Guía de Base de Datos Informix

OBJETIVO

Proporcionar una guía que permita a las diferentes áreas de soporte y desarrollo de nuevas tecnologías, tener al alcance las mejores prácticas en el desarrollo e implementación de objetos en las bases de datos.

ALCANCE

Este documento solo está enfocado al SMDB de informix, pero las recomendaciones podrán ser aplicadas a los otros SMDB utilizados en el banco.

CONSIDERACIONES PARA LA CREACIÓN DE OBJETOS

Existen instrucciones para creación de objetos o alteración de los mismos que sólo pueden ser realizados a través del usuario “informix” lo cual debe venir indicado en el C. C.

- **Creación de bases de datos**
- **Creación de Tablas / Vistas**
- **Alter a las tablas**
- **Creación de Triggers**
- **Creacion de Indices**
- **Creación de SPL’s**
- **Creación de vistas**

STORE PROCEDURE LANGUAGE (SPL)

El nombre del STORE PROCEDURE LANGUAJE en la sentencia de creación, debe de estar escritos con minúsculas sin espacio en blanco, el propietario debe de ser “informix”, ejemplo:

CREATE PROCEDURE “informix”. sp_venta_cartera

Siempre que se desee crear o reemplazar un SPL deberá estar construido de manera inicial como:

DROP PROCEDURE “informix”. sp_venta_cartera

(Si se desea remplazar “dropear”, en caso contrario comentar la línea)

SET EXPLAIN ON;

SET OPTIMIZATION HIGH;

Julio, 2023

CREATE PROCEDURE "informix". sp_venta_cartera

Existe la declaración **SET DEBUG FILE**, y **TRACE ON** para rastrear en tiempo de ejecución los que está haciendo el procedimiento almacenado y lo deposita en un archivo de salida, dicha declaración se debe de poner dentro de la declaración **ON EXCEPTION**, FUERA DE ESTA DECLARACIÓN DEBERÁ ESTAR COMENTADA.

Después de la declaración del **ON EXCEPTION**, agregar solo una vez las sentencias:

SET ISOLATION TO DIRTY READ;
SET LOCK MODE TO WAIT 3.

Cuando ejecuten una sentencia en la cláusula **WHERE** no utilizar funciones escalares como son:

ABS, ADD_MONTHS, CASE, CEIL, CONCAT, DATE, DAY, DECODE, FLOOR, LAST_DAY, LEN, LENGTH, LOWER, LPAD, LTRIM, MAX, MIN, MOD, MONTH, MONTHS_BETWEEN, NEXT_DAY, NVL, POW, POWER, RANGE, ROUND, RPAD, RTRIM, SUBSTR, SUBSTRING, TODAY, TRIM, TRUNC, UNITS DAY, UNITS FRACTION, UNITS HOUR, UNITS MINUTE, UNITS SECOND, UPPER, WEEKDAY, YEAR.

Siempre que se mande a ejecutar un SPL deberá indicar al propietario del SPL que en este caso es siempre informix ejemplo:

EXECUTE PROCEDURE "informix".arrpagoint();

En caso de que la ejecución de los SPLs lleven parámetros deberá indicarlo dentro de la sentencia EXECUTE PROCEDURE, si lleva más de un parámetro debe separarlos usando coma (,), si el parámetro es Alfanumérico o fecha en código duro debe encerrarlo entre comillas (" ") si es un parámetro numérico no lleva comillas, ejemplo:

EXECUTE PROCEDURE "informix".sp_rep_cartera_quebrantar("001",6);

Siempre que dentro de un SPL se mande a crear una Tabla Temporal deberá agregar la sentencia **WITH NO LOG** al final ejemplo:

SELECT * FROM customer INTO TEMP cust_temp with no log;
CREATE TEMP TABLE tab2 (fname CHAR(15), lname CHAR(15)) WITH NO LOG;

Al igual que las tablas físicas, las temporales tienen la capacidad de organizar su información al utilizar índices, por lo que se recomienda la creación de 1 a 2 índices por los campos a utilizar en el filtro WHERE donde se utilice la tabla con información menor a 5 millones de registros, ejemplo:

CREATE INDEX "informix".idx_br_burofisicas_clon ON "informix".tmp_ br_burofisicas_clon(numreg);

No utilizar las instrucciones BEGIN – COMMIT, así como el PDQPRIORITY en creación de índices
No se permite la creación de tablas temporales con más de 5 millones de registros, se deben de considerar tablas físicas.

A continuación proporcionaremos algunas consideraciones para ejemplificar los costos de las sentencias

Cuando se manda sin ningún filtro realiza **SEQUENTIAL SCAN**, esto no normal, el problema es el volumen alto de registros a procesar, por eso se traduce en un costo alto.

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 20:28:49)
------
select *
 from sc_maechq

Estimated Cost: 7601872
Estimated # of Rows Returned: 67075328

  1) informix.sc_maechq: SEQUENTIAL SCAN
```

Cuando se realiza la búsqueda sobre la misma tabla por cuenta y existe un índice adecuado el costo es bajo. En este caso el índice es:

CREATE UNIQUE INDEX "informix".idx_maechq1 on "informix".sc_maechq (cuenta,sucursal)

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 20:34:00)
------
select *
  from sc_maechq
  where  cuenta='10000005016'

Estimated Cost: 4
Estimated # of Rows Returned: 2

  1) informix.sc_maechq: INDEX PATH

    (1) Index Name: informix.idx_maechq1
        Index Keys: cuenta sucursal   (Serial, fragments: ALL)
        Lower Index Filter: informix.sc_maechq.cuenta = '10000005016'
```

Diferente filtro mismo resultado, ya que la llave primaria es por el campo de “cuenta”, pero con menor costo (aunque no es muy significativo y utiliza el mismo índice, razón por la cual tendrán que realizar el análisis de la tabla e índices, para escoger el mejor filtro acorde al índice.

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 18:26:54)
------
select * from sc_maechq
where cuenta='10000005016'
and sucursal='0011'

Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) informix.sc_maechq: INDEX PATH

    (1) Index Name: informix.idx_maechq1
        Index Keys: cuenta sucursal   (Serial, fragments: ALL)
        Lower Index Filter: (informix.sc_maechq.cuenta = '10000005016' AND informix.sc_maechq.sucursal = '0011' )
```

Cuando se utiliza la declaración “**ORDER BY**”

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 19:17:33)
------
select unique s.cuenta,s.status_cta,s.producto,s.sdo_actual-s.sdo_retenido-s.sdo_cong,t.motivo
        from sc_maechq as s, sc_tarjeta as t
        where  s.empresa = '001'
      and t.cuenta = s.cuenta
      order by s.cuenta

Estimated Cost: 129782104
Estimated # of Rows Returned: 63175820
Temporary Files Required For: Order By

  1) informix.s: SEQUENTIAL SCAN

        Filters: informix.s.empresa = '001'

  2) informix.t: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.t.cuenta = informix.s.cuenta
```

Cuando no se utiliza la declaración “**ORDER BY**”

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 19:18:26)
------
select unique s.cuenta,s.status_cta,s.producto,s.sdo_actual-s.sdo_retenido-s.sdo_cong,t.motivo
        from sc_maechq as s, sc_tarjeta as t
        where  s.empresa = '001'
      and t.cuenta = s.cuenta

Estimated Cost: 78638536
Estimated # of Rows Returned: 63175820

  1) informix.s: SEQUENTIAL SCAN

       Filters: informix.s.empresa = '001'

  2) informix.t: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.t.cuenta = informix.s.cuenta
```

La información que se trae esta consulta es bastante, aquí se observa dos situaciones, por un lado dada la cantidad de información que extrae realiza un **SEQUENTIAL SCAN** y sin el **ORDER BY** el costo es menor, cuando no sea necesario omitir el **ORDER BY**.

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-09-2023 21:24:17)
------
select unique s.cuenta,s.status_cta,s.producto,s.sdo_actual-s.sdo_retenido-s.sdo_cong,t.motivo
from sc_maechq as s, sc_tarjeta as t
where t.cuenta='10000005016'
and   t.cuenta = s.cuenta
order by s.cuenta


Estimated Cost: 8
Estimated # of Rows Returned: 3
Temporary Files Required For: Order By

  1) informix.t: INDEX PATH

    (1) Index Name: informix.ix_tarjeta4
        Index Keys: cuenta   (Serial, fragments: ALL)
        Lower Index Filter: informix.t.cuenta = '10000005016'

  2) informix.s: INDEX PATH

    (1) Index Name: informix.idx_maechq1
        Index Keys: cuenta sucursal   (Serial, fragments: ALL)
        Lower Index Filter: informix.t.cuenta = informix.s.cuenta
NESTED LOOP JOIN
```

Cuando en la misma sentencia pero pasando el parámetro de la cuenta “10000005016”, el costo es bajo.

Es importante notar que no hay sentencia buena o mala, sino todo depende de la lógica de negocio, sin embargo, es importante hacer hincapié y sensibilizarnos en detectar cuales de estas sentencias son de poca transacción y cuando son para analítica, y de eso dependerá de a qué hora tienes que ejecutarlas.

