes del storage group usando la declaración ALTER STOGROUP.

Creando Databases DB2.

Cuando se define una Database, su nombre identificará una colección de tablas e índices asociados.

Para crear una Database se usa la declaración CREATE DATABASE de SQL.

```
CREATE DATABASE    DABA0
  STOGROUP         STGR0
  BUFFERPOOL       BP0;
```

La cláusula STOGROUP establece un storage group identificado como *STGR0* para la database identificada como *DABA0*.

La cláusula BUFFERPOOL asigna un buffer pool llamado *BP0* para las tablas e índices dentro de la database.

La declaración ALTER DATABASE permite alterar la definición lógica de la database, es decir se puede cambiar el nombre del storage group y el buffer pool.

Creando Table Spaces

Los table spaces son los espacios físicos que contienen tablas. Un table space puede tener una o más tablas. Cada table space tiene un rango de direccionamiento máximo de 64 gigabytes (especificación límite del manual de instalación) y puede estar compuesto por uno o más data sets VSAM. Los table spaces están divididos en unidades de igual tamaño llamadas páginas que pueden ser de 4k o 32k .

Existen 3 tipos de table spaces:

- Simple. Un table space que puede contener más de una tabla. El espacio esta compuesto de páginas y cada página puede contener renglones de muchas tablas.

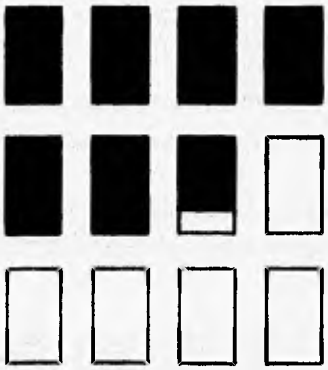


Figura 3.7. Table space simple con múltiples tablas.

- Segmentado. Puede contener más de una tabla. El espacio esta compuesto de grupos de páginas llamados segmentos. Cada segmento esta dedicado para alojar renglones de un sola tabla.

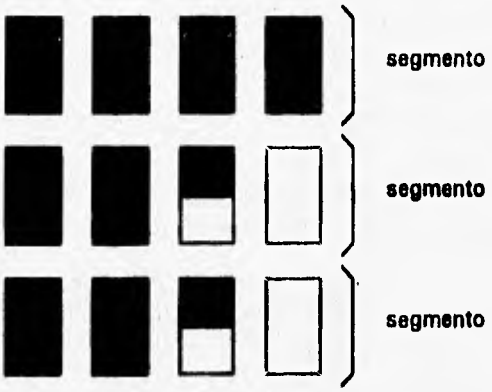


Figura 3.8. Table space segmentado con múltiples tablas.

- Particionado. Puede contener sólo una tabla. El espacio esta subdividido en particiones basadas en rangos de la llave del índice asociado, denominado índice de particionamiento.

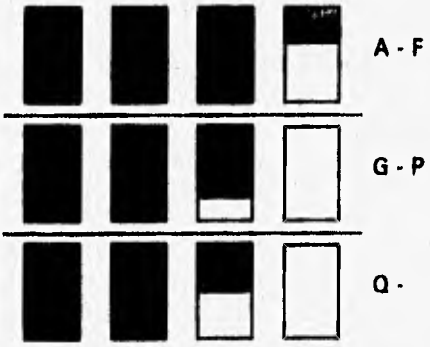


Figura 3.9. Table space particionado.

Creando un table space simple.

Se puede poner más de una tabla en un table space simple, la siguiente declaración crea el table space simple *TASP0* en la database *DABA0* y usa el storage group *STGR*:

```
CREATE TABLESPACE    TASP0
  IN                 DABA0
  USING STOGROUP     STGR0
    PRIQTY           20
    SECQTY           20
  LOCKSIZE           ANY
  BUFFERPOOL         BP0
  CLOSE              NO
  PCTFREE            10;
```

Las cláusulas de la declaración CREATE TABLESPACE son las siguientes:

La cláusula IN. Se le indica una database especifica. La database nombrada debe estar definida. Si no se da el nombre de una database, DB2 utiliza una database de default.

La cláusula USING. Se refiere al storage group que alojará al tablespace, si no incluye esta cláusula DB2 utiliza un storage group de default.

- Utilizar STOGROUP y el nombre de un storage group para crear el table space dentro sus voúmenes.
- PRIQTY y SECQTY. Estas cláusulas permiten especificar las asignaciones de espacio primario y secundario para el table space en kbytes.
- ERASE. Indica si los datasets definidos para DB2 serán borrados cuando el tablespace es liberado. Determina que acción se tomará cuando un table space de un storage group es borrado (dropped). Si se utiliza ERASE YES, los datos son sobreescritos con ceros, como medida de seguridad. Usando el ERASE NO borra los datos completamente cuando se borra el table space.

La cláusula BUFFERPOOL. La selección de un nombre de buffer pool determina el tamaño de la página. Si se selecciona los buffer pools BP0, BP1 o BP2 dan un tamaño de página de 4k, y BP32k da un tamaño de página de 32k.

La cláusula LOCKSIZE. Esta cláusula permite seleccionar el nivel y tamaño de bloqueo para DB2. Este bloqueo puede ser a nivel página, tabla o tablespace. Por default se asigna el LOCKSIZE ANY, este default provoca que las decisiones de bloqueo sean realizadas por DB2.

La cláusula CLOSE. Con esta cláusula se le indica a DB2 cerrar los data sets. CLOSE YES cierra los data sets cuando éstos no sean utilizados por mucho tiempo. CLOSE NO evita repetir las operaciones de abrir o cerrar data sets, es decir los mantiene abiertos durante su procesamiento.

La cláusula PCTFREE y FREEPAGE. El uso de PCTFREE coloca el porcentaje de cada página que estará reservado como espacio libre para ser utilizado durante las operaciones de carga y reorganización de datos (LOAD y REORG respectivamente). EL espacio puede ser utilizado más tarde insertando o actualizando. El valor de default es 5 (especificación de default en el manual de administración de DB2).

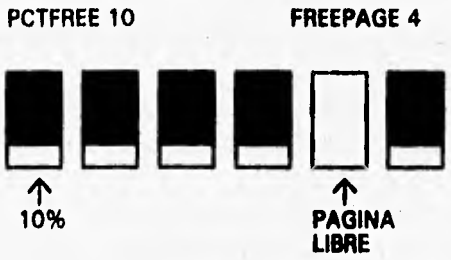


Figura 3.10. Espacio libre.

EL uso de FREEPAGE permite especificar el número de páginas que serán ocupadas seguida de una página con espacio libre que será utilizado durante las operaciones de carga y reorganización de datos (LOAD y REORG respectivamente). El espacio está disponible para insertar o actualizar en la misma manera que en el espacio libre en una página.

Creando un table space segmentado.

Un ejemplo de una declaración CREATE TABLESPACE, para un table space segmentado, es la siguiente:

```
CREATE TABLESPACE    TASPO
  IN                 DABAO
  USING STOGROUP     STGRO
    PRIQTY           160
    SECQTY           80
  SEGSIZE            4
  LOCKSIZE           TABLE
  BUFFERPOOL         BPO
  CLOSE              NO
```

La cláusula SEGSIZE. El valor dado es el número de páginas en cada segmento; éste debe ser un múltiple de 4, desde 4 hasta 64 (norma especificada en la guía de instalación). La selección del valor depende del conocimiento del tamaño de las tablas que serán almacenadas.

La cláusula LOCKSIZE. La opción TABLE es válido sólo para table spaces segmentados (norma especificada en manual guía de administración DB2). Esto significa que DB2 puede bloquear sólo una tabla, en vez del table space entero.

La cláusula FREEPAGE. El uso de esta cláusula ya se explicó anteriormente, sólo que al utilizarse en table spaces segmentados al menos debe de existir una página libre en cada segmento (norma especificada en la guía de instalación).

Creando un table space particionado.

Un ejemplo de CREATE TABLESPACE, para un table space particionado, es:

```
CREATE TABLESPACE    TASP0
   IN                DADA0
   USING STOGROUP    STGR0
      PRIQTY         12
      SECQTY         12
      ERASE          NO
   NUMPARTS          4
      (PART 3  USING STOGROUP    STGR1
         PRIQTY      12
         SECQTY      12)
   LOCKSIZE          PAGE
   BUFFERPOOL        BP1
   CLOSE             YES;
```

El ejemplo anterior crea un table space particionado, ésta incluye la cláusula NUMPARTS en la declaración CREATE TABLESPACE.

Un table space particionado puede tener sólo una tabla , y hasta 64 particiones (especificado en la guía de administración del DB2). El tamaño máximo de cualquier table space es de 64 gigabytes, pero el tamaño máximo de una partición es de 4 gigabytes. (normas especificadas en la guía de administración del DB2).

La declaración del ejemplo anterior crea un table space con 4 particiones. Este también ilustra como las particiones pueden ser almacenadas sobre diferentes tipos de dispositivos; la partición 3 utiliza el storage group *STGR1* y los otros tres usan el *STGR0*.

La declaración CREATE TABLESPACE nos dice como el table space está dividido en un conjunto de particiones. La declaración del ejemplo crea un table space, *TASP0*, que más tarde contendrá la tabla a utilizar. Cuando se defina el índice de particionamiento para la tabla, es aquí donde se declara que registros empleados estarán almacenados en cada partición.

TASP0 es un table space particionado y puede contener sólo una tabla.

La cláusula USING. Establece el storage group y las asignaciones de espacio para todas las particiones.

La cláusula NUMPARTS. Especifica el número de particiones.

La cláusula PART. Especifica la partición y la definición de espacio físico para ésta.

Creando Tablas.

Para crear una tabla que se ha diseñado, se utiliza la declaración CREATE TABLE. Cuando se crea una tabla, DB2 registra una definición de la tabla en el catálogo DB2.

La siguiente declaración muestra un ejemplo para crear una tabla de empleados:

```
CREATE TABLE  EMPLEADO
   ( EMPNO      CHAR(6)      NOT NULL,
     FIRSTNME   CHAR(12)     NOT NULL,
     MIDINIT    CHAR(1)      NOT NULL,
     LASTNAME   CHAR(15)             ,
     WORKDEPT   CHAR(3)              ,
     PHONENO    CHAR(4)              ,
     HIREDATE   DATE                 ,
     JOB        CHAR(8)              ,
     EDLEVEL    SMALLINT             ,
     SEX        CHAR(1)              ,
     BIRTHDATE  DATE                 ,
     SALARY     DECIMAL(9,2)         ,
     BONUS      DECIMAL(9,2)         ,
     COMM       DECIMAL(9,2)         ,
     PRIMARY KEY (EMPNO)
     FOREIGN KEY (WORKDEPT) REFERENCES   DEPTO
          ON DELETE SET NULL          )
IN   DABAO.TASPO;
```

La cláusula PRIMARY KEY. La llave primaria de una tabla identifica cada una de las ocurrencias de una entidad. Esta cláusula nombra la columna o columnas de la llave primaria. En el ejemplo anterior el número de empleados es la llave primaria de la tabla de empleados y la cláusula PRIMARY KEY identifica la columna de los números de empleados *(EMPNO)*.

Si se nombra una llave primaria, se deberá también definir un único índice (el índice primario), sobre el mismo conjunto de columnas. El valor de la llave primaria debe entonces ser único y no puede ser nulo.

La cláusula FOREIGN KEY. La llave foránea de una tabla es una llave que está especificada en la definición de una restricción referencial. La llave foránea debe tener el mismo número de columnas, con las mismas descripciones, como la llave primaria de la tabla padre.

La cláusula IN. Se utiliza para nombrar la database o la database y el table space para ser utilizada por la tabla.

Creando Indices.

Los índices proveen acceso eficiente a los datos. Una tabla puede tener más de un índice y una llave índice es una columna o una colección ordenada de columnas sobre las cuales un índice está definido.

La declaración CREATE INDEX se utiliza para crear un índice. Se recomienda crear todos los índices sobre una tabla antes de cargar la tabla para optimizar el performance de acceso a los datos.

DB2 permite duplicar valores en una columna que es llave. Si no se quiere duplicar valores se debe utilizar la declaración CREATE UNIQUE INDEX.

El siguiente ejemplo crea un índice con una llave compuesta, conteniendo las columnas *PROJNO, ACTNO* y *ACSTDATE.* Las llaves ASC y DESC en el ejemplo indican el orden ascendente y descendente. ASC es el default.

```
CREATE UNIQUE INDEX     OWNER.INDICE0
    ON                  OWNER.TABLA0
       (PROJNO ASC,
       ACTNO ASC,
       ACSTDATE DESC)
    USING STOGROUP      STGR0
       PRIQTY       12
       ERASE        NO
    SUBPAGES        8
    BUFFERPOOL      BP0
    CLOSE           NO
```

Para especificar un agrupamiento de índices se utiliza la cláusula CLUSTER (sólo para el caso de table space particionado) en la declaración del CREATE INDEX.

Cuando se almacena una tabla en un table space particionado, se debe decidir como dividir los datos entre las particiones utilizando la cláusula PART de una declaración CREATE INDEX. El índice que divide los datos es llamado el índice particionado.

El siguiente ejemplo muestra como crear un índice particionado sobre la columna del número de departamento *(EMPNO)* de la tabla de empleados. El nombre del índice es *INDICE1*. Las entradas del índice están en orden ascendente (el default).

El ejemplo también ilustra como los valores del índice están asignados por partes, utilizando la cláusula VALUES. Cada valor introducido por VALUES es el más alto valor de la llave índice (o el más bajo para una columna utilizando el orden descendente) asignado en la partición especificada. En el ejemplo, *099999* es el valor más alto asignado en la partición 1. Sin embargo la partición 1 tiene el rango de valores *00000* a *099999*. El valor para la última partición, en este caso es *999999*.

```
CREATE UNIQUE INDEX     OWNER.INDICE1
   ON                   OWNER.EMPLEADOS
      (EMPNO ASC)
   USING STOGROUP       STGR0
      PRIQTY        12
      ERASE         NO
   SUBPAGES         8
   CLUSTER
      (PART         1 VALUES('099999'),
      PART          2 VALUES('199999'),
      PART          3 VALUES('299999'),
      PART          4 VALUES('999999'))
   BUFFERPOOL       BP0
   CLOSE            NO
```

En el ejemplo anterior, cada partición utiliza el storage group *STGR0*.

El tamaño de la página de un índice es 4k. Las páginas, sin embargo, pueden ser subdividas en 1, 2, 4, 8, o 16 subpáginas; el default es 4.

La subpágina es una unidad de bloqueo. El propósito de tener más de una subpágina es para incrementar la concurrencia para las aplicaciones que actualizan, en vez de las que insertan. Las aplicaciones que insertan frecuentemente debe reducir el número de subpáginas para la concurrencia.

Las cláusulas PCTFREE y FREEPAGE, así como las cláusulas BUFFERPOOL y CLOSE son esencialmente las mismas como las usadas con el CREATE TABLESPACE. Para los índices, sin embargo, no se puede utilizar el buffer pool de 32k (norma establecida en la guía de instalación).

Después de usada una base de datos relacional es posible realizar cambios en algunos aspectos del diseño. Esto es, para alterar la definición de un objeto DB2 se puede utilizar la declaración ALTER de SQL o borrando el objeto y recreándolo con diferentes especificaciones.

Con la declaración ALTER de SQL se pueden modificar la definición lógica de storage groups, databases, table spaces, tablas e índices. ALTER cambia la manera en que éstos objetos están definidos en el catálogo de DB2, pero éste no realiza todos los cambios; por ejemplo, no se puede borrar una columna de una tabla con ALTER.

Cuando no se pueda hacer cambios con ALTER, se debe realizar lo siguiente:

1. Usar la declaración DROP para remover el objeto.
2. Usar la declaración CREATE para recrear el objeto.

La declaración DROP tiene un efecto en cascada; los objetos dependientes de los objetos borrados serán también borrados. Antes de borrar un objeto, se debe verificar el catálogo de DB2 para determinar el impacto de la operación.

Alterando Storage Groups.

Se puede utilizar la declaración ALTER STOGROUP para añadir o remover volúmenes de un storage group. Si quiere emigrar a otro tipo de dispositivo, se necesita mover los datos.

Se pueden utilizar hasta 133 volúmenes en un storage group (limite especificada en la guía de administración de DB2). Todos los volúmenes deben ser del mismo tipo; y cuando un storage group es utilizado para extender un data set, los volúmenes deberán tener el mismo tipo de dispositivo así como los volúmenes utilizados cuando el data set fue definido.

Los cambios que se realicen a la lista de volúmenes por ALTER STOGROUP no tienen efecto sobre el storage existente. Los cambios toman efecto cuando los nuevos objetos son definidos o cuando las utilerías de reorganización, recuperación o carga de datos (REORG, RECOVER, o LOAD respectivamente) son utilizadas sobre estos objetos. Por ejemplo, si se utiliza el ALTER STOGROUP para remover el volumen *VOL01* del storage group *SG01*, los datos de DB2 sobre este volúmen permanecen intactos. Sin embargo, cuando un nuevo table space es definido utilizando a *SG01*, el volumen *VOL01* no está disponible para la asignación del espacio.

Alterando Databases.

La declaración ALTER DATABASE permite modificar todas las cláusulas utilizadas para crear una database:

- STOGROUP. Permite cambiar el nombre del storage group de default para soportar requerimientos de espacio físico DASD para table spaces e índices dentro de la database. El nuevo storage group es sólo utilizado por nuevos table spaces e índices; las definiciones existentes no cambian.
- BUFFERPOOL. Permite cambiar el nombre del buffer pool de default para los table spaces e índices dentro de la database. Esto sólo se aplica para nuevos table spaces e índices; las definiciones existentes no cambian.

Aterando Table Spaces.

La declaración ALTER TABLESPACE permite cambiar las siguientes cláusulas:

- BUFFERPOOL. Permite nombrar el bufferpool para ser asociado con el table space. El tamaño de la página debe ser el mismo.
- LOCKSIZE. Permite especificar el nivel de bloqueo para el table space. Para un table space simple o particionado, se pueden utilizar las opciones PAGE, TABLESPACE, o ANY. Para un table space segmentado, se puede también usar la opción TABLE; no se puede cambiar el tamaño del segmento.
- CLOSE. Permite cerrar los data sets soportados cuando no haya usuarios activos del table space.
- PART. Permite identificar una partición del table space.
- FREEPAGE. Permite especificar la frecuencia de una página de espacio libre cuando el table space es cargado o reorganizado.
- PCTFREE. Permite especificar el porcentaje de espacio libre para cada página cuando un table space es cargado o reorganizado.
- USING. Se puede cambiar de un storage group a otro cambiando el valor de STOGROUP. Los cambios no tienen efecto inmediato sobre los data sets existentes; los nuevos data sets son utilizados cuando se usen las utilerías reorganización, recuperación y carga de datos (REORG, RECOVER, o LOAD) para el table space.
- PRIQTY. Permite especificar la asignación de espacio primario para un data set del table space o partición.
- SECQTY. Permite especificar la asignación de espacio secundario para un data set del table space o partición.

- ERASE. Permite especificar que los contenidos de un data set para el table space o partición serán borrados cuando el table space es borrada.

Se pueden cambiar las opciones para PRIQTY, SECQTY, y ERASE en el mismo storage group o en el nuevo. Se puede cambiar los atributos de almacenamiento para una partición del table space usando la cláusula PART.

Alterando Tablas.

Con la declaración ALTER TABLE se puede cambiar una tabla.

Con ALTER TABLE se puede hacer lo siguiente:

- Añadir una nueva columna.
- Añadir o borrar una llave primaria o foránea.

Cuando se realiza algún cambio en la definición lógica de la tabla, el catálogo de DB2 es actualizado.

Cuando se utiliza el ALTER TABLE para añadir una columna, la nueva columna debe ser la columna de más a la derecha. Los registros físicos no son actualmente cambiados hasta que los valores son insertados en la nueva columna.

Algunos cambios para una tabla no pueden ser realizadas con la declaración ALTER TABLE. Cuando se requiere cambiar el tipo de dato de una columna para hacer sus campos más grandes, por ejemplo, una especificación original de CHAR(20) convertirlo en CHAR(25) o una columna teniendo un tipo de dato convertirlo a otro tipo.

Para realizar tales cambios, se necesita borrar la tabla y redefinirla.

Alterando Indices.

La declaración ALTER INDEX permite cambiar los siguientes atributos de índice:

- BUFFERPOOL. Permite nombrar el buffer pool para ser asociado con el índice.
- CLOSE. Permite especificar si los data sets para el índice serán cerrados cuando el número de procesos utilizando el índice llega a ser cero.
- PART. Permite identificar una partición del índice.
- FREEPAGE. Permite especificar la página libre cuando el índice o partición es cargada o reorganizada.
- PCTFREE. Permite especificar la cantidad de espacio libre en cada página cuando el índice o partición es cargada o reorganizada.

- USING. Así como los table spaces, se puede cambiar de un storage group a otro cambiando el valor del STOGROUP.
- PRIQTY. Permite especificar la asignación de espacio primario.
- SECQTY. Permite especificar la asignación de espacio secundario.
- ERASE. Permite especificar si el contenido de un data set para el índice o partición será borrado cuando el índice es borrado.

Para borrar cualquier otra cláusula de la definición de índice, se debe borrar el índice y redefinirlo. Borrando un índice no provoca que DB2 borre cualquier otro objeto.

3.6 DECISIONES DE DISEÑO

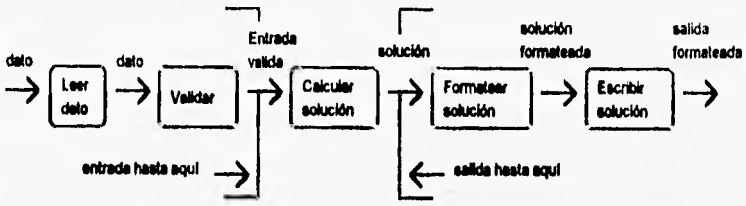
Existen decisiones de diseño que afectan tanto al diseño lógico como al físico y que tienen sus efectos en el rendimiento. El considerar tales decisiones nos conduce a reducir problemas como:

- Periodicidad poco confiable de respaldos, reorganizaciones y depuraciones.
- Elevado costo de tiempo en la consulta.
- Abortaciones de programas por espacio físico.
- Errores humanos en el cálculo de parámetros.

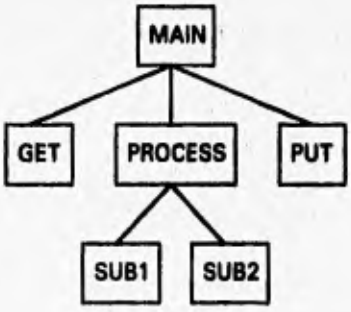
Por lo tanto analizaremos todas aquellas decisiones que ayuden a mejorar el rendimiento y a reducir la problemática presentada.

Cuando realizamos el diseño físico de una base de datos para obtener un alto performance y un uso adecuado de almacenamiento, es necesario la utilización de herramientas tendientes a auxiliar las decisiones de diseño:

- Diagrama de flujo de datos. Proporciona al administrador de base de datos una idea de los procesos que afectan las entidades. Esta herramienta puede ayudarnos a determinar el tipo de table space (segmentado, simple o particionado), parámetros de espacio libre (pctfree y freepage), la utilización de índices y otros parámetros de diseño.



- Procesamiento de flujo de datos. Esta herramienta proporciona información acerca de los procesos importantes y el número de ellos que se ejecutarán dentro de un día. Esta información es necesaria pra desarrollar un diseño físico con un alto performance y que sea flexible para futuras necesidades.



- Pseudo SQL. Nos auxilia para generar un prototipo de las instrucciones que implementarán el diseño físico. Este Pseudo SQL no tiene que ser correcto sintácticamente.

```
SELECT PACKAGE, DATE
FROM PACKAGE
WHERE PACKAGE = ?
ORDER BY DATE
```

El objeto de la fase del diseño físico es producir un modelo físico basado en las características de uso y criterios de performance. La nueva información requerida de los usuarios para completar esta fase es primero, un estimado del volumen y uso de las datos (ocurrencias de las entidades y frecuencias de las transacciones), y segundo,

prioridades de performance (tiempo de respuesta o prioridades de ejecución de los procesos).

Para comenzar con un análisis de uso, el primer dato que se requiere es el número de ocurrencias de cada entidad. Esto es importante para poder estimar requerimientos de almacenamiento, pero es de mayor importancia para determinar la proporción de procesamiento, por ejemplo, si nosotros tenemos 200 departamentos y 2000 empleados, conoceremos que una transacción que regresa una lista de empleados para un departamento dado accesará un promedio de 10 empleados.

Las especificaciones para cada una de las transacciones y los programas en batch deben ser analizadas para determinar la frecuencia de acceso de cada dato de la entidad y el argumento de búsqueda utilizado para cada acceso. Esto puede ser sumarizado para cada uno de los programas y así obtener el número de accesos requeridos por ejecución de programa y para cada uno de los datos de la entidad obtener la frecuencia de acceso para cada argumento de búsqueda. Esta información de frecuencia es importante en la obtención de un óptimo performance.

La anterior información es útil para poder llevar a cabo las siguientes decisiones de diseño:

- Uso de Nulls.
- Desnormalización.
- Indices.
- Tipo de table space (Simple, Segmentado y Particionado).
- Porcentaje libre (freespace).

Uso de Nulls.

DB2 utiliza un indicador de valor especial, llamado el valor nulo (null), que utiliza para un valor desconocido o un valor perdido. Un valor nulo es un valor que no es un cero, un blanco o una cadena vacia. Este es una valor especial interpretado por DB2 que implica que un dato no ha sido suministrado.

Los valores nulos pueden ser utilizados cuando los valores actuales son desconocidos o inaplicables; por ejemplo cuando un valor debe ser fijado en la columna de costo de una tabla de información de un producto para un registro cuyo costo aun no ha sido puesto, un valor de cero es inexacto porque el producto seguramente tiene un costo. Un valor nulo resuelve el problema porque mostraría que el registro contiene un costo pero este es desconocido.

Una búsqueda que es realizada en una tabla que contiene columnas nulas hace que el tiempo de ejecución sea más rápido debido a que los nulos no son tomados en cuenta en la búsqueda. Sin embargo, los nulos no satisfacen cualquier declaración de SQL.

Es necesario hacer notar que se debe de realizar una consideración de NOT NULL (no nulo) para columnas en que se requiere crear un índice único. Una columna que es considerada llave primaria no debe de contener nulos.

Desnormalización.

La normalización proporciona un método de diseño que auxilia a los desarrolladores entender y depurar las tablas que ellos diseñan. Las reglas de normalización producen diseños que ofrecen un eficiente performance y minimizan las inconsistencias en los datos. En algunos casos, sin embargo, las ventajas del performance pueden ser obtenidas violando la normalización, a esta técnica se le llama desnormalización, que deberá ser practicada con gran cuidado.

La desnormalización reintroduce la posibilidad de actualizar y borrar anomalías, pero los daños son pocos si la base de datos es utilizada cuidadosamente.

La decisión de llevar a cabo la desnormalización depende de las siguientes consideraciones:

- La redundancia de datos es estable.
- La cantidad de redundancia de datos es pequeña.
- Los datos deberán estar en primera forma normal.

NORMALIZACION	DESNORMALIZACION
Máxima flexibilidad	Mínimas relaciones
Renglón demasiado corto	Renglón demasiado largo
Recuperación de I/O de muchos renglones	Recuperación de I/O de muchas columnas
Bloqueos a muchos renglones	Bloqueos a muchas columnas

Figura 3.11. Efectos de la normalización y desnormalización.

Indices.

Un índice lista valores de una o más columnas de una tabla. DB2 deberá encontrar los datos más eficientemente.

Un índice estándar puede referenciar valores duplicados; en cambio un índice único no puede tenerlos. Los índices únicos son requeridos para columnas que son llaves primarias, que por definición deben ser únicas, porque los índices únicos no permiten valores duplicados. Los índices únicos y estándares pueden agrupar renglones, manteniendo la colocación física de los renglones en orden secuencial. DB2 sólo permite un índice por index space. Cuando es creado un índice, emitiendo la declaración CREATE INDEX antes de cargar los datos proporciona ventajas de performance significantes.

El costo de una aplicación dada con un número dado de accesos a la base de datos es relativo para el costo de cada uno de estos accesos. El costo de un acceso a la base de datos es principalmente debido a:

- El número de páginas de datos e índices recorridas por DB2.
- El número de entradas de índices y renglones recorridas por DB2.
- La cantidad de renglones sorteados por DB2.
- El número de I/O's.

Un índice puede mejorar la ruta de acceso en diferentes maneras tal como:

- Reduciendo el número de renglones y páginas recorridos.
- Eliminando un sorteo, si la solicitud de ordenamiento corresponde al índice.

En estos casos, el índice puede mejorar el performance.

Cuando se utiliza la integridad referencial DB2, antes de iniciar con el diseño de índices, debe considerarse lo siguiente:

- Un índice debe ser definido sobre cada llave primaria.
- Un índice debe ser definido sobre cada llave foránea.

Un índice puede mejorar el performance dando un acceso más eficiente. En particular, el uso de índices debe ser evaluado cuidadosamente en los siguientes casos:

- Pocos datos. Cuando se utilizan pocos datos se deben tener pocos índices o ninguno. Este es el caso cuando los datos son añadidos o borrados el mismo día o en pocos días. En este caso, la eficiencia en unas cuantas transacciones no sobrecarga las inserción y borrado.
- Valores duplicados. Los índices con un gran número de valores duplicados tienen un alto costo cuando se borran entradas. Considérese el caso donde uno o pocos valores tienen un gran número de duplicados. En este caso se consideran también los valores nulos, ceros o blancos en una columna. Para borrar una entrada con índice, DB2 rastrea sobre un promedio medio las entradas de índice.

La siguiente figura muestra un ejemplo del impacto del número de índices definidos cuando se realizan operaciones sobre una tabla específica. El costo de CPU está expresado en términos relativos. Para este ejemplo, el costo de CPU de una inserción es más del doble cuando 4 índices son definidos en lugar de uno. Los índices utilizados aquí no tienen cadenas de sinónimos muy grandes, en este caso, un mayor impacto debe ser expectado.

Número de Indices Definidos	Costo de CPU Relativo cuando se hace un INSERT
0	2
1	3
2	5
4	8
8	14

Como se puede observar los índices necesitan procesamiento de recursos. Por ejemplo, cada vez que un renglón es insertado o borrado una correspondiente operación debe ser realizada sobre cada una de sus tablas de índices. Finalmente, los índices requieren de espacio en disco. Sin embargo, los beneficios obtenidos dependen de la manera en que los datos serán utilizados y al tamaño de la tabla.

Analizando lo antes dicho habrá casos en los que no conviene indexar, debido a que se consumirían más recursos si se utilizaran índices. Por lo que en los siguientes casos se debe evitar el uso o menor uso de índices:

- Evitar indexado de columnas:
  - Que tengan valores redundantes.
  - Que sean frecuentemente actualizadas.
  - Que sean mayores que 30 bytes.
  - Que provoquen problemas de contención de bloqueo.

- Evitar indexado de tablas:
  - Que tengan menos de 7 páginas al menos que se utilicen frecuentemente en relaciones o en verificación de la integridad referencial.

Tipo de Table Space (Simple, Segmentado y Particionado).

Si se crea una sola tabla, se puede seleccionar cualquiera de los tipos de table spaces. Si se crean múltiples tablas en table space, se puede seleccionar un table space simple o segmentado. La selección deberá depender de los requerimientos de manejo de datos en particular. La siguiente discución de los tres tipos de table spaces auxilia la selección de uno de ellos.

Table Spaces simples. Una ventaja de table spaces simples es que se tiene el control sobre el orden en que los renglones están almacenados. El orden en que los renglones están almacenados es el orden en que se cargan los renglones en la tabla. Si la tabla no está sujeta a demasiadas inserciones o borrados este tipo de table spaces puede mejorar la localidad de referencia en la manera en que se cargan los datos.

Sin embargo los table spaces simples tiene muchas desventajas, como las siguientes:

- Toma más tiempo buscar por todos los renglones de una tabla cuando las tablas están entremezcladas en un table space. Buscando en una tabla requiere buscar en todo el table space.
- Cuando una tabla es borrada, el espacio ocupado por la tabla no está disponible para reusarse inmediatamente.
- El table space completo no está disponible cuando ciertas utilerías DB2 están en ejecución.
- Si se bloquea una tabla, las demás tablas del table space también se bloquearán.

- Si se bloquea una página de un table space simple puede significar que la información de otras tablas esté bloqueada también.

Tables spaces segmentados. Se usa un table space segmentado para almacenar más de una tabla. Las páginas del espacio están organizadas en segmentos y cada segmento aloja registros de una sola tabla. Cada segmento contiene el mismo número de páginas, que pueden ser un múltiplo de 4 (desde 4 a 64). Cada tabla usa tantos segmentos como ésta necesite y un space map registra los asignamientos de segmentos para las tablas.

Cuando los registros son insertados por INSERT o LOAD los segmentos de una misma tabla no se encuentran juntos. Al reorganizar el table space coloca los segmentos de una misma tabla juntos.

Algunas de las ventajas de los table spaces segmentados son las siguientes:

- Cuando se busca en una tabla, sólo los segmentos asignados a esta tabla son accesados. Las páginas de segmentos vacíos son ignorados.
- Cuando se bloquea una tabla, el bloqueo no interfiere con el acceso a segmentos de otras tablas.
- Cuando una tabla es borrada, sus segmentos vuelven a estar disponibles para reusarse inmediatamente; no es necesario esperar cuando se usa la utilería REORG.
- Un borrado en masa de todos los renglones de una tabla opera mucho más rápido y produce menos información de log.
- Incluso si el table space contiene una sóla tabla, segmentándolo significa que el COPY no deberá copiar páginas que estén vacías a causa de un drop de una tabla o un borrado masivo.

Una de las principales ventajas para un table space segmentado es la asignación mínima de almacenamiento por tabla. Por esta razón, los table spaces segmentados son preferibles a table spaces simples.

Table spaces particionados. Se usa un table space particionado para almacenar una sola tabla. Cada una de las particiones contiene una parte de la tabla. Se recomienda usar un table space particionado para tablas grandes (10 millones de renglones o más).

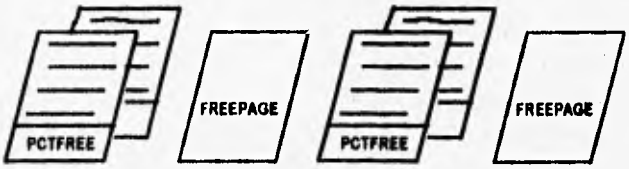
Un table space particionado necesita un cluster index porque los renglones son asignados a las particiones de acuerdo a los valores de sus índices.

Se debe considerar el uso de una tabla particionada en las siguientes circunstancias:

- Si se quiere extender una tabla grande sobre varios storage groups. Todas las particiones de la tabla no tienen que usar el mismo storage group.
- Si se necesita trabajar sobre una parte de los datos a la vez. Los table spaces particionados permiten a las utilerías trabajar sobre una parte de los datos a la vez.

Espacio libre (Free space).

Se pueden utilizar dos parámetros para indicar la cantidad y distribución de espacio libre en las páginas dentro de los datasets de un tablespace o indexspace cuando los datos son cargados o reorganizados. Estos dos páramentros vistos anteriormente en la implementación del diseño pueden ser especificados en la declaración CREATE (PCTFREE y FREEPAGE). PCTFREE especifica el porcentaje de espacio libre en cada página. FREESPACE indica el número de páginas para ser cargadas entre cada página libre. Ambos deben ser utilizados en la misma declaración. Los valores de 20 y 2 en los parámetros de PCTFREE y FREEPAGE respectivamente deberán producir una carga de datos semejante a esta:



El espacio libre proporciona un sitio para la inserción de nuevos renglones con una secuencia o para la expansión de renglones. La página es la unidad básica de almacenamiento físico. Los parámetros de espacio libre pueden ser utilizados para un table space entero o un index space o para una partición. La selección realizada en ambos parámetros afecta a las operaciones de I/O, la utilización de CPU y la frecuencia con que los datos deberán ser reorganizados.

3.7 DESCRIPCION DE LAS UTILERIAS DE DB2

Las utilerías (utility's) consisten en un conjunto de programas o procedimientos integrados dentro del esquema de recuperación (recovery) de DB2. También las utilerías son parte del conjunto de mecanismos para auxiliar y controlar el manejo de los datos en las estructuras de almacenamiento de DB2. Su principal función de las utilerías es el ahorrarnos trabajo en ciertas tareas, tales como la administración de datos e implantación de esquemas de recuperación.

La importancia de analizar las utilerías de una base datos estriba en que son herramientas que permiten dar mantenimiento y prevenir escenarios para el caso de contigencia. Problemas tales como periodicidad en los respaldos y reorganización de los datos se minimizarán si se analiza el uso y efecto de los parámetros de las herramientas de una manera adecuada.

DB2 ofrece un número de utilerías para el análisis y administración del almacenamiento físico de datos.

Las utilerías de DB2 tienen las siguientes características:

- Trabajan sobre table spaces.
- Corren conectados a DB2.
- Requieren autorización de DB2.
- Corren en un ambiente batch MVS.

Las siguientes utilerías son las más importantes y usadas en la administración de las bases de datos:

- COPY. Crea copias de un table space o un conjunto de datos con un table space. Hay dos tipos de copias:
  -. Un full image copy es una copia de todas las páginas de un table space o conjunto de datos.
  -. Un incremental image copy es una copia sólo de páginas que han sido modificadas desde la última vez que se usó la utilería COPY.

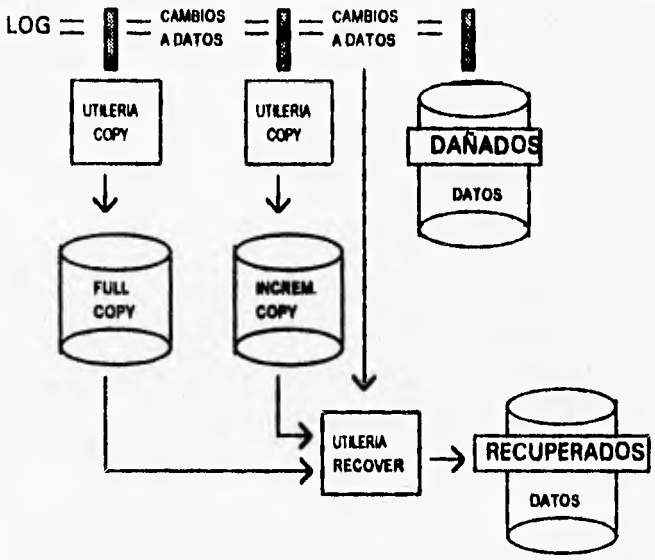


Figura 3.12. Uso de la utilería COPY.

- LOAD. Carga datos dentro de una o más tablas en un table space, remplaza el contenido de una sola partición, o remplaza o añade el contenido de un table space entera. Su función es la operación de mover datos de un data set secuencial dentro de un table space.

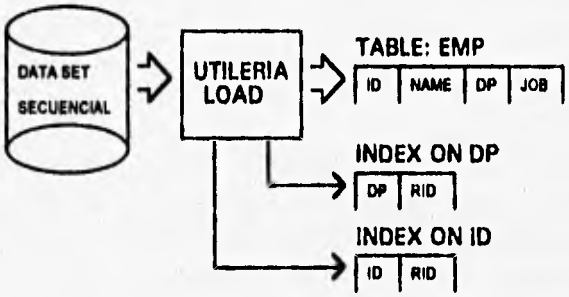


Figura 3.13. Utilería LOAD.

- RECOVER. Recupera datos para el estado actual. Se puede recuperar con una copia o un log específico. La unidad más grande de recuperación de datos es el table space; la más pequeña es la página. El dato es recuperado de copias de un table space y de logs de database. RECOVER también recupera múltiples índices sobre tablas que residen en un table space común.

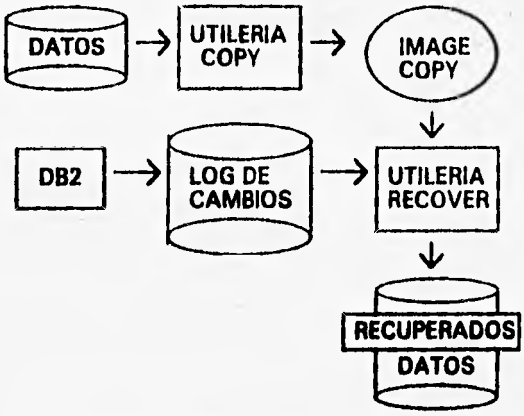


Figura 3.14. Uso de la utilería RECOVER

- REORG. Reorganiza un table space, la tarea la realiza descargando los renglones desde el tablespace a un archivo temporal en el orden de su índice primario también llamado cluster (que rige el orden de las llaves o campos para el ordenamiento de los renglones en la carga o descarga de datos), enseguida realiza un ordenamiento (fase de sort) para posteriormente realizar la carga de datos ordenados en la secuencia marcada por el índice primario (cluster) y por último reconstruye los índices asociados (fase de build) al tablespace de acuerdo al nuevo ordenamiento de los datos. En adición, la utilería puede reorganizar una sola partición ya sea de un índice particionado o un table space particionado.

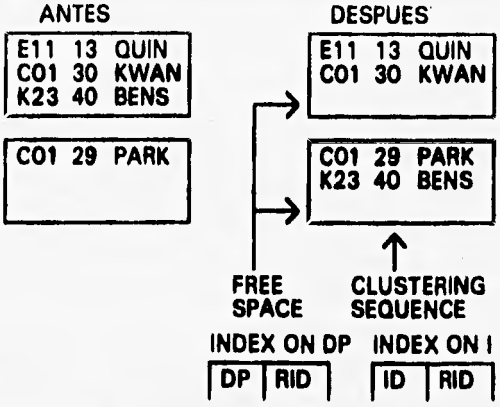


Figura 3.15. Resultados de la utilería REORG.

- QUIESCE. Establece un punto global de consistencia (RBA-Relative Byte address) en el cual un conjunto de table space está dentro de un estado consistente. El propósito primario de la utilería QUIESCE es permitir recuperar las estructuras de integridad referencial lo cual garantiza la consistencia de los datos, la estructura referencial del table space incluido en la utilería es reconstruida.

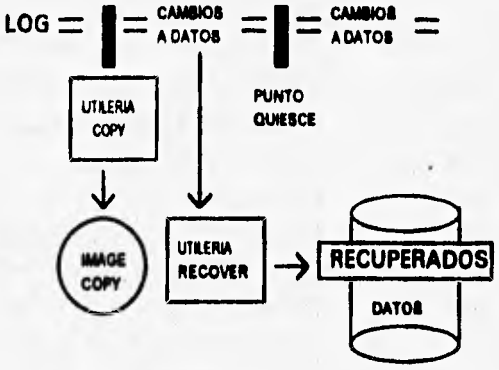


IMAGE COPY = MAS RECIENTE FULL COPY
+ COPIAS INCREMENTALES SUBSECUENTES

PUNTO QUIESCE = TIEMPO CUANDO LAS MODIFICACIONES ESTAN COMPLETADAS Y LOS DATOS ESTAN CONSISTENTES

Figura 3.16. Utilería QUIESCE.

Las utilerías usan un conjunto de archivos (data sets) para realizar su trabajo. La función de los data sets es almacenar los datos de salida de las distintas fases de cada utilería y salvar datos que pudiesen ser desechados durante la ejecución de la utilería para su análisis posterior (por ejemplo en la utilería load se desechan datos repetidos de entrada). Los data sets son referenciados como parámetros de entrada o llamados dentro del procedimiento de la utilería, sus atributos de definición lógica son estándar y serán automáticos dentro del DB2DA, por otro lado la definición espacio físico dependerá del volumen de datos a procesar.

La siguiente tabla muestra la lista de data sets, sus nombres estándares que necesita, la utilería que los requiere y por último el propósito y función de cada uno:

SORTOUT	LOAD, REORG	Toma las llaves sorteadas (salida ordenada) para permitir la face de SORT
SORTWKnn	LOAD,REORG, RECOVER	Trabaja el data set para sorteado de índices, nn es un número de 2 dígitos; se pueden usar varios data sets.
SYSCOPY	COPY	Salida del data set para copias.
SYSDISC	LOAD	Descarta registros
SYSERR	LOAD	Contiene información acerca de los errores encontrados durante el procesamiento.
SYSIN	LOAD, REORG, COPY	Entrada del data set para las declaraciones de utilería
SYSMAP	LOAD	Contiene información acerca de donde los registros de entrada del data set son cargados
SYSPRINT	LOAD,COPY, RECOVER, REORG	Salida de mensajes e impresiones.
SYSREC	LOAD, REORG	Entrada del data set para LOAD
SYSUT	LOAD,REORG, RECOVER	Temporalmente trabaja el data set que toma las llaves sorteadas para la entrada de la fase de SORT.

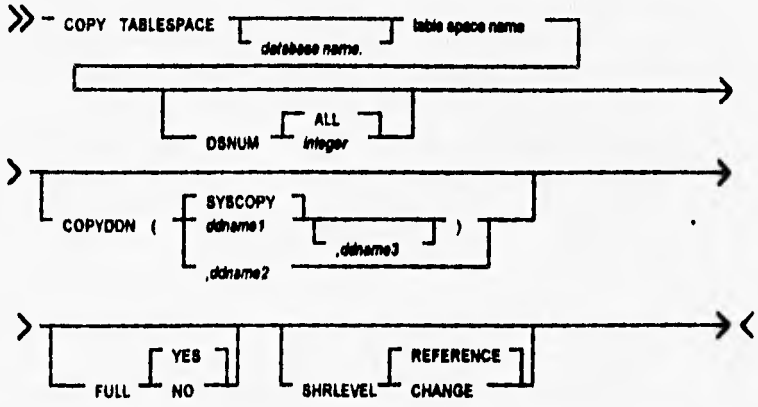
Se pueden emplear distintos métodos para ejecutar cada una de las utilerías, su facilidad de uso es de acuerdo a la optimización de sus parámetros de entrada en el ambiente de ejecución. Existen cuatro metódos para la ejecución de utilerías:

1. A través del panel interactivo de utilerías que ofrece DB2, donde se genera el codigo del procedimiento y su ejecución
2. Usando la facilidad del comando procesador DSNU (en sistema operativo MVS) para generar el código del procedimiento y ejecutarlo como un proceso batch.
3. Corriendo el procedimento (JCL) de la utilería proporcionándolo y conectándolo con DB2 dentro del procedimiento DSNUPROC (facilidad de administración de DB2), el cual usa los parámetros que se proveen para construir las setencias de ejecución para invocar la utilería de DB2.
4. Editando el procedimiento usted mismo (a través del DSNPROC) y optimizando los parámetros de entrada para ejecutarlo como un proceso batch.

Para el ambiente de bases de datos del DB2DA se utilizará el metódo No. 4, es decir, generará código como procedimientos codificados en JCL, el programa que utiliza DB2 es el DSNUTILB (módulo de carga de DB2) para la ejecución de las utilerías proporcionándole los parámetros adecuados.

Se describirá y ejemplicará la implementación de las utilerías que utilizará el DB2DA (Image Copy, Reorg y Quiesce) en la parte de definición de utilerías y frecuencia.

Implementación de la Utilería COPY.



Descripción de los parámetros y llaves:

- TABLESPACE. Especifica el table space (y , opcionalmente, la data base a la que pertenece) que será copiado.
  - *database name*. Es el nombre de la database a la que el table space pertenece.
  - *table space name*. Es el nombre del table space para ser copiado.
- DSNUM. Identifica una partición o data set, con el table space, esto es para ser copiado, o copia el table space entero.
  - ALL. Copia el table space entero.
  - integer. Es el número de una partición o data set para ser copiado, para un table space particionado, el entero es su número de partición.
- COPYDDN *ddname1*. Especifica el nombre de la tarjeta de referencia o declaración del data set en el procedimiento de la utilería el cual indica el nombre del data set del copiado primario o respaldo (backup) para el image copy.
  - *ddnamex*. Es el nombre de la tarjeta de referencia o definición del data set en el procedimiento de la utilería.
    El default es SYSCOPY para la copia primaria.
    No se pueden tener image copy de data sets duplicados.
- FULL. Realiza ya sea un incremental image copy o un full image copy.

- YES. Realiza un full image copy. Creando un full image copy da reset al status de copia pendiente para el table space.
  El default es YES.
- NO. Realiza sólo un full image copy. Sólamente cambia desde el último image copy que se realizó.

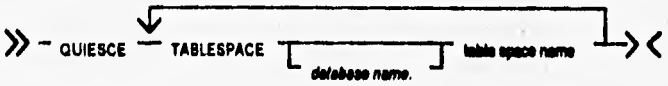
- SHRLEVEL. Indica si otros programas que accesan el table space deberán usar un acceso de sólo lectura mientras COPY este ejecutándose.
  - REFERENCE. Asigna el acceso de sólo lectura para otros programas.
  - CHANGE. Permite a otros programas o procesos modificar datos en el table space.

Ejemplo de un procedimiento (JCL) que realiza la utilería image copy de dos tablespaces:

```
//UUUUUU JOB (ABD,01),'UUUUUU',REGION=2048K,          TARJETA DE
//            CLASS=W,MSGCLASS=L,MSGLEVEL=(1,1),      CONTROL DEL
//            NOTIFY=UUUUUU                           USUARIO UUUUUU
//*
*
//*                    ==>  IMAGE  COPY    BACKUP  FOR  TABLESPACE
*
//*****-------------------------------------------------------------
*****
//*
//CDFPU020 EXEC PGM=DSNUTILB,                 EJECUCION DEL PROGRAMA
//         REGION=4M,                         DSNUTILB DE DB2 PARA LA
//         PARM='DB2D,UTILID'                 EJECUCION DE LA UTILERIA
//*
//*********************************************
//* BIBLIOTECA DE CARGA DE DB2
//*********************************************
//STEPLIB   DD DSN=DSILNT.PPDB2D.DSNLOAD,DISP=SHR
//*
//SYSPRINT  DD SYSOUT=*            OPCION DE IMPRESION A PANTALLA
//UTPRINT   DD SYSOUT=*            OPCION DE SALIDA A PANTALLA
//SYSUDUMP  DD DUMMY               OPCION DE UN DUMP O VACIADO DE
//*                                DE MENSAJES EN CODIGO MAQUINA
//*                                        EN CASO DE ABORTAR EL
PROCEDIMIENTO
//*
//*************************************************************
//* TARJETAS DE REFERENCIA DE DEFINICION LOGICA Y FISICA  DE
//* DATA SETS PARA EL RESPALDO (BACKUP) DE TABLESPACES
//*************************************************************
//COPY0 DD    DSN=SYS1.MODELO,DISP=SHR,
//            DCB=(SYS1.MODELO,LRECL=4096,BLKSIZE=16384,RECFM=FB)
//COPY1 DD    VOL=(,RETAIN,,),
```

```
//             DISP=(NEW,CATLG),
//             UNIT=TAPE80,
//             DSN=DSIFNB.CDDPB.JUL30.EAA001,
//             DCB=*.COPY0,
//             LABEL=(1,SL,EXPDT=99000)
//COPY2 DD     VOL=(,RETAIN,,REF=*.COPY1),
//             DISP=(NEW,CATLG),
//             UNIT=TAPE80,
//             DSN=DSIFNB.CDDPB.JUL30.EAA002,
//             DCB=*.COPY0,
//             LABEL=(2,SL,EXPDT=99000)
//*************************************************************
//* ENTRADA DE PARAMETROS DE ENTRADA DE LA UTILERIA DEL
//* PROCEDIMIENTO INDICE COPY   DONDE:
//*  BAABP001 ES LA DATABASE
//*  EAA001  ES EL TABLESPACE
//*************************************************************
//SYSIN     DD *
  COPY TABLESPACE  BAABP001.EAA001  DSNUM ALL  COPYDDN COPY1
   FULL YES  SHRLEVEL REFERENCE
  COPY TABLESPACE  BAABP001.EAA002  COPYDDN COPY2
   FULL YES  SHRLEVEL REFERENCE
//*************          FIN DEL PROCEDIMENTO      **************
```

Implementación de la Utilería QUIESCE.



Descripción de los parámetros y llaves:

- TABLESPACE *database name.table space name*. Especifica el table space, para establecer su punto de consistencia.
  - *database name*. El nombre de database a la que el table space pertenece.
  - *table space name*. El nombre de table space para establecer su punto de consistencia.

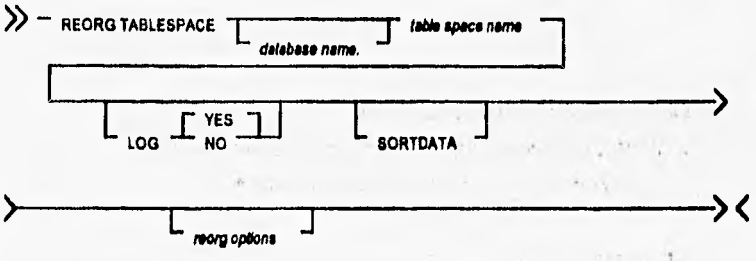
Se puede crear una lista de table spaces para establecer sus puntos de consistencia como un grupo, para esto hay que repetir la llave TABLESPACE.

Ejemplo de un procedimento de la utilería QUIESCE de un tablespace:

```
//UUUUUUUU JOB (AB,01,DB2),'UUUUUUU',REGION=4096K,        TARJETA DE
//             CLASS=W,MSGCLASS=L,MSGLEVEL=(1,1),         CONTROL DEL
//             NOTIFY=UUUUUUU                             USUARIO UUUUUUU
//*
//** ------------------------------------------------------------------ **
//**       PROCEDIMIENTO QUE REALIZA UN PUNTO GLOBAL DE                 **
//**         CONSISTENCIA A UN TABLESPACE DB2                           **
//*****-------------------------------------------------------------*****
//*
//CDDPU021 EXEC PGM=DSNUTILB,REGION=4M,     EJECUCION DEL PROGRAMA
//             PARM='DB2D,CDD1MVA3,'        DSNTILB DB2 PARA LA
//*                                         EJECUCION DE LA UTILERIA
//*
//*************************************************
//* LIBRERIA O BIBLIOTECA DE CARGA DE DB2
//*************************************************
//STEPLIB   DD DSN=DSILNT.PPDB2D.DSNLOAD,DISP=SHR
//*
//SYSPRINT  DD SYSOUT=*              OPCION DE IMPRESION A PANTALLA
//UTPRINT   DD SYSOUT=*              OPCION DE SALIDA A PANTALLA
//SYSUDUMP  DD DUMMY                 OPCION DE UN DUMP O VACIADO
//*                                  DE MENSAJES EN CODIGO MAQUINA
//*                                  EN CASO DE ABORTAR EL PROCEDIMIENTO
//*********************************************************
//*  DEFINICION DE ESPACIO FISICO PARA MENSAJES DE SORT A LA PANTALLA
//*********************************************************
//SORTOUT   DD UNIT=TMPDA,SPACE=(CYL,(0,5))
//*
//*********************************************************
//*  DATA SET DE ENTRADA VACIO PARA EL CASO DE QUIESCE
//*********************************************************
//SYSREC    DD DUMMY
//*
//*********************************************************
```

```
//*  ENTRADA DE PARAMETROS PARA LA UTILERIA  QUIESCE
//*  SE ESTABLECERA UN PUNTO GLOBAL DE CONSISTENCIA PARA
//*  LOS TABLESPACE    EAA001 Y EAA002 DE LA DATABASE
//*  LLAMADA BAABP001
//*************************************************************
//SYSIN     DD *
   QUIESCE TABLESPACE BAABP001.EAA001
   QUIESCE TABLESPACE BAABP001.EAA002
//*****        FIN DE PROCEDIMIENTO QUIESCE    ************
```

Implementación de la Utilería REORG.



Descripción de los parámetros y llaves:

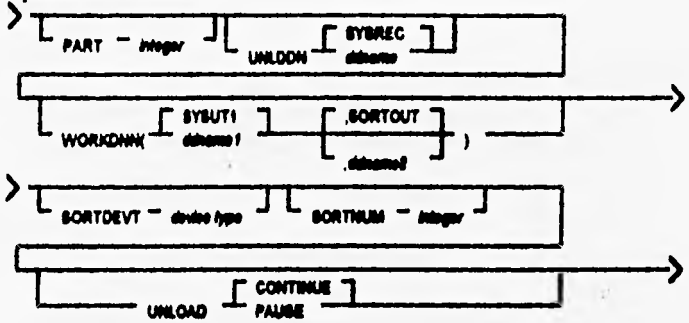
- TABLESPACE *database name.table space name*. Especifica el table space (y opcionalmente la database a la que pertenece) para ser reorganizado.
  Si se reorganiza un table space, sus índices soportados también son reorganizados.
  - *database name*. Es el nombre de la database a la que el table space pertenece.
  - *table space name*. Es el nombre del table space para ser reorganizado.
- LOG. Decide si el reacomodo de datos es registrado en el log (históricos) de DB2 durante la fase de recarga del REORG. Si el reacomodo de datos no fue registrado en el log, el table space es recuperable sólo después de que un image copy ha sido tomado.
  - YES. Registra el reacomodo de datos en el log, el default es LOG YES.
  - NO. No registra el reacomodo de datos en log y regresa el estatus de copia pendiente en el table espace.
- SORTDATA. Especifica que los datos serán descargados para el table space examinado, entonces sortea usando el índice.

Implementación de la utilería REORG INDEX.

Descripción de los parámetros y llaves:

- INDEX *index name*. Especifica un índice para ser reorganizado.
  - *index name*. Es el nombre calificado del índice.

Opciones del REORG:



Descripción de los parámetros y llaves:

- PART *integer*. Identifica una partición para ser organizada.
  - *integer*. Es el número de la partición.
- UNLDDN *ddname*. Especifica un data set donde serán descargados los datos.
  - *ddname*. Es el nombre de la tarjeta de referencia (DD definition data set) del data set de descarga en el procedimiento de la utilería.
- WORKDDN*(ddname1,ddname2)*. Para el REORG de un table space, *ddname* especifica la declaración (DD definition data set en el procedimiento) para un data set temporal utilizado para una salida intermedia. Para el REORG de un índice, ddname especifica la declaración (DD definition data set en el procedimiento) para el data set de descarga.
  - *ddname1*. Es el nombre (DD definition data set) del archivo de trabajo temporal para un ordenamiento (sort) de entrada. Un data set de trabajo para un ordenamiento (sort) de entrada es requerido para las tablas con índices. El default es SYSUT1.

- *ddname2*. Es el nombre (DD definition data set) del archivo de trabajo temporal para un ordenamiento (sort) de salida. El default es SORTOUT.

- SORTDEVT *device type*. Especifica el tipo de dispositivo para data sets temporales.
  - *device type*. Es el tipo de dispositivo.
- SORTNUM *integer*. Especifica el número de data sets temporales para ser dinámicamente asignados por el programa sort (ordenamiento).
  - *integer*. Es el número de data sets temporales.
- UNLOAD. Especifica si el job de la utilería debe continuar procesando o terminar después de que los datos han sido descargados. Los datos pueden ser recargados dentro de la misma tabla y table space.
  - CONTINUE. Especifica que, después de que los datos han sido descargados, la utilería continua procesando. El default es CONTINUE.
  - PAUSE. Especifica que después de que los datos han sido descargados, el procesamiento termina.

Ejemplo de un procedimento de la utilería REORG de un tablespace:

```
//UUUUUUUU JOB (0000,01),'UUUUUUU',MSGCLASS=L,      TARJETA DE CONTROL
//   NOTIFY=UUUUUUU,MSGLEVEL=(1,1),CLASS=W          PARA EL USUARIO
//*                                                 UUUUU PARA LA EJE-
//*                                                 CUCION DEL
//*                                                 PROCEDIMIENTO
//*******************************************************************
//*   PROCEDIMIENTO QUE REALIZA LA UTILERIA REORG A UN TABLESPACE
//*******************************************************************/
//*   EJECUCION DEL PROGRAMA DB2 DSNTILB SE LE INDICAN LOS PARAMETROS
//*   DE REGION DE MEMORIA, EL NOMBRE DEL SUBSISTEMA DB2 Y UN
//*   IDENTIFICADOR DE LA UTILERIA  (UTILID)
//*******************************************************************/
//PASO5    EXEC PGM=DSNUTILB,
//         REGION=4M,PARM='DB2D,UTILID,'
//*
//*******************************************************************/
//* DEFINICION O REFERENCIA DE LIBRERIAS DE CARGA DE DB2 Y MODULOS DE
//*   DE CARGA PARA LAS OPERACIONES DE SORT(ORDENAMIENTO) INTERNO
//*   DE LA UTILERIA REORG
//*******************************************************************/
//STEPLIB  DD DSN=DSILNT.PPDB2D.DSNLOAD,DISP=SHR
//         DD DSN=DSILNT.PPSORTLN.LOAD,DISP=SHR
//SORTLIB  DD DSN=DSILNT.PPSORTSU.LOAD,DISP=SHR
//*
//*******************************************************************/
//* DEFINICION O REFERENCIA DE OPCIONES DE SALIDA A IMPRESION Y
//* MENSAJES DE LA EJECUCION A PANTALLA, OPCION DE DUMP A ARCHIVO
//* VACIO EN EL CASO DE ABORTAR EL PROCEDIMIENTO.
//*******************************************************************/
//SYSPRINT DD SYSOUT=*
//UTPRINT  DD SYSOUT=*
//SYSUDUMP DD DUMMY
//*
//*******************************************************************/
```

```
//* DEFINICION Y REFERENCIA LOGICA Y FISICA DE LOS ARCHIVOS DE TRABAJO
//* DE LA UTILERIA REORG
//*  SYSUT1 : ALMACENA TEMPORALMENTE LAS LLAVES ORDENADAS PARA LA
//*           CONSTRUCCION DE LOS INDICES.
//*  SORTOUT : ALMACENA TEMPORALMENTE LOS DATOS DEL TABLESPACE PARA
//*            ORDENARLOS Y POSTERIORMENTE CARGARLOS AL TABLESPACE
//*  SYSREC : ALMACENA TEMPORALMENTE LA DESCARGA DE LOS DATOS DEL
//*           TABLESPACE
//*****************************************************************/
//SYSUT1   DD DSN=DSIFNB.CDDPB.SYSUT1,
//         DISP=(NEW,CATLG),
//         SPACE=(CYL,(400,30),,,ROUND),
//         UNIT=WRKDA
//SORTOUT  DD DSN=DSIFNB.CDDPB.SORTOUT,
//         DISP=(NEW,CATLG),
//         SPACE=(CYL,(500,30),,,ROUND),
//         UNIT=WRKDA
//SYSREC   DD DSN=DSIFNB.CDDPB.SYSREC,
//         DISP=(NEW,CATLG),
//         SPACE=(CYL,(500,30),,,ROUND),
//         UNIT=WRKDA
//*
//*****************************************************************/
//* ENTRADA DE PARAMETROS DE LA UTILERIA REORG:
//* SE REORGANIZARA EL TABLESPACE LLAMADO EAA001 QUE PERTENECE A LA
//* DATABASE BAABP001
//*****************************************************************
 REORG TABLESPACE BAABP001.EAA001  LOG NO
 SORTDATA  PART ALL UNLDDN SYSREC WORKDDN SYSUT1
 UNLOAD CONTINUE
/************  FIN DEL PROCEDIMIENTO REORG      **********************
```

3.8 RENDIMIENTO DE LAS UTILERIAS EN EL DISEÑO FISICO

Cuando es iniciada la planeación extensa de una Data Base, las expectaciones de performance deben ser definidas claramente. Estas deben ser relativas para un ambiente particular de software en este caso MVS y DB2, por lo tanto necesitamos incluir en la planeación del procesamiento de datos (tales como la ejecución de utilerías en la administración de bases de datos) los siguientes requerimientos y mediciones:

- Elapsed time (tiempo transcurrido en ejecutarse un programa batch) y tiempo de CPU.
- Volatibilidad de los datos en las tablas.
- Requerimientos de disponibilidad de datos.
- Requerimientos de concurrencia.

Una vez que las tablas e índices DB2 han sido implementados físicamente, este diseño puede ser validado a través de las principales aplicaciones, las cuales usarán los datos. Esta validación puede mostrarnos que cambios de diseño son requeridos al encontrarse con las espectaciones de performance.

Para la ejecución de procedimientos de utilerías las mediciones de tiempo de ejecución (elapsed time), tiempos de CPU, volatibilidad de las tablas (frecuencia de actualización a datos), nivel de concurrencia y disponibilidad de la información, son muy importantes para medir u optimizar su rendimento (performance).

El identificar los beneficios de las utilerías que definirá el DB2DA a través de la selección adecuada de sus parámetros ayudará al conjunto de mediciones anteriormente señaladas a obtener valores para alcanzar un mejor rendimiento (performance) en la administración de las bases de datos.

El sistema DB2DA recomendará la frecuencia de ejecución de utilerías para el mantenimiento de las Data Bases, particularmente discutiremos los beneficios y funciones de las utilerías Image Copy, Quiesce y Reorg, que son las necesarias para dar el respaldo y mantenimento de una database en DB2.