Julio, 2023

En los ejemplos anteriores nos damos cuenta que puede haber una diversidad de sentencia y para revisar cada sentencia de todos los SPL's, sería una labor titánica, razón por lo cual trataremos en esta guía de ejemplificar y/o clasificar de forma ordenada las posibles sentencia que utilizan dentro de los SPL.

La sentencia JOIN (unir, combinar) de SQL permite combinar registros de una o más tablas en una base de datos. En el Lenguaje de Consultas Estructurado (SQL) hay tres tipos de JOIN: interno, externo y cruzado. El estándar ANSI del SQL especifica cinco tipos de JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER y CROSS. Una tabla puede unirse a sí misma, produciendo una autocombinación, SELF-JOIN.

Todas las explicaciones que están a continuación usan las siguientes dos tablas para ilustrar el efecto de diferentes clases de uniones JOIN.

Empleado	
Apellido	**IDDepartamento**
Andrade	31
Jordán	33
Steinberg	33
Róbinson	34
Zolano	34
Gaspar	36

Departamento	
NombreDepartamento	**IDDepartamento**
Ventas	31
Ingeniería	33
Producción	34
Mercadeo	35

La tabla Empleado contiene los apellidos de los empleados junto al número del departamento al que pertenecen, mientras que la tabla Departamento contiene los nombres de los departamentos de la empresa.

Existen empleados que tienen asignado un número de departamento que no se encuentra en la tabla Departamento (Gaspar). Igualmente, existen departamentos a los cuales no pertenece ningún empleado (Mercadeo). Esto servirá para presentar algunos ejemplos más adelante.

Combinación Interna (INNER JOIN)

Con esta operación cada registro en la tabla A es combinado con los correspondientes de la tabla B que satisfagan las condiciones que se especifiquen en el predicado del JOIN. Cualquier registro de la tabla A o de la tabla B que no tenga uno correspondiente en la otra tabla es excluido, y solo aparecerán los que tengan correspondencia en la otra tabla. Este es el tipo de JOIN más utilizado, por lo que es considerado el tipo de combinación predeterminado.

Es necesario tener especial cuidado cuando se combinan columnas con valores nulos NULL, ya que el valor nulo no se combina con otro valor o con otro nulo, excepto cuando se le agregan predicados tales como IS NULL o IS NOT NULL.

Como ejemplo, la siguiente consulta toma todos los registros de la tabla Empleado y encuentra todas las combinaciones en la tabla Departamento. La sentencia JOIN compara los valores en la columna IDDepartamento en ambas tablas. Cuando no existe esta correspondencia entre algunas combinaciones, estas no se muestran; es decir, que, si el número de departamento de un empleado no coincide con los números de departamento de la tabla Departamento, no se mostrará el empleado con su respectivo departamento en la tabla resultante.
Las dos consultas siguientes son similares y se realizan de manera explícita (A) e implícita (B)

Ejemplo de la sentencia INNER JOIN explícita:

```
Select * From empleado
 INNER JOIN departamento
ON empleado.IDDepartamento=departamento.IDDepartamento
```

Ejemplo de la sentencia INNER JOIN implícita:

```
Select * From empleado, departamento
WHERE  empleado.IDDepartamento = departamento.IDDepartamento
```

	Empleado		Departamento	
	Apellido	**IDDepartamento**	**NombreDepartamento**	**IDDepartamento**
	Zolano	34	Producción	34
	Jordán	33	Ingeniería	33
	Róbinson	34	Producción	34
	Steinberg	33	Ingeniería	33
	Andrade	31	Ventas	31

El empleado Gaspar y el departamento de Mercadeo no son presentados en los resultados ya que ninguno de estos tiene registros correspondientes en la otra tabla. No existe un departamento con número 36 ni existe un empleado con número de departamento 35.

Evidencia donde se puede observar el tipo de unión que realiza “hjoin o hash JOIN” y se trae 5 elementos, y como no tiene índices realiza un SEQUENTIAL SCAN

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 16:50:00)
------
Select * From empleado
 INNER JOIN departamento
ON empleado.IDDepartamento=departamento.IDDepartamento


Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) informix.empleado: SEQUENTIAL SCAN

  2) informix.departamento: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 empleado
  t2                 departamento

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     6          1         6          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     4          1         4          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ------------------------------------------------------------------------------
  hjoin    5          1         4         6         0        00:00.00   4
```

Combinación externa (OUTER JOIN)

Mediante esta operación no se requiere que un registro en una tabla tenga un registro relacionado en la otra tabla. El registro es mantenido en la tabla combinada, aunque no exista el correspondiente en la otra tabla.

Existen tres tipos de combinaciones externas, el Left Join, el Right Join y el Full Join, donde se toman todos los registros de la tabla de la izquierda, o todos los de la tabla derecha, o todos los registros respectivamente.

Left Join

El resultado de esta operación siempre contiene todos los registros de la tabla de la izquierda (la primera tabla que se menciona en la consulta), más los elementos comunes de la tabla de derecha.

retorna un valor nulo NULL en los campos de la tabla derecha cuando no haya correspondencia.

A diferencia del resultado presentado en los ejemplos de combinación interna donde no se mostraba el empleado cuyo departamento no existía, en el siguiente ejemplo se presentarán los empleados con su respectivo departamento, y adicionalmente se presenta un empleado cuyo departamento no existe.

```
SELECT *
 FROM   empleado
      LEFT JOIN departamento
        ON empleado.IDDepartamento = departamento.IDDepartamento
```

	Empleado		Departamento	
	Apellido	**IDDepartamento**	**NombreDepartamento**	**IDDepartamento**
	Jordán	33	Ingeniería	33
	Andrade	31	Ventas	31
	Róbinson	34	Producción	34
	Zolano	34	Producción	34
	Gaspar	36	NULL	NULL
	Steinberg	33	Ingeniería	33

Evidencia del LEFT JOIN

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 16:58:21)
------
SELECT *
 FROM   empleado
        LEFT JOIN departamento
           ON empleado.IDDepartamento = departamento.IDDepartamento


Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) informix.empleado: SEQUENTIAL SCAN

  2) informix.departamento: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 empleado
  t2                 departamento

  type      table   rows_prod  est_rows  rows_scan  time        est_cost
  ------------------------------------------------------------------------
  scan      t1      6          1         6          00:00.00    2

  type      table   rows_prod  est_rows  rows_scan  time        est_cost
  ------------------------------------------------------------------------
  scan      t2      4          1         4          00:00.00    2

  type      rows_prod  est_rows  rows_bld  rows_prb  novrflo  time        est_cost
  ------------------------------------------------------------------------------
  hjoin     6          1         4         6         0        00:00.00    4
```

LEFT JOIN excluyendo la intersección

Si se quieren mostrar solo los registros de la primera tabla que no tengan correspondientes en la segunda, se puede agregar la condición adecuada en la cláusula WHERE. Esto nos dará los empleados que no estén asignados a ningún departamento, que en el diagrama de la derecha se representan en amarillo.

```
SELECT *
 FROM   empleado
      LEFT OUTER JOIN departamento
        ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE  departamento.IDDepartamento IS NULL
```

	Empleado		Departamento	
	Apellido	IDDepartamento	NombreDepartamento	IDDepartamento
	Gaspar	36	NULL	NULL

LEFT JOIN excluyendo la intersección.

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 17:18:09)
------
SELECT *
 FROM   empleado
        LEFT OUTER JOIN departamento
           ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE  departamento.IDDepartamento IS NULL


Estimated Cost: 4
Estimated # of Rows Returned: 1

     1) informix.empleado: SEQUENTIAL SCAN

     2) informix.departamento: SEQUENTIAL SCAN

    ON-Filters:informix.empleado.iddepartamento = informix.departamento.iddepartamento
    DYNAMIC HASH JOIN (LEFT OUTER JOIN)
        Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento

PostJoin-Filters:informix.departamento.iddepartamento IS NULL


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 empleado
  t2                 departamento

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     6          1         6          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     4          1         4          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ------------------------------------------------------------------------------
  hjoin    1          1         4         6         0        00:00.00   4
```

RIGHT OUTER JOIN o RIGHT JOIN

Esta operación es una imagen refleja de la anterior; el resultado de esta operación siempre contiene todos los registros de la tabla de la derecha (la segunda tabla que se menciona en la consulta), independientemente de si existe o no un registro correspondiente en la tabla de la izquierda.

La sentencia RIGHT OUTER JOIN retorna todos los valores de la tabla derecha con los valores de la tabla de la izquierda correspondientes, si los hay, o retorna un valor nulo NULL en los campos de la tabla izquierda cuando no haya correspondencia.

Ejemplo de right join para la combinación externa:

```
SELECT *
 FROM   empleado
      RIGHT OUTER JOIN departamento
        ON empleado.IDDepartamento = departamento.IDDepartamento
```

	Empleado		Departamento	
	Apellido	IDDepartamento	NombreDepartamento	IDDepartamento
	Zolano	34	Producción	34
	Jordán	33	Ingeniería	33
	Róbinson	34	Producción	34
	Steinberg	33	Ingeniería	33
	Andrade	31	Ventas	31
	NULL	NULL	Mercadeo	35

En este caso el área de Mercadeo fue presentada en los resultados, aunque aún no hay empleados registrados en dicha área.

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 17:22:54)
------
SELECT *
 FROM   empleado
        RIGHT OUTER JOIN departamento
           ON empleado.IDDepartamento = departamento.IDDepartamento


Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) informix.departamento: SEQUENTIAL SCAN

  2) informix.empleado: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                departamento
  t2                empleado

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     4          1         4          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     6          1         6          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ------------------------------------------------------------------------------
  hjoin    6          1         6         4         0        00:00.00   4
```