Funciones y beneficios de la utilería REORG.

La funciones de la utilería se divide en cuatro fases:

1. Descarga de datos o renglones desde el tablespace y ordenamiento de estos (UNLOAD y SORTDATA)
2. Precarga de datos al tablespace (RELOAD)
3. Ordenamiento de llaves para los índices asociados al tablespace (SORT)
4. La construcción de los índices (INDEXSPACE) asociados.

Los renglones serán descargados en la secuencia de índice primario (cluster) de la tabla, es decir, se ordenarán dentro de la secuencia del índice cluster además de descargar (unload) los datos (utilizando la opción SORTDATA).

Los beneficios de la utilería REORG son:

- Restauración de la secuencia física del índice primario (cluster).
- Recuperación de espacio retenido por renglones de tablas retiradas (DROP TABLE), es decir, se eliminan huecos con espacios libres entre datos.
- Restablecimiento del parámetro espacio libre a nivel página (free space), el cual indica que cada página de datos debe tener un porcentaje de espacio libre (free space), dato aplica para índices y tablespaces.
- Remueve los punteros overflow, es decir, son eliminados los desbordes en las paginas y reacomodados .
- Repara u optimiza notablemente los resultados del tiempo de obtención de páginas de datos para el índice.
- Un índice puede ser reorganizado individualmente, es decir, sin reorganizar el table space .

Funciones y beneficios de la utilería IMAGE COPY.

La utilería Image Copy forma parte del esquema de recuperación de una database en DB2. Si un tablespace o indexspace es perdido por medio de una falla de hardware (por ejemplo en el dispositivo de almacenamiento), quizás las utilerías Image Copy y Recovery permitan su restauración. La instalación y uso periodico de los procedimientos de la utilería Image Copy, para un tablespace o sólo las páginas que contengan modificaciones, auxiliarán en la recuperación de una database en caso de contingencia (Las formas del Image Copy son llamadas *Full image copy e incremental image copy*). La utilería Merge Copy permite combinar un full image copy y un incremental image copy para crear un nuevo image copy. La utilería Image Copy registra la operación que ha sido realizada dentro del catálogo de DB2 (tabla llamada SYSCOPY, registra todas la operaciones de las utilerías de DB2 realizadas con éxito). La utilería Recovery determina desde el catálogo que copias están disponibles y requeridas para la restauración de datos.

DB2 registra antes y despúes de un Image Copy toda la actividad de actualizaciones, inserciones y borrado en su log dataset (archivo histórico que registra toda la actividad de DB2), si una falla ocurre con la utilería Image Copy al ser requerida para una recuperación, como última opción se utiliza el log data set para la restauración del tablespace, la desventaja es que va requerir mayor tiempo que el image copy.

Las siguientes recomendaciones describen la manera de aumentar el performance y sintonizar la utilería IMAGE COPY para un mínimo overhead:

- Si una operación REORG es exitosa y la opción LOG fué colocado en NO, el status de copia pendiente es colocado para este table space, entonces es necesario realizar una full image copy. Una incremental image copy no se podrá realizar.
- Utilizar un Image copy primario y secundario son recomendables después de una operación de REORG especificado con LOG NO. Si el image copy primario no está disponible, la recuperación puede ser posible utilizando el image copy secundario.
- Tomar en cuenta las restricciones pendientes: No se puede copiar un table space que está en status de verificación o recuperación pendiente.
- Si más del 15% de páginas de datos están cambiadas realizar un full image copy.
- Al utilizar el parámetro SHRLEVEL REFERENCE garantizar que el image copy represente un punto de consistencia para el table space (quiesce).

- Si un table space está particionado, pueden copiarse varias o todas las particiones independientemente en separados jobs en forma paralela. Esto reduce considerablemente el tiempo que toma al realizarce un image copy.
- Los incrementals image copies se refieren a copias parciales realizadas, basadas de acuerdo a que datos han sido modificados desde la última copia. Este tipo de copia ahorra tiempo y espacio en el dispositivo de almacenamiento.

Funciones y beneficios de la Utilería QUIESCE.

La utilería QUIESCE es usada para tener una fotografía de los datos en el tiempo a través de una dirección llamada RBA (Relatived Byte Address) en el log dataset (archivo de actividad histórica de DB2).

Ejecutar la utilería QUIESCE exitósamente garantiza que nuestro table space tenga un punto de consistencia global de consistencia, es decir, que los datos del tablespace están íntegros y consistentes. Para lograr esta consistencia en el table space se deben considerar las siguientes observaciones:

- No se puede usar QUIESCE sobre un table space que esté en status de copia pendiente, verificación pendiente o recuperación pendiente.
- Se puede recuperar un table space, a través de su punto de consistencia establecido por QUIESCE, con la utilería RECOVER utilizando el log dataset de DB2 (archivo que registra la actividad histórica de DB2).

Las utilerías en las bases de datos son importantes apuntando hacia el rendimiento (performance) de los sistemas que utilizan tales estructuras (tablas e índices) para almacenar sus datos en dos caminos. Primero, las utilerías permiten un manejo eficiente del almacenamiento físico. Segundo, usarlas apropiadamente contribuye a un rendimiento óptimo de las bases de datos. Las utilerías requieren recursos (memoria principal, memoria secundaria) para ellas, sin embargo el uso cuidadoso de las utilerías puede salvar costos de procesamiento de datos y ahorrarnos trabajo.

4 DISEÑO DEL SISTEMA

4.1 Carta de estructura primaria del sistema.

4.2 Esquema de tablas para el sistema.

4.3 Diagrama de flujo de datos.

4.4 El catálogo (Diccionario de datos) de DB2 en el sistema.

4.4.1 Diagrama entidad-relación de la base de datos del sistema DB2DA.

4.5 Diseño de la estructura modular del sistema.

4.5.1 Inicialización de las tablas auxiliares.

4.5.2 Actualización de las tablas auxiliares.

4.5.3 Definición del diseño físico.

4.5.4 Definición de las utilerías y frecuencia.

4.5.5 Generación del JCL (Programa) para utilerías.

4.5.6 Consulta a tablas auxiliares.

4.5.7 Actualización de tablas auxiliares.

4.5.8 Depuración de tablas auxiliares.

4.6 Modularidad e interfaces del sistema.

4.7 Pseudocódigo del diseño.

4.8 Lenguaje de implementación (REXX).

4.1 CARTA DE ESTRUCTURA PRIMARIA DEL SISTEMA

El sistema DB2DA consistirá de 5 pasos a través de los cuales se producirá el Diseño Físico y la Definición de Utilerías. Además, contará con tres opciones adicionales para consultar, actualizar y depurar las Tablas (TABLAS AUXILIARES) que almacenan las Definiciones Generadas.

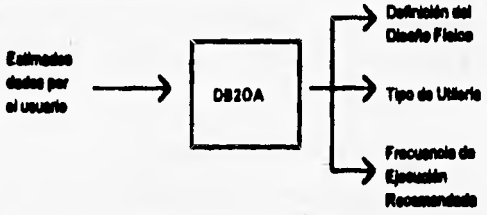


Diagrama que muestra al sistema DB2DA a nivel 0.

Diagrama que muestra la carta de estructura desarrollada:

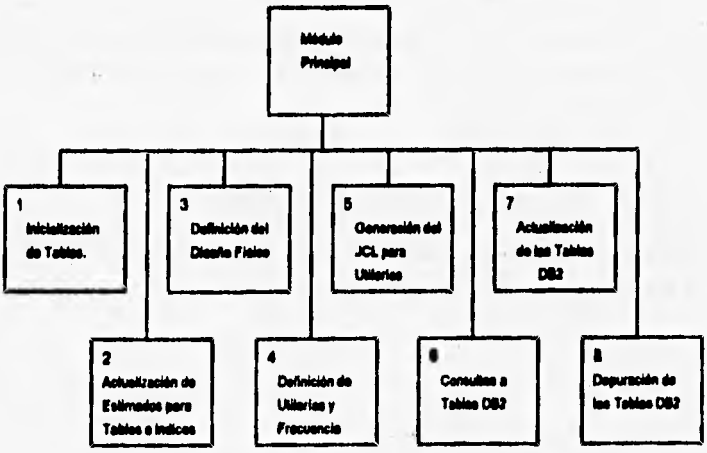


Diagrama que muestra la carta de estructura primaria.

Paso 1. Inicialización de Tablas.

En este paso se consultará el catálogo de DB2 para extraer nombres de Tablespaces e Indices pertenecientes al Database que indique el usuario. Estos datos serán insertados en las tablas DIS_TABLAS, DIS_INDICES y DIS_UTILERIAS, tablas Auxiliares del Sistema.

Paso 2. Actualización de Estimados para Tablas e Indices.

En esta etapa del proceso se presentarán al usuario cada una de las Tablas e Indices del Database elegido, para que introduzcan o modifiquen los estimados solicitados.

Paso 3. Definición del Diseño Físico.

Aquí se calcularán los parámetros del Diseño Físico para todas las Tablas e Indices del Database seleccionado por el usuario. Estos cálculos se realizarán a partir de los estimados proporcionados en el paso 2.

Paso 4. Definición de Utilerías y Frecuencias.

En este etapa se definirán que utilerías y con que Frecuencia ejecutarlas. La Definición se hará para todos los Tablespaces e Indices del Database que haya escogido el usuario. Al igual que en el paso 3, estas definiciones se hacen a partir de los estimados dados por el usuario.

Paso 5. Generación de JCL para Utilerías.

En este paso se producirá el JCL correspondiente a las utilerías definidas en el paso 4, para los Tablespaces e Indices del Database escogido.

Opciones Adicionales:

Opción C. Consultas a Tablas.

Esta opción permitirá consultar las definiciones generadas para el diseño físico o utilierías. Se empleará el table editor de QMF.

Opción A.- Actualización de las tablas.

Esta opción permitirá modificar las definiciones generadas para el diseño físico o utilierías. Se empleará el table editor de QMF.

Opción D. Depuración de tablas.

Esta opción permitirá depurar las definiciones generadas para el diseño físico o utilierías. Se empleará el table editor de QMF.

4.2 ESQUEMA DE TABLAS PARA EL SISTEMA.

El origen de las tablas auxiliares surge con la necesidad de documentar la información de entrada del DB2DA, el poder consultar y modificar dicha información para su procesamiento, así como registrar los datos de salida. De acuerdo al análisis de la problemática y a los requerimientos para su solución se diseñaron las tablas:

- DIS_TABLAS. Esta tabla guarda un renglón por cada tabla en un database. En la descripción de cada columna se indica si su valor es inicializado (I), estimado por el usuario (U) o calculado (C).
- UTILERIAS. Esta tabla guarda un renglón por cada utilería aplicada a una partición, tablespace o índice en un database. En cada columna se indica si su valor es inicializado (I), estimado por el usuario (U) o calculado (C).
- DIS_INDICES. Esta tabla guarda un renglón por cada índice en un database. En los comentarios de cada columna se indica si su valor es inicializado (I), estimado por el usuario (U) o calculado (C).
- DIS_UTILID. Esta tabla representa un catálogo de códigos de las utilerías.

Las cuales tienen la siguiente definición lógica:

La tabla **DIS_TABLAS** tiene la siguiente estructura:

Campo	Tipo	
DBNAME	CHAR(8)	NOT NULL WITH DEFAULT
TSNAME	CHAR(8)	NOT NULL WITH DEFAULT
TBCREATOR	CHAR(8)	NOT NULL
TBNAME	CHAR(18)	NOT NULL
VERSION	CHAR(3)	NOT NULL
CARD	INTEGER	NOT NULL WITH DEFAULT
ROW_LENOTH	SMALLINT	NOT NULL WITH DEFAULT
CREC_MENSUAL	CHAR(4)	NOT NULL WITH DEFAULT
VOLAT_DIARIA	CHAR(4)	NOT NULL WITH DEFAULT
FREC_LOAD_REP	CHAR(12)	NOT NULL WITH DEFAULT
FUENTE_LOAD_REP	CHAR(12)	NOT NULL WITH DEFAULT
DB2_RI	CHAR(2)	NOT NULL WITH DEFAULT
PER_TIPICO_ACT	CHAR(12)	NOT NULL WITH DEFAULT
ACT_MAS_COMUN	CHAR(10)	NOT NULL WITH DEFAULT
PER_TIPICO_DEP	CHAR(12)	NOT NULL WITH DEFAULT
INSERT_MASIVO	CHAR(2)	NOT NULL WITH DEFAULT
USO_LINEA	CHAR(2)	NOT NULL WITH DEFAULT
ACT_BATCH	CHAR(2)	NOT NULL WITH DEFAULT
ACT_LINEA	CHAR(2)	NOT NULL WITH DEFAULT
CONV_LIN_BAT	CHAR(2)	NOT NULL WITH DEFAULT
STO_GROUP	CHAR(8)	NOT NULL WITH DEFAULT
TIPO_TBSP	CHAR(4)	NOT NULL WITH DEFAULT
NPAGES	INTEGER	NOT NULL WITH DEFAULT
PARTICIONES	SMALLINT	NOT NULL WITH DEFAULT
PRI_QTY_TOT	INTEGER	NOT NULL WITH DEFAULT
PRI_QTY	INTEGER	NOT NULL WITH DEFAULT
SEC_QTY	INTEGER	NOT NULL WITH DEFAULT
PCT_FREE	DECIMAL(2,0)	NOT NULL WITH DEFAULT
FREE_PAGE	SMALLINT	NOT NULL WITH DEFAULT
CLOSERULE	CHAR(3)	NOT NULL WITH DEFAULT
LOCK_SIZE	CHAR(4)	NOT NULL WITH DEFAULT
SEQ_SIZE	SMALLINT	NOT NULL WITH DEFAULT
IDXMAX	SMALLINT	NOT NULL WITH DEFAULT
FECHA	DATE	NOT NULL WITH DEFAULT
USERID	CHAR(8)	NOT NULL WITH DEFAULT

Descripción de cada uno de los campos de la tabla **DIS_TABLAS**:

Campo	Descripción
DBNAME	Nombre del database al que pertenece la tabla (I)
TSNAME	Nombre del tablespace donde se encuentra almacenada la tabla (I)
TBCREATOR	Propietario de la tabla (I)
TBNAME	Nombre de la tabla (I)
VERSION	Versión del diseño físico (U)
CARD	Cantidad de renglones en la tabla (U)
ROW_LENGTH	Longitud física del renglón de la tabla (U)
CREC_MENSUAL	Crecimiento mensual esperado (alto, bajo) (U)
VOLAT_DIARIA	Volatilidad diaria (alta, baja) (U)
FREC_LOAD_REP	Frecuencia de uso de LOAD REPLACE LOG NO (U)
FUENTE_LOAD_REP	Indica si se conserva el archivo fuente del LOAD REPLACE (U)
DB2_RI	Indica si se emplea integridad referencial de DB2 (si,no) (U)
PER_TIPICO_ACT	Periodo típico de actualización (diario, semanal, etc.) (U)
ACT_MAS_COMUN	Actualización más común (INSERT, DELETE, UPDATE) (U)
PER_TIPICO_DEP	Periodo típico de depuración (diario, semanal, etc.) (U)
INSERT_MASIVO	Indica si se emplea INSERT masivo (si, no) (U)
USO_LINEA	Indica si se emplea la tabla en línea (si, no) (U)
ACT_BATCH	Indica si se hacen actualizaciones en batch (si, no) (U)
ACT_LINEA	Indica si se hacen actualizaciones en línea (si, no) (U)
CONV_LIN_BAT	Indica si conviven el servicio de línea con el batch (si, no) (U)
STO_GROUP	Storage group en el que se almacenará el tablespace (U)
TIPO_TBSP	Tipo de tablespace (segmentado, particionado, etc.) (C)
NPAGES	Páginas calculadas para la tabla (C)
PARTICIONES	Particiones calculadas para la tabla (0 si no es particionado) (C)
PRI_QTY_TOT	Primary quantity total considerando todas las particiones de la tabla será igual al valor de la columna PRI_QTY si el tablespace es particionado (C)
PRI_QTY	Primary quantity calculada para una partición del tablespace (C)
SEC_QTY	Secondary quantity calculada para una partición del tablespace (C)
PCT_FREE	Percent free calculado para el tablespace (C)
FREE_PAGE	Free pages calculadas para el tablespace (C)
CLOSERULE	Definición del parámetro CLOSE para el tablespace (C)
LOCK_SIZE	Definición del parámetro LOCKSIZE para el tablespace (C)
SEG_SIZE	Definición del parámetro SEGSIZE para el tablespace (C)
IDXMAX	Máximo número de índices recomendados para esta tabla (C)

La llave primaria para la tabla **DIS_TABLAS** está compuesta por los siguientes campos:

- TBCREATOR
- TBNAME
- VERSION

La tabla **UTILERIAS** tiene la siguiente estructura:

Campo	Tipo	
DBNAME	CHAR(8)	NOT NULL WITH DEFAULT
VERSION	CHAR(3)	NOT NULL
CALIFICADOR	CHAR(8)	NOT NULL
NOMBRE	CHAR(18)	NOT NULL
PARTICION	SMALLINT	NOT NULL
TIPO_OBJ	CHAR(4)	NOT NULL WITH DEFAULT
UTILID	CHAR(3)	NOT NULL
FRECUENCIA	CHAR(12)	NOT NULL WITH DEFAULT
GRUPO	SMALLINT	NOT NULL WITH DEFAULT
GDG_ENTRIES	SMALLINT	NOT NULL WITH DEFAULT
SHRLEVEL	CHAR(8)	NOT NULL WITH DEFAULT
SECUENCIA	SMALLINT	NOT NULL WITH DEFAULT
PAG_TOT	INTEGER	NOT NULL WITH DEFAULT
PAG_PART	INTEGER	NOT NULL WITH DEFAULT
FECHA	DATE	NOT NULL WITH DEFAULT
USERID	CHAR(8)	NOT NULL WITH DEFAULT

Descripción de cada uno de los campos de la tabla **UTILERIAS**:

Campo	Descripción
DBNAME	Nombre del database al que pertenece el tablespace o índice (I)
VERSION	Versión del diseño de utilerías (U)
CALIFICADOR	Propietario del tablespace o índice (I)
NOMBRE	Nombre del tablespace o índice (I)
PARTICION	Número de la partición de un tablespace o índice (0 si no es particionado) (I)
TIPO_OBJ	Tipo de objeto (tablespace, índice)
UTILID	Utilería a aplicar (REORG, RUNSTATS, FULL IMAGE COPY,) (C)
FRECUENCIA	Frecuencia de uso de la utilería (diario, semanal, etc.) (C)
GRUPO	Grupo al que pertenece un conjunto de utilerías del mismo periodo se usa para poder dividir jobs de un mismo periodo. Se inicializa en 1 y el usuario puede modificar su valor para formar grupos
GDG_ENTRIES	Número de entradas en un GDG para respaldos (C)
SHRLEVEL	Define el valor del parámetro SHRLEVEL (C)
PAG_TOT	Páginas totales del tablespace o índice (C)
PAG_PART	Páginas totales de una partición del tablespace o índice (C)

La llave primaria para la tabla **UTILERIAS** está compuesta por los siguientes campos:

- DBNAME
- VERSION
- CALIFICADOR
- NOMBRE
- PARTICION
- UTILID
- FRECUENCIA

La tabla **DIS_INDICES** tiene la siguiente estructura:

Campo	Tipo	
DBNAME	CHAR(8)	NOT NULL WITH DEFAULT
IXCREATOR	CHAR(8)	NOT NULL
IXNAME	CHAR(18)	NOT NULL
VERSION	CHAR(3)	NOT NULL
TBNAME	CHAR(18)	NOT NULL WITH DEFAULT
LONGITUD	SMALLINT	NOT NULL WITH DEFAULT
STO_GROUP	CHAR(8)	NOT NULL WITH DEFAULT
CLUSTERED	CHAR(2)	NOT NULL WITH DEFAULT
UNQ	CHAR(2)	NOT NULL WITH DEFAULT
PARTICIONADO	CHAR(2)	NOT NULL WITH DEFAULT
FULLKEYCARD	INTEGER	NOT NULL WITH DEFAULT
NLEAF	INTEGER	NOT NULL WITH DEFAULT
NLEVELS	SMALLINT	NOT NULL WITH DEFAULT
PRI_QTY_TOT	INTEGER	NOT NULL WITH DEFAULT
PRI_QTY	INTEGER	NOT NULL WITH DEFAULT
SEC_QTY	INTEGER	NOT NULL WITH DEFAULT
PCT_FREE	DECIMAL(2,0)	NOT NULL WITH DEFAULT
FREE_PAGE	SMALLINT	NOT NULL WITH DEFAULT
CLOSERULE	CHAR(3)	NOT NULL WITH DEFAULT
SUB_PAGES	SMALLINT	NOT NULL WITH DEFAULT
FECHA	DATE	NOT NULL WITH DEFAULT
USERID	CHAR(8)	NOT NULL WITH DEFAULT

Descripción de cada uno de los campos de la tabla **DIS_INDICES**:

Campo	Descripción
DBNAME	Nombre del database al que pertenece el índice (I)
IXCREATOR	Propietario de la índice (I)
IXNAME	Nombre de la índice (I)
VERSION	Versión del diseño físico (U)
TBNAME	Nombre de la tabla en la que se basa el índice (I)
LONGITUD	Longitud física de la llave del índice (U)
STO_GROUP	Storage group en el que se almacenará el índice (U)
CLUSTERED	Indica si el índice es CLUSTER (si, no) (U)
UNQ	Indica si el índice es UNIQUE (si, no) (U)
PARTICIONADO	Indica si el índice es particionado (si, no) (U)
FULLKEYCARD	Número de valores distintos en la llave del índice (U)
NLEAF	LEAF PAGES calculadas para el índice (C)
NLEVELS	Niveles calculados para el índice (C)
PRI_QTY_TOT	Primary quantity total considerando todas las particiones del índice. Será igual al valor de la columna PRI_QTY si el índice no es particionado (C)
PRI_QTY	Primary quantity calculada para una partición del índice (C)
SEC_QTY	Secondary quantity calculada para una partición del índice (C)
PCT_FREE	Percent free calculado para el índice (C)
FREE_PAGE	Free pages calculadas para el índice (C)
CLOSERULE	Definición del parámetro CLOSE para el índice (C)
SUB_PAGES	Cantidad de subpages calculadas para el índice (C)

La llave primaria para la tabla **DIS_INDICES** está compuesta por los siguientes campos:

- **IXCREATOR**
- **IXNAME**
- **VERSION**

La tabla **DIS_UTILID** tiene la siguiente estructura:

Campo	Tipo	
UTILID	CHAR(3)	NOT NULL WITH DEFAULT

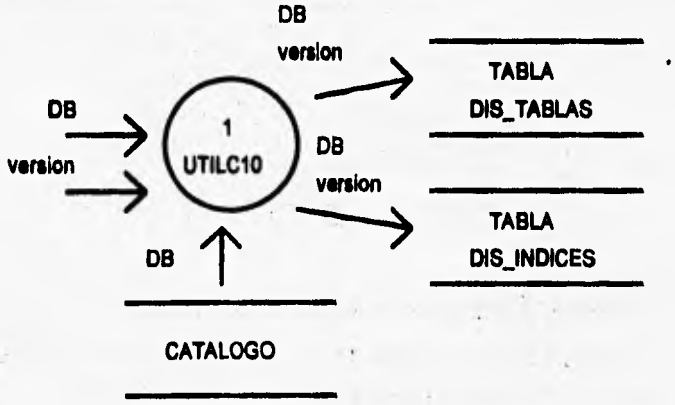
Descripción de cada uno de los campos de la tabla DIS_UTILID:

Campo	Descripción
UTILID	Utileria a aplicar (REORG, RUNSTATS, FULL IMAGE COPY,) (C)

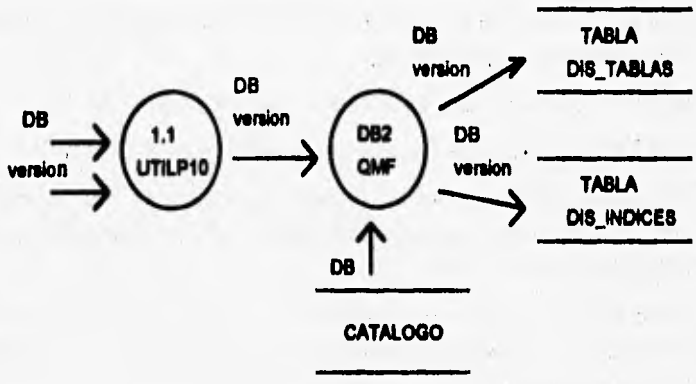
4.3 DIAGRAMA DE FLUJO DE DATOS

1) Inicialización de las Tablas auxiliares.

⇒ UTILC10 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database especifica.

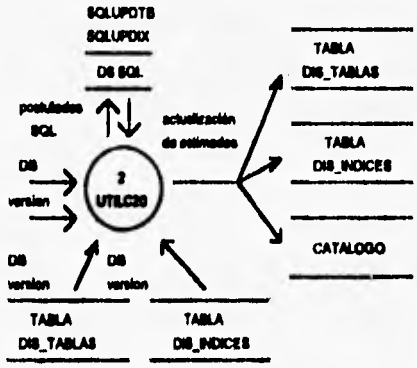


⇒ UTILP10 : Obtiene los valores de DB y versión



2) Actualización de estimados en tablas auxiliares

⇒ UTILC20 : Actualización de Estimados para Table spaces e Indices.



EL módulo UTILC20 se divide en los siguientes submódulos:

⇒ **UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.**

⇒ **UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.**

⇒ **UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.**

⇒ **UTILP20 : Inicia la actualización de la tabla de ISPF denominada UTILTAB.**

⇒ **UTILP21 : Inicia la actualización de la tabla de ISPF denominada UTILIDX.**

⇒ **UTILUPD1 : Con los datos almacenados en la Tabla de ISPF denominada UTILIDX se generan postulados UPDATE de SQL para actualizar la tabla DIS_INDICES de DB2.**

⇒ **UTILUPD2 : Con los datos almacenados en la Tabla de ISPF denominada UTILTAB se generan postulados UPDATE de SQL para actualizar la tabla DIS_TABLAS de DB2.**

⇒ **UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen las tablas (DIS_TABLAS y DIS_INDICES), así como el catálogo de DB2.**

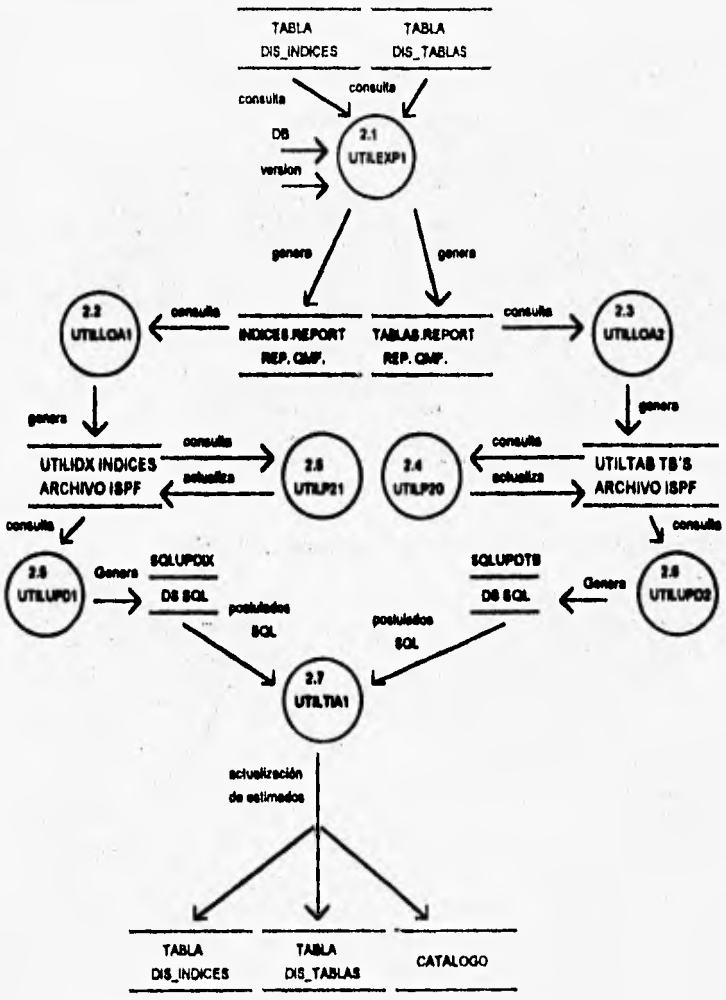
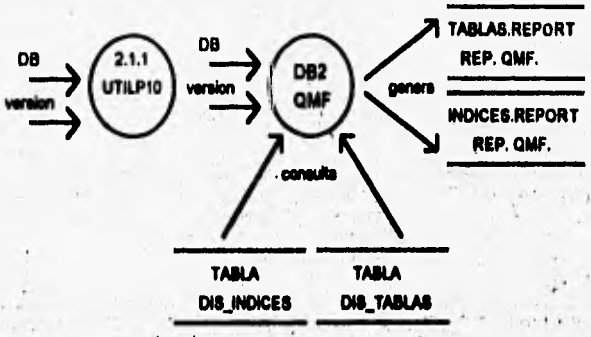


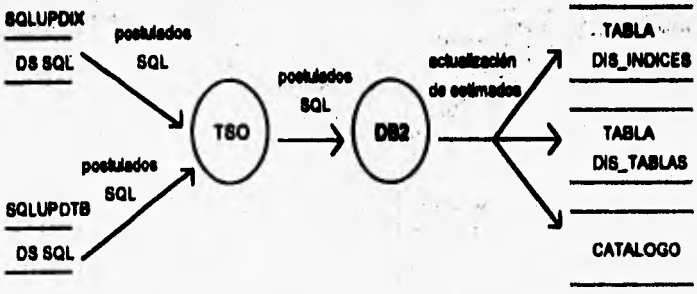
TABLA
DIS_INDICES
TABLA
DIS_TABLAS
consulta
consulta
DB
version
2.1
UTILEXP1
genera
genera
2.2
UTILLOA1
consulta
INDICES.REPORT
REP. CMF.
TABLAS.REPORT
REP. CMF.
consulta
2.3
UTILLOA2
genera
genera
UTILIDX INDICES
ARCHIVO ISPF
consulta
actualiza
2.5
UTILP21
2.4
UTILP20
consulta
actualiza
UTILTAB TB'S
ARCHIVO ISPF
consulta
consulta
2.6
UTILUP01
Genera
SQLUPDIX
DB SQL
SQLUPDTB
DB SQL
Genera
2.6
UTILUP02
postulados
SQL
postulados
SQL
2.7
UTILTIA1
actualización
de estimados
TABLA
DIS_INDICES
TABLA
DIS_TABLAS
CATALOGO

EL módulo UTILEXP1 se divide en los siguientes submódulos:

⇒ UTILP10 : Obtiene los valores de DB y versión

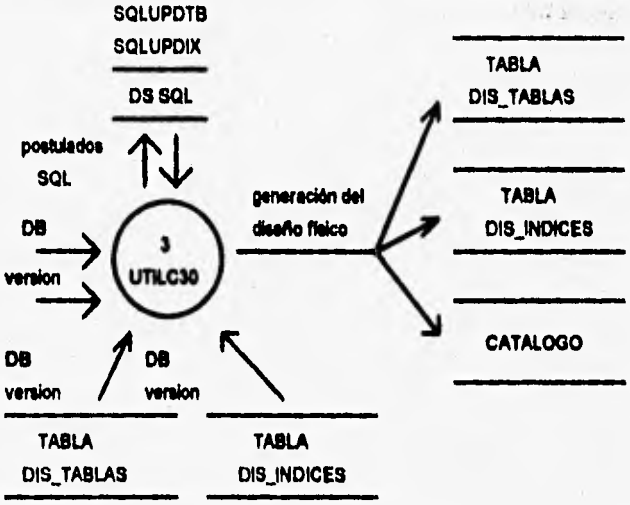


EL módulo UTILTIA1 se divide en los siguientes submódulos:



3) Definición del diseño físico.

⇒ UTILC30 : A partir de los Estimados dados por el usuario, se efectúa el Diseño Físico que incluye: decidir tipo de tablespace, valores de los parámetros CLOSE, PCTFREE, PRIQTY, SECQTY, LOCKSIZE, cantidad de particiones y máximo número de índices.



EL módulo UTILC30 se divide en los siguientes submódulos:

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

⇒ REALIZACION DEL DISEÑO FISICO : Inicia el Diseño Físico de Tablespaces y el Diseño Físico de Indices.

⇒ UTILUPD1 : Con los datos almacenados en la Tabla de ISPF denominada UTILIDX se generan postulados UPDATE de SQL para actualizar la tabla DIS_INDICES de DB2.

⇒ UTILUPD2 : Con los datos almacenados en la Tabla de ISPF denominada UTILTAB se generan postulados UPDATE de SQL para actualizar la tabla DIS_TABLAS de DB2.

⇒ **UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen las tablas (DIS_TABLAS y DIS_INDICES), así como el catálogo de DB2.**

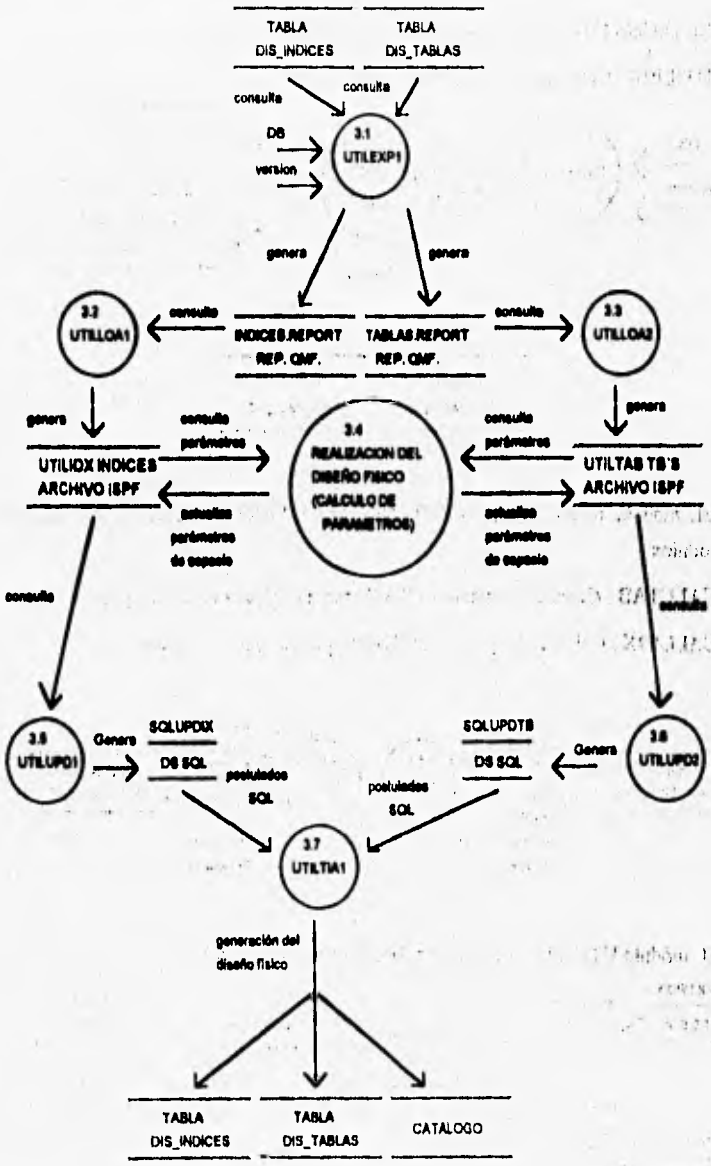
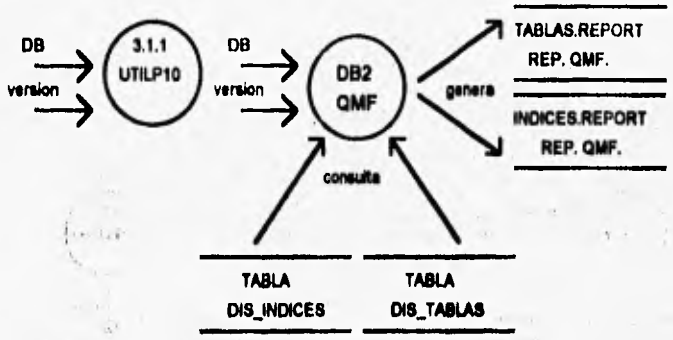


TABLA DIS_INDICES
TABLA DIS_TABLAS
consulta
consulta
DB
version
3.1 UTILEXP1
genera
genera
3.2 UTILLOA1
consulta
INDICES.REPORT REP. CMF.
TABLAS.REPORT REP. CMF.
consulta
3.3 UTILLOA2
genera
consulta parámetros
3.4 REALIZACION DEL DISEÑO FISICO (CALCULO DE PARAMETROS)
consulta parámetros
genera
UTILIDX INDICES ARCHIVO ISPF
solicita parámetros de espacio
UTILTAB TB'S ARCHIVO ISPF
solicita parámetros de espacio
consulta
consulta
3.6 UTILUPD1
Genera
SQLUPDIX
DB SQL
postulados SQL
SQLUPDTB
DB SQL
Genera
3.5 UTILUPD2
postulados SQL
3.7 UTILTIA1
generación del diseño físico
TABLA DIS_INDICES
TABLA DIS_TABLAS
CATALOGO

EL módulo UTILEXP1 se divide en los siguientes submódulos:

⇒ **UTILP10** : Es un módulo de obtención de datos.

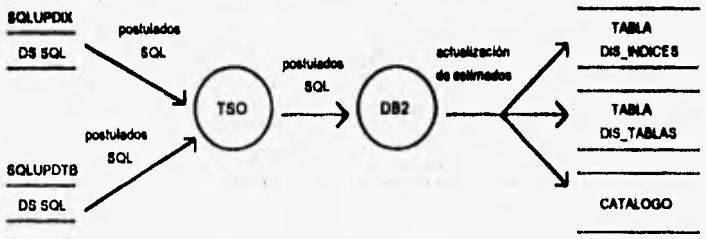


EL módulo REALIZACION DEL DISEÑO FISICO se divide en los siguientes submódulos:

⇒ **CALCTAB** : Calcula el espacio en DASD requerido para un Tablespace.

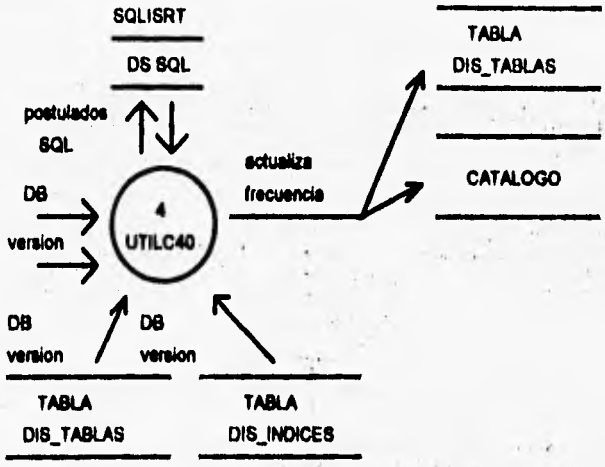
⇒ **CALCIDX** : Calcula el espacio en DASD requerido para un Indice.

EL módulo UTILTIA1 se divide en los siguientes submódulos:



4) Definición de utilerías y frecuencia.

⇒ UTILC40 : A partir de los Estimados dados por el usuario, se efectúa la definición de Utilerías que incluye: Tipo de Utilería (REORG, IMAGE COPY FULL, IMAGE COPY INCREMENTAL, RUNSTATS, ETC.) y Frecuencia de ejecución recomendada.



EL módulo UTILC40 se divide en los siguientes submódulos:

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

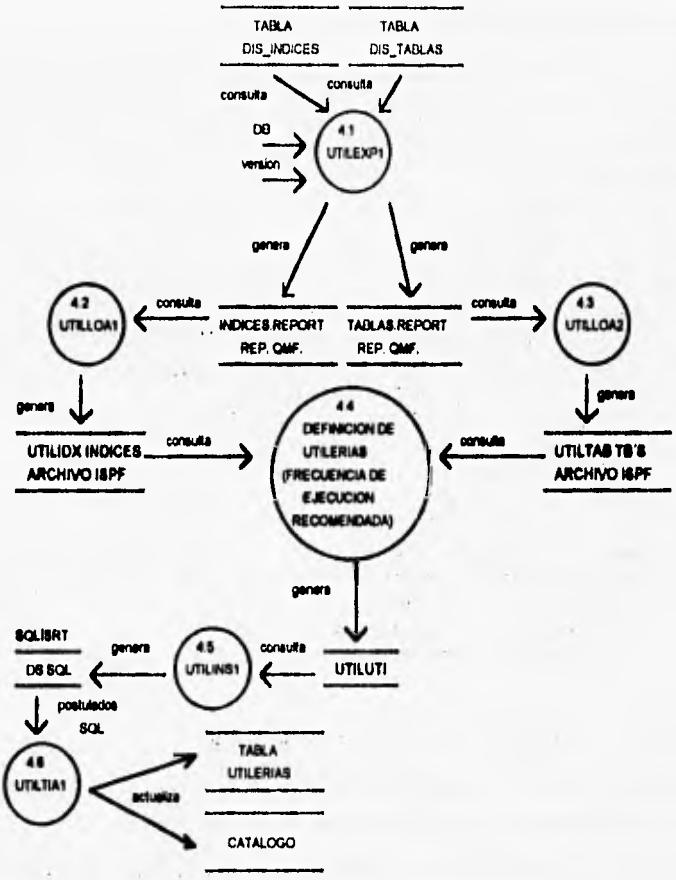
⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

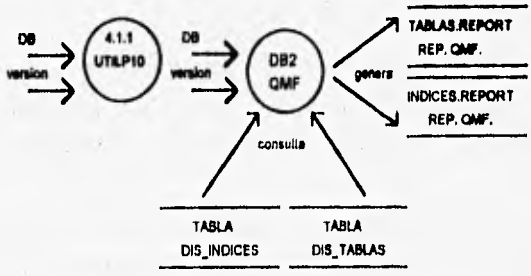
⇒ DEFINICION DE UTILERIAS : Inicia la Definición de Utilerías.

⇒ UTILINS1 : Con los datos almacenados en la Tabla de ISPF denominada UTILUTI se generan postulados INSERT de SQL para actualizar la tabla UTILERIAS de DB2.

⇒ UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen la tabla (UTILERIAS), así como el catálogo de DB2.



EL módulo UTILEXP1 se divide en los siguientes submódulos:



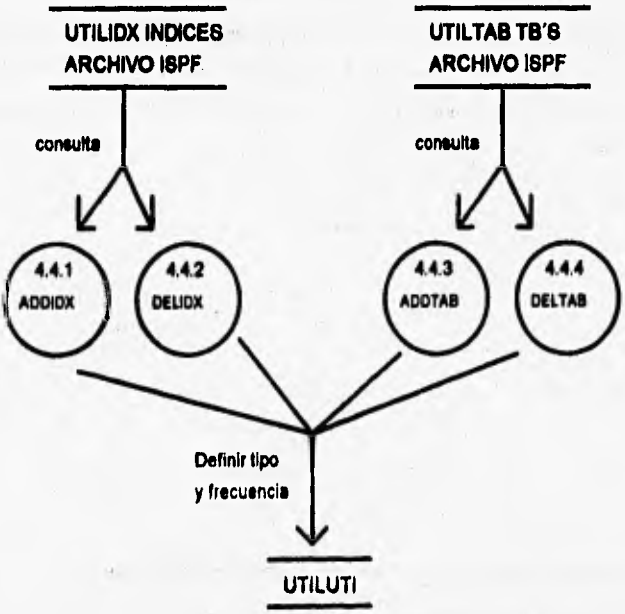
EL módulo DEFINICION DE UTILERIAS se divide en los siguientes submódulos:

⇒ **ADDTAB** : Añade al archivo **UTILUTI** parámetros con el tipo de Utilería y frecuencia recomendada para tablas.

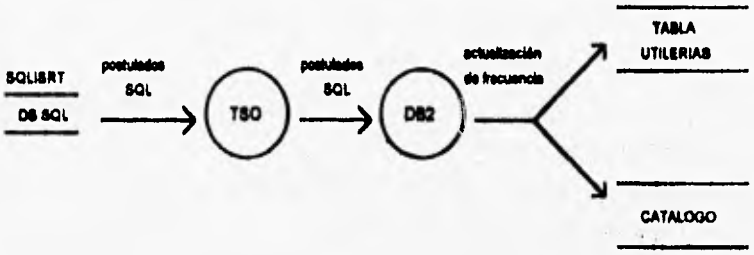
⇒ **DELTAB** : Borra al archivo **UTILUTI** parámetros con el tipo de Utilería y frecuencia recomendada para tablas.

⇒ **ADDIDX** : Añade al archivo **UTILUTI** parámetros con el tipo de Utilería y frecuencia recomendada para índices.

⇒ **DELIDX** : Borra al archivo **UTILUTI** parámetros con el tipo de Utilería y frecuencia recomendada para índices.

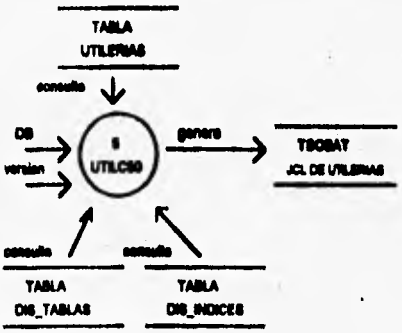


EL módulo UTILTIA1 se divide en los siguientes submódulos:



5) Generación de JCL para utilerías.

⇒ UTILC50 : A partir de la tabla denominada UTILERIAS, generada en el paso previo, se generan diferentes jobs con el JCL y postulados de ejecución para los objetos de una Database/Versión, de acuerdo con la definición almacenada en esa tabla.



EL módulo UTILC50 se divide en los siguientes submódulos:

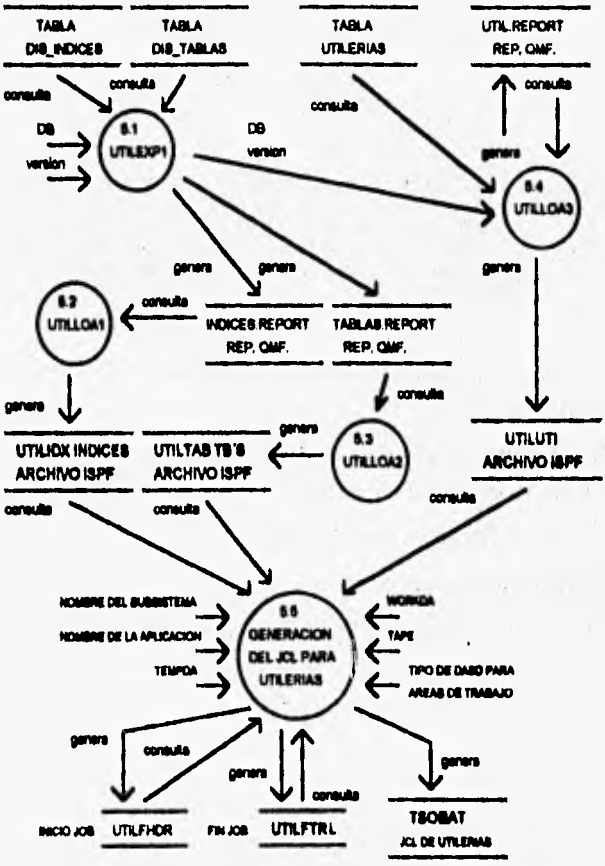
⇒ QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de de ISPF denominada UTILIDX.

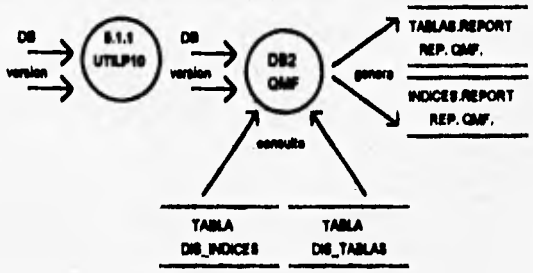
⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

⇒ UTILLOA3 : Exporta la tabla UTILERIAS en formato de reporte de QMF para ser cargada como tabla ISPF. A partir del reporte exportado de la tabla UTILERIAS, carga una tabla ISPF denominada UTILUTI.

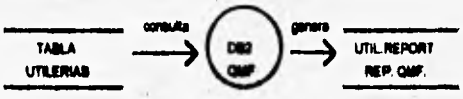
⇒ GENERACION DEL JCL PARA UTILERIAS : Es una interfaz entre los archivos ISPF y los módulos encargados de generar el JCL para utilerías, además de ser un panel de obtención de datos.



EL módulo UTILEXP1 se divide en los siguientes submódulos:



EL módulo UTILLOA3 se divide en los siguientes submódulos:



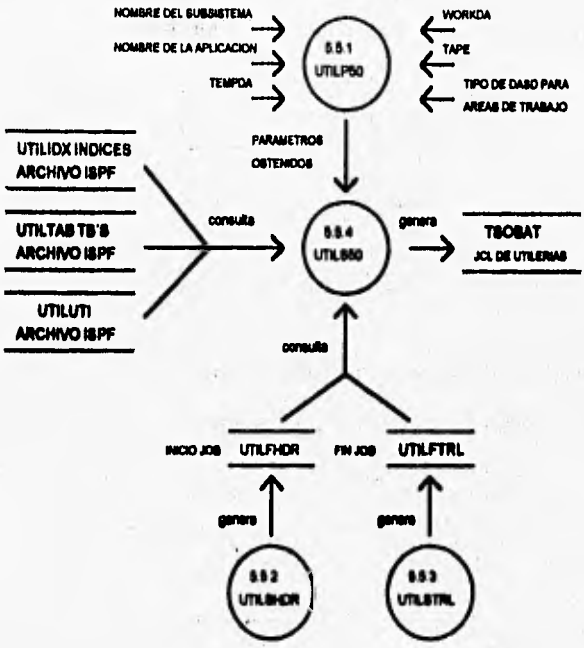
EL módulo GENERACION DEL JCL PARA UTILERIAS se divide en los siguientes submódulos:

⇒ **UTILP50 : Panel de obtención de datos (nombre del subsistema, nombre de la aplicación, tempda, workda, tape, tipo de DASD para áreas de trabajo).**

⇒ **UTILSHDR : Establece el inicio del job.**

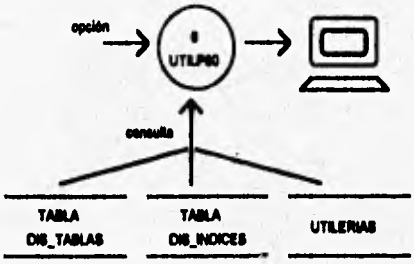
⇒ **UTILSTRL : Establece el final del job.**

⇒ **UTILS50 : Genera el archivo TSOBAT que contiene el JCL de utilerías.**

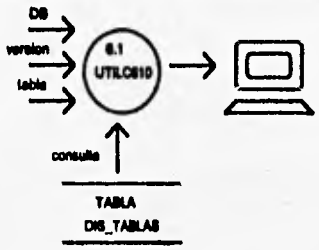


C)Consulta a tablas auxiliares.

⇒ **UTILP60 : Panel que selecciona la tabla auxiliar (DIS_TABLAS, DIS_INDICES o UTILERIAS) para realizar su consulta.**

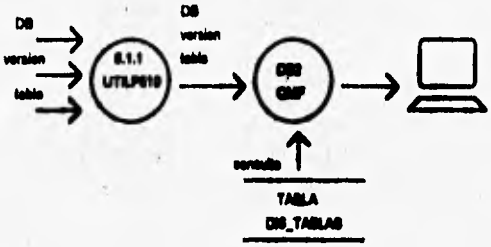


⇒ UTILC610 : Módulo que consulta la tabla DIS_TABLAS.

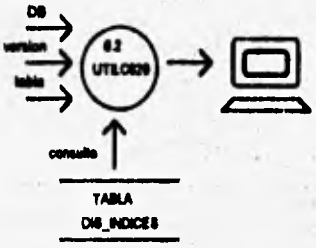


El módulo UTILC610 se divide en los siguientes módulos:

⇒ UTILP610: Obtiene los valores de DB, versión y tabla.

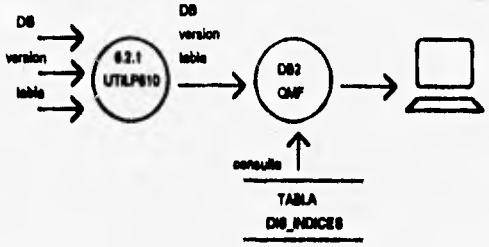


⇒ UTILC620 : Módulo que consulta la tabla DIS_INDICES.

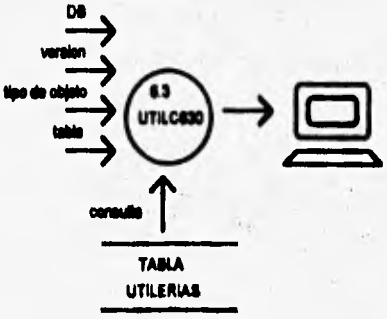


El módulo UTILC620 se divide en los siguientes módulos:

⇒ **UTILP610**: Obtiene los valores de DB, versión y tabla.

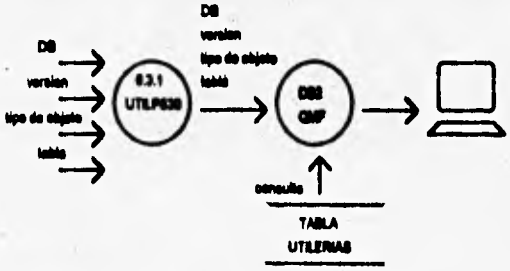


⇒ **UTILC620 : Módulo que consulta la tabla UTILERIAS.**



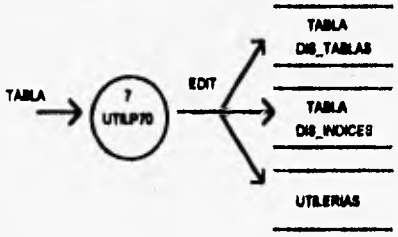
El módulo UTILC630 se divide en los siguientes módulos:

⇒ **UTILP630 : Obtiene los valores de DB, versión, tipo de objeto y tabla.**



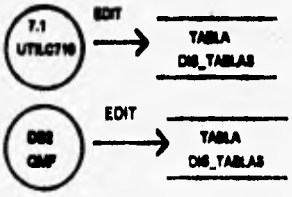
A)Actualización de tablas auxiliares.

⇒ UTILP70 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database específica.

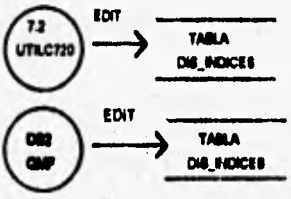


El módulo UTILP70 se divide en los siguientes módulos:

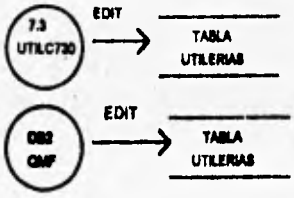
⇒ UTILC710 : Actualiza la tabla DIS_TABLAS.



⇒ UTILC720 : Actualiza la tabla DIS_INDICES.

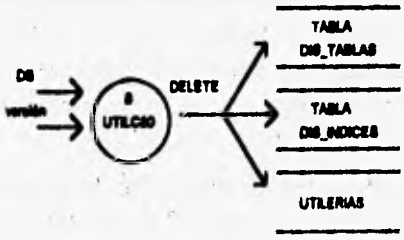


⇒ UTILC730 : Actualiza la tabla DIS_UTILERIAS.



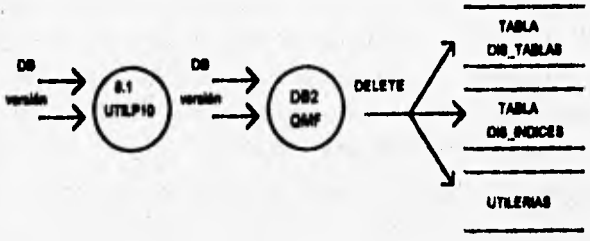
D)Depuración de tablas auxiliares.

⇒ UTILC80 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database específica.



El módulo UTILC80 se divide en los siguientes módulos:

⇒ UTILP10 : Obtiene los valores de DB y versión.



4.4 EL CATALOGO (DICCIONARIO DE DATOS) DE DB2 EN EL SISTEMA

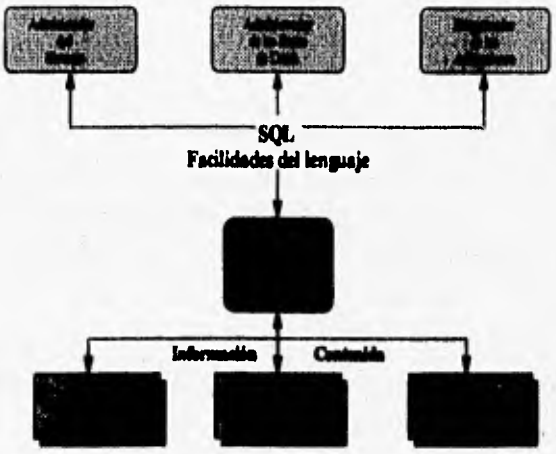
La estructura interna de DB2 es bastante compleja y el sistema tiene las espectativas de proveer todas las funciones normalmente encontradas en un moderno DBMS (Data Base Management System). El producto contiene un número grande de componentes internos.

Dentro de la estructura interna de DB2, se encuentra el componente de servicios de las bases de datos (*databases services*), el cual soporta la definición, recuperación y actualización del usuario y datos del sistema.

El propósito primario del componente de servicios de las bases de datos es el soportar la definición, la recuperación y actualización de datos, de las bases de datos del DB2, en otras palabras, los servicios consisten en la implementación de las funciones en el lenjuage de SQL que suministren el soporte necesario para permitir la operación y respuesta del DB2 tal como, la preparación de programas de aplicaciones, para su ejecución y subsecuentes ejecuciones.

Los servicios de las bases de datos se apoyan para su funcionamiento del manejo de un conjunto de tablas que son propias del sistema (DB2) llamadas tablas del sistema (para distinguirlas de las tablas de usuario). Estas tablas contienen el control y descripción de información considerando datos de tablas, de usuario y sus columnas, además contiene datos para el caso de operaciones de recuperación de las bases de datos (tablas e índices). Las tablas del sistema se dividen en dos grupos, conocidas como el catálogo y el directorio de DB2. Desde el punto de vista del usuario la diferencia entre ellas es la siguiente: las tablas del catálogo son accesibles por medio de setencias de manejo de datos de SQL, las cuales pueden ser usadas para producir reportes para uso de los administradores de las bases de datos. Las tablas del Directorio no pueden ser accesadas vía sentencias de SQL y es puramente intentado el acceso por el propio uso interno del DB2.

El catálogo también llamado diccionario de datos en DB2 es una base de datos del sistema, consiste en un conjunto de tablas las cuales, contienen y describen información concerniente a varios objetos que son de interés del sistema propio. Ejemplos de tales objetos son; tablespaces, tablas, columnas, índices, privilegios, etc.



El catálogo de datos DB2

Los datos en las tablas del catálogo están disponibles para usuarios autorizados de DB2 a través de consultas SQL , sin embargo es usado primeramente por DB2 y está por lo tanto sujeto a cambios. Las tablas del catálogo son actualizadas por DB2, durante su operación normal en respuesta a las sentencias de definición de datos, de control, SQL y ciertamente de comandos y utilerías de DB2.

Una ventaja signifcante de un sistema relacional como DB2 es que el catálogo esta compuesto dentro del mismo sistema de relaciones o tablas; como resultado los usuarios pueden interrogar a las tablas del catálogo usando las facilidades estándar que ofrece su lenguage normal de consultas (SQL en el caso de DB2).

Dentro de DB2 especialmente, el catálogo consiste de 30 tablas, no es nuestro propósito dar una descripción extensa del catálogo, sin embargo, se ha mencionado lo básico para dar idea de como la información dentro del catálogo puede auxiliarnos como usuarios de DB2 y al sistema DB2DA.

Las tablas del catálogo que intervienen en el modelo del sistema DB2DA son las siguientes:

SYSINDEXES	Contiene un renglón por cada índice.
SYSINDEXEXPART	Contiene un renglón por cada índice no particionado y un renglón por cada partición de un índice particionado.
SYSTABLEPART	Contiene un renglón por cada tablespace no particionado y un renglón por cada partición de un tablespace particionado.
SYSTABLES	Contiene un renglón por cada tabla, vista o alias.
SYSTABLESPACE	Contiene un renglón por cada tablespace.

El sistema DB2DA obtendrá de estas la descripción necesaria de los objetos a analizar (tablas e índices), dados por el usuario

No todas las columnas de las tablas del catálogo son parte de la interfaz de uso de programación, es decir, cuando en la programación de una aplicación en sus programas se codifican sentencias de SQL para actualizar (definición y alteración de objetos) a las tablas del catálogo sólo ciertas columnas son directamente actualizadas, ya que otras columnas son del uso exclusivo del DB2 y algunas no se usan.

Enseguida ilustramos las tablas del catálogo que forman parte del modelo entidad relación del sistema DB2DA sólo con las columnas que se utilizan en dicho diseño y en la programación del sistema.

nota: Las tablas del catálago tienen como propietario (owner) el calificador SYSIBM (Especificación técnica del manual de administración de DB2)

SYSIBM.SYSTABLEPART

PARTITION	SMALLINT	Número de partición; 0 si el tablespace es no particionado.
TSNAME	CHAR(8)	Nombre del tablespace.
DBNAME	CHAR(8)	Nombre de la database que contiene el tablespace.
IXNAME	VARCHAR(18)	Nombre del índice particionado. Esta columna esta en blanco si el tablespace no es particionado.
IXCREATOR	CHAR(8)	Identificador de autorización del propietario del índice particionado. Esta columna esta en blanco si el tablespace no es particionado.
PQTY	INTEGER	Espacio primario de asignación en unidades de bloques de 4K-bytes. El valor de esta columna es 0 si un storage group no se esta usando. PQTY esta basado sobre el valor del parámetro PRIQTY dentro de la apropiada sentencia de CREATE o ALTER TABLESPACE. Sin embargo PRIQTY pregunta por el espacio en unidades de 1K-bytes.
SQTY	SMALLINT	Espacio secundario de asignación en unidades de bloques de 4K-bytes. El valor de esta columna es 0 si un storage group no se está usando. SQTY esta basado sobre el valor del parámetro SECQTY dentro de la apropiada sentencia de CREATE o ALTER TABLESPACE. Sin embargo SECQTY pregunta por el espacio en unidades de 1K-bytes.
STORTYPE	CHAR(1)	Tipo de asignación de almacenamiento. E Explícito (storage group no usado) I Implícito(storage group usado)
STORNAME	CHAR(8)	Nombre del storage group usado para la asignación de almacenamiento. Blanco si el storage group no es usado.
VCATNAME	CHAR(8)	Nombre de calificador del catálogo maestro del sistema (integrated catalog facility) usado para la asignación de espacio.
CARD	INTEGER	Número de renglones dentro del tablespace o partición. -1 si no tiene estadísticas generadas.
LIMITKEY	VARCHAR(512)	Valor de la llave límite en la partición en un formato externo. 0 si el tablespace no es particionado.
FREEPAGE	SMALLINT	El número de páginas cargadas antes de que una página es dejada como espacio libre.
PCTFREE	SMALLINT	El porcentaje de cada página dejada como espacio libre.
CHECKFLAG	CHAR(1)	'C': La partición del tablespace esta en modo pendiente de checar (CHECK PENDING) y hay renglones que pueden violar restricciones de integridad referencial. Blanco: Si el tablespace no es particionado o no contiene renglones que puedan violar restricciones de integridad referencial.

SYSIBM.SYSTABLES

NAME	VARCHAR(18)	Nombre de la tabla, vista o alias.
CREATOR	CHAR(8)	Identificador de autorización del propietario de la tabla, vista, o alias.
TYPE	CHAR(1)	Tipo de objeto: A alias T table V Vista
DBNAME	CHAR(8)	Para una tabla, o vista de tablas, el nombre de la database que contiene el tablespace mencionado en TSNAME. Para un alias o vista de una vista, el valor es DSNDB06.
TSNAME	CHAR(8)	Para una tabla o vista de una tabla, el nombre del tablespace que contiene la tabla. Para una vista de más de una tabla el nombre del tablespace que contiene una de las tablas. Para una vista de una vista, el valor es SYSVIEWS. Para un alias el valor es SYSDBAUT.
CARD	INTEGER	Número de renglones dentro del tablespace o partición. -1 si no tiene estadísticas generadas o el renglón describe a una vista o alias. Esta es una columna actualizable.
REMARKS	VARCHAR(254)	Un string (conjunto) de caracteres suministrados por el usuario en el parámetro COMMENT en la sentencia de definición de la tabla.
KEYCOLUMS	SMALLINT	El número de columnas en la llave primaria de la tabla. 0 si el renglón describe una vista o alias.
CREATEDBY	CHAR(8)	Identificador de autorización primaria del usuario quien creo la tabla, vista o alias.
TBCREATOR	CHAR(8)	Para un alias, el identificador de la autorización del dueño de la tabla que hace referencia; blanco si es cualquier otro valor.
TBNAME	VARCHAR(18)	Para un alias, el nombre de la tabla o vista que hace referencia; blanco si es cualquier otro valor.