RIGHT JOIN excluyendo la intersección

Si se quieren mostrar solo los registros de la tabla de Departamento que no tengan correspondientes en la tabla de Empleado, se puede agregar la condición adecuada en la cláusula WHERE. Esto nos dará los departamentos que no tengan asignados ningún empleado.

```
SELECT *
 FROM   empleado
       RIGHT OUTER JOIN departamento
         ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE empleado.iddepartamento IS NULL
```

	Empleado		Departamento	
	Apellido	IDDepartamento	NombreDepartamento	IDDepartamento
	NULL	NULL	Mercadeo	35

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 17:25:39)
------
SELECT *
 FROM   empleado
       RIGHT OUTER JOIN departamento
         ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE empleado.iddepartamento IS NULL


Estimated Cost: 4
Estimated # of Rows Returned: 1

      1) informix.departamento: SEQUENTIAL SCAN

      2) informix.empleado: SEQUENTIAL SCAN

    ON-Filters:informix.empleado.iddepartamento = informix.departamento.iddepartamento
    DYNAMIC HASH JOIN (LEFT OUTER JOIN)
        Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento

PostJoin-Filters:informix.empleado.iddepartamento IS NULL


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                departamento
  t2                empleado

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     4          1         4          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     6          1         6          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ------------------------------------------------------------------------------
  hjoin    1          1         6         4         0        00:00.00   4
```

Combinación completa (FULL OUTER JOIN)

Esta operación presenta los resultados de tabla izquierda y tabla derecha, aunque alguna no tenga correspondencia en la otra tabla. La tabla combinada contendrá, entonces, todos los registros de ambas tablas y presentará valores nulos NULLs para registros sin pareja.

```
SELECT *
 FROM   empleado
       FULL OUTER JOIN departamento
         ON empleado.IDDepartamento = departamento.IDDepartamento
```

	Empleado		Departamento	
	Apellido	IDDepartamento	NombreDepartamento	IDDepartamento
	Zolano	34	Producción	34
	Jordán	33	Ingeniería	33
	Róbinson	34	Producción	34
	Gaspar	36	NULL	NULL
	Steinberg	33	Ingeniería	33
	Andrade	31	Ventas	31
	NULL	NULL	Mercadeo	35

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 17:27:30)
------
SELECT *
 FROM   empleado
        FULL OUTER JOIN departamento
           ON empleado.IDDepartamento = departamento.IDDepartamento


Estimated Cost: 4
Estimated # of Rows Returned: 1

     1) informix.empleado: SEQUENTIAL SCAN

     2) informix.departamento: SEQUENTIAL SCAN

   ON-Filters:informix.empleado.iddepartamento = informix.departamento.iddepartamento
   DYNAMIC HASH JOIN (FULL OUTER JOIN)
       Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                empleado
  t2                departamento

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     6          1         6          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     4          1         4          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ---------------------------------------------------------------------------
  hjoin    7          1         4         6         0        00:00.00   4
```

FULL JOIN excluyendo la intersección

Si se quieren mostrar solo los registros de las tablas que no tengan correspondencia en la otra, se pueden agregar las condiciones adecuadas en la cláusula **WHERE**.

```
SELECT *
 FROM   empleado
      FULL OUTER JOIN departamento
        ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE  (empleado.IDDepartamento IS NULL)  OR  (departamento.IDDepartamento is NULL)
```

	Empleado		Departamento	
	Apellido	IDDepartamento	NombreDepartamento	IDDepartamento
	Gaspar	36	NULL	NULL
	NULL	NULL	Mercadeo	35

```
QUERY: (OPTIMIZATION TIMESTAMP: 06-12-2023 17:29:32)
------
SELECT *
 FROM   empleado
        FULL OUTER JOIN departamento
          ON empleado.IDDepartamento = departamento.IDDepartamento
 WHERE  (empleado.IDDepartamento IS NULL)  OR  (departamento.IDDepartamento is NULL)


Estimated Cost: 4
Estimated # of Rows Returned: 1

      1) informix.empleado: SEQUENTIAL SCAN

      2) informix.departamento: SEQUENTIAL SCAN

    ON-Filters:informix.empleado.iddepartamento = informix.departamento.iddepartamento
    DYNAMIC HASH JOIN (FULL OUTER JOIN)
        Dynamic Hash Filters: informix.empleado.iddepartamento = informix.departamento.iddepartamento

PostJoin-Filters:(informix.empleado.iddepartamento IS NULL OR informix.departamento.iddepartamento IS NULL )


Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                empleado
  t2                departamento

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     6          1         6          00:00.00   2

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     4          1         4          00:00.00   2

  type     rows_prod  est_rows  rows_bld  rows_prb  novrflo  time       est_cost
  ------------------------------------------------------------------------------
  hjoin    2          1         4         6         0        00:00.00   4
```

Uso de Directivas de métodos de acceso

La siguiente tabla describe cada una de las directivas de método de acceso e indica cómo afecta el plan de consulta del optimizador, las presentamos con la intención de que conozcan cual es la utilidad de cada directiva, y se realice el análisis adecuado, para su utilización.

Palabras clave	Efecto	Acción del optimizador
AVOID_FULL	No hay exploración de tabla completa en la tabla enumerada	El optimizador considera los diversos índices que puede escanear. Si no existe ningún índice, el optimizador realiza un análisis de tabla completa.

AVOID_INDEX	No utiliza ninguno de los índices especificados.	El optimizador considera los índices restantes y un escaneo de tabla completa. Si se especifican todos los índices de una tabla, el optimizador utiliza una exploración de tabla completa para acceder a la tabla.
AVOID_INDEX_SJ	No utiliza una ruta de auto unión de índice para los índices especificados	El optimizador no considera el índice especificado para escanear la tabla en una ruta de auto unión de índice.
AVOID_MULTI_INDEX	No utiliza una ruta de escaneo de múltiples índices para la tabla especificada	El optimizador no tiene en cuenta una ruta de exploración multiíndice para la tabla especificada.
FULL	Realiza un escaneo de tabla completa	Incluso si existe un índice en una columna, el optimizador utiliza una exploración de tabla completa para acceder a la tabla.
INDEX	Utiliza el índice especificado para acceder a la tabla.	Si se especifica más de un índice, el optimizador elige el índice que produce el menor costo. Si no se especifican índices, se consideran todos los índices disponibles.
INDEX_ALL *or* MULTI_INDEX	Acceda a la tabla utilizando los índices especificados (escaneo de índice múltiple)	Estas palabras clave son sinónimos. Para obtener información sobre el uso, consulte "Exploración de índices múltiples" a continuación.
INDEX_SJ	Use el índice especificado para escanear la tabla en una ruta de auto unión de índice.	El optimizador se ve obligado a escanear la tabla utilizando una ruta de auto unión de índice con el índice especificado (o elegir el índice menos costoso en una lista de índices para una ruta de auto unión de índice).

Las palabras clave AVOID_FULL e INDEX especifican que el optimizador debe evitar un análisis completo de una tabla. Sin embargo, se recomienda que utilice la palabra clave AVOID_FULL para especificar la intención de evitar un análisis completo de la tabla.

La directiva AVOID_MULTI_INDEX no acepta una lista de índices como argumento. Esto se debe a que la directiva AVOID_INDEX también evita que el índice especificado se use en una ruta de ejecución de escaneo de múltiples índices.

Escaneos de múltiples índices

Se pueden definir hasta dieciséis (16) índices en una tabla. Una ruta de búsqueda basada en un método de acceso que utiliza más de un índice en la misma tabla se denomina exploración de índices múltiples. La directiva MULTI_INDEX o INDEX_ALL obliga al optimizador de consultas a considerar una exploración de múltiples índices para buscar filas calificadas en la tabla especificada. La lista de argumentos para la directiva MULTI_INDEX o INDEX_ALL tiene esta semántica:

Si especifica una tabla como el único argumento de la directiva, el optimizador considera todos los índices disponibles en esa tabla y los utiliza todos (o un subconjunto) cuando busca filas calificadas en la tabla.
Si especifica una tabla y solo un índice, el optimizador considera usar solo ese índice para escanear la tabla.
Si especifica una tabla y más de un índice, el optimizador considera una ruta de búsqueda que utiliza todos los índices especificados.

Escaneo de múltiples índices con métodos de acceso de escaneo salteado

Una ruta de exploración de índices múltiples accede a una tabla mediante un método de acceso de exploración omitida, utilizando una lista ordenada de ROWID. La lista ordenada generalmente se genera a partir de un método de acceso de escaneo de múltiples índices, utilizando todos los índices que especifican la directiva INDEX_ALL o MULTI_INDEX.