SYSIBM.SYSINDEXES

NAME	VARCHAR(18)	Nombre del índice.
CREATOR	CHAR(8)	ID de autorización del dueño del índice.
TBNAME	VARCHAR(18)	Nombre de la tabla sobre el índice está definido.
TBCREATOR	CHAR(8)	ID de autorización del dueño de la tabla.
UNIQUERULE	CHAR(1)	Si el índice es único: D no (duplicados son permitidos). U yes. P índice primario (ùnico). C el índice es utilizado para forzar la restricción única.
COLCOUNT	SMALLINT	El número de columnas en la llave.
CLUSTERING	CHAR(1)	Si el CLUSTER fue especificado cuando el índice fue creado: N no Y yes
CLUSTERED	CHAR(1)	Si la tabla es actualmente agrupado por el índice: N No: un número significativo de renglones no está en orden agrupado. Y Yes: La mayor parte de los renglones está en orden agrupado.
DBNAME	CHAR(8)	Nombre de la database que contiene el índice.
INDEXSPACE	CHAR(8)	Nombre del espacio del índice.
BPOOL	CHAR(8)	Nombre del buffer pool usado por el índice.
PGSIZE	SMALLINT	Tamaño, en bytes, de las subpáginas en el índice: 256, 512, 1024, 2048, o 4096.
DBNAME	CHAR(8)	Nombre de la database que contiene el índice.
ERASERULE	CHAR(1)	Si los data sets son borrados.
SPACE	INTEGER	Número de kbytes de asignación de almacenamineto (DASD) para el índice.
CREATEDBY	CHAR(8)	ID de autorización primario del usuario quien creo el índice.

SYSIBM.SYSTABLESPACE

NAME	CHAR(8)	Nombre del table space.
CREATOR	CHAR(8)	ID de autorización del dueño del table space
DBNAME	CHAR(8)	Nombre de la database que contiene el table space.
BPOOL	CHAR(8)	Nombre del buffer pool usado por el table space.
PARTITIONS	SMALLINT	Número de particiones del table space; 0 si el table space no es particionado
LOCKRULE	CHAR(1)	Tamaño de bloqueo del table space: A any P page S table space T table
PGSIZE	SMALLINT	Tamaño de las páginas en el table space en Kbytes.
ERASERULE	CHAR(1)	Si los data sets serán borrados con un DROP. El valor es significativo si el table space es particionado: N no erase Y erase
NTABLES	SMALLINT	Número de tablas definidas en el table space.
CLOSERULE	CHAR(1)	Si los data sets son candidatos para cerrarse cuando el límite sobre el número de data sets abiertos es alcanzado: Y yes N no
SPACE	INTEGER	Número de Kbytes de almacenamiento DASD asignado para el table space, como el determinado por la última ejecución de la utilería STOSPACE. 0 si el table space no está relacionado con un storage group. Si el table space es particionado, y diferentes storage groups han sido especificados, éste es el número de kilobytes de almacenamiento DASD asignado para la partición como el determinado por la última ejecución de la utilería STOSPACE.
SEGSIZE	SMALLINT	El número de páginas en cada segmento de un table space segmentado. Cero si el table space no es segmentado.
CREATEBY	CHAR(8)	ID de autorización primario del usuario quién creó el table space.

SYSIBM.SYSINDEXPART

Nombre de la columna	Tipo de dato	Descripción
PARTITION	SMALLINT	Número de partición; 0 si el índice no es particionado.
IXNAME	VARCHAR(18)	Nombre del índice.
IXCREATOR	CHAR(8)	ID de autorización del dueño del índice.
PQTY	INTEGER	Asignación de espacio primario en unidades de bloques de 4k-byte de almacenamiento. Cero si un storage group no es utilizado.
SQTY	SMALLINT	Asignación de espacio secundario en unidades de bloques de 4k-byte de almacenamiento. Cero si un storage group no es utilizado.
STORTYPE	CHAR(1)	Tipo de asignación de almacenamiento. E Explícito (storage group no usado) I Implícito(storage group usado)
STORNAME	CHAR(8)	Nombre del storage group o integrated catalog facility catalog utilizado para la asignación de espacio.
VCATNAME	CHAR(8)	Nombre del integrated catalog facility usado para la signación de espacio.
FREEPAGE	SMALLINT	El número de páginas que son cargadas antes de que un página es puesta como espacio libre
PCTFREE	SMALLINT	El porcentaje de cada subpágina que es puesto como espacio libre.

4.4.1 Diagrama entidad-relacion de la base de datos del sistema DB2DA

La estructura lógica general del sistema DB2DA puede expresarse en forma gráfica a través de un diagrama entidad relación. Este diagrama nos ayudará a representar el modelo de datos conceptual del sistema DB2DA, es decir, los datos, sus atributos y las ligas con su correspondiente cardinalidad.

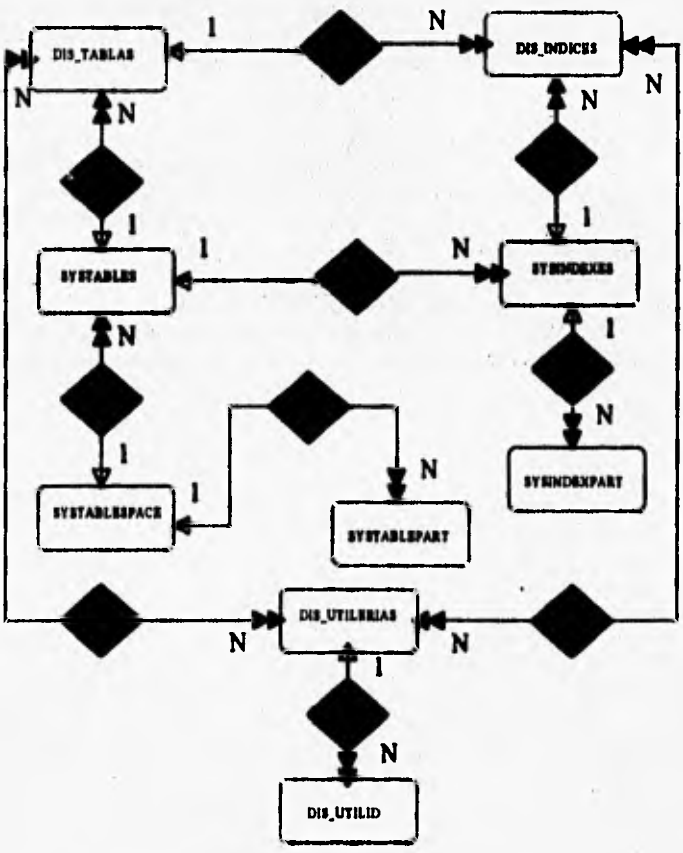


Figura 4.4.1 Diagrama entidad-relación de la base de datos del sistema DB2DA.

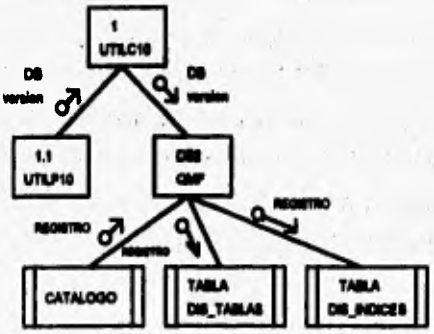
4.5 DISEÑO DE LA ESTRUCTURA MODULAR DEL SISTEMA

La estructura modular del sistema comprende el desarrollo de una visión conceptual del sistema, el establecimiento de una estructura, la identificación de los datos y su almacenamiento, la identificación de relaciones e interconexiones entre componentes y el desarrollo.

4.5.1 INICIALIZACION DE LAS TABLAS AUXILIARES.

⇒ UTILC10 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database especifica.

⇒ UTILP10 : Obtiene los valores de DB y versión



4.5.2 ACTUALIZACION DE LAS TABLAS AUXILIARES

⇒ UTILC20 : Actualización de Estimados para Table spaces e Indices.

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

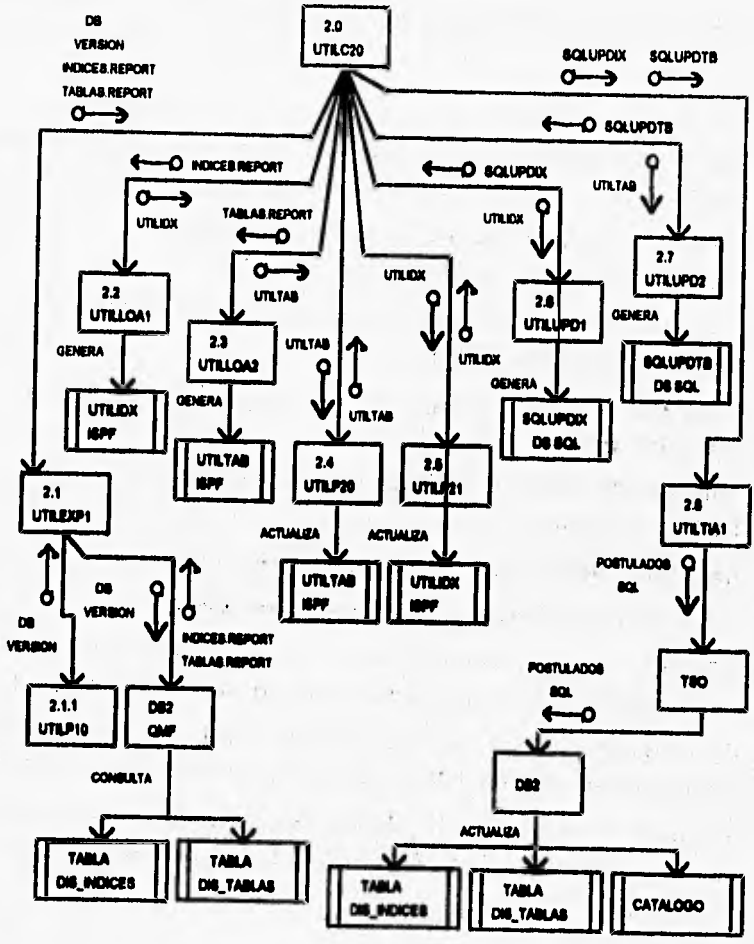
⇒ UTILP20 : Inicia la actualización de la tabla de ISPF denominada UTILTAB.

⇒ UTILP21 : Inicia la actualización de la tabla de ISPF denominada UTILIDX.

⇒ UTILUPD1 : Con los datos almacenados en la Tabla de ISPF denominada UTILIDX se generan postulados UPDATE de SQL para actualizar la tabla DIS_INDICES de DB2.

⇒ UTILUPD2 : Con los datos almacenados en la Tabla de ISPF denominada UTILTAB se generan postulados UPDATE de SQL para actualizar la tabla DIS_TABLAS de DB2.

⇒ UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen las tablas (DIS_TABLAS y DIS_INDICES), así como el catálogo de DB2.


DB
VERSION
INDICES.REPORT
TABLAS.REPORT
2.0
UTILC20
SQLUPDIX
SQLUPDTB
SQLUPDTB
INDICES.REPORT
SQLUPDIX
UTILTAB
UTILIDX
TABLAS.REPORT
UTILIDX
UTILIDX
2.2
UTILLOA1
UTILTAB
2.3
UTILLOA2
UTILIDX
2.8
UTILUPD1
2.7
UTILUPD2
GENERA
GENERA
GENERA
SQLUPDTB
DB SCL
UTILIDX
ISPF
UTILTAB
ISPF
UTILTAB
2.4
UTILP20
2.5
UTILP21
SQLUPDIX
DB SCL
2.1
UTILEXP1
2.6
UTILTIA1
ACTUALIZA
ACTUALIZA
POSTULADOS
SCL
DB
VERSION
DB
VERSION
INDICES.REPORT
TABLAS.REPORT
UTILTAB
ISPF
UTILIDX
ISPF
TSO
POSTULADOS
SCL
2.1.1
UTILP10
DB2
CMF
DB2
CONSULTA
ACTUALIZA
TABLA
DIS_INDICES
TABLA
DIS_TABLAS
TABLA
DIS_INDICES
TABLA
DIS_TABLAS
CATALOGO

4.5.3 DEFINICION DEL DISEÑO FISICO

⇒ UTILC30 : A partir de los Estimados dados por el usuario, se efectúa el Diseño Físico que incluye: decidir tipo de tablespace, valores de los parámetros CLOSE, PCTFREE, PRIQTY, SECQTY, LOCKSIZE, cantidad de particiones y máximo número de índices.

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

⇒ REALIZACION DEL DISEÑO FISICO : Inicia el Diseño Físico de Tablespaces y el Diseño Físico de Indices.

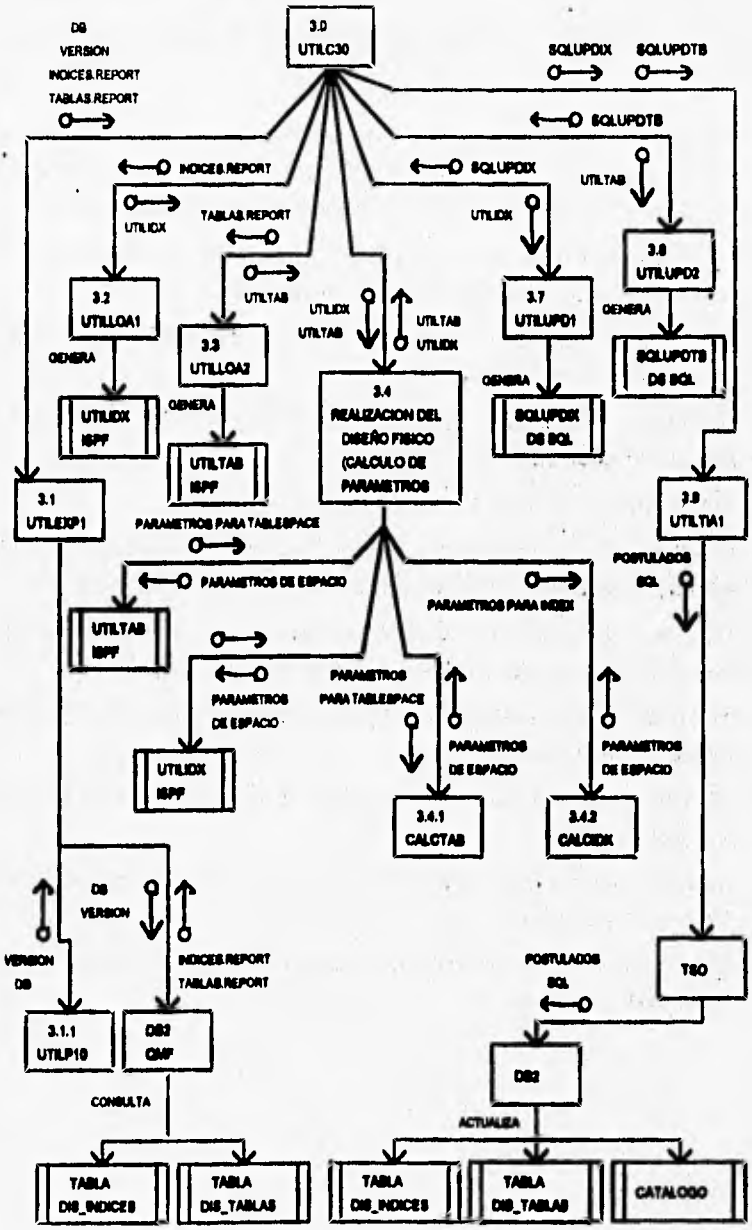
⇒ CALCTAB : Calcula el espacio en DASD requerido para un Tablespace.

⇒ CALCIDX : Calcula el espacio en DASD requerido para un Indice.

⇒ UTILUPD1 : Con los datos almacenados en la Tabla de ISPF denominada UTILIDX se generan postulados UPDATE de SQL para actualizar la tabla DIS_INDICES de DB2.

⇒ UTILUPD2 : Con los datos almacenados en la Tabla de ISPF denominada UTILTAB se generan postulados UPDATE de SQL para actualizar la tabla DIS_TABLAS de DB2.

⇒ UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen las tablas (DIS_TABLAS y DIS_INDICES), así como el catálogo de DB2.


DB
VERSION
INDICES.REPORT
TABLAS.REPORT
3.0
UTILC30
SQLUPDIX
SQLUPDTB
SQLUPDTB
INDICES.REPORT
SQLUPDIX
UTILTAB
UTILIDX
TABLAS.REPORT
UTILIDX
3.8
UTILUPD2
3.2
UTILLOA1
UTILTAB
UTILIDX
UTILTAB
3.7
UTILUPD1
GENERA
3.3
UTILLOA2
UTILTAB
UTILIDX
SQLUPDTB
DB SQL
GENERA
GENERA
3.4
REALIZACION DEL
DISEÑO FISICO
(CALCULO DE
PARAMETROS
GENERA
UTILIDX
ISPF
UTILTAB
ISPF
SQLUPDIX
DB SQL
3.1
UTILEXP1
PARAMETROS PARA TABLESPACE
3.9
UTILTIA1
POSTULADOS
SQL
PARAMETROS DE ESPACIO
PARAMETROS PARA INDEX
UTILTAB
ISPF
PARAMETROS
DE ESPACIO
PARAMETROS
PARA TABLESPACE
PARAMETROS
DE ESPACIO
PARAMETROS
DE ESPACIO
UTILIDX
ISPF
3.4.1
CALCTAB
3.4.2
CALCIDX
DB
VERSION
VERSION
DB
INDICES.REPORT
TABLAS.REPORT
POSTULADOS
SQL
TSO
3.1.1
UTILP10
DB2
CMF
DB2
CONSULTA
ACTUALIZA
TABLA
DB_INDICES
TABLA
DB_TABLAS
TABLA
DB_INDICES
TABLA
DB_TABLAS
CATALOGO

4.5.4 DEFINICION DE UTILERIAS Y FRECUENCIA

⇒ UTILC40 : A partir de los Estimados dados por el usuario, se efectúa la Definición de Utilerías que incluye: Tipo de Utilería (REORG, IMAGE COPY FULL, IMAGE COPY INCREMENTAL, RUNSTATS, ETC.) y Frecuencia de ejecución recomendada.

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

⇒ DEFINICION DE UTILERIAS : Inicia la Definición de Utilerías.

⇒ UTILINS1 : Con los datos almacenados en la Tabla de ISPF denominada UTILUTI se generan postulados INSERT de SQL para actualizar la tabla UTILERIAS de DB2.

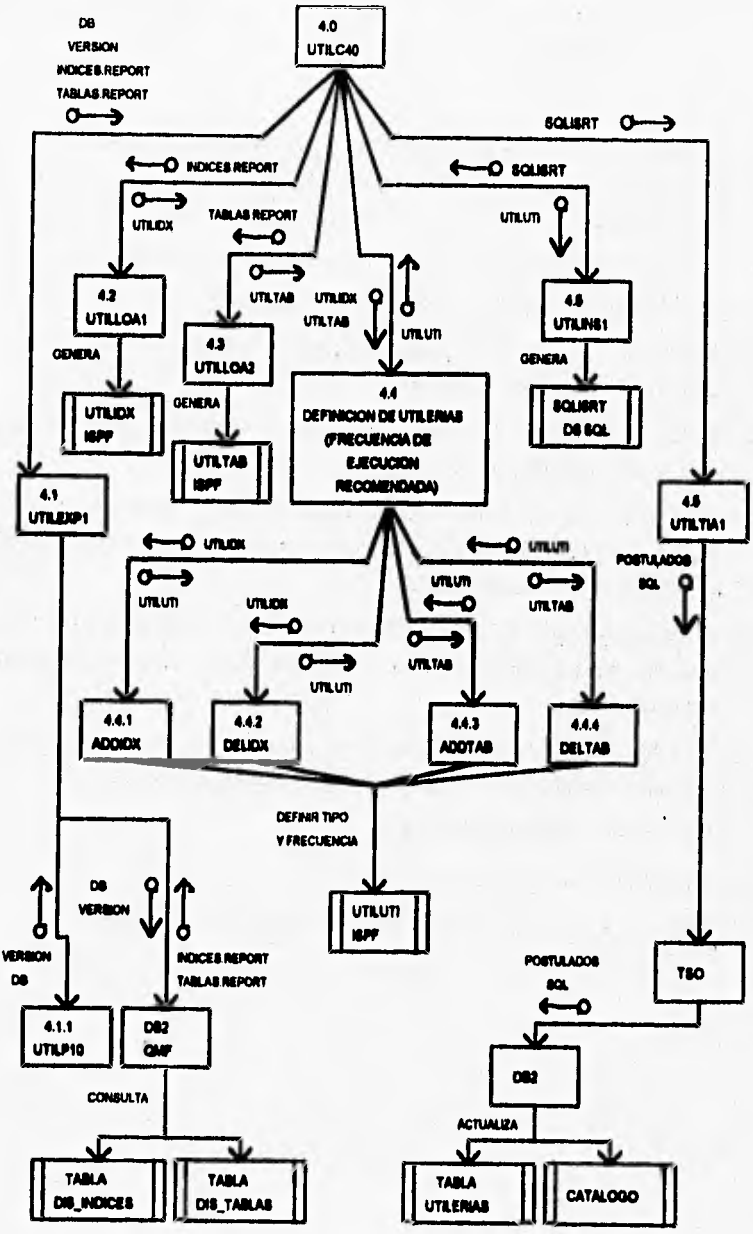
⇒ UTILTIA1 : Se conecta a DB2 vía TSO para que los postulados UPDATE de SQL se ejecuten y actualicen la tabla (UTILERIAS), así como el catálogo de DB2.

⇒ ADDTAB : Añade al archivo UTILUTI parámetros con el tipo de Utilería y frecuencia recomendada para tablas.

⇒ DELTAB : Borra al archivo UTILUTI parámetros con el tipo de Utilería y frecuencia recomendada para tablas.

⇒ ADDIDX : Añade al archivo UTILUTI parámetros con el tipo de Utilería y frecuencia recomendada para índices.

⇒ DELIDX : Borra al archivo UTILUTI parámetros con el tipo de Utilería y frecuencia recomendada para índices.


4.0
UTILC40
DB
VERSION
INDICES.REPORT
TABLAS.REPORT
SQLISRT
INDICES REPORT
SQLISRT
UTILIDX
TABLAS REPORT
UTILUTI
UTILTAB
UTILIDX
UTILTAB
UTILUTI
4.2
UTILLOA1
4.3
UTILLOA2
4.4
DEFINICION DE UTILERIAS
(FRECUENCIA DE
EJECUCION
RECOMENDADA)
4.5
UTILINS1
GENERA
GENERA
GENERA
UTILIDX
ISPF
UTILTAB
ISPF
SQLISRT
DB SQL
4.1
UTILEXP1
4.6
UTILTIA1
UTILIDX
UTILUTI
UTILUTI
UTILTAB
POSTULADOS
SQL
UTILIDX
UTILUTI
UTILTAB
UTILUTI
4.4.1
ADDIDX
4.4.2
DELIDX
4.4.3
ADDTAB
4.4.4
DELTAB
DEFINIR TIPO
Y FRECUENCIA
UTILUTI
ISPF
DB
VERSION
VERSION
DB
INDICES REPORT
TABLAS REPORT
POSTULADOS
SQL
TSO
4.1.1
UTILP10
DB2
QMF
DB2
CONSULTA
ACTUALIZA
TABLA
DB_INDICES
TABLA
DB_TABLAS
TABLA
UTILERIAS
CATALOGO

4.5.5 GENERACION DEL JCL (PROGRAMA) PARA UTILERIAS

⇒ UTILC50 : A partir de la tabla denominada UTILERIAS, generada en el paso previo, se generan diferentes jobs con el JCL y postulados de ejecución para los objetos de una Database/Versión, de acuerdo con la definición almacenada en esa tabla.

⇒ UTILEXP1 : Exporta las tablas DIS_TABLAS y DIS_INDICES en formato de reporte de QMF para ser cargadas posteriormente como tablas ISPF.

⇒ UTILLOA1 : A partir de un reporte exportado de la tabla DIS_INDICES, carga una tabla de ISPF denominada UTILIDX.

⇒ UTILLOA2 : A partir de un reporte exportado de la tabla DIS_TABLAS, carga una tabla de ISPF denominada UTILTAB.

⇒ UTILLOA3 : Exporta la tabla UTILERIAS en formato de reporte de QMF para ser cargada como tabla ISPF. A partir del reporte exportado de la tabla UTILERIAS, carga una tabla ISPF denominada UTILUTI.

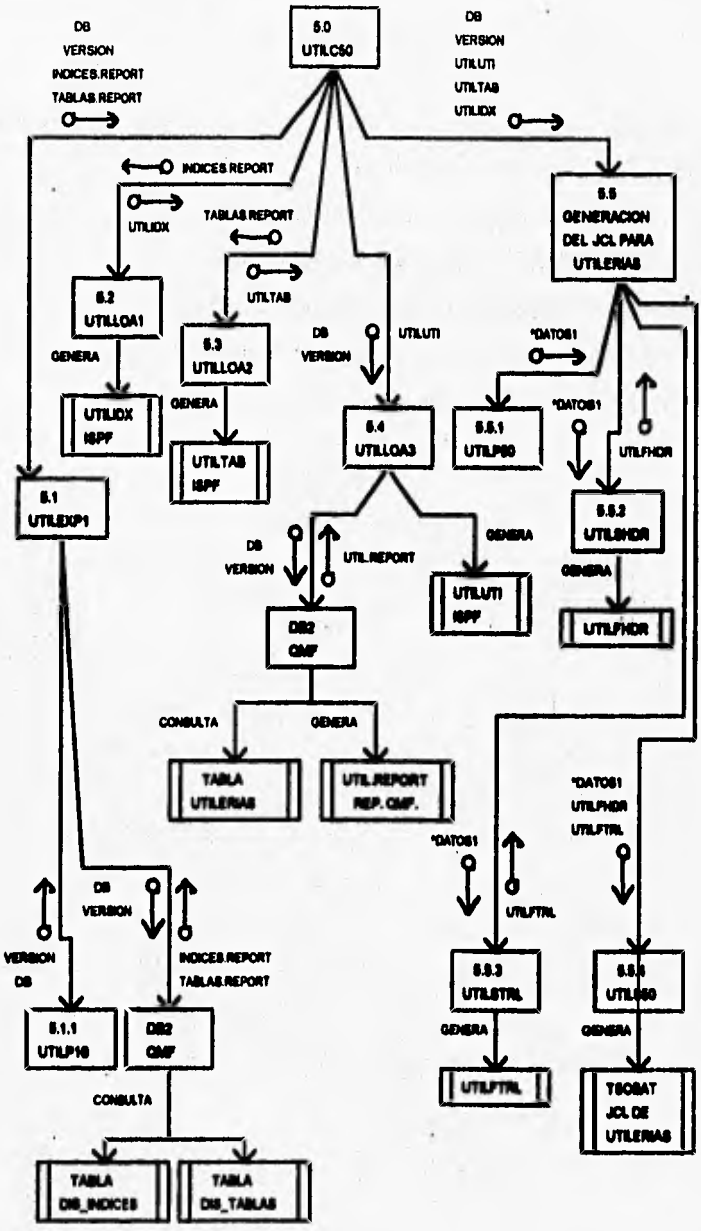
⇒ GENERACION DEL JCL PARA UTILERIAS : Es una interfaz entre los archivos ISPF y los módulos encargados de generar el JCL para utilerías, además de ser un panel de obtención de datos.

⇒ UTILP50 : Panel de obtención de datos (nombre del subsistema, nombre de la aplicación, tempda, workda, tape, tipo de DASD para áreas de trabajo).

⇒ UTILSHDR : Establece el inicio del job.

⇒ UTILSTRL : Establece el final del job.

⇒ UTILS50 : Genera el archivo TSOBAT que contiene el JCL de utilerías.


5.0
UTILC50
DB
VERSION
INDICES.REPORT
TABLAS.REPORT
DB
VERSION
UTILUTI
UTILTAB
UTILIDX
INDICES.REPORT
UTILIDX
TABLAS.REPORT
UTILTAB
5.5
GENERACION
DEL JCL PARA
UTILERIAS
5.2
UTILLOA1
5.3
UTILLOA2
DB
VERSION
UTILUTI
*DATOS1
GENERA
GENERA
UTILIDX
ISPF
UTILTAB
ISPF
5.4
UTILLOA3
5.5.1
UTILP60
*DATOS1
UTILFHDR
5.1
UTILEXP1
5.5.2
UTILSHDR
DB
VERSION
UTIL.REPORT
GENERA
GENERA
UTILUTI
ISPF
UTILFHDR
DB2
QMF
CONSULTA
GENERA
TABLA
UTILERIAS
UTIL.REPORT
REP. QMF.
*DATOS1
UTILFHDR
UTILFTRL
*DATOS1
UTILFTRL
DB
VERSION
VERSION
DB
INDICES.REPORT
TABLAS.REPORT
5.5.3
UTILSTRL
5.5.4
UTILJ60
5.1.1
UTILP10
DB2
QMF
GENERA
GENERA
CONSULTA
UTILFTRL
TSOBAT
JCL DE
UTILERIAS
TABLA
DB_INDICES
TABLA
DB_TABLAS

4.5.6 CONSULTA A TABLAS AUXILIARES

⇒ **UTILP60 : Panel que selecciona la tabla auxiliar (DIS_TABLAS, DIS_INDICES o UTILERIAS) para realizar su consulta.**

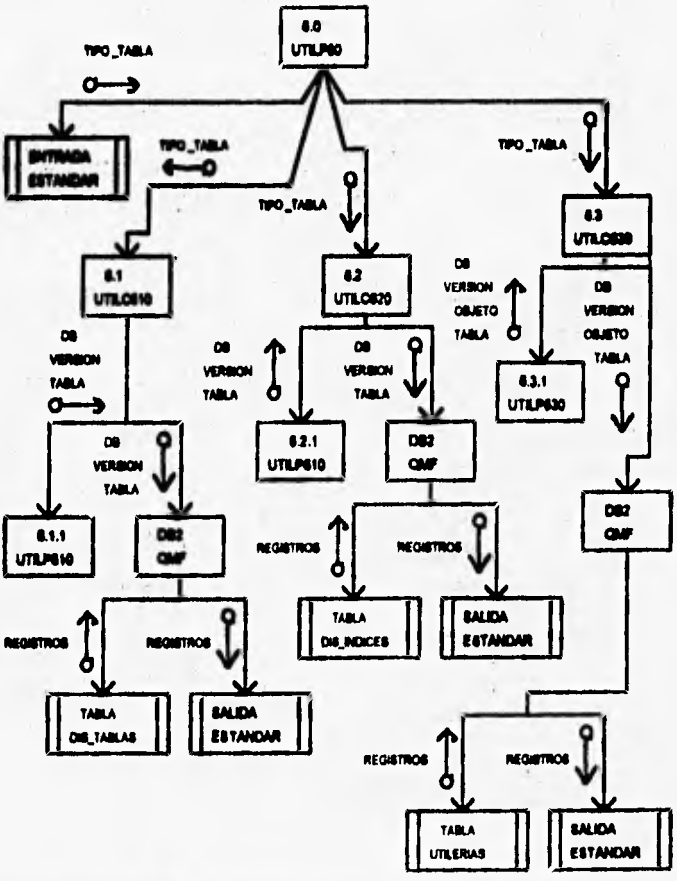
⇒ **UTILC610 : Módulo que consulta la tabla DIS_TABLAS.**

⇒ **UTILP610: Obtiene los valores de DB, versión y tabla.**

⇒ **UTILC620 : Módulo que consulta la tabla DIS_INDICES.**

⇒ **UTILC630 : Módulo que consulta la tabla UTILERIAS.**

⇒ **UTILP630 : Obtiene los valores de DB, versión, tipo de objeto y tabla.**



4.5.6 CONSULTA A TABLAS AUXILIARES

⇒ UTILP60 : Panel que selecciona la tabla auxiliar (DIS_TABLAS, DIS_INDICES o UTILERIAS) para realizar su consulta.