Por ejemplo, si los predicados de la consulta especifican col1 <= 10 y col2 BETWEEN 15 AND 25, el plan de ejecución puede usar dos índices: el primer índice en col1 y el segundo índice en col2. Cada exploración de índice devuelve todos los ROWID que cumplen la condición de búsqueda para el índice respectivo. La intersección lógica de las dos listas de ROWID incluye solo las filas que cumplen ambas condiciones de

búsqueda. Luego, el servidor de la base de datos ordena la lista ROWID combinada y usa esta lista ordenada para escanear la tabla en busca del conjunto de resultados de la consulta.

Si la consulta incluye predicados en más de dos columnas indexadas, la lista de ROWID que devuelve cada examen de índice debe combinarse para producir una lista ordenada de ROWID de todas las filas que califican.

Debido a que cada ROWID representa la ubicación física de una fila (en qué página y en qué ranura), la ruta de ejecución simplemente accede a esa ubicación física para recuperar la fila. Como sugiere el término "exploración por omisión", normalmente hay espacios entre un ROWID y el siguiente en la lista ordenada, de modo que el servidor de la base de datos "salta" de una fila calificada a la siguiente fila calificada del conjunto de resultados.

La lista de ROWID ordenados se puede generar a partir de múltiples escaneos de índice, como se describe anteriormente, o desde un solo escaneo de índice. En el caso de un solo índice, la ruta de ejecución de exploración omitida realiza estas acciones:

El escaneo de índice único crea una lista desordenada de los ROWID de todas las filas calificadas.
Esta lista desordenada se ordena por valor de ROWID.
El servidor de la base de datos luego recupera las filas calificadas en el orden de sus ROWID.

Un método de acceso de escaneo salteado se asemeja a un escaneo secuencial, pero a veces puede ser más eficiente. Un escaneo secuencial recupera todas las filas de la tabla, pero un escaneo salteado solo recupera las filas que tienen ROWID calificados.

Restricciones en las rutas de escaneo de múltiples índices para la ejecución de consultas

El nivel de aislamiento de la transacción afecta si la directiva MULTI_INDEX o INDEX_ALL puede forzar una ruta de ejecución de exploración multiíndice, que no está disponible mientras el nivel de aislamiento es Estabilidad del cursor o Lectura confirmada con la opción ÚLTIMO COMPROMETIDO. (Sin embargo, esta directiva se admite en los niveles de aislamiento de Lectura sucia y Lectura repetible, y en Lectura confirmada sin la opción ÚLTIMO COMPROMETIDO).

Las siguientes restricciones adicionales se aplican a las rutas de acceso de escaneo de múltiples índices:

Los índices deben ser índices de árbol B. Estos pueden ser índices adjuntos o separados.
Estas directivas se ignoran para los índices de árbol R, los índices funcionales y los índices basados en la interfaz de índice virtual (VII).

La tabla no puede ser una tabla remota, una pseudotabla, una tabla de catálogo del sistema, una tabla externa o una tabla jerárquica.

Una exploración de varios índices no puede admitir predicados de unión como filtros de índice en las exploraciones de índice subyacentes.
Un escaneo de múltiples índices ignora todas las columnas de un índice compuesto excepto la columna inicial.

Las sentencias DML que realizan eliminaciones en cascada o declarar variables locales de sentencia (SLV) no pueden usar un escaneo de múltiples índices.

Las consultas de actualización que activan una acción desencadenada POR CADA FILA no pueden usar una exploración de índice múltiple.

En las bases de datos compatibles con ANSI, la directiva MULTI_INDEX o INDEX_ALL no se sigue para una declaración SELECT que no tiene cláusula ORDER BY, cláusula GROUP BY ni cláusula FOR READ ONLY, si la cláusula FROM específica solo una tabla.

(En este caso especial, la consulta tiene un comportamiento de cursor implícito que entra en conflicto con una ruta de acceso de escaneo de múltiples índices).

Combinaciones de directivas de métodos de acceso

En general, puede especificar solo una directiva de método de acceso por tabla. Solo las siguientes combinaciones de directivas de método de acceso son válidas para la misma tabla en la misma consulta:

INDEX, AVOID_INDEX_SJ
AVOID_FULL, AVOID_INDEX
AVOID_FULL, AVOID_INDEX_SJ
AVOID_INDEX, AVOID_INDEX_SJ
AVOID_FULL, AVOID_INDEX, AVOID_INDEX_SJ
AVOID_FULL, AVOID_MULTI_INDEX
AVOID_INDEX, AVOID_MULTI_INDEX
AVOID_INDEX_SJ, AVOID_MULTI_INDEX
AVOID_FULL, AVOID_INDEX_SJ, AVOID_MULTI_INDEX
AVOID_INDEX, AVOID_INDEX_SJ, AVOID_MULTI_INDEX

Cuando especifica las directivas de método de acceso AVOID_FULL y AVOID_INDEX, el optimizador evita realizar un análisis completo de la tabla y evita usar el índice o los índices especificados. Esta combinación de directivas negativas permite que el optimizador use índices que se crean después de especificar las directivas de método de acceso.

Debido a que el optimizador considera automáticamente la ruta de autounión del índice si especifica la directiva INDEX o AVOID_FULL, use la directiva INDEX_SJ solo para forzar una ruta de autounión del índice usando el índice especificado (o eligiendo el índice menos costoso en una lista separada por comas). de índices). La directiva INDEX_SJ puede mejorar el rendimiento cuando un índice de varias columnas incluye columnas que solo proporcionan una baja selectividad como filtros de clave de índice.

Especificar la directiva INDEX_SJ evita el requisito habitual del optimizador para las estadísticas de distribución de datos en las claves principales del índice. Esta directiva hace que el optimizador considere una ruta de autounión de índice, incluso si las estadísticas de distribución de datos no están disponibles para las columnas clave de índice principales. En este caso, el optimizador solo incluye el número mínimo de columnas de clave de índice como claves principales para satisfacer la directiva.

Por ejemplo, si se define un índice en las columnas **c1** , **c2** , **c3** , **c4** , y la consulta especifica filtros en las cuatro columnas pero no hay distribuciones de datos disponibles en ninguna columna, entonces especificar INDEX_SJ en este índice dará como resultado la columna **c1** se utiliza como clave principal en una ruta de autounión de índice. Si desea que el optimizador use un índice pero no considere la ruta de autounión del índice, debe especificar una directiva INDEX o AVOID_FULL para elegir el índice, y también debe especificar una directiva AVOID_INDEX_SJ para evitar que el optimizador considere cualquier otro ruta de autounión del índice.

Si AVOID_INDEX_SJ se usa junto con la directiva INDEX, ya sea como una directiva INDEX explícita o como la combinación equivalente de AVOID_FULL y AVOID_INDEX, los índices especificados en la directiva AVOID_INDEX_SJ deben ser un subconjunto de los índices especificados en la directiva INDEX. Para obtener más información sobre los efectos de las directivas INDEX_SJ y AVOID_INDEX_SJ, consulte el capítulo de IBM® Informix® Performance Guide que describe las directivas del optimizador.

Especificar la directiva MULTI_INDEX o INDEX_ALL evita el requisito habitual del optimizador para las estadísticas en la tabla especificada. El optimizador normalmente requiere al menos estadísticas de bajo nivel en la tabla antes de considerar la ruta de exploración multiíndice en la tabla.

Ejemplos de directivas de métodos de acceso

Suponga que tiene una tabla llamada **emp** que contiene las columnas **emp_no** , **dept_no** y **job_no** , y para la cual los siguientes índices **ids_dept_no** están definidos en la columna **dept_no** y el índice **idx_job_no** está definido en la columna **job_no .** Cuando realiza un SELECT que incluye la tabla **emp** en la cláusula FROM, puede indicarle al optimizador que acceda a la tabla de una de las siguientes maneras:

Ejemplo usando una directiva positiva:

SELECT {+INDEX(emp idx_dept_no)}

En el siguiente ejemplo, la directiva de método de acceso obliga al optimizador a considerar el uso de un análisis de múltiples índices, en función de los resultados combinados del análisis del índice idx_dept_no en la columna dept_no y el índice idx_job_no en la columna job_no.

SELECT {+MULTI_INDEX(emp idx_dept_no ids_job_no)} ...

Ejemplo usando directivas negativas:

SELECT {+AVOID_INDEX(emp idx_loc_no, idx_job_no), AVOID_FULL(emp)} ...

Este ejemplo incluye varias directivas de métodos de acceso. Estas directivas obligan a escanear el índice **idx_dept_no** en la columna **dept_no** indicando al optimizador que no escanee los índices **idx_loc_no** e **idx_job_no** , y que no realice un escaneo completo de la tabla **emp** . Sin embargo, si se crea un nuevo índice **idx_emp_no para table emp** , estas directivas no impiden que el optimizador lo considere.

Tenga en cuenta también que el término directiva negativa se refiere a la cadena "AVOID_" en una directiva de método de acceso y no tiene nada que ver con el símbolo + que sigue al indicador de comentario que comienza cada directiva de optimización.