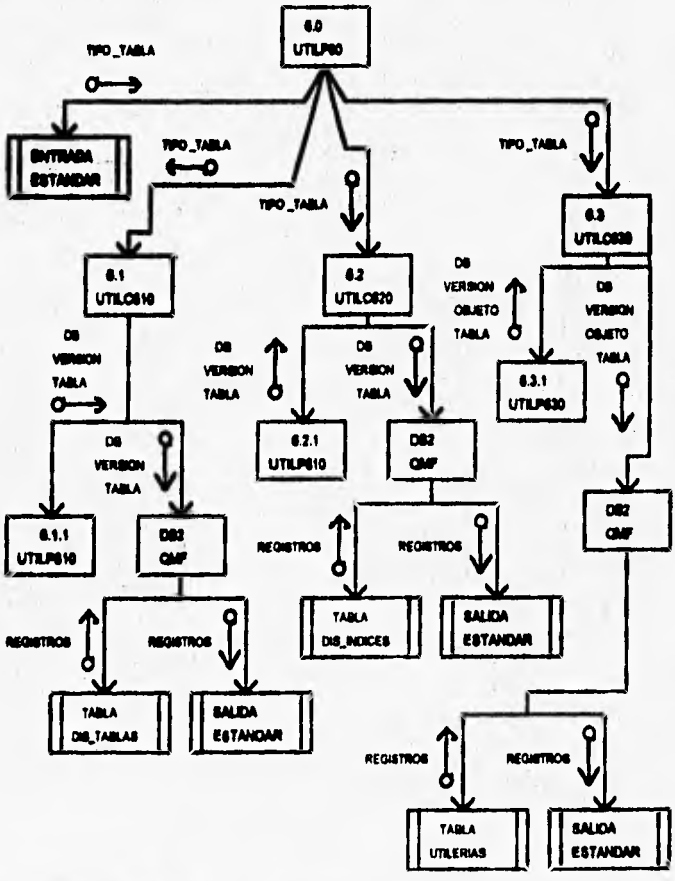
⇒ UTILC610 : Módulo que consulta la tabla DIS_TABLAS.

⇒ UTILP610: Obtiene los valores de DB, versión y tabla.

⇒ UTILC620 : Módulo que consulta la tabla DIS_INDICES.

⇒ UTILC630 : Módulo que consulta la tabla UTILERIAS.

⇒ UTILP630 : Obtiene los valores de DB, versión, tipo de objeto y tabla.



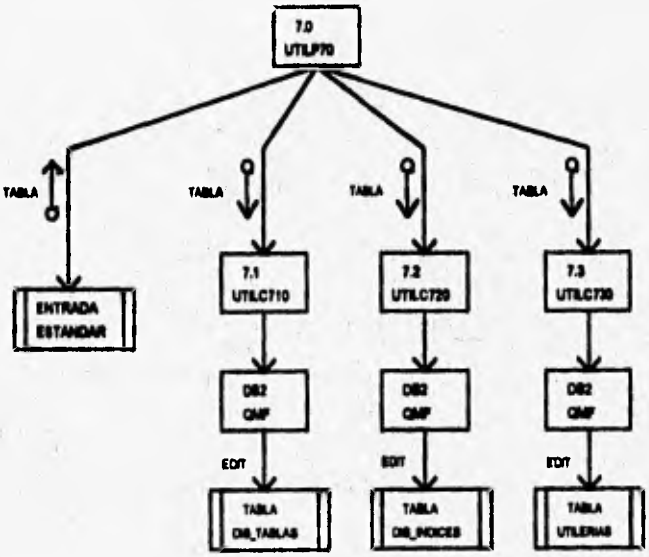
4.5.7 ACTUALIZACION DE TABLAS AUXILIARES

⇒ UTILP70 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database especifica.

⇒ UTILC710 : Actualiza la tabla DIS_TABLAS.

⇒ UTILC720 : Actualiza la tabla DIS_INDICES.

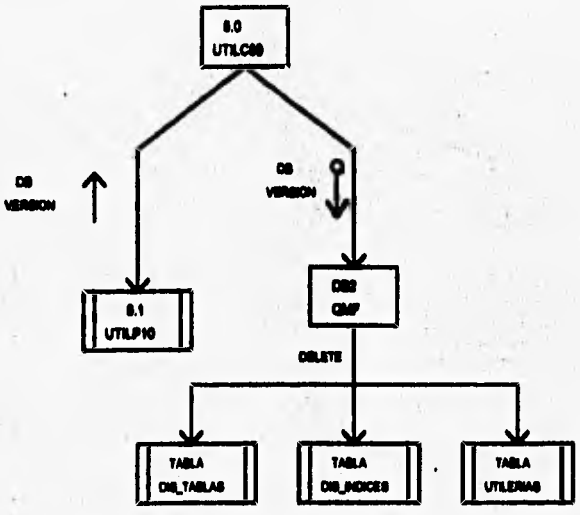
⇒ UTILC730 : Actualiza la tabla DIS_UTILERIAS.



4.5.8 DEPURACION DE TABLAS AUXILIARES

⇒ **UTILC80 : Inicializa las tablas DIS_TABLAS y DIS_INDICES tomando valores del Catálogo de DB2 para una Database específica.**

⇒ **UTILP10 : Obtiene los valores DB y versión.**



4.6 MODULARIDAD E INTERFACES DEL SISTEMA

Los sistemas modulares consisten en unidades claramente definidas y manejables con las interfaces especificadas entre los diversos módulos. El sistema DB2DA es un sistema modular que cumple con los siguientes criterios:

- Cada abstracción de un proceso es un subsistema claramente definido y con el potencial de ser útil para otras aplicaciones.
- Cada función en cada abstracción tiene un propósito específico, claramente definido.
- Las funciones comparten datos globales en forma selectiva; es ciertamente fácil de identificar todas las rutinas que comparten una estructura de datos principal.
- Las funciones que manejan las instancias de un tipo abstracto de datos quedan encapsuladas con la estructura de datos en cuestión.
- Los módulos contienen instrucciones, lógica de proceso y estructuras de datos.
- Los módulos quedan incluidos dentro de un programa principal.
- Los módulos pueden usar a otros módulos.

Esta modularidad ciertamente mejora la claridad del diseño, que a su vez facilita la instrumentación, la depuración, las pruebas, la documentación y el mantenimiento de programación. Una meta fundamental en el diseño de los productos de la programación es la de estructurar al producto de tal forma que el número y la complejidad de las interacciones entre los diversos módulos sea minimizada; un conjunto de procedimientos heurísticos atractivos para llegar a esta meta incluye los conceptos de acoplamiento y cohesión.

La fuerza de acoplamiento entre dos módulos está influida por la complejidad de la interfaz, por el tipo de conexión y por el tipo de comunicación existentes; se obtienen relaciones obvias a partir de una menor complejidad que de grandes y obscuras complejidades. Las conexiones establecidas por medio de la referencia a otros nombres de módulos se encuentran ligeramente acopladas comparadas con las conexiones establecidas por medio de la referencia a los elementos internos de otros módulos. La comunicación entre módulos incluye el pasaje de datos, de elementos, de control (tales como banderas, interruptores, etiquetas). El grado de acoplamiento es menor para la comunicación de datos, mayor los conceptos de control.

El acoplamiento entre módulos del sistema DB2DA puede ser considerado dentro de una escala del 1 (la má fuerte) a 100 (la más débil) en el rango del 70 al 80 de la siguiente forma:

- Existe acoplamiento de control, incluye el pasaje de banderas de control ya sea como parámetros o en forma global, entre módulos de tal forma que un módulo controla la secuencia de proceso de otro. Ejemplos: conexión con QMF, conexión a DB2 via TSO.
- Se identifica acoplamiento por zonas de datos donde los elementos globales son compartidos en forma selectiva entre las diversas rutinas que requieren de los datos. Ejemplos: las variables DB y VERSION, las tablas dis_tablas, dis_indices y dis_utilerías.
- En el acoplamiento de datos incluye el uso de listas de parámetros para pasar a los elementos entre rutinas. Ejemplos: los archivos tablas_report, indices_report, utilidx, utiltab, postulados_sql.

La cohesión interna de un módulo se mide en términos de la fuerza de unión de los elementos dentro del módulo; la cohesión del sistema DB2DA se pretende que ocurra dentro de una escala de 1 (las más débil) a 100 (la más fuerte) en el rango de 80 al 90 en el siguiente orden:

- La cohesión secuencial ocurre cuando la salida de un elemento es la entrada para el siguiente.Ejemplo: si existe DB en catálogo de DB2 entonces ejecutar consulta de SQL a la tabla DIS_TABLAS.
- La cohesión funcional representa un tipo fuerte, y por ende deseable, de amarre de los elementos de un módulo debido a que todos los elementos se encuentran relacionados al desempeño de una sola función. Ejemplos: en las instrucciones del módulo UTILP20, abrir archivo(utilidx), obtener estimados de entrada para tablas, actualizar el archivo utiltab, cerrar archivo.
- La cohesión informacional de elementos en un módulo ocurre cuando el módulo contiene una estructura de datos compleja, así como varias rutinas que manejan dicha estructura; cada rutina del módulo presenta unión funcional; esta cohesión es la realización total de la abstracción de los datos. Ejemplos: los módulos en el pseudocódigo utilc20, utilc30.

La meta de la modularización del sistema DB2DA por el uso de criterios del acoplamiento y cohesión es la de producir un sistema que tenga acoplamiento de zonas de datos y acoplamiento de datos entre los módulos y además que cuenten con cohesión funcional e informacional en los elementos de cada módulo.