El bucle FOREACH para definir cursores

Un ciclo FOREACH comienza con la palabra clave FOREACH y termina con END FOREACH. Entre FOREACH y END FOREACH, puede declarar un cursor o usar EJECUTAR PROCEDIMIENTO o EJECUTAR FUNCIÓN.

```
CREATE_PROCEDURE increase_by_pct( pct INTEGER )
  DEFINE s INTEGER;

  FOREACH sal_cursor FOR
    SELECT salary INTO s FROM employee
      WHERE salary > 35000
    LET s = s + s * ( pct/100 );
    UPDATE employee SET salary = s
      WHERE CURRENT OF sal_cursor;
  END FOREACH;

END PROCEDURE;
```

La rutina anterior realiza estas tareas dentro del ciclo FOREACH:
Declara un cursor
Selecciona un valor de salario a la vez del empleado
Aumenta el salario en un porcentaje.
Actualiza empleado con el nuevo salario

Obtiene el siguiente valor de salario
La declaración SELECT se coloca dentro de un cursor porque devuelve todos los salarios en la tabla mayores que35000.

La cláusula WHERE CURRENT OF de la instrucción UPDATE solo actualiza la fila en la que se encuentra actualmente el cursor y establece un cursor de actualización en la fila actual. Un cursor de actualización coloca un bloqueo de actualización en la fila para que ningún otro usuario pueda actualizar la fila hasta que ocurra su actualización.

Una rutina SPL establecerá un cursor de actualización automáticamente si una declaración UPDATE o DELETE dentro del ciclo FOREACH usa la cláusula WHERE CURRENT OF. Si usa WHERE CURRENT OF, debe hacer referencia explícita al cursor dentro de la instrucción FOREACH. Si está utilizando un cursor de actualización, puede agregar una instrucción BEGIN WORK antes de la instrucción FOREACH y una instrucción COMMIT WORK después de END FOREACH.

```
BEGIN WORK;
  FOREACH sal_cursor FOR
    SELECT salary INTO s FROM employee WHERE salary > 35000;
    LET s = s + s * ( pct/100 );
    UPDATE employee SET salary = s WHERE CURRENT OF sal_cursor
  END FOREACH;
COMMIT WORK;
```

Para cada iteración del ciclo FOREACH en la figura anterior, se adquiere un nuevo bloqueo (si usa el bloqueo de nivel de fila). La declaración COMMIT WORK libera todos los bloqueos (y confirma todas las filas actualizadas como una sola transacción) después de la última iteración del bucle FOREACH.

Para confirmar una fila actualizada después de cada iteración del bucle, debe abrir el cursor CON HOLD e incluir las sentencias BEGIN WORK y COMMIT WORK dentro del bucle FOREACH.

```
CREATE PROCEDURE serial_update();
   DEFINE p_col2 INT;
   DEFINE i INT;
   LET i = 1;
   FOREACH cur_su WITH HOLD FOR
      SELECT col2 INTO p_col2 FROM customer WHERE 1=1
      BEGIN WORK;
         UPDATE customer SET customer_num = p_col2 WHERE CURRENT OF cur_su;
      COMMIT WORK;
      LET i = i + 1;
   END FOREACH;
END PROCEDURE;
```

La rutina serial_update() de SPL confirma cada fila como una transacción separada.

<u>Restricción para bucles FOREACH</u>

Dentro de un ciclo FOREACH, la consulta SELECT debe completar la ejecución antes de cualquier operación DELETE, INSERT o UPDATE que cambie el conjunto de datos del cursor SELECT. Una forma de asegurarse de que se complete la consulta SELECT es usar una cláusula ORDER BY en la declaración SELECT. La cláusula ORDER BY crea un índice en las columnas y previene los errores causados por las sentencias UPDATE, INSERT, DELETE modificando los resultados de la consulta de la sentencia SELECT en el mismo bucle FOREACH.

Es importante que se agregue la rutina de realizar commit cada 1000 registros.

CREACIÓN DE TRIGGERS

Todos los nombres de los triggers y de los archivos que los contienen deben estar en minúsculas.

Todos los nombres de los triggers y de los archivos que los contienen no deben contener espacios en blanco.

Todos los nombres de los archivos que contienen trigger deben terminar con la extensión .sql

Todos los nombres de los triggers que se vayan a crear, modificar o eliminar deberán agregarles el nombre del propietario que en este caso será **siempre el usuario informix entre comillas**.

Sintaxis:

```
CREATE TRIGGER "informix".nombre_trigger
-- INSERT | DELETE | UPDATE OF COLUMN_NAME
ON -- "informix".nombre_tabla
-- REFERENCING NEW AS name OLD AS name
-- FOR EACH ROW | BEFORE | AFTER
-- ( contenido )

Ejemplo:

create trigger "informix".tr_tabla1_delete delete on "informix".tabla1
referencing old as ant
    for each row
              (
              delete from baseDatos:"informix".tabla2
          where (campo1 = ant.campo1 ) );
```

CREACIÓN DE TABLAS

El nombre de las TABLAS en la sentencia de creación, debe de estar escritos con minúsculas sin espacio en blanco, el propietario debe de ser “informix”, ejemplo:

CREATE TABLE “informix”.ss_concilsdocont

Es necesario agregar el **extent size … next size … lock mode row;** al final de la tabla (los tamaños siempre van en KB = 1,024 bytes = 1,024 caracteres).

Ejemplo:

CREATE TABLE “informix”.comerciodudoso
(numerodeafiliacion char(19),
descripcion char(40),
primary key (numerodeafiliacion) **)**
extent size … next size … lock mode row;

Es necesario que recuerden que el tamaño mayor va en la primer sentencia **extent size …** y el siguiente tamaño menor en la sentencia **next size** (los tamaños siempre van en KB = 1,024 bytes = 1,024 caracteres o múltiplos del mismo).

Para tablas con alta volumetría de datos requiere que los tamaños de extents no sean los mínimos (estándar) de 32 kb. Deben estar calculados en base al total de registros a insertar.

Tomar en cuenta el **tamaño del registro** con el comando:
dbschema –d nombre_base_de_datos –t nombre_tabla –ss

```
{ TABLE "informix".br_traslado_hist row size = 812 number of columns = 8 index size = 70 }

create table "informix".br_traslado_hist
  (
    institucion char(2),
    numcte char(20) default '',
    num_solicitud char(20),
    envio char(255),
    envio1 char(255),
    envio2 char(255),
    status char(1),
    fecha_insert date default  ''
  ) extent size 32 next size 32 lock mode row;
```

Cada Extent se mide por lo general en registros de 4 Kb. Se deben restar **28 Bytes** que son reservados para la conformación del mismo **(4096-28=4068)**, con lo que quedamos con **4068** Bytes por extent.
Utilizar el **tamaño del** registro donde cada fila de la tabla mide 812 Bytes. Posterior calcular el tamaño promedio de las filas de la tabla:

"A mayor cantidad de extents menor es el rendimiento del sistema, pero mejor el aprovechamiento de recursos de disco"

Además para mejorar el desempeño en inserciones y lecturas masivas deben hacer la distribución de los datos en forma circular (**ROUND ROBIN**) en mínimo 3 (optimo 5) dbspaces (dividiendo el tamaño del **EXTENT SIZE** del punto anterior entre 3 (o 5) dbspaces haciendo lo mismo con el tamaño del **NEXT SIZE** y **solicitar la creación de los dbspaces nuevos para crear estas tablas con los espacios que calcularon**.

Ejemplo:

```
create table "informix".sb_regreso_hist(
institucion char(2),
num_solicitud char(20),
regreso char(1000),
status char(1)
)
```

fragment by round robin in dbs_edoctahis1, dbs_edoctahis2, dbs_edoctahis3, dbs_edoctahis4
extent size **94627** next size **9462** lock mode row;

Los índices para este tipo de tablas deben **crearse en un Dbspace separado de los datos** (por la distribución circular) **mejor si tiene tamaños de página superiores a los 4Kb**.

Es necesario que establezcan una llave primaria, una clave primaria es una columna o un conjunto de columnas en una tabla cuyos valores identifican de forma exclusiva una fila de la tabla. Una base de datos relacional está diseñada para imponer la exclusividad de las claves primarias permitiendo que haya sólo una fila con un valor de clave primaria específico en una tabla.

SELECT {+INDEX(emp idx_dept_no)}

Es importante que sus tablas tengan estos constraint.

Claves primarias y foráneas

Las tablas se relacionan con otras tablas mediante una *relación de clave primaria o de clave foránea*. Las relaciones de claves primarias y foráneas se utilizan en las bases de datos relacionales para definir relaciones de muchos a uno entre tablas.

Las relaciones de claves primarias y foráneas entre tablas en un esquema de estrella o copo de nieve, a veces llamadas relaciones de muchos a uno, representan las vías de acceso a través de las cuales las tablas relacionadas se unen en la base de datos. Estas vías de acceso de unión son la base para formar consultas de datos históricos.

Claves primarias

Una clave primaria es una columna o un conjunto de columnas en una tabla cuyos valores identifican de forma exclusiva una fila de la tabla. Una base de datos relacional está diseñada para imponer la exclusividad de las claves primarias permitiendo que haya sólo una fila con un valor de clave primaria específico en una tabla.

Claves foráneas

Una clave foránea es una columna o un conjunto de columnas en una tabla cuyos valores corresponden a los valores de la clave primaria de otra tabla. Para poder añadir una fila con un valor de clave foráneo específico, debe existir una fila en la tabla relacionada con el mismo valor de clave primaria.

Relaciones de muchos a uno

Una relación de muchos a uno hace referencia a una tabla o entidad que contiene valores y hace referencia a otra tabla o entidad que tiene valores exclusivos. Las relaciones de muchos a uno con frecuencia son impuestas por las relaciones de clave foránea y clave primaria, y generalmente las relaciones se establecen entre las tablas de hechos y las entidades o tablas de dimensiones y entre los niveles de una jerarquía.

La relación se utiliza con frecuencia para describir clasificaciones o agrupaciones. Por ejemplo, en un esquema geográfico que tenga las tablas Región, Estado y Ciudad muchos estados pertenecen a una región determinada, pero los mismos estados no pueden pertenecer a dos regiones diferentes. Lo mismo ocurre con las ciudades, una ciudad sólo está en un estado (las ciudades que tienen el mismo nombre pero están en más de un estado se deben tratar de forma algo distinta). Cada ciudad existe en un solo estado, pero un estado puede tener muchas ciudades, de ahí el término muchos a uno.

Normalizar un modelo de datos

Cuando se normaliza un modelo de datos, se pueden conseguir los siguientes objetivos. Es posible:

- ✔ Producir una mayor flexibilidad en el diseño.
- ✔ Asegurarse que los atributos se colocan en las tablas correctas.
- ✔ Reducir la redundancia de datos.
- ✔ Aumentar la eficacia del programador.
- ✔ Reducir los costes de mantenimiento de la aplicación.
- ✔ Maximizar la estabilidad de la estructura de datos.

La normalización consta de varios pasos para reducir las entidades a propiedades físicas más deseables. Estos pasos se denominan normas de normalización o *formularios normales*.

Cada formulario normal restringe los datos en mayor medida que el último formulario. Por este motivo, debe conseguir el primer formulario normal para poder conseguir el segundo, y debe conseguir el segundo para poder conseguir el tercero.

Primer formulario normal

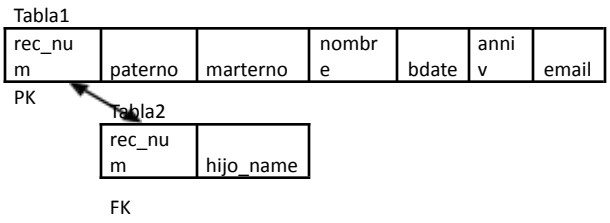
Una entidad está en el primer formulario normal si no contiene ningún grupo repetitivo. En términos relacionales, una tabla está en el primer formulario normal si no contiene columnas repetitivas. Las columnas repetitivas reducen la flexibilidad de los datos, malgastan espacio de disco y dificultan la búsqueda de datos, Ejemplo:

Tabla1

rec_num	paterno	marterno	nombre	bdate	anniv	email	hijo1	hijo2	hijo3
								Columnas Repetidas	

Puede ver varios problemas en la tabla actual. La tabla siempre reserva espacio en disco para los tres registros de hijos, tanto si la persona tiene hijos como si no los tiene. El número máximo de hijos que puede registrar es tres, pero algunas de las personas pueden tener cuatro o más hijos. Para buscar un determinado hijo, debe buscar en las tres columnas de cada fila.

Para eliminar las columnas repetitivas e incorporar la tabla en el primer formulario normal, divida la tabla en dos tablas. Coloque las columnas repetitivas en una de las tablas. La asociación entre las dos tablas se establece con una combinación de clave primaria y clave foránea. Puesto que un hijo no puede existir sin una asociación en la tabla1, puede hacer referencia a la tabla2 con una clave foránea, rec_num, Ejemplo:



Segundo formulario normal

Una entidad está en el segundo formulario normal si todos sus atributos dependen de la clave entera (primaria). En términos relacionales, cada columna de una tabla debe ser funcionalmente dependiente de la clave primaria entera de dicha tabla. La dependencia funcional indica que existe un enlace entre los valores de dos columnas diferentes.

Si el valor de un atributo depende de una columna, el valor del atributo debe cambiar si cambia el valor de la columna. El atributo es una función de la columna. Las siguientes explicaciones lo especifican mejor:

Si la tabla tiene una clave primaria de una columna, el atributo debe depender de dicha clave.

Si la tabla tiene una clave primaria compuesta, el atributo debe depender de los valores de todas sus columnas tomadas en conjunto, no de una o de algunas de ellas.

Si el atributo también depende de otras columnas, deben ser columnas de una clave candidata; es decir, columnas que sean exclusivas en cada fila.

Si no convierte el modelo al segundo formulario normal, se arriesga a sufrir redundancia de datos y dificultados al modificar datos. Para convertir tablas del primer formulario normal en tablas del segundo formulario normal, elimine las columnas que no dependen de la clave primaria.

Tercer formulario normal

Una entidad está en el tercer formulario normal si está en el segundo formulario normal y ninguno de sus atributos es dependiente de forma transitiva de la clave primaria. Dependencia transitiva significa que dichos atributos de clave de descriptor dependen no sólo de la clave primaria entera, sino también de otros atributos de clave de descriptor que, a su vez, dependen de la clave primaria. En términos de SQL, el tercer formulario normal significa que ninguna columna de una tabla depende de una columna de descriptor que, a su vez, depende de la clave primaria.

Para convertir al tercer formulario normal, elimine los atributos que dependen de otros atributos de clave de descriptor.

CREACIÓN DE ÍNDICES

El nombre de los INDICES en la sentencia de creación, debe de estar escritos con minúsculas sin espacio en blanco, el propietario debe de ser “informix”, ejemplo:

CREATE INDEX "informix".idx_indexes ON "informix".sc_tabla(cuenta) online;

No utilizar las instrucciones BEGIN – COMMIT, así como el PDQPRIORITY en creación de índices

Los índices son una estructura de datos que mejora la velocidad de recuperación de datos en una tabla.

Los índices se utilizan para localizar datos rápidamente sin tener que buscar en todos los registros de una tabla.

El escaneo secuencial por el contrario es aquel que lee cada registro de la tabla en su totalidad, y lee en el orden que se almacenan las páginas en disco.

En casos aislados cuando la tabla no es utilizada con una alta transaccionalidad el escaneo secuencia puede ser el mejor método de acceso a los datos cuando se desea buscar más del 20-25% de los registros de la tabla.

A continuación se presenta un ejemplo del escaneo secuencial versus localización por índice:

§ Tabla de 100.000 filas con 4 niveles.
§ Un escaneo secuencial de la tabla tomará 100,000 lecturas (1 lectura por registro)
§ Un índice tomará 4 lecturas para cada nivel y 1 lectura para la registró (5 lecturas por registro

La utilización de los índices debe de ser de la siguiente forma:
1) El índice debe evitar el buscar demasiada información, es decir el índice debe de descartar la mayor cantidad posible de registros.
2) El índice debe de cumplir con el filtrado de información que cumpla con las reglas de negocio.
El índice consume espacio y recursos del manejador, por lo tanto, no se debe de hacer un uso excesivo de creación de índices.

Para las sentencias DML se tienen lo siguiente.

1) SELECT. Para las consultas requerirá leer varios niveles del índice para acceder a las páginas donde están los datos.

2) INSERT. Cuando se realiza un Insert a una tabla, se deberá de agregar apuntadores al registro por cada índice existente, si no hay páginas existentes se crean nuevas, así como más niveles, lo cual resulta costoso en el rendimiento y almacenamiento.
El rendimiento se puede mejorar para cargas grandes deshabilitando los índices y luego reconstruyendo los índices después de la carga, se requiere para esto conocimiento previo de las operaciones del negocio para no afectar cuando se deshabilitan.

3) DELETE. Cuando se borra un registro, se deben de borrar todos los apuntadores del registro en cada uno de los índices, más el registro en la tabla.

4) UPDATE. Para realizar una actualización de un registro y/o campo, se deberá de tener presente la actualización de los apuntadores del registro en cada página de índices, es decir; una actualización se realiza como una eliminación del apuntador anterior del registro y luego una inserción en la nueva página, lo cual produce doble costo uno para eliminar y otro para insertar.

En resumen, cuando realizamos inserciones y/o borrados de registros realizamos muchas operaciones de I/O, dependiendo de los niveles de cada índices afectado, y cuando se realiza una actualización se puede decir que realizas el doble de operación de I/O sobre cada índice afectado y/o relacionado con él o los registros, ya que elimina e inserta.

Nota: Cuando se realiza un Rollback de alguna transacción, es importante considerar lo anterior ya que puede presentar tiempo y costo en deshacer dicha transacción.

El bloquear un registro, hará que se bloqueen los índices de los apuntadores de los registros relacionados.

Una tabla con 5 índices generará 6 bloqueos.

Insertar, actualizar o eliminar un registro crear automáticamente un candado en la clave de índice y las tablas con bloqueo a nivel de página mantendrán bloqueos en la página de índices relacionados afectando a los registros contenidos en la página.

La pregunta que nos debemos hacer es ¿Cuándo se requiere un índice? La respuesta pueden ser muchas, pero tomemos en cuanta las siguientes seis:

1) Cuando las consultas de la tabla solo devuelven de pocos renglones utilizando condiciones (WHERE).
2) Cuando requieres el control de los datos de una o varias columnas sean únicos.
3) Cuando requieres de una restricción (constraint) como llave primaria o foránea.
4) Cuando haces uso de JOIN'S, las columnas utilizadas en el JOIN deben de formar parte del índice.
5) Para mejorar el rendimiento de una consulta que utiliza ORDER BY.
6) Para mejorar el rendimiento de una consulta que utiliza GROUP BY.