4.7.- PSEUDOCODIGO DEL DISEÑO

Se presenta el pseudocódigo del diseño:

```
INICIALIZACION DE LAS TABLAS AUXILIARES

MODULO UTILP10(DB,VERSION)
BEGIN
        LEER DB,VERSION /*DB=DATABASE, VERSION=VERSIÒN DEL DISEÑO FISICO*/
END


MODULO UTILC10
/*-------------------------------------------------*/
/* DESCRIPCION:                                    */
/* INICIALIZA LAS TABLAS DIS_TABLAS Y DIS_INDICES TOMANDO VALORES  */
/* DEL CATALOGO DE DB2 PARA UN DATABASE ESPECIFICO                 */
/*-------------------------------------------------*/
BEGIN
        CALL UTILP10(DB,VERSION)
        CONEXION CON QMF
        IF CONEXION EXITOSA THEN
                EJECUTAR SELECT SQL A CATALOGO DB2 (DB)
                IF DB EXISTE EN CATALOGO DB2 THEN
                        EJECUTAR SELECT SQL A TABLA DIS_TABLAS (DB,VERSION)
                        IF DB Y VERSION EXISTEN EN TABLA DIS_TABLAS THEN
                                SAY "COMBINACION DATABASE Y VERSION YA FUE
                                        INICIALIZADA"
                        ELSE
                                /* SE OBTIENEN VALORES DE INICIALIZACION DEL CATALOGO*/
                                /* DB2: NOMBRE DE LA DATABASE, CREADOR, NOMBRE DE LA*/
                                /* TABLA,VERSION, TABLESPACE, TAMAÑO DEL RENGLON,*/
                                /* INDICE*/
                                OBTENER VALORES DE INICIALIZACION EJECUTANDO SELECT
                                        SQL A CATALOGO DE DB2(DB)
                                /*SE INICIALIZAN TABLAS DIS_TABLAS Y DIS_INDICES*/
                                INSERTAR VALORES DE INICIALIZACION EN TABLAS
                                        DIS_TABLAS Y DIS_INDICES
                                        EJECUTANDO INSERT
                        ENDIF
                ELSE
                        SAY "NO EXISTE DATABASE EN CATALOGO"
                ENDIF
        ENDIF
END
```

```
ACTUALIZACIÓN DE ESTIMADOS PARA TABLAS E INDICES

MODULO UTILEXP1(DB,VERSION,TABLAS.REPORT,INDICES.REPORT)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* EXPORTA LAS TABLAS DIS_TABLAS Y DIS_INDICES EN FORMATO DE REPORTE */
/* DE QMF PARA SER CARGADAS POSTERIORMENTE COMO TABLAS ISPF       */
/*-------------------------------------------------------*/
BEGIN
        CALL UTILP10(DB,VERSION)
        CONEXION CON QMF
        IF CONEXION EXITOSA THEN
                EJECUTAR SELECT SQL A TABLAS DIS_TABLAS Y DIS_INDICES(DB,VERSION)
                IF DB Y VERSION EXISTEN EN TABLA DIS_TABLAS THEN
                        EXPORTAR TABLAS DIS_TABLAS Y DIS_INDICES A FORMATO
                QMF(INDICES.REPORT, TABLAS.REPORT)
                ELSE
                        SAY " NO SE HA INICIALIZADO, LA DATABASE Y VERSION, EJECUTAR EL
                PASO DE LA INICIALIZACION"
                ENDIF
        ELSE
                SAY "NO SE PUDO HACER CONEXION CON QMF"
        ENDIF
END


MODULO UTILLOA1(INDICES.REPORT,UTILIDX)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* A PARTIR DE UN REPORTE EXPORTADO DE LA TABLA DIS_INDICES, CARGA  */
/* UNA TABLA DE ISPF DENOMINADA UTILIDX.                 */
/*-------------------------------------------------------*/
BEGIN
        /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO INDICES.REPORT RENGLON*/
                /*POR RENGLON*/
        ABRIR ARCHIVO(FD1,INDICES.REPORT)
        /*FD2 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILIDX RENGLON POR */
                /*RENGLON*/
        /*UTILIDX ES UNA TABLA ISPF*/
        CREAR ARCHIVO (FD2,UTILIDX) FORMATO ISPF NAMES",
                "IDBNAME IXCREA IXNAME IVERSION ITBNAME ILONG ISTOGRP",
                "ICLUST IUNQ IPART IFULLCRD INLEAF INLEVELS ",
                "IQTYTOT IFRIQTY ISECQTY IPCTFREE IFREEPAG ICLOSE ",
                "ISUBPAG IFECHA IUSUARIO)"
        DO WHILE FD1 NO SEA FIN DE ARCHIVO
                /*SE BORRAN CARACTERES DEL FORMATO QMF (FD1)*/
                DATOS = DELSTR(FD1,1,16)
                /*SE COMIENZAN A CARGAR DATOS EN FORMATO ISPF
                IDBNAME  = $UBWORD(DATOS,1,1)
                IXCREA   = $UBWORD(DATOS,2,1)
                IXNAME   = $UBWORD(DATOS,3,1)
                IVERSION = $UBWORD(DATOS,4,1)
                ITBNAME  = $UBWORD(DATOS,5,1)
                ILONG    = $UBWORD(DATOS,6,1)
                ISTOGRP  = $UBWORD(DATOS,7,1)
                ICLUST   = $UBWORD(DATOS,8,1)
                IUNQ     = $UBWORD(DATOS,9,1)
                IPART    = $UBWORD(DATOS,10,1)
                IFULLCRD = $UBWORD(DATOS,11,1)
                INLEAF   = $UBWORD(DATOS,12,1)
                INLEVELS = $UBWORD(DATOS,13,1)
                IQTYTOT  = $UBWORD(DATOS,14,1)
                IFRIQTY  = $UBWORD(DATOS,15,1)
                ISECQTY  = $UBWORD(DATOS,16,1)
                IPCTFREE = $UBWORD(DATOS,17,1)
                IFREEPAG = $UBWORD(DATOS,18,1)
                ICLOSE   = $UBWORD(DATOS,19,1)
                ISUBPAG  = $UBWORD(DATOS,20,1)
                IFECHA   = $UBWORD(DATOS,21,1)
                IUSUARIO = $UBWORD(DATOS,22,1)
                /*EL APUNTADOR SE MUEVE AL SIGUIENTE RENGLON*/
```

```
            SKIP FD1 Y FD2
        ENDO
        CERRAR ARCHIVOS
END


MODULO UTILLOA3(TABLAS.REPORT,UTILTAB)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* A PARTIR DE UN REPORTE EXPORTADO DE LA TABLA DIS_TABLAS, CARGA  */
/* UNA TABLA DE ISPF DENOMINADA UTILTAB.                 */
/*-------------------------------------------------------*/
BEGIN
        /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO TABLAS.REPORT RENGLON*/
                /*POR RENGLON*/
        ABRIR ARCHIVO(FD1,TABLAS.REPORT)
        /*FD2 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILTAB RENGLON POR */
                /*RENGLON*/
        /*UTILTAB ES UNA TABLA ISPF*/
        CREAR ARCHIVO (FD2,UTILTAB) FORMATO ISPF NAMES"(
                "DBNAME  TBNAME TBCREA  TBNAME  VERSION  CARD LONG",
                "CREMEN  VOLDIA  FRECLOA  FTELOAD  DB2RI  PERACT",
                "ACTCOM  PERDEP  ISRTMAS  UBOLIN ACTBAT  ACTLIN  CONVIV",
                "STOGRP  TIPTBSP  NPAGES  PARTS  QTYTOT  PRIQTY",
                "SECQTY  PCTFREE  FREEPAG  CLOSE  LOCKSZ  SEGSIZE  IDXMAX",
                "FECHA   USUARIO )"
        DO WHILE FD1 NO SEA FIN DE ARCHIVO
                /*SE BORRAN CARACTERES DEL FORMATO QMF (FD1)*/
                DATOS = DELSTR(FD1,1,16)
                /*SE COMIENZAN A CARGAR DATOS EN FORMATO ISPF
                DBNAME = SUBWORD(DATOS,1,1)
                TBNAME = SUBWORD(DATOS,2,1)
                TBCREA = SUBWORD(DATOS,3,1)
                TBNAME = SUBWORD(DATOS,4,1)
                VERSION = SUBWORD(DATOS,5,1)
                CARD  = SUBWORD(DATOS,6,1)
                LONG  = SUBWORD(DATOS,7,1)
                CREMEN = SUBWORD(DATOS,8,1)
                VOLDIA = SUBWORD(DATOS,9,1)
                FRECLOA = SUBWORD(DATOS,10,1)
                FTELOAD = SUBWORD(DATOS,11,1)
                DB2RI  = SUBWORD(DATOS,12,1)
                PERACT = SUBWORD(DATOS,13,1)
                ACTCOM = SUBWORD(DATOS,14,1)
                PERDEP = SUBWORD(DATOS,15,1)
                ISRTMAS = SUBWORD(DATOS,16,1)
                UBOLIN = SUBWORD(DATOS,17,1)
                ACTBAT = SUBWORD(DATOS,18,1)
                ACTLIN = SUBWORD(DATOS,19,1)
                CONVIV = SUBWORD(DATOS,20,1)
                STOGRP = SUBWORD(DATOS,21,1)
                NPAGES = SUBWORD(DATOS,22,1)
                PARTS  = SUBWORD(DATOS,23,1)
                QTYTOT = SUBWORD(DATOS,24,1)
                PRIQTY = SUBWORD(DATOS,25,1)
                SECQTY = SUBWORD(DATOS,26,1)
                PCTFREE = SUBWORD(DATOS,27,1)
                FREEPAG = SUBWORD(DATOS,28,1)
                SEGSIZE = SUBWORD(DATOS,29,1)
                IDXMAX = SUBWORD(DATOS,30,1)
                FECHA  = SUBWORD(DATOS,31,1)
                USUARIO = SUBWORD(DATOS,32,1)
                /*EL APUNTADOR SE MUEVE AL SIGUIENTE RENGLON*/
                SKIP FD1 Y FD2
        ENDO
        CERRAR ARCHIVOS
END
```

```
MODULO UTILP20(UTILTAB)
/*DESCRIPCION: OBTENER ESTIMADOS PARA TABLAS Y ACTUALIZAR EL ARCHIVO UTILTAB*/
BEGIN
        /*ENTRE LOS ESTIMADOS QUE SE OBTIENEN SON:*/
        /*LONGITUD FISICA DEL RENGLON */
        /*CANTIDAD DE RENGLONES*/
        /*CRECIMIENTO MENSUAL*/
        /*VOLATILIDAD DIARIA*/
        /*FRECUENCIA DE ACTUALIZACION*/
        /*FRECUENCIA DE DEPURACION*/
        /*TIPO DE ACTUALIZACION*/
        /*NOMBRE DEL STORAGE GROUP*/
        ABRIR ARCHIVO(UTILTAB)
        OBTENER ESTIMADOS DE ENTRADA PARA TABLAS
        ACTUALIZAR EL ARCHIVO UTILTAB
        CERRAR ARCHIVO
END


MODULO UTILP21(UTILIDX)
/*DESCRIPCION: OBTENER ESTIMADOS PARA INDICES Y ACTUALIZAR EL ARCHIVO UTILIDX*/
BEGIN
        /*ENTRE LOS ESTIMADOS QUE SE OBTIENEN SON:*/
        /*LONGITUD FISICA DE LA LLAVE DEL INDICE */
        /*CANTIDAD DE VALORES DISTINTOS EN EL INDICE*/
        /*INDICE CLUSTER (SI O NO)*/
        /*INDICE UNIQUE (SI O NO)*/
        /*STORAGE GROUP*/
        ABRIR ARCHIVO(UTILIDX)
        OBTENER ESTIMADOS DE ENTRADA PARA INDICE
        ACTUALIZAR EL ARCHIVO UTILIDX
        CERRAR ARCHIVO
END


MODULO UTILUPD1(UTILIDX)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* CON LOS DATOS ALMACENADOS EN LA TABLA DE ISPF DENOMINADA     */
/* UTILIDX SE GENERAN POSTULADOS UPDATE DE SQL PARA             */
/* ACTUALIZAR LA TABLA DIS_INDICES DE DB1.                      */
/*-------------------------------------------------------*/
BEGIN
        ABRIR ARCHIVO(UTILIDX)
        IDENTIFICAR ESTIMADOS DE ENTRADA DE INDICES
        GENERAR CON LOS ESTIMADOS POSTULADOS UPDATE DE SQL
        ESCRIBIR LOS POSTULADOS UPDATE DE SQL EN EL ARCHIVO SQLUPDIX
        CERRAR ARCHIVO
END


MODULO UTILUPD2(UTILTAB)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* CON LOS DATOS ALMACENADOS EN LA TABLA DE ISPF DENOMINADA     */
/* UTILTAB SE GENERAN POSTULADOS UPDATE DE SQL PARA             */
/* ACTUALIZAR LA TABLA DIS_TABLAS DE DB2.                       */
/*-------------------------------------------------------*/
BEGIN
        ABRIR ARCHIVO(UTILTAB)
        IDENTIFICAR ESTIMADOS DE ENTRADA DE TABLAS
        GENERAR CON LOS ESTIMADOS POSTULADOS UPDATE DE SQL
        ESCRIBIR LOS POSTULADOS UPDATE DE SQL EN EL ARCHIVO SQLUPDTB
        CERRAR ARCHIVO
END
```

```
MODULO UTILTIA1(POSTULADOS_SQL)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* EJECUTA LOS POSTULADOS UPDATE                         */
/* QUE ACTUALIZAN LAS TABLAS DB2 A PARTIR DE LAS TABLAS ISPF.    */
/*-------------------------------------------------------*/
BEGIN
        ABRIR ARCHIVO(POSTULADOS_SQL)
        CONECTARSE A DB2 VIA TSO
        EJECUTAR POSTULADOS SQL CONTENIDOS EN POSTULADOS_SQL
        DB2 ACTUALIZA CON LOS POSTULADOS LAS TABLAS (DIS_INDICES, DIS_TABLAS Y
        CATALOGO)
END


MODULO UTILC20
BEGIN
        CALL UTILEXP1(DB,VERSION,INDICES.REPORT,TABLAS.REPORT)
        CALL UTILLOA1(INDICES.REPORT,UTILIDX)
        CALL UTILLOA2(TABLAS.REPORT,UTILTAB)
        CALL UTILP20(UTILTAB)
        CALL UTILP21(UTILIDX)
        CALL UTILUPD1(UTILIDX,SQLUPDIX)
        CALL UTILUPD2(UTILTAB,SQLUPDTB)
        CALL UTILTIA1(SQLUPDTB)
        CALL UTILTIA1(SQLUPDIX)

END


DEFINICION DEL DISEÑO FISICO

MODULO CALCTAB(LON,CAR,PCT,FRE,PTP,PAR,SEG)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* ESTA SUBRUTINA CALCULA EL ESPACIO EN DASD REQUERIDO PARA UN    */
/* TABLESPACE.                                           */
/* ENTRADA:                                              */
/*  LON - LONGITUD FISICA DEL RENGLON SIN CONSIDERAR HEADER     */
/*      (6 BYTES) Y ID (2 BYTES)                         */
/*  CAR - NUMERO DE RENGLONES ESTIMADOS PARA EL TABLESPACE      */
/*  PCT - VALOR DEL PARAMETRO PCTFREE                    */
/*  FRE - VALOR DEL PARAMETRO FREEPAGE                   */
/*  PTP - PORCENTAJE DE ESPACIO QUE DEBE ASIGNARSE AL SECQTY    */
/*      RESPECTO AL PRIQTY                               */
/*  PAR - NUMERO DE PARTICIONES DEL TABLESPACE (0 PARA NO PART)  */
/*  SEG - TAMAÑO DEL SEGMENTO (0 PARA NO SEGMENTADOS)           */
/* SALIDA:                                               */
/*  QTYTOT - ESPACIO TOTAL REQUERIDO POR EL TABLESPACE EN KB     */
/*  NPAGES - PAGINAS TOTALES REQUERIDAS POR EL TABLESPACE       */
/*  PRIQTY - VALOR DEL PARAMETRO PRIQTY PARA UN PARTICION DEL    */
/*       TABLESPACE. SERA IGUAL A QTYTOT PARA TABLESPACES       */
/*       NO PARTICIONADOS. SI SE REQUIEREN 5 O MAS CILINDROS    */
/*       DE ESPACIO, SE REDONDEA A CILINDROS, DE LO CONTRARIO   */
/*       SE REDONDEA A TRACKS.                                  */
/*  SECQTY - VALOR DEL PARAMETRO SECQTY. SE REDONDEA A CILINDROS  */
/*       O PISTAS USANDO EL MISMO CRITERIO QUE EN PRIQTY.       */
/*-------------------------------------------------------*/
BEGIN
        PAGE_SIZE = (4096 - 22) * ((100 - PCT)/100)
        ROWS_PER_PAGE = TRUNC(PAGE_SIZE / LON)
        IF ROWS_PER_PAGE > 127 THEN
                ROWS_PER_PAGE = 127
        ENDIF
        PAGES_USED = 2 + TRUNC((CAR / ROWS_PER_PAGE) + 1)
        IF FREEPAGE > 0 THEN
                TOTAL_PAGES = TRUNC(PAGES_USED * ((1 + FRE) / FRE))
        ELSE
                TOTAL_PAGES = PAGES_USED
        ENDIF
        IF SEG > 0 THEN
```

```
            IF TOTAL_PAGES // SEG > 0 THEN
                TOTAL_PAGES = TOTAL_PAGES + (SEG - (TOTAL_PAGES // SEG))
        ENDIF
        IF PAR = 0 THEN
            NPAGES = TOTAL_PAGES
            QTYTOT = TOTAL_PAGES * 4
            SELECT
                WHEN QTYTOT >= 3600 AND QTYTOT // 720 <> 0 THEN
                    PRIQTY = QTYTOT + (720 - (QTYTOT//720))
                WHEN QTYTOT < 3600 AND QTYTOT // 48 <> 0 THEN
                    PRIQTY = QTYTOT + (48 - (QTYTOT//48))
                OTHERWISE
                    PRIQTY = QTYTOT
            ENDSELECT
            SECQTY = TRUNC(PRIQTY * PTP)
            SELECT
                WHEN SECQTY >= 3600 & SECQTY // 720 <> 0 THEN
                    SECQTY = SECQTY + (720 - (SECQTY//720))
                WHEN SECQTY < 3600 & SECQTY // 48 <> 0 THEN
                    SECQTY = SECQTY + (48 - (SECQTY//48))
                OTHERWISE
                    NOP
            ENDSELECT
        ELSE
            PRIQTY = TRUNC(QTYTOT/PAR)
            SELECT
                WHEN PRIQTY >= 3600 & PRIQTY // 720 <> 0 THEN
                    PRIQTY = PRIQTY + (720 - (PRIQTY//720))
                WHEN PRIQTY < 3600 & PRIQTY // 48 <> 0 THEN
                    PRIQTY = PRIQTY + (48 - (PRIQTY//48))
                OTHERWISE
                    NOP
            ENDSELECT
            SECQTY = TRUNC(PRIQTY * 0.2)
            SELECT
                WHEN SECQTY >= 3600 & SECQTY // 720 <> 0 THEN
                    SECQTY = SECQTY + (720 - (SECQTY//720))
                WHEN SECQTY < 3600 & SECQTY // 48 <> 0 THEN
                    SECQTY = SECQTY + (48 - (SECQTY//48))
                OTHERWISE
                    NOP
            ENDSELECT
        ENDIF
END


MODULO CALCIDX(LON,CAR,PCT,FRE,SUB,DUP,PTP,IPAR,PAR)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* ESTA SUBRUTINA CALCULA EL ESPACIO EN DASD REQUERIDO PARA UN    */
/* INDICE.                                               */
/* ENTRADA:                                              */
/*  LON - LONGITUD DE LA LLAVE DEL INDICE                */
/*  CAR - NUMERO DE RENGLONES ESTIMADOS PARA LA TABLA    */
/*  PCT - VALOR DEL PARAMETRO PCTFREE                    */
/*  FRE - VALOR DEL PARAMETRO FREEPAGE                   */
/*  FRE - VALOR DEL PARAMETRO SUBPAGES                   */
/*  DUP - PROMEDIO DE VALORES DUPLICADOS EN LA LLAVE DEL INDICE.  */
/*      1 PARA INDICES UNIQUE.                           */
/*  PTP - PORCENTAJE DE ESPACIO QUE DEBE ASIGNARSE AL SECQTY    */
/*      RESPECTO AL PRIQTY                               */
/*  IPAR - INDICA SI SE TRATA O NO DE UN INDICE PARTICIONADO    */
/*  PAR - NUMERO DE PARTICIONES DEL TABLESPACE           */
/* SALIDA:                                               */
/*  IQTYTOT - ESPACIO TOTAL REQUERIDO POR EL INDICE EN KB    */
/*  INLEAF  - PAGINAS LEAF TOTALES REQUERIDAS POR EL INDICE    */
/*  INLEVELS- NUMERO DE NIVELES QUE TENDRA EL INDICE         */
/*  IPRIQTY - VALOR DEL PARAMETRO PRIQTY PARA UN PARTICION DEL   */
/*        INDICE.  SERA IGUAL A QTYTOT PARA INDICES          */
/*        NO PARTICIONADOS. SI SE REQUIEREN 5 O MAS CILINDROS */
/*        DE ESPACIO, SE REDONDEA A CILINDROS, DE LO CONTRARIO */
```

```
/*        SE REDONDEA A TRACKS.                    */
/*  ISECQTY - VALOR DEL PARAMETRO SECQTY. SE REDONDEA A CILINDROS */
/*        O PISTAS USANDO EL MISMO CRITERIO QUE EN PRIQTY.    */
/*-----------------------------------------------------------*/
BEGIN
        INLEVELS = 1
        NONLEAF_PAGES_AC = 0
        S = (100-PCT) / 100
        SUBPAGE_SIZE = TRUNC((4067/SUB) - 21)
        IF DUP = 1 THEN
                ENTRIES_PER_PAGE = SUB*TRUNC(S*SUBPAGE_SIZE/(LON+4))
        ELSE
                ENTRIES_PER_PAGE = DUP*SUB*TRUNC(S*SUBPAGE_SIZE/(LON+(4*DUP)+6))
        EDIF
        IF ENTRIES_PER_PAGE < SUB THEN
                ENTRIES_PER_PAGE = SUB
        ENDIF
        ENTRIES_PER_PAGE = TRUNC(ENTRIES_PER_PAGE)
        LEAF_PAGES = TRUNC(CAR/ENTRIES_PER_PAGE)+1)
        IF LEAF_PAGES > 1 THEN
                ENTRIES_PER_NONLEAF = TRUNC((S * 4047) / (LON + 3))
                INLEVELS = INLEVELS + 1
                NONLEAF_PAGES = TRUNC((LEAF_PAGES/ENTRIES_PER_NONLEAF)+1)
                NONLEAF_PAGES_AC = NONLEAF_PAGES + NONLEAF_PAGES_AC
                DO WHILE NONLEAF_PAGES > 1
                        INLEVELS = INLEVELS + 1
                        NONLEAF_PAGES =
                                TRUNC((NONLEAF_PAGES/ENTRIES_PER_NONLEAF)+1)
                        NONLEAF_PAGES_AC = NONLEAF_PAGES + NONLEAF_PAGES_AC
                ENDDO
        ENDIF
        SUB_TOTAL = LEAF_PAGES + NONLEAF_PAGES_AC
        IF FREEPAGE > 0 THEN
                TOTAL_PAGES = TRUNC(SUB_TOTAL * ((1 + FRE) / FRE))
        ELSE
                TOTAL_PAGES = SUB_TOTAL
        ENDIF
        TOTAL_PAGES = TOTAL_PAGES + 2
        INLEAF = LEAF_PAGES
        IQTYTOT = TOTAL_PAGES * 4
        IF IPAR = 'NO' THEN
                SELECT
                        WHEN IQTYTOT >= 3600 & IQTYTOT // 720 <> 0 THEN
                                IPRIQTY = IQTYTOT + (720 - (IQTYTOT//720))
                        WHEN IQTYTOT < 3600 & IQTYTOT // 48 <> 0 THEN
                                IPRIQTY = IQTYTOT + (48 - (IQTYTOT//48))
                        OTHERWISE
                                IPRIQTY = IQTYTOT
                ENDSELECT
                ISECQTY = TRUNC(IPRIQTY * PTP)
                SELECT
                        WHEN ISECQTY >= 3600 & ISECQTY // 720 <> 0 THEN
                                ISECQTY = ISECQTY + (720 - (ISECQTY//720))
                        WHEN ISECQTY < 3600 & ISECQTY // 48 <> 0 THEN
                                ISECQTY = ISECQTY + (48 - (ISECQTY//48))
                        OTHERWISE
                                NOP
                ENDSELECT
        ELSE
                IPRIQTY = TRUNC(IQTYTOT/PAR)
                SELECT
                        WHEN IPRIQTY >= 3600 & IPRIQTY // 720 <> 0 THEN
                                IPRIQTY = IPRIQTY + (720 - (IPRIQTY//720))
                        WHEN IPRIQTY < 3600 & IPRIQTY // 48 <> 0 THEN
                                IPRIQTY = IPRIQTY + (48 - (IPRIQTY//48))
                        OTHERWISE
                                NOP
                ENDSELECT
                ISECQTY = TRUNC(IPRIQTY * 0.25)
                SELECT
                        WHEN ISECQTY >= 3600 & ISECQTY // 720 <> 0 THEN
```

```
/*          SE REDONDEA A TRACKS.                    */
/*  ISECQTY - VALOR DEL PARAMETRO SECQTY. SE REDONDEA A CILINDROS */
/*          O PISTAS USANDO EL MISMO CRITERIO QUE EN PRIQTY.    */
/*-------------------------------------------------------*/
BEGIN
        INLEVELS = 1
        NONLEAF_PAGES_AC = 0
        S = (100-PCT) / 100
        SUBPAGE_SIZE = TRUNC((4067/SUB) - 21)
        IF DUP = 1 THEN
                ENTRIES_PER_PAGE = SUB*TRUNC(S*SUBPAGE_SIZE/(LON+4))
        ELSE
                ENTRIES_PER_PAGE = DUP*SUB*TRUNC(S*SUBPAGE_SIZE/(LON+(4*DUP)+6))
        EDIF
        IF ENTRIES_PER_PAGE < SUB THEN
                ENTRIES_PER_PAGE = SUB
        ENDIF
        ENTRIES_PER_PAGE = TRUNC(ENTRIES_PER_PAGE)
        LEAF_PAGES = TRUNC((CAR/ENTRIES_PER_PAGE)+1)
        IF LEAF_PAGES > 1 THEN
                ENTRIES_PER_NONLEAF = TRUNC((S * 4047) / (LON + 3))
                INLEVELS = INLEVELS + 1
                NONLEAF_PAGES = TRUNC((LEAF_PAGES/ENTRIES_PER_NONLEAF)+1)
                NONLEAF_PAGES_AC = NONLEAF_PAGES + NONLEAF_PAGES_AC
                DO WHILE NONLEAF_PAGES > 1
                        INLEVELS = INLEVELS + 1
                        NONLEAF_PAGES =
                                TRUNC((NONLEAF_PAGES/ENTRIES_PER_NONLEAF)+1)
                        NONLEAF_PAGES_AC = NONLEAF_PAGES + NONLEAF_PAGES_AC
                ENDDO
        ENDIF
        SUB_TOTAL = LEAF_PAGES + NONLEAF_PAGES_AC
        IF FREEPAGE > 0 THEN
                TOTAL_PAGES = TRUNC(SUB_TOTAL * ((1 + FRE) / FRE))
        ELSE
                TOTAL_PAGES = SUB_TOTAL
        ENDIF
        TOTAL_PAGES = TOTAL_PAGES + 2
        INLEAF = LEAF_PAGES
        IQTYTOT = TOTAL_PAGES * 4
        IF IPAR = 'NO' THEN
                SELECT
                        WHEN IQTYTOT >= 3600 & IQTYTOT // 720 <> 0 THEN
                                IPRIQTY = IQTYTOT + (720 - (IQTYTOT//720))
                        WHEN IQTYTOT < 3600 & IQTYTOT // 48 <> 0 THEN
                                IPRIQTY = IQTYTOT + (48 - (IQTYTOT//48))
                        OTHERWISE
                                IPRIQTY = IQTYTOT
                ENDSELECT
                ISECQTY = TRUNC(IPRIQTY * PTP)
                SELECT
                        WHEN ISECQTY >= 3600 & ISECQTY // 720 <> 0 THEN
                                ISECQTY = ISECQTY + (720 - (ISECQTY//720))
                        WHEN ISECQTY < 3600 & ISECQTY // 48 <> 0 THEN
                                ISECQTY = ISECQTY + (48 - (ISECQTY//48))
                        OTHERWISE
                                NOP
                ENDSELECT
        ELSE
                IPRIQTY = TRUNC(IQTYTOT/PAR)
                SELECT
                        WHEN IPRIQTY >= 3600 & IPRIQTY // 720 <> 0 THEN
                                IPRIQTY = IPRIQTY + (720 - (IPRIQTY//720))
                        WHEN IPRIQTY < 3600 & IPRIQTY // 48 <> 0 THEN
                                IPRIQTY = IPRIQTY + (48 - (IPRIQTY//48))
                        OTHERWISE
                                NOP
                ENDSELECT
                ISECQTY = TRUNC(IPRIQTY * 0.25)
                SELECT
                        WHEN ISECQTY >= 3600 & ISECQTY // 720 <> 0 THEN
```

```
                                ISECQTY = ISECQTY + (720 - (ISECQTY//720))
                        WHEN ISECQTY < 3600 & ISECQTY // 48 <> 0 THEN
                                ISECQTY = ISECQTY + (48 - (ISECQTY//48))
                        OTHERWISE
                                NOP
                ENDSELECT
        ENDIF
END


MODULO REALIZACION_DEL DISEÑO_FISICO(UTILIDX,UTILTAB)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* A PARTIR DE LOS ESTIMADOS DADOS POR EL USUARIO, SE EFECTUA EL    */
/* DISEÑO FISICO QUE INCLUYE: DECIDIR TIPO DE TABLESPACE, VALORES   */
/* DE LOS PARAMETROS CLOSE, PCTFREE, FREEPAGE, PRIQTY, SECQTY,      */
/* LOCKSIZE, CANTIDAD DE PARTICIONES Y MAXIMO NUMERO DE INDICES     */
/* RECOMENDADO.                                          */
/*-------------------------------------------------------*/
BEGIN
        /* SE INICIA EL DISEÑO FISICO DE TABLAS */
        /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILTAB RENGLON*/
                /*POR RENGLON*/
        ABRIR ARCHIVO(FD1,UTILTAB)
        DO WHILE FD1 NO SEA FIN DE ARCHIVO
                IF CREMEN = 'ALTO' OR CREMEN = 'BAJO' THEN
                        PCTFREE = 5
                        FREEPAG = 0
                        PCT_PRIQTY = 0.10
                        CLOSE = 'YES'
                        LOCKSZ = 'ANY'
                        TIPTBSP = 'SEGM'
                        PARTS = 0
                        SEGSIZE = 64
                        IDXMAX = 3
                        CALL CALCTAB(LONG,CARD,PCTFREE,FREEPAGE,PCT_PRIQTY,PARTS,
                                SEGSIZE)
                        I=64
                        DO WHILE NPAGES / I < 10 AND I>= 8
                                I=I-4
                        ENDDO
                        SEGSIZE = I
                        IF NPAGES > 50000 THEN
                                IDXMAX = 2
                        ENDIF
                        IF NPAGES > 500000 OR NPAGES < 150 THEN
                                IDXMAX = 1
                        ENDIF
                        IF NPAGES <= 20 THEN
                                IDXMAX = 0
                        ENDIF
                        IF NPAGES <= 50000 AND CREMEN = 'ALTO' AND CONVIV = 'NO' THEN
                                PCTFREE = 15
                                FREEPAG = 15
                                PCT_PRIQTY = 0.20
                                IF ISRTMAS = 'SI' THEN
                                        PCTFREE = 0
                                        FREEPAG = 0
                                ENDIF
                                CALL CALCTAB(LONG,CARD,PCTFREE,FREEPAGE,PCT_PRIQTY,
                                        PARTS,SEGSIZE)
                        ENDIF
                        IF NPAGES > 50000 THEN
                                SEGSIZE = 0
                                TIPTBSP = 'PART'
                                PARTS = TRUNC((NPAGES / 50000) + 1)
                                IF PARTS > 64 THEN
                                        PARTS = 64
                                ENDIF
                                CARD_PART = TRUNC((CARD / PARTS) + 0.5)
                                IF CREMEN = 'ALTO' THEN
```

```
                                PCTFREE = 10
                                FREEPAG = 31
                                PCT_PRIQTY = 0.15
                        ENDIF
                        IF ISRTMAS = 'SI' THEN
                                PCTFREE = 0
                                FREEPAG = 0
                        ENDIF
                        CALL CALCTAB(LONG,CARD_PART,PCTFREE,FREEPAGE,
                                PCT_PRIQTY,PARTS,RECSIZE)
                ENDIF
                IF UBOLIN = 'SI' THEN
                        CLOSE = 'NO'
                ENDIF
                IF VOLDIA = 'ALTA' AND UBOLIN = 'SI' THEN
                        LOCKSZ = 'PAGE'
                ENDIF
                IF CONVIV = 'SI' THEN
                        LOCKSZ = 'PAGE'
                ENDIF
                ACTUALIZAR EL ARCHIVO(UTILTAB)
        END
        SKIP FD1
ENDDO
CERRAR ARCHIVO (UTILTAB)
/* TERMINA EL DISEÑO FISICO DE TABLAS */

/* SE INICIA EL DISEÑO FISICO DE INDICES */
ABRIR ARCHIVO(FD1,UTILTAB)
ABRIR ARCHIVO(FD2,UTILIDX)
TBCREA = IXCREA
TBNAME = ITBNAME
BUSCAR EN UTILTAB (TBNAME)
DO WHILE FD2 NO SEA FIN DEARCHIVO
        IF (IUNQ = 'SI' AND (CREMEN = 'ALTO' OR CREMEN = 'BAJO')) OR
           (IUNQ = 'NO' AND (CREMEN = 'ALTO' OR CREMEN = 'BAJO')) THEN
                IPCTFREE = 10
                IFREEPAG = 0
                IPCT_PRIQTY = 0.10
                ICLOSE = 'YES'
                ISUBPAG = 4
                IPART = 'NO'
                IF ICLUST = 'SI' & TIPTBSP = 'PART' THEN
                        IPART = 'SI'
                ENDIF
                DUPL = IFULLCRD/CARD
                IF DUPL < 1 THEN
                        DUPL = 1
                ENDIF
                IF DUPL > 255 THEN
                        DUPL = 255
                ENDIF
                CALL CALCIDX(ILONG,CARD,IPCTFREE,IFREEPAG,ISUBPAG,
                        DUPL,IPCT_PRIQTY,IPART,PARTS)
                IF NPAGES <= 50000 AND CREMEN = 'ALTO' AND CONVIV = 'NO' THEN
                        IPCTFREE = 20
                        IFREEPAG = 7
                        ISUBPAG = 1
                        IPCT_PRIQTY = 0.20
                        IF ISRTMAS = 'SI' THEN
                                IPCTFREE = 0
                                IFREEPAG = 0
                        ENDIF
                        CALL CALCIDX(ILONG,CARD,IPCTFREE,IFREEPAG,ISUBPAG,
                                DUPL,IPCT_PRIQTY,IPART,PARTS)
                ENDIF
                IF NPAGES > 50000 THEN
                        IF  CREMEN = 'ALTO' THEN
                                IPCTFREE = 15
                                IFREEPAG = 15
                                ISUBPAG = 1
```

```
                                        IPCT_PRIQTY = 0.15
                                ENDIF
                                IF ISRTMAS = 'SI' THEN
                                        IPCTFREE = 0
                                        IFREEPAG = 0
                                ENDIF
                                CALL CALCIDX(ILONG,CARD,IPCTFREE,IFREEPAG,ISUBPAG,
                                        DUPI,IPCT_PRIQTY,IPART,PARTS)
                        ENDIF
                        IF USOLIN = 'SI' THEN
                                ICLOSE = 'NO'
                        ENDIF
                        ACTUALIZAR EL ARCHIVO (UTILIDX)
                ENDIF
                SKIP FD2
                TBCREA = IXCREA
                TBNAME = ITBNAME
                MOVER AL TOPE DE UTILTAB FD1
                BUSCAR EN UTILTAB (TBNAME)
        ENDDO
        CERRAR ARCHIVOS(UTILIDX Y UTILTAB)
        /* TERMINA EL DISEÑO FISICO DE INDICES */
END


MODULO UTILC30
BEGIN
        CALL UTILEXP1(DB,VERSION,INDICES.REPORT,TABLAS.REPORT)
        CALL UTILLOA1(INDICES.REPORT,UTILIDX)
        CALL UTILLOA2(TABLAS.REPORT,UTILTAB)
        CALL REALIZACION_DEL DISEÑO_FISICO(UTILIDX,UTILTAB)
        CALL UTILUPD1(UTILIDX,SQLUPDIX)
        CALL UTILUPD2(UTILTAB,SQLUPDTB)
        CALL UTILTIA1(SQLUPDTB)
        CALL UTILTIA1(SQLUPDIX)
END


DEFINICION DE UTILERIAS Y FRECUENCIA

MODULO ADDTAB(UTI, FRE, GDG, SHR, PAR)
BEGIN
        UTILID = UTI
        FREC = FRE
        GDGENT = GDG
        SHRLVL = SHR
        IF PAR = 0 THEN
                PART = 0
                AÑADE DATOS EN ARCHIVO(UTILUTI)
        ELSE
                I=1
                DO WHILE I <= PAR
                        PART = I
                        AÑADE DATOS EN ARCHIVO(UTILUTI)
                ENDDO
        ENDIF
END


MODULO DELTAB(UTI, FRE, GDG, SHR, PAR)
BEGIN
        UTILID = UTI
        FREC = FRE
        GDGENT = GDG
        SHRLVL = SHR
        IF PAR = 0 THEN
                PART = 0
                BORRA DATOS EN ARCHIVO(UTILUTI)
        ELSE
                I=1
```

```
                DO WHILE I=I <= PAR
                        PART = I
                        BORRA DATOS EN ARCHIVO(UTILUTI)
                ENDDO
        ENDIF
END


MODULO ADDIDX(UTI, FRE, GDG, SHR, PAR, NPAR)
BEGIN
        UTILID = UTI
        FREC = FRE
        GDGENT = GDG
        SHRLVL = SHR
        IF PAR = 'NO' OR NPAR = 0 THEN
                PART = 0
                AÑADE DATOS EN ARCHIVO(UTILUTI)
        ENDIF
        IF PAR = 'SI' AND NPAR > 0 THEN
                I=1
                DO WHILE I <= NPAR
                        PART = I
                        AÑADE DATOS EN ARCHIVO(UTILUTI)
                ENDDO
        ENDIF
END


MODULO DELIDX(UTI, FRE, GDG, SHR, PAR, NPAR)
BEGIN
        UTILID = UTI
        FREC = FRE
        GDGENT = GDG
        SHRLVL = SHR
        IF PAR = 'NO' OR NPAR = 0 THEN
                PART = 0
                BORRA DATOS EN ARCHIVO(UTILUTI)
        ENDIF
        IF PAR = 'SI' AND NPAR > 0 THEN
                I=1
                DO WHILE I <= NPAR
                        PART = I
                        BORRA DATOS EN ARCHIVO(UTILUTI)
                ENDDO
        ENDIF
END


MODULO DEFINICION_DE UTILERIAS(UTILIDX,UTILTAB,UTILUTI)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* A PARTIR DE LOS ESTIMADOS DADOS POR EL USUARIO, SE EFECTUA LA   */
/* DEFINICION DE UTILERIAS QUE INCLUYE: TIPO DE UTILERIA (RECRO,   */
/* IMAGE COPY FULL, IMAGE COPY INCREMENTAL, RUNSTATS, ETC.) Y      */
/* FRECUENCIA DE EJECUCION RECOMENDADA.                  */
/*-------------------------------------------------------*/
BEGIN
        /*SE INICIA LA DEFINICION DE UTILERIAS */
        /*FD3 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILUTI*/
        CREAR ARCHIVO (FD3,UTILUTI) FORMATO ISPF
                "KEYS(CALIF NAME PART UTILID SECUEN)",
                "NAMES(DDNAME  VERSION TIPOOBJ FREC",
                "GRUPO  GDGENT  SHRLVL  PAGTOT PAGPART",
                "SYSUTIP  SYSUTIS  PARTSEC  FECHA  USUARIO)"
        /*FD2 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILIDX */
        ABRIR ARCHIVO (FD2,UTILIDX)
        /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILTAB*/
        ABRIR ARCHIVO (FD1,UTILTAB)
        SYSUTIP = 0
        SYSUTIS = 0
        PARTSEC = 0
```

```
SECUEN = 0
GRUPO = 1
FECHA = DATE(E)
USUARIO = USERID()
DO WHILE FDI NO SEA FIN DE ARCHIVO
        IF CREMEN = 'ALTO' OR CREMEN = 'BAJO' THEN
                TBNAME = TBNAME
                CALIF = DBNAME
                NAME = TSNAME
                TIPOOBJ = 'TBSP'
                PAGTOT = NPAGES
                PAGPART = TRUNC((PRIQTY/4) + 0.5)
                CALL ADDTAB('FIC', 'SEMANAL   ', 5, 'REFER', PARTS)
                CALL ADDTAB('MOD', 'MENSUAL   ', 0, '   ', PARTS)
                CALL ADDTAB('QUI', 'DIARIO    ', 0, '   ', 0)
                CALL ADDTAB('RET', 'MENSUAL   ', 0, '   ', PARTS)
                CALL ADDTAB('STT', 'MENSUAL   ', 0, 'REFER', 0)
                IF PERACT <> 'DIARIO' AND PERACT <> 'SEMANAL'
                        AND NPAGES > 50000 THEN
                        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
                        CALL ADDTAB('RET', 'BIMESTRAL ', 0, '   ', PARTS)
                        CALL DELTAB('QUI', 'DIARIO    ', 0, '   ', PARTS)
                ENDIF
                IF PERACT <> 'DIARIO' AND PERACT <> 'SEMANAL'
                        AND NPAGES <= 50000 THEN
                        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
                        CALL ADDTAB('RET', 'MENSUAL   ', 0, '   ', PARTS)
                        CALL DELTAB('QUI', 'DIARIO    ', 0, '   ', PARTS)
                ENDIF
                IF VOLDIA = 'BAJA' AND NPAGES > 50000 AND PERACT = 'DIARIO' THEN
                        CALL ADDTAB('FIC', 'QUINCENAL ', 5, 'REFER', PARTS)
                        CALL ADDTAB('IIC', 'DIARIO    ', 10, 'REFER', PARTS)
                ENDIF
                IF ACTBAT = 'NO' AND ACTLIN = 'NO' AND PERACT = 'NULO' THEN
                        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
                        CALL DELTAB('RET', 'MENSUAL   ', 0, '   ', PARTS)
                ENDIF
                IF NPAGES <= 50000 AND FTELOAD = 'SI' AND FRECLOA = 'DIARIO' THEN
                        CALL DELTAB('FIC', 'SEMANAL   ', 0, '   ', PARTS)
                        CALL DELTAB('QUI', 'DIARIO    ', 0, '   ', PARTS)
                        CALL DELTAB('MOD', 'MENSUAL   ', 0, '   ', PARTS)
                        CALL DELTAB('RET', 'MENSUAL   ', 0, '   ', PARTS)
                        CALL DELTAB('STT', 'MENSUAL   ', 0, '   ', PARTS)
                ENDIF
                IF NPAGES > 50000 AND FTELOAD = 'SI' AND AND FRECLOA = 'DIARIO'
                        THEN
                        CALL DELTAB('FIC', 'SEMANAL   ', 0, '   ', PARTS)
                        CALL DELTAB('QUI', 'DIARIO    ', 0, '   ', PARTS)
                        CALL DELTAB('MOD', 'MENSUAL   ', 0, '   ', PARTS)
                        CALL DELTAB('RET', 'MENSUAL   ', 0, '   ', PARTS)
                ENDIF
                IF DB2RI = 'SI' THEN
                        CALL ADDTAB('CKD', 'MENSUAL   ', 0, '   ', 0)
                ENDIF
                IF VOLDIA = 'ALTA' AND NPAGES > 50000 AND ACTCOM = 'INSERT' THEN
                        SECUEN = 1
                        CALL ADDTAB('FIC', 'SEMANAL   ', 5, 'REFER', PARTS)
                        SECUEN = 0
                        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
                ENDIF
                IF VOLDIA = 'ALTA' AND NPAGES <= 50000 AND ACTCOM = 'INSERT'
                        THEN
                        SECUEN = 1
                        CALL ADDTAB('FIC', 'DIARIO    ', 5, 'REFER', PARTS)
                        SECUEN = 0
                        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
                        CALL ADDTAB('RET', 'QUINCENAL ', 0, '   ', PARTS)
                ENDIF
                IF ACTBAT = 'SI' AND ACTLIN = 'SI' THEN
                        CALL ADDTAB 'QUI', 'DIARIO DOBLE', 0, '   ', 0
                ENDIF
```

```
IF VOLDIA = 'BAJA' AND NPAGES > 50000 AND PERACT = 'SEMANAL'
        THEN
        CALL ADDTAB('FIC', 'QUINCENAL  ', 5, 'REFER', PARTS)
        CALL ADDTAB('RET', 'BIMESTRAL  ', 0, '  ', PARTS)
ENDIF
IF VOLDIA = 'BAJA' AND NPAGES > 50000
        AND (PERACT = 'QUINCENAL' OR PERACT = 'MENSUAL') THEN
        CALL ADDTAB('FIC', 'MENSUAL   ', 3, 'REFER', PARTS)
        CALL ADDTAB('RET', 'TRIMESTRAL ', 0, '  ', PARTS)
ENDIF
IF CONVIV = 'S' THEN
        UTILID = 'FIC'
        MOVER AL TOPE DE UTILUTI FD3
        BUSCAR EN UTILUTI (CALIF NAME UTILID)
        DO WHILE FD3 NO SEA FIN DE ARCHIVO
                $HRLVL = 'CHANGE'
        ACTUALIZAR ARCHIVO (FD3,UTILUTI)
                BUSCAR EN UTILUTI(CALIF NAME UTILID) FD3
        ENDDO
        UTILID = 'RET'
        MOVER AL TOPE DE UTILUTI FD3
        BUSCAR EN UTILUTI(CALIF NAME UTILID) FD3
        DO WHILE FD3 NO SEA FIN DE ARCHIVO
                $HRLVL = 'CHANGE'
                ACTUALIZAR ARCHIVO (FD3,UTILUTI)
                BUSCAR EN UTILUTI(CALIF NAME UTILID) FD3
        ENDDO
ENDIF
MOVER AL TOPE DE UTILIDX FD2
BUSCAR EN UTILIDX (ITEMNAME) FD2
DO WHILE FD2 NO SEA FIN DE ARCHIVO
        CALIF = DOCREA
        NAME = DONAME
        TIPOOBJ = 'DOC'
        PAGTOT = TRUNC(DQTYTOT/4) + 0.5)
        PAGPART = TRUNC(DPRQTY/4) + 0.5)
        CALL ADDIDX('REF', 'QUINCENAL  ', 0, '  ', IPART, PARTS)
        CALL ADDIDX('RET', 'QUINCENAL  ', 0, 'REFER', IPART, 0)
        IF PERACT <> 'DIARIO' AND PERACT <> 'SEMANAL' THEN
                CALL ADDIDX('REF', 'MENSUAL   ', 0, '  ', IPART, PARTS)
                CALL ADDIDX('RET', 'MENSUAL   ', 0, 'REFER', IPART, 0)
        ENDIF
        IF ACTBAT = 'NO' AND ACTLIN = 'NO' AND PERACT = 'NULO' THEN
                CALL DELIDX('REF', 'QUINCENAL  ', 0, '  ', IPART, PARTS)
        ENDIF
        IF NPAGES <= 50000 AND FTELOAD = 'S'
                AND FRECLOA = 'DIARIO' THEN
                CALL DELIDX('REF', 'QUINCENAL  ', 0, '  ', IPART, PARTS)
                CALL DELIDX('RET', 'QUINCENAL  ', 0, '  ', IPART, 0)
        ENDIF
        IF NPAGES > 50000 AND FTELOAD = 'S' AND FRECLOA = 'DIARIO'
                THEN
                CALL DELIDX('REF', 'QUINCENAL  ', 0, '  ', IPART, PARTS)
        ENDIF
        IF VOLDIA = 'ALTA' AND ACTCOM = 'INSERT'
                AND NPAGES <= 50000   THEN
                CALL ADDIDX('REF', 'SEMANAL   ', 0, '  ', IPART, PARTS)
                CALL ADDIDX('RET', 'SEMANAL   ', 0, 'REFER', IPART, 0)
        ENDIF
        IF VOLDIA = 'BAJA' AND NPAGES > 50000
                AND (PERACT = 'SEMANAL' OR PERACT = 'QUINCENAL'
                OR PERACT = 'MENSUAL') THEN
                CALL ADDIDX('REF', 'MENSUAL   ', 0, '  ', IPART, PARTS)
                CALL ADDIDX('RET', 'MENSUAL   ', 0, 'REFER', IPART, 0)
        ENDIF
        IF ISRTMAS = 'S' AND NPAGES <= 50000   THEN
                CALL ADDIDX('REF', 'SEMANAL   ', 0, '  ', IPART, PARTS)
                CALL ADDIDX('RET', 'SEMANAL   ', 0, 'REFER', IPART, 0)
        ENDIF
        IF CONVIV = 'S' THEN
                UTILID = 'RET'
```

```
                                        MOVER AL TOPE DE UTILUTI A FD3
                                        BUSCAR EN UTILUTI (CALIF NAME UTILID) FD3
                                        DO WHILE FD3 NO SEA FIN DE ARCHIVO
                                                SHRLVL = 'CHANGE'
                                                ACTUALIZAR ARCHIVO(FD3,UTILUTI)
                                                BUSCAR EN UTILUTI(CALIF NAME UTILID) FD3
                                        ENDDO
                                ENDIF
                                BUSCAR EN UTILIDX (ITBNAME) FD2
                        ENDDO
                ENDIF
                SKIP FD1
        ENDDO
        CERRAR ARCHIVO(UTILTAB)
        CERRAR ARCHIVO(UTILIDX)
        ORDENAR ARCHIVO(UTILUTI) POR LOS CAMPOS(TIPOOBJ,C,A CALIF,C,A NAME,C,A
                UTILID,C,A PART,N,A)
        CERRAR ARCHIVO(UTILUTI)
        /*TERMINA LA DEFINICION DE UTILERIAS */
END

MODULO UTILINS1(UTILUTI,SQLISRT)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* CON LOS DATOS ALMACENADOS EN LA TABLA DE ISPF DENOMINADA     */
/* UTILUTI SE GENERAN POSTULADOS INSERT DE SQL PARA             */
/* ACTUALIZAR LA TABLA UTILERIAS DE DB2.                        */
/*-------------------------------------------------------*/
BEGIN
        ABRIR ARCHIVO(UTILUTI)
        IDENTIFICAR ESTIMADOS DE ENTRADA PARA UTILERIAS
        GENERAR CON LOS ESTIMADOS POSTULADOS INSERT DE SQL
        ESCRIBIR LOS POSTULADOS UPDATE DE SQL EN EL ARCHIVO SQLISRT
        CERRAR ARCHIVO
END

MODULO UTILC40
BEGIN
        CALL UTILEXP1(DB,VERSION,INDICES.REPORT,TABLAS.REPORT)
        CALL UTILLOA1(INDICES.REPORT,UTILIDX)
        CALL UTILLOAS(TABLAS.REPORT,UTILTAB)
        CALL DEFINICION_DE UTILERIAS(UTILIDX,UTILTAB,UTILUTI)
        CALL UTILINS1(UTILUTI,SQLISRT)
        CALL UTILTIA1(SQLISRT)
END

GENERACION DEL JCL PARA UTILERIAS

MODULO UTILLOAS(DB,VERSION,UTILUTI)
/*-------------------------------------------------------*/
/* DESCRIPCION:                                          */
/* EXPORTA LA TABLAS UTILERIAS EN FORMATO DE REPORTE DE QMF PARA SER */
/* CARGADA COMO TABLA ISPF.                              */
/* A PARTIR DEL REPORTE EXPORTADO DE LA TABLA UTILERIAS, CARGA     */
/* UNA TABLA DE ISPF DENOMINADA UTILUTI.                 */
/*-------------------------------------------------------*/
BEGIN
        CONEXION CON QMF
        IF CONEXION EXITOSA THEN
                EJECUTAR SELECT SQL A TABLA UTILERIAS(DB,VERSION)
                IF DB Y VERSION EXISTEN EN TABLA DIS_TABLAS THEN
                        EXPORTAR TABLA UTILERIAS A FORMATO QMF(UTILERIAS.REPORT)
                ELSE
                        SAY "NO SE HAN DEFINIDO UTILERIAS PARA LA DATABASE"
                        SAY "Y VERSION ESPECIFICADAS"
                        SAY "EJECUTAR EL PASO DE DEFINICION DE UTILERIAS "
                ENDIF
```

```
        ELSE
                SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
        ENDIF
        /*SE CARGAN EN ISPF LOS DATOS DE LA TABLA UTILERIAS */
        /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILUTI RENGLON POR */
                /*RENGLON*/
        /*UTILUTI ES UNA TABLA ISPF*/
        CREAR ARCHIVO(FD1,UTILUTI) FORMATO ISPF
                KEYS(CALIF,NAME, PART, UTILID, SECUEN),
                NAMES(DBNAME, VERSION, TIPOOBJ, FREC ,GRUPO,GDOENT, SHRLVL,PAGTOT,
        PAGPART,SYSUTIP, SYSUTIS,PARTSEC, DISPST,FECHA ,USUARIO)"
        DO WHILE FD1 NO SEA FIN DE ARCHIVO
                /*SE BORRAN CARACTERES DEL FORMATO QMF (FD1)*/
                DATOS = DELSTR(FD1,1,16)
                /*SE COMIENZAN A CARGAR DATOS EN FORMATO ISPF
                CALIF  = SUBWORD(DATOS,1,1)
                NAME   = SUBWORD(DATOS,2,1)
                PART   = SUBWORD(DATOS,3,1)
                UTILID = SUBWORD(DATOS,4,1)
                SECUEN = SUBWORD(DATOS,5,1)
                DBNAME = SUBWORD(DATOS,6,1)
                VERSION = SUBWORD(DATOS,7,1)
                TIPOOBJ = SUBWORD(DATOS,9,1)
                FREC   = SUBWORD(DATOS,9,1)
                GRUPO  = SUBWORD(DATOS,10,1)
                GDOENT = SUBWORD(DATOS,11,1)
                SHRLVL = SUBWORD(DATOS,12,1)
                PAGTOT = SUBWORD(DATOS,13,1)
                PAGPART = SUBWORD(DATOS,14,1)
                FECHA  = SUBWORD(DATOS,15,1)
                USUARIO = SUBWORD(DATOS,16,1)
                SYSUTIP = 0
                SYSUTIS = 0
                PARTSEC = 0
                DISPST = 0
                /*EL APUNTADOR SE MUEVE AL SIGUIENTE RENGLON*/
                SKIP FD1
        ENDDO
        CERRAR ARCHIVO
END


MODULO UTILPS0(*DATOS1)
BEGIN
        /*SSN=    ES EL SUBSISTEMA*/
        /*APLIC= ES EL NOMBRE DE LA PALICACION*/
        /*TEMPDA=         UNIDAD PARA AREAS DE TRABAJO */
        /*WORKDA=         UNIDAD PARA AREAS DE SORT */
        /*TAPE=  UNIDAD DE CINTA */
        /*DASD=  ES TIPO DE DASD PARA AREAS DE TRABAJO */
        /*SE INICIALIZAN VARIABLES*/
        SSN = 'DB2D'
        TEMPDA = 'TMPDA'
        WORKDA = 'WRKDA'
        TAPE = 'TAPE80'
        DASD = '3390'
        /*SE LEEN AQUELLAS VARIABLES QUE SE QUIERAN CAMBIAR*/
        LEER SSN,APLIC,TEMPDA,WORKDA,TAPE,DASD
        *DATOS1=(SSN,APLIC,TEMPDA,WORKDA,TAPE,DASD)
END
```

```
MODULO UTILSHDR(*DATOS1,UTILFHDR)
/*-------------------------------------------------------*/
/* ESTABLECE EL INICIO DE JOB */
/* PARAMETROS SIMBOLICOS: */
/* PARMS= BIBLIOTECA DE PARAMETROS */
/* CLASEA= CLASE DE SALIDA DE MENSAJES */
/* CLASE9= CLASE DE SALIDA DE DUMPS */
/* GEN= VERSION A CREAR DEL GDG */
/* PREFIX1= PREFIJO DE ARCHIVOS */
/*-------------------------------------------------------*/
BEGIN
    /*CON LOS PARAMETROS SIMBOLICOS Y CON *DATOS1 SE ESTABLECE INICIO DE JOB*/
    CREAR ARCHIVO(UTILFHDR)
    ESTABLECER_INICIO_DE_JOB(UTILFHDR,*DATOS1,PARAMETROS SIMBOLICOS)
    CERRAR ARCHIVO
END

MODULO UTILSTRL(*DATOS1,UTILFTRL)
/*SE ESTABLECE EL FIN DEL JOB*/
BEGIN
    CREAR ARCHIVO(UTILFTRL)
    ESTABLECER_FIN_DE_JOB(UTILFTRL)
    CERRAR ARCHIVO
END

MODULO GENERACION_DEL_JCL(UTILTAB,UTILIDX,UTILUTI,DB,VERSION,*DATOS1,UTILFHDR,UTILFTRL)
/*-------------------------------------------------------*/
/* DESCRIPCION: */
/* A PARTIR DE LA TABLA DENOMINADA UTILERIAS, GENERADA EN EL PASO */
/* PREVIO, SE GENERAN DIFERENTES JOBS CON EL JCL Y POSTULADOS DE */
/* EJECUCION PARA LOS OBJETOS DE UN DATABASE/VERSION, DE ACUERDO */
/* CON LA DEFINICION ALMACENADA EN ESA TABLA. */
/*-------------------------------------------------------*/
BEGIN
    DBSAVE = DB
    VERSAVE = VERSION
    SYSUTIP = 0
    CALL UTILP50(*DATOS1)
    SELECT
        WHEN DASD = 3380 THEN
            RECXTRK = 10
            TRKXCYL = 15
        WHEN DASD = 3390 THEN
            RECXTRK = 12
            TRKXCYL = 15
        OTHERWISE
        RECXTRK = 12
        TRKXCYL = 15
    ENDSELECT
    /*FD1 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILTAB RENGLON*/
        /*POR RENGLON*/
    ABRIR ARCHIVO(FD1,UTILTAB)
    /*FD2 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILIDX RENGLON*/
        /*POR RENGLON*/
    ABRIR ARCHIVO(FD2,UTILIDX)
    /*FD3 ES UN APUNTADOR QUE RECORRERA EL ARCHIVO UTILTAB RENGLON*/
        /*POR RENGLON*/
    ABRIR ARCHIVO(FD3,UTILUTI)
    IF FALLA AL ABRIR ARCHIVO THEN
        SAY "NO SE HA GENERADO LA DEFINICION DE UTILERIAS"
        SAY "FAVOR DE EJECUTAR EL PASO 4 ANTES DE CONTINUAR"
        EXIT
    ENDIF
    MOVER AL TOPE DE UTILUTI FD3
    IF DBNAME <> DBSAVE OR VERSION <> VERSAVE THEN
        SAY "NO SE HA GENERADO LA DEFINICION DE UTILERIAS"
        SAY "FAVOR DE EJECUTAR EL PASO 4 ANTES DE CONTINUAR"
        EXIT
    ENDIF
```

```
        DO WHILE FD3 NO SEA FIN DE ARCHIVO
                IF TIPOOBJ = 'TBSP' THEN
                        TSNAME = NAME
                        MOVER AL TOPE DE UTILUTI FD1
                        BUSCAR EN UTILTAB (TSNAME)
                        IF CARD = '' THEN
                                CARD = 0
                        IF PARTS = 0 THEN
                                DISPST = TRUNC(CARD / 5)
                        ELSE
                                DISPST = TRUNC((CARD/PARTS) / 5)
                        ENDIF
                        ITBNAME = TBNAME
                        MOVER AL TOPE DE UTILIDX FD2
                        BUSCAR EN UTILIDX (ITBNAME)
                        DO WHILE FD2 NO SEA FIN DE ARCHIVO
                                SYSUTIP = SYSUTIP + TRUNC(IPRIQTY/4)
                                BUSCAR EN UTILIDX (ITBNAME)
                        END
                ELSE
                        IXNAME = NAME
                        MOVER AL TOPE DE UTILIDX FD2
                        BUSCAR EN UTILIDX (IXNAME)
                        SYSUTIP = TRUNC(IPRIQTY/4)
                ENDIF
                SYSUTIP = TRUNC((SYSUTIP/RECXTRK/TRKXCYL)+1)
                SYSUTIB = TRUNC((SYSUTIP * 0.2)+1)
                PAGPART = TRUNC((PAGPART/RECXTRK/TRKXCYL)+1)
                PARTSEC = TRUNC((PAGPART * 0.2)+1)
                ACTUALIZAR ARCHIVO(FD1,UTILUTI'
                SKIP FD1
        END
        ORDENAR ARCHIVO UTILUTI POR (FREC,C,A TIPOOBJ,C,A CALIF,C,A NAME,C,A,
                UTILID,C,A PART,N,A)
        CERRAR ARCHIVO UTILUTI
        CERRAR ARCHIVO UTILIDX
        CERRAR ARCHIVO UTILTAB
        CALL UTILSHDR('DATOS1',UTILFHDR)
        CALL UTILFHDR('DATOS1',UTILFTRL)
SAY '=== EL JCL RESULTANTE SERA PRODUCIDO POR EL SIGUIENTE ',
    'JOB BATCH ==='
END


MODULO UTILS50('DATOS1',UTILFHDR,UTILFTRL)
BEGIN
        /* EL JCL RESULTANTE SERA PRODUCIDO POR EL SIGUIENTE JOB BATCH */
        CREAR ARCHIVO TSOBAT
        GENERAR_EL_TSOBAT('DATOS1',UTILFHDR,UTILFTRL)
        CERRAR ARCHIVO TSOBAT
END


MODULO UTILC50
BEGIN
        CALL UTILEXP1(DB,VERSION,INDICES.REPORT,TABLAS.REPORT)
        CALL UTILLOA1(INDICES.REPORT,UTILIDX)
        CALL UTILLOA2(TABLAS.REPORT,UTILTAB)
        UTILLOA3(DB,VERSION,UTILUTI)
        GENERACION_DEL_JCL(UTILTAB,UTILIDX,UTILUTI,DB,VERSION,'DATOS1',UTILFHDR,
                UTILFTRL)
        UTILS50('DATOS1',UTILFHDR,UTILFTRL)
END
```