Mantenimiento de índices.

· Para obtener la mayor eficiencia de los índices se deberá de programar la reconstrucción de los índices después de un tiempo, el cual depende de que tanta información cambie por medio de las sentencias DML y DDL.
· Se deberá analizar la fragmentación de los espacios.
· Revisar el número de extents, número de páginas alojadas, etc.
· Por supuesto de no dejar de correr estadísticas.
· El acceso al storage (discos físicos), que contienen datos e índices debe de hacerse por canales diferentes.

Tomando en cuenta todas las consideraciones anteriores, analizaremos los índices de la tabla, sc_movdia los cuales se presentan a continuación:

Información del 17 de Marzo 2023.

#	Nombre	Columnas				levels	leaves	nunique	clust	nunique	%
1	idx_movdia1a	cuenta				4	6357	430041	549846	515583	63%
2	idx_movdia4a	usuario				3	2549	4256	174517	7620	1%
3	idx_movdia6a	cancelad				3	3494	2	51145	2	0%
4	idx_movdia10	fech_hor				3	2901	33282	47205	35072	4%
5	idx_sc_movdia9	folio_suc				3	6698	502188	288380	628787	77%
6	idx_movdia_referencia	referencia				4	12032	521628	450724	628987	77%
7	idx_movdia2a	empresa	folio_suc			3	7326	1	288326	1	0%
8	idx_movdia3a	empresa	num_tarjeta			4	3959	1	258026	1	0%
9	idx_movdia7a	cuenta	folio_suc			4	11456	430084	550571	515627	63%
10	idx_movdia8a	fech_alt	cuenta			3	6398	3	549934	3	0%
11	idx_movdia_fechaserial	fech_alt	num_serial			3	3475	3	47212	3	0%
12	idx_sc_movdia_02	cancelad	num_serial			3	5885	2	51180	2	0%
13	idx_sucempbdicheq	sucursal	empresa			3	4881	1312	197790	1321	0%
14	ind_sc_movdia_01	fech_alt	folio_suc	monto_tot		4	9835	3	288406	3	0%
15	idx_movdiamspei	transacc	fecha_val	cancelad	referencia	4	13907	183	467190	209	0%

La tabla tiene 15 índices creados en 3 dbspaces, los cuales por las consideraciones anteriores se deberá de analizar por los administradores del storage, si están creados en porciones de disco distintos, tanto físicamente como lógicamente, esto para garantizar el mejor rendimiento. (Nota: Esto no índica que no se pueda tener en los mismos espacios, tanto físicos como lógicos, solo que es recomendable seguir las mejores prácticas.)

Los 15 índices, tienen distintos niveles, 6 "nivel 4", 9 "nivel 3" , cabe recordar que entre más niveles tengan más I/O realizan, a la hora de insertar y/o borrar un registro y el doble cuando realizan una actualización, por tanto se observa, que estos índices tienen unos niveles no adecuado.

La optimización de los índices conlleva a la reconstrucción de los mismos, cabe mencionar que se recomienda separar los datos de los índices en espacios de disco diferentes tanto físico como lógico.
La columna empresa no proporciona un filtrado de información, ya que siempre es el mismo valor y no descarta ningún registro, por lo que en muchos casos genera overhead (consumo excesivo de recursos).

En la última columna se muestra un porcentaje donde indica que entre más grande más eficiente el índice, entre otras cosas.

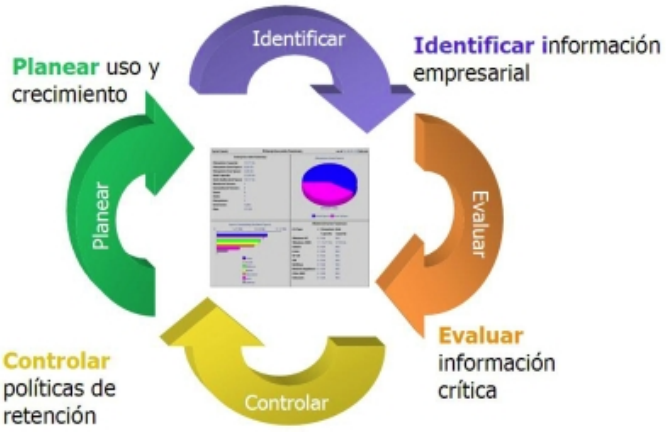
Por otra parte, muestro la siguiente tabla del uso de los índices en cuanto "IOS", donde se puede contrastar con la eficiencia mostrada en la tabla anterior, es decir, hay índices que se utilizan y tienen una buena eficiencia y otros índices tienen un alto uso pero poca eficiencia.

Información del 17 de Marzo 2023.

Dbspace	Indice	IS_ios	bufios	pageios	Tot_IO	% USU-IOS
dbs_movdia_idx02	idx_movdia8a	465,698,560	2,289,512,831	50,259,804	2,805,471,195	21%
dbs_movdia_idx02	idx_movdia1a	249,429,456	1,387,230,849	51,816,311	1,688,476,616	12%
dbs_movdia_idx01	idx_movdiamspei	231,846,019	1,096,183,840	44,941,295	1,372,971,154	10%
dbs_movdia_idx01	idx_movdia7a	38,384,656	809,834,793	69,789,095	918,008,544	7%
datos02_idx	idx_movdia_referencia	37,269,098	754,223,113	41,142,203	832,634,414	6%
dbs_movdia_idx02	idx_sc_movdia9	34,396,766	713,414,747	29,237,508	777,049,021	6%
dbs_movdia_idx03	idx_movdia2a	33,450,695	713,222,587	30,099,361	776,772,643	6%
dbs_movdia_idx02	ind_sc_movdia_01	32,445,192	694,482,431	33,162,775	760,090,398	6%
dbs_movdia_idx01	idx_movdia3a	32,369,947	673,621,514	31,257,602	737,249,063	5%
dbs_movdia_idx03	idx_movdia_fechaserial	106,687,288	600,639,395	2,501,765	709,828,448	5%
dbs_movdia_idx01	idx_sucempbdicheq	38,752,834	546,544,118	9,526,978	594,823,930	4%
dbs_movdia_idx02	idx_movdia4a	38,090,817	511,462,691	5,279,771	554,833,279	4%
dbs_movdia_idx03	idx_sc_movdia_02	33,074,753	349,548,331	2,842,884	385,465,968	3%
dbs_movdia_idx03	idx_movdia6a	32,582,440	334,767,821	1,423,144	368,773,405	3%
dbs_movdia_idx01	idx_movdia10	34,113,561	314,649,488	1,258,374	350,021,423	3%
					13,632,469,501	100%

CICLO DE VIDA DE LA INFORMACION

La elección de la mejor solución dependerá en gran medida de las propias reglas del negocio, del tipo y cantidad de la información que hay que resguardar, la vigencia de la misma, la seguridad y disponibilidad, y, además, que vaya acompañada del mejor costo total de propiedad de la solución que elija



Julio, 2023

Una base de datos no es más que un componente de un sistema de información. Por tanto, el ciclo de vida del sistema de información incluye el ciclo de vida de la base de datos que forma parte de él. En particular, desde el punto de vista de la base de datos, centraremos principalmente nuestra atención en las siguientes actividades:

- Definición del sistema: Durante la etapa de análisis de requerimientos del sistema, nos fijaremos especialmente en todos los requerimientos asociados a los datos con los que ha de trabajar nuestro sistema.
- Diseño de la base de datos (tablas): El análisis de los requerimientos del sistema nos permitirá organizar los datos con los que nuestro sistema habrá de trabajar. Este proceso de diseño, que está íntimamente ligado a la futura base de datos de nuestro sistema, lo descompondremos en tres fases:

· Diseño conceptual
· Diseño lógico
· Diseño físico

- Implementación de la base de datos (la parte de la implementación del sistema correspondiente a la creación de la base de datos).

- Limitantes a nivel de tablas: Como parte del ciclo se tendrá la recomendación de iniciar dicho proceso al alcanzar el 50% del máximo número de páginas alcanzada (16,775,134).

- Suprimir permanentemente un elemento de una base de datos: Es el proceso mediante el que los datos antiguos se eliminan de la base de datos del sistema, el cual debe seguirse bajo las siguientes recomendaciones:

- Tablas de alta transaccionalidad 1 día de información
- Tablas de alta - media transaccionalidad 3 meses de información
- Tablas de mediana transaccionalidad 6 meses de información
- Tablas de baja transaccionalidad 1 año de información

Las depuraciones minimizan el número de registros de bases de datos no utilizados para aumentar la eficiencia de las búsquedas y reducir el tamaño del disco físico necesario.