CONSULTA A TABLAS AUXILIARES

```
MODULO UTILP610(DB,VERSION,TABLA)
/* CONSULTAS A LAS TABLAS DB2 DE DISEÑO FISICO */
/* DB          =DATABASE*/
/* VERSION     =NOMBRE DE LA VERSION*/
/*TABLA        =NOMBRE DE LA TABLA*/
BEGIN
        LEER(DB,VERSION,TABLA)
END


MODULO UTILC610(TIPO_TABLA)
/* DESCRIPCION:                          */
/* CONSULTA LA TABLA DIS_TABLAS          */
BEGIN
        IF TIPO_TABLA="T" THEN
                CALL UTILP610(DB,VERSION,TABLA)
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EJECUTAR SELECT SQL A TABLA DIS_TABLAS (DB,VERSION,TABLA)
                        IF SELECT SQL EXITOSO THEN
                                DESPLEGAR_RESULTADO_DEL_SQL_A_PANTALLA
                        ELSE
                                SAY "ERROR AL EJECUTAR SELECT SQL"
                ELSE
                        SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END


MODULO UTILC620(TIPO_TABLA)
/* DESCRIPCION:                          */
/* CONSULTA LA TABLA DIS_INDICES         */
BEGIN
        IF TIPO_TABLA="I" THEN
                CALL UTILP610(DB,VERSION,TABLA)
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EJECUTAR SELECT SQL A TABLA DIS_INDICES(DB,VERSION,TABLA)
                        IF SELECT SQL EXITOSO THEN
                                DESPLEGAR_RESULTADO_DEL_SQL_A_PANTALLA
                        ELSE
                                SAY "ERROR AL EJECUTAR SELECT SQL"
                ELSE
                        SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END


MODULO UTILP630(DB,VERSION,OBJETO,TABLA)
/* DB          =NOMBRE DEL DATABASE*/
/*VERSION      =NOMBRE DE LA VERSION*/
/*OBJETO       =TIPO DE OBJETO*/
/*TABLA        =NOMBRE DE LA TABLA*/
BEGIN
        LEER DB,VERSION,OBJETO,TABLA)
END


MODULO UTILC630(TIPO_TABLA)
/* DESCRIPCION:                          */
/* CONSULTA LA TABLA UTILERIAS           */
BEGIN
        IF TIPO_TABLA="U" THEN
                CALL UTILP630(DB,VERSION,OBJETO,TABLA)
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EJECUTAR SELECT SQL A TABLA UTILERIAS
```

```
                        (DB,VERSION,OBJETO,TABLA)
                    IF SELECT SQL EXITOSO THEN
                        DESPLEGAR_RESULTADO_DEL_SQL_A_PANTALLA
                    ELSE
                        SAY "ERROR AL EJECUTAR SELECT SQL."
                ELSE
                    SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END


MODULO UTILP60
BEGIN
        LEER TIPO_TABLA
        SELECT
                WHEN TIPO_TABLA = "T" THEN
                        /* CONSULTA LA TABLA DE DISEÑO FISICO DE TABLAS*/
                        CALL UTILC610(TIPO_TABLA)
                WHEN TIPO_TABLA ="I" THEN
                        /*CONSULTA LA TABLA DE DISEÑO FISICO DE INDICES*/
                        CALL UTILC620(TIPO_TABLA)
                WHEN TIPO_TABLA="U" THEN
                        /*CONSULTA LA TABLA DE DEFINICION DE UTILERIAS*/
                        CALL UTILC630(TIPO_TABLA)
                ENDSELECT
END


ACTUALIZACION DE TABLAS AUXILIARES

MODULO UTILC710(TABLA)
/* DESCRIPCION:                                  */
/* INVOCA AL TABLE EDITOR DE QMF PARA EDITAR LA TABLA DIS_TABLAS   */
BEGIN
        IF TABLA="DIS_TABLA"
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EDIT DIS_TABLA
                ELSE
                        SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END


MODULO UTILC720(TABLA)
/* DESCRIPCION:                                  */
/* INVOCA AL TABLE EDITOR DE QMF PARA EDITAR LA TABLA DIS_INDICES   */
BEGIN
        IF TABLA="DIS_INDICES"
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EDIT DIS_INDICES
                ELSE
                        SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END
```

```
MODULO UTILC730(TABLA)
/* DESCRIPCION:                                          */
/* INVOCA AL TABLE EDITOR DE QMF PARA EDITAR LA TABLA UTILERIAS   */
BEGIN
        IF TABLA="UTILERIAS"
                CONEXION CON QMF
                IF CONEXION EXITOSA THEN
                        EDIT UTILERIAS
                ELSE
                        SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
                ENDIF
        ENDIF
END


MODULO UTILP70
BEGIN
        LEER TABLA
        /*ACTUALIZACION DE LA TABLA DE DISEÑO FISICO DE TABLAS*/
        CALL UTILC710(TABLA)
        /*ACTUALIZACION DE LA TABLA DE DISEÑO FISICO DE INDICES*/
        CALL UTILC720(TABLA)
        /*ACTUALIZACION DE LA TABLA DE DEFINICION DE UTILERIAS*/
        CALL UTILC730(TABLA)
END


DEPURACION DE TABLAS AUXILIARES


MODULO UTILC80
/* DESCRIPCION:                                          */
/* DEPURA LAS TABLAS DIS_TABLAS, DIS_INDICES Y UTILERIAS PARA UN    */
/* DATABASE Y VERSION DADO POR EL USUARIO.                */
BEGIN
        CALL UTILP10(DB,VERSION)
        CONEXION CON QMF
        IF CONEXION EXITOSA THEN
                EJECUTAR DELETE SQL A TABLAS DIS_TABLAS, DIS_INDICES Y UTILERIAS
                        (DB,VERSION)
        ELSE
                SAY "ERROR AL INTENTAR LA CONEXION CON QMF"
        ENDIF
END
```

4.8.- LENGUAJE DE IMPLEMENTACION (REXX)

Para incrementar la probabilidad de éxito en la implantación del DB2DA, debe utilizarse una herramienta de programación, con la que se realice una conexión transparente con DB2 y otras herramientas que se usan para explotar las bases de datos, como QMF. El sistema que se está utilizando es DB2 y corre bajo el sistema operativo MVS. Por tal motivo se ha seleccionado el lenguaje de programación REXX, para realizar la parte de codificación del DB2DA, ya que además de correr bajo el mismo ambiente puede interconectarse con DB2 vía QMF y esto hace que la información sea fácil de reunir, extraer y manipular para así obtener los resultados que se buscan.

REXX es un lenguaje de programación de propósito general. Cumple con las características de la programación estructurada, es legible y tienen un formato libre. REXX puede ser combinado con comandos de diferentes ambientes.

Ampliando las características del lenguaje REXX y mencionando otras adicionales se tiene que:

- El lenguaje de programación REXX es fácil de leer y escribir porque muchas instrucciones tienen un significado a las palabras en inglés. Las instrucciones de REXX tiene palabras comunes tales como: SAY, IF..THEN..ELSE.., DO..END y EXIT.
- Hay pocas reglas acerca del formato de REXX. No se requiere iniciar una instrucción en una columna en particular, se pueden tener espacios en una línea o saltar líneas enteras, se puede tener una instrucción generada en muchas líneas o tener múltiples instrucciones en una sóla línea, las variables no requieren ser predefinidas y las instrucciones pueden estar en mayúsculas, minúsculas o mezcladas.
- REXX soporta funciones que realizan operaciones tales como procesamiento, búsquedas y comparación para texto y números. Otras funciones tiene la capacidad de formateo y realizar cálculos aritméticos.
- El lenguaje REXX es un lenguaje intérprete. Cuando REXX ejecuta un programa, el procesador de lenguaje directamente procesa cada instrucción.
- Lo más importante de REXX es que tiene un soporte de comunicaciones común y una interfaz de programación común que permiten conectar aplicaciones, sistemas, redes y dispositivos.

CONCLUSIONES

Inicialmente se proporcionó un panorama general sobre la importancia y el objetivo de las bases de datos haciendo énfasis en el modelo relacional. Se trataron los fundamentos del diseño lógico referenciando tópicos como: modelo entidad relación, modelo relacional, restricciones de integridad y normalización. Se explicaron los componentes funcionales, características, utilerías y diccionario de datos de un Sistema Manejador de Base de Datos (DBMS). Este apartado mostró de manera introductoria el diseño físico de las bases de datos.

Posteriormente se expuso la situación actual de la organización de la empresa, así como la problemática detectada al liberar una aplicación de un ambiente de pruebas a un ambiente de producción teniendo su origen en un deficiente diseño físico. Se propuso una metodología de diseño y la automatización de la generación de parámetros del diseño físico de tablas e índices, además de recomendar las utilerías a usar y con que frecuencia ejecutarlas como una solución a la problemática identificada.

Enfocándose al diseño físico en DB2, se analizaron tópicos tales como: los objetos de una base de datos, el método de acceso que se utiliza para la manipulación de la información, las consideraciones del diseño lógico y como se realiza su implemetación física, las decisiones de diseño que se deben tomar en cuenta para tener un buen performance, las utilerías que se aplican en el mantenimiento de las bases de datos.

Finalmente se presentó el diseño estructurado del sistema iniciando con la carta de estructura primaria. Se esquematizaron las tablas auxiliares detallándose la descripción de sus campos y como se hace la conexión con el diccionario de datos de DB2. Se plantearon los diagramas de flujo de datos y el diseño modular estructurado a detalle, junto con las interfaces funcionales. Finalmente se presentó el pseudocódigo del sistema como el producto terminal del diseño.

El desarrollo de la presente tesis nos llevó a concluir que poner énfasis en el diseño de las bases de datos se tienen datos más confiables, disponibles y consistentes, y con ello se logra proteger el activo más importante de la empresa que es la información. Por esto es que la presente ofrece el diseño de un sistema integral automatizado que genera recomendaciones para el diseño físico y da la frecuencia de ejecución de las utilerías de mantenimiento de una base de datos en DB2 para un alto rendimiento, y sobre todo esta tesis es en sí una metodología estándar para la elaboración del diseño de una base de datos relacional.

El contar con una metodología estándar de diseño de bases de datos relacionales se da solución a problemas como: errores humanos en el cálculo de parámetros para el espacio físico, abortación de programas por falta de espacio, elevado número de cambios en la definición de objetos, etc.

Los beneficios obtenidos con la implementación y uso de la metodología son:

- Acelera la liberación de una aplicación de una ambiente de pruebas a un ambiente producción
- Aumenta la estabilidad de un sistema en producción con la adecuada definición de los parámetros de espacio y la programación de las utilerías de mantenimiento y respaldo.
- Mejora el rendimiento para la explotación de las bases de datos.

La funcionalidad, productividad y el aumento del rendimiento en la explotación de las bases de datos son factores adicionales que inducen a preferir la utilización de la metodología.

GLOSARIO.

Actividad del archivo. La razón de actividad de un archivo se define del siguiente modo:

razón de actividad de archivo = (#de registros leídos y usados en la pasada)/(# de registros explorados).

El dispositivo de acceso directo resulta entonces mucho más rápido que la cinta y evita muchas largas operaciones de exploración.

Afinación. Es un reajuste frecuentemente con el objeto de mejorar el rendimiento del sistema.

Algebra relacional . Es un lenguaje de consulta procedural, esto quiere decir que el usuario da instrucciones al sistema para que realice una secuencia de operaciones en la base de datos para calcular el resultado deseado. Consta de un conjunto de operaciones que toman una o dos relaciones como entrada y producen una relación como resultado. Sus operaciones fundamentales son seleccionar, proyectar, producto cartesiano, unión y diferencia de conjuntos. Además de las operaciones fundamentales existen otras operaciones a saber, intersección de conjuntos, producto natural, división y asignación. Estas operaciones se definen en términos de las operaciones fundamentales. Cuando escribimos una expresión en álgebra relacional, damos una secuencia de procedimientos que genera la respuesta a nuestra consulta.

Cálculo relacional de tuplas es un lenguaje de consultas no procedural, es decir, describe la información deseada sin dar un procedimiento específico para obtener esa información. Un ejemplo que muestra el uso del cálculo y el álgebra relacional es la definición y utilización de vistas. Las vistas son mecanismos útiles para simplificar consultas de la base de datos.

Concurrencia : Se presenta cuando múltiples usuarios acceden al mismo tiempo la misma información.

Consistencia : Es cuando se obtiene la misma información por peticiones similares en un momento dado, es decir múltiples usuarios en un mismo lapso de tiempo reciben la misma información.

Datos Integros : Son aquellos que cumplen con las reglas de integridad.

Disponibilidad. Consiste en el efecto del manejo de fallas, tal como lo ve el usuario, a menudo se mide como la disponibilidad, a la fracción de tiempo que el sistema es capaz de producir. Es importante que la mayoría de los sistemas continúen trabajando todo el tiempo posible, es decir, deben tener un alto grado de disponibilidad. Los dispositivos de almacenamiento están sujetos a fallas. Para remediar esta situación se recurre en ocasiones a duplicación de los datos en otro dispositivo para que así siga en línea una copia de los datos originales.

Economía del espacio. Hay una amplia variedad de técnicas, las más conocidas son el emblocamiento de los registros y la compactación de datos, cuyo objeto es el de maximizar la cantidad de datos que se almacenan en los diversos volúmenes. Existen diferentes medios de almacenamiento, estos medios de almacenamiento se clasifican según la velocidad con que puede tenerse acceso a los datos, según el coste por unidad de datos y según la fiabilidad que tienen. Entre los medios con que normalmente se cuentan están:

- Memoria cache. Es la forma de almacenamiento más rápida y costosa. Su tamaño es muy pequeño y el sistema operativo gestiona su utilización.
- Memoria principal. Es el medio de almacenamiento que se emplea para los datos disponibles sobre los cuales se va operar. También las instrucciones de máquina operan sobre la memoria principal. Es demasiada pequeña para almacenar la base de datos completa.
- Almacenamiento en disco. Es el medio de almacenamiento de datos a largo plazo. Lo común es que toda la base de datos esté almacenado en disco. Los datos deben trasladarse del disco a la memoria principal para poder operar sobre de ellos. Después de realizar las operaciones, los datos deben de devolverse al disco. Su acceso es directo porque es posible leer los datos en cualquier orden.
- Almacenamiento en cinta. Se usa principalmente para copias de seguridad (Backups) y datos de archivo. El acceso a los datos es mucho más lento porque la cinta debe leerse secuencialmente desde el principio. Los dispositivos de cinta son menos complejos que los de disco, por lo que son más fiables.

- Una manera de disminuir el número de accesos a disco es mantener en la memoria principal tantos bloques como sea posible. Puesto que no es posible mantener todos los bloques en la memoria principal , se necesita administrar la asignación del espacio disponible en ella para almacenar bloques. Los sistemas de buffers es la parte de la memoria principal que está disponible para almacenar copias de los bloques del disco. El DBMS tiene un administrador de buffer encargado de realizar la tarea de asignar el espacio de buffer.

Eficiencia **(Rendimiento, Performance) : Cuando es indispensable asegurar el acceso en tiempo real a los datos, los usuarios del sistema se interesan al extremo en el tiempo de respuesta del sistema, es decir, se debe medir el tiempo de ejecución de las aplicaciones contra el tiempo de respuesta del sistema. También se refiere a la optimización o ajuste y hasta cambiar fundamentalmente la organización en memoria secundaria despúes que el sistema ha entrado en servicio y se han aclarado suficientemente las pautas de uso. En muchos casos, el uso de las bases de datos evoluciona continuamente, a medida que más personas se van familiarizando con ella y se crean más programas de aplicación.**

Expandibilidad **En algunos sistemas este crecimiento es rápido. Requiere organizaciones de archivos que acepten la expansión y que no esten limitadas por el tamaño de los volúmenes, tamaños de los bloques de índices y otras restricciones.**

Inconsistencia **: Significa que se obtienen diferentes salidas para peticiones similares en un momento dado.**

Indexación. **Muchas consultas hacen referencia a sólo una pequeña parte de los registros. Es ineficiente que el sistema tenga que leer todos los registros. Lo ideal es que el sistema pueda localizar estos registros. Para permitir estas formas de acceso se utiliza la construcción de índices para el diseño físico. Para permitir el acceso aleatorio rápido a los registros de un archivo se utiliza una estructura de índice, cada estructura de índice está asociada con una clave de búsqueda determinada. Si el archivo está ordenado secuencialmente y elegimos incluir varios índices en diferentes claves de búsqueda, el índice cuya clave de búsqueda especifica el orden secuencial del archivo es el índice primario. Los demás se llaman índices secundarios. la clave de búsqueda de un índice primario es normalmente la clave primaria.**

Hay dos tipos de índices que pueden usarse:

- Indice denso. Aparece un registro índice para cada valor de la clave de búsqueda en el archivo. El registro contiene el valor de la clave de búsqueda y un puntero al registro.

- Indice escaso. Se crean registros índices solamente para algunos de los registros. Para localizar un registro, encontramos el registro índice con el valor de la clave de búsqueda más grande que sea menor o igual que el valor de la clave de búsqueda que estamos buscando. Empezamos en el registro al que apunta el registro índice y seguimos los punteros del archivo hasta encontrar el registro deseado.

Interval (Intervalo) : Se refiere a la unión de lo que de otra manera serian muchos archivos separados.

Integridad : Cuando una base de datos incluye información utilizada por muchos usuarios, es importante que no puedan destruirse los datos almacenados, ni las relaciones que existen en grupos de datos (items), para lo cual se aplican ciertas reglas que los datos deben cumplir (dictados por el mundo real).

Procesamiento Los registros que siempre se usan secuencialmente y en el mismo orden, se almacenan de la misma forma ventajosamente. En cambio, los registros de los cuales se pide acceso en cualquier orden, exigen medios de localización más elaborados, y es entonces posible que los métodos de direccionamiento resulten factores dominantes para la distribución física de los registros.

Recuperación (Recover) : Consiste en la capacidad de restaurar la integridad y consistencia después de una falla del sistema o la restauración de transacciones y datos en un momento dado.

Redundancia: Consiste en la repetición de datos, que se hallan simultáneamente almacenados en varios volúmenes con distintas finalidades y también con diferentes fechas de actualización, por lo tanto, la redundancia es costosa porque ocupa más espacio de almacén que el necesario y requiere múltiples operaciones de actualización, esto acarrea con el tiempo a la inconsistencia de los datos.

Seguridad : Se refiere a la protección de los datos contra accesos, modificaciones o pérdidas, ya sea en forma intencional o no intencional.

Tiempo de respuesta. Si un item de datos se utiliza con mucha frecuencia, es por lo general deseable que se le pueda obtener rápidamente a través de un dispositivo adecuado.

Volatibilidad de los datos. Es frecuente en muchos archivos el agregado de registros nuevos y la eliminación de los obsoletos. Los archivos en los que esta actividad es grande se denominan archivos volátiles. Existen varias técnicas para estas operaciones de inserción y eliminación. Cada una de ellas ofrece ventajas (o desventajas), pero todas afectan de algún modo la distribución física óptima de los datos. Las operaciones de inserción se convierten en un factor determinante para el diseño del alamcenamiento cuando los archivos son particularmente volátiles.

APENDICE

A REGLAS DE CALCULO PARA EL ESPACIO.

El cálculo de parámetros del diseño físico se hacen de acuerdo a fórmulas recomendades por el manual de administración de DB2

Las fórmulas utilizadas son las siguientes:

- Número de renglones por página:

$$\frac{((\text{pg size}) - 22) \times ((100 - \text{pctfree}) / 10}{\text{Record Length}}$$

Nota: El número de renglones máximo por página es 127.

- Para el número de Kbytes requerido dentro de la especificación PRIQTY en la clausula USING STOGROUP:

1. # pages requeridas = (total de renglones) / (# renglones por page)
2. # free pages=(# pages requeridas) / FREEPAGE, o 0 si FREEPAGE = 0
3. Total de pages=(# pages requeridas) + (# free pages)
4. # kbytes=(total de pages) x 4 , porque 4k es el tamaño de page.

   o

   =(total de pages) x 32, si 32k es el tamaño de page

- Para el número de TRACKS/CYLINDER requeridos:

  1. # pages requeridas = (total de renglones) / (# renglones por page)
  2. # free pages=(# pages requeridas) / FREEPAGE, o 0 si FREEPAGE = 0
  3. Total de pages=(# pages requeridas) + (# free pages)
  4. # intervalos de control vsam (CIs)=(page size / 4k) x (total de pages)
  5. Usar la siguiente referencia para determinar tracks o cylinders:

TIPO DE DISPOSITIVO	3380	3390
CIs 4k por track	10	12
Tracks por cylinder	15	15

B FORMATOS

Se presentan los siguientes formatos :

- **D821.-** Información general por tabla.
- **D822.-** Relación de utilerías por tabla.
- **D824.-** Información del diseño físico por tabla.

[illegible] Formato

DB21 **INFORMACION GENERAL POR TABLA**

[illegible]	
[illegible]	

[illegible]	
[illegible]	

TABLE			[illegible]		[illegible]						[illegible]				DEPURACIÓN		[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	%	[illegible] %

[illegible]	
SIMP	SIMPLE
S nn	SEGMENTADO Y SEGSIZE
P nn	PARTICIONADO Y NUMERO DE PARTICION

[illegible]	
I	INSERT
D	DELETE
U	UPDATE
L	LOAD

[illegible]			
EVE	EVENTUAL	MES	MENSUAL
DIA	DIARIO	BIM	BIMESTRAL
SEM	SEMANAL	SEI	SEMESTRAL
QUI	QUINCENAL	ANU	ANUAL

Hoja:	
Fecha:	

DB22

RELACION DE UTILERIAS POR TABLA

Aplicación:	
Base de Datos:	

TABLE		TABLESPACE		[illegible] ONLINE		BATCH		IMAGE COPY						QUIESCE	MODIFY	REORG			Run	Re	DSN TIAUL		CHECK	
Name	Card	Name	Tipo	Oper	Freq	Oper	Freq	Full	# Ver	Inc r	# Ver	Merge	# Ver	CE	FY	Table	Index	LOG Yes ó No	Stats	Bind	frec	# Ver	Data	Index

En Modify, normalmente se usará el parámetro AGE = 90.
En Copy, 'Full No' es un respaldo incremental.
'# Ver' se refiere a versiones de GDG's para cada tipo de copia en las diferentes frecuencias que apliquen.

TABLESPACE TYPE	
SIMP	SIMPLE
S SS	SEGMENTADO Y SEGSIZE
P NN	PARTICIONADO Y NUMERO DE PARTICION

FREQUENCY			
EVE	EVENTUAL	MES	MENSUAL
DIA	DIARIO	BIM	BIMESTRAL
SEM	SEMANAL	SEI	SEMESTRAL
QUI	QUINCENAL	ANU	ANUAL

D824

[illegible]	[illegible]
[illegible]	[illegible]

[illegible]	[illegible]
[illegible]	[illegible]

TABLE																			
[illegible]																			

En Herramienta indicar con que instrumento se realizó el cálculo.
En Tablespoon Cyls y en Indexes Cyls indicar el total de espacio requerido.
En Prkply expresar el valor en Cylinders y Kbytes.
En Seqty redondear al multiplo de 4 superior.

[illegible]	
SIMP	SIMPLE
S SB	SECUENTIADO Y SESGUE
P RN	PARTICIONADO Y NUMERO DE PARTICION

BIBLIOGRAFIA

- IBM Database 2 versión 2.
  SQL reference, release 3.
  Tercera Edición.
  Internacional Business Machines Corporation.
  U.S.A, Marzo 1992.

- IBM Database 2 versión 2.
  Administration Guide , Volumen I, II y III.
  Tercera Edición.
  Internacional Business Machines Corporation.
  U.S.A, Marzo 1992.

- IBM Database 2 versión 2.
  Command and Utility reference, release 3.
  Tercera Edición.
  Internacional Business Machines Corporation.
  U.S.A, Marzo 1992.

- Hery F. Kort, Abraham Silberschartz.
  Fundamentos de Bases de Datos.
  Segunda Edición.
  Editorial McGRAW- HILL.
  Mexico D.F, 1989.

- James Martin.
  Organización de Bases de Datos.
  Primera Edición.
  Editorial McGRAW- HILL.
  Mexico D.F, 1989.

- Gio Wiederhold .
  Diseño de Bases de Datos.
  Segunda edición.
  Editorial McGRAW- HILL.
  México D.F, 1993.

- Richard Fairley.
  Ingeniería de Software.
  Primera edición.
  Editorial McGRAW- HILL.
  México D.F, 1990.

- Daniel Martin.
  Advanced Database Techniques.
  Segunda Edición.
  Editorial The MIT Press.
  Cambridge,Massachusetts London, England, 1989.

- Shaku Atre.
  Data base Structured Techniques for Design, Performance and Management, second edition.
  John Wiley & Sons Editors.
  U.S.A, Septiembre 1990.

- Date with Colin J. White.
  A Guide to DB2.
  Tercera Edición.
  Addison-Wesley Publishing Company.
  U.S.A, Septiembre 1990.

- Gabrielle Wiorkowski, David Kull.
  DB2, Design & Develoment Guide.
  Segunda Edición.
  Addison-Wesley Publishing Company.
  U.S.A, Junio 1990.