- Verificación y validación: Como en todo sistema de información, deberemos verificar que la base de datos y las aplicaciones funcionan correctamente. Además, deberemos comprobar que el sistema construido se ajusta a las necesidades reales que promovieron su proyecto de desarrollo (esto es, validar el sistema y sus requerimientos).
- Operación, supervisión y mantenimiento: Finalmente, una vez puesto en marcha el sistema, se llega a la etapa "final" del ciclo de vida de todo sistema de información (en la que, como ya vimos, se repetirá todo el ciclo cada vez que tengamos que realizar modificaciones sobre el sistema ya existente).

ELIMINAR INFORMACION DE UNA TABLA (TRUNCATE TABLE)

TRUNCATE es una palabra clave de SQL que suprime con rapidez las filas activas de una tabla y las estructuras de árbol B de sus índices. Se suprimen las filas activas sin descartar la tabla ni el esquema, los privilegios de acceso, los desencadenantes, las restricciones ni otros atributos de la misma. Con TRUNCATE TABLE, es posible vaciar una tabla local y liberar (o reutilizar para la misma tabla) el espacio de almacenamiento que anteriormente retenía las filas de datos y las estructuras de árbol B.

Para eliminar todas las filas y cualquier índice relacionado de la tabla ejecutar:

TRUNCATE TABLE *nombre_tabla* **DROP STORAGE;**

CARGAS Y DESCARGAS

Todas las cargas y/o descargas siempre deben contener:

set isolation to dirty read;

set lock mode to wait 3;

Antes del **unload** y/o **load;** **sin son descargas masivas de información direccionarlas al filesystem, que tiene más espacio libre:** **Solicitarlo a sistemas operativos (Ext. 500132).**

Ejemplo

```
SET ISOLATION TO DIRTY READ;
SET LOCK MODE TO WAIT 3;
unload to /RESPALDOSNEW/co_historico.unl
SELECT * FROM "informix".co_historico
   WHERE usuario = '92696295' and control_poliza = 257424
      and fecha_captura = '11012011'
         and secuencia > 0 and empresa = '001';
```

Creaciones de CRON

Todas las creaciones de CRON deben de indicar/contener lo siguiente:

En que Servidor:

Solicitar en el cron la aplicación en la servidor réplica (para garantizar su correcta función en caso de DRP que están en el segmento de Monterrey).

- **En cual cuenta de Usuario:** “sysconau” por ejemplo (del cual ya se haya solicitado su creación a nivel S. O. via el formato de seguridad que ya conocen) **se va a colgar dicho proceso.**
- **Fecha de ejecución:** Que fecha/día.
- **Script:** proporcionar el archivo .sql con las sentencias que serán mandadas a ejecutar por el proceso CRON a programar.
- **Shell:** proporcionar el archivo .sh con las sentencias que serán mandadas a ejecutar por el proceso CRON a programar.
- **Hora de ejecución:** A que hora y/o minutos (no existen segundos).
- **Parámetros:** si los lleva en código duro o hay que ingresarlos/capturarlos manualmente.
- **Correos:** a que dirección (es) de correo electrónico se debe enviar el resultado de su ejecución.
- **Bitácora:** especificar si es necesario conservar la historia de ejecuciones o solo enviar la ejecución del día.

Ejecución de Scripts

En los scripts que realicen operaciones SELECT (para consultas), UPDATE (para actualizaciones), INSERT (para inserciones) y/o DELETE (para borrar registros) deben colocar como primera instrucción **Begin** al comienzo de los bloques y **Commit** al final de los mismos, **siempre y cuando no sean masivos > 5,000 por que pueden bloquear la instancia al ser una sola transacción muy larga en su ejecución;** para que quede de la siguiente manera:

```
begin,
      ……
      ……
commit;
```

Las sentencias de SELECT (para consultas), UPDATE (para actualizaciones), DELETE (para borrar) e INSERT (para inserciones) que se hagan en la misma base de datos deberán tener siempre al usuario informix entre comillas “ “.

Correcto

```
BEGIN;
INSERT INTO "informix".si_bpiusuarios(empresa, numcte, id_status, folio_contrato, usuario, pass, f_pass,
pass1, f_pass1, pass2, f_pass2, pass3, f_pass3, f_status, f_ultimo_acceso, f_actualizacion, suc_registro,
f_registro) VALUES('001', '004271940', 10, '0104840T9U20', '', '', '', '', '', '', '', '', '', '', '', '0116', '05/16/2009');
COMMIT;
```

Solicitar para las sentencias DELETE y UPDATE el filtro de información a afectar (where) debe estar lo suficientemente acordado para no generar una actualización o borrado FULL y llegue a causar una transacción larga.

```
begin;
DELETE FROM  "informix". si_bpiusuarios
where empresa = '001'
and numcte = '0000009992222333';
 commit;

begin;
UPDATE  "informix". si_bpiusuarios
set fecha_registro = '05/20/2009'
where empresa = '001'
      and numcte = '0000009992222333';
commit;
```

Las sentencias de UPDATE (para actualizaciones) e INSERT (para inserciones) que lleven fechas deberán seguir el formato que se muestra a continuación **05/16/2009 – MM/DD/AAAA (MM = MES / DD = DIA / AAAA = AÑO).**

```
INSERT INTO "informix".si_bpiusuarios(empresa, numcte, id_status, folio_contrato, usuario, pass,
f_pass, pass1, f_pass1, pass2, f_pass2, pass3, f_pass3, f_status, f_ultimo_acceso, f_actualizacion,
suc_registro, f_registro) VALUES('001', '004271940', 10, '0104840T9U20', '', '', '', '', '', '', '', '', '', '', '',
'0116', '05/16/2009');

UPDATE  "informix". si_bpiusuarios
set fecha_registro = '05/16/2009'
where empresa = '001'
and numcte = '0000009992222333';
```

En los scripts que realicen operaciones TRUNCATE (para borrado de toda la información que contenga la tabla) deben colocar como primera instrucción **Begin** al comienzo de los bloques y **Commit** al final de los mismos, **siempre debe ser una sola instrucción a ser ejecutada; (No puede ir definida ninguna otra instrucción debajo de una orden truncate por que manda error)** para que quede de la siguiente manera:

Debe ser una sola sentencia.

```
begin;
INSERT INTO "informix".si_bpiusuarios(empresa, numcte, id_status, folio_contrato, usuario,
pass, f_pass, pass1, f_pass1, pass2, f_pass2, pass3, f_pass3, f_status, f_ultimo_acceso,
f_actualizacion, suc_registro, f_registro) VALUES('001', '004271940', 10, '0104840T9U20', '', '', '',
'', '', '', '', '', '', '', '', '', '0116', '05/16/2009');
commit;

begin;
SELECT empresa, numcte, id_status, folio_contrato, usuario, pass, f_pass, pass1, f_pass1,
pass2, f_pass2, pass3, f_pass3, f_status, f_ultimo_acceso, f_actualizacion, suc_registro,
f_registro
from "informix". si_bpiusuarios;
commit;

begin;
DELETE FROM "informix". si_bpiusuarios
where empresa = '001'
and numcte = '0000009992222333';
commit;

begin;
UPDATE "informix". si_bpiusuarios
set fecha_registro = '05/20/2009'
where empresa = '001'
and numcte = '0000009992222333';
commit;
```

Cargas Masivas (millones de registros)

Cuando a través de un proceso o shell se requiere la carga de información esta deberá ser mediante el uso del comando dbload que tiene la siguiente sintaxis.

Crear el archivo Shell

Debe comenzar con **dbload_** seguido del nombre de la tabla a la que se va a cargar la información con extensión .sh

Ejemplo: dbload_sd_edocta_sdos.sh

Debe contener el comando '**date'** para registrar la fecha y hora de inicio de la ejecución.
Omitir el comando "**nice –n -30**" para que corra con baja prioridad a nivel S. O.
La sintaxis del comando dbload

dbload -d nombre_basededatos –c nombre_tabla.com –l nombre_tabla.log –e numero_maximo_de_registros –n 1000 (para una carga bloqueando la tabla –k en forma esclusiva para una carga sin bloquear la tabla –r compartida)

Debe contener el comando '**date'** para registrar la fecha y hora de fin de la ejecución.
Llamada al comando '**dbaccess'** para hacer la actualización de estadísticas mediana una vez terminada la carga masiva de registros.

Ejemplo:

```
date
dbload -d bdicred -c sd_edocta_sdos.com -l sd_edocta_sdos.log -e 2500000 -n 1000 -r
date
dbaccess bdicred -<<EOF
update statistics medium for table sd_edocta_sdos;
EOF
date
```

Proporcionar el archivo de comandos que usa el '**dbload'** para poder insertar en la tabla destino el cual debe apuntar al Filesystem con espacio superior a 90 Gb:

El nombre del archivo .com debe ser nombre_tabla.com
Ejemplo:

sd_edocta_sdos.com

Debe contener lo siguiente.
nombre del archivo .unl que contiene los datos a cargar
el número de columnas que tiene la tabla (eso se obtiene generando el dbschema de la tabla: dbschema –d nombre_bd –t nombre_tabla –ss aparece esta información en las primeras dos líneas de texto)
el nombre de la tabla a la cual se le van a insertar los datos

Ejemplo:

```
FILE /RESPALDOSNEW/sd_edocta_sdos.unl DELIMITER '|' 34;
INSERT INTO sd_edocta_sdos;
```

Tomar en consideración que cualquier operación de tipo masivo (insercion / borrado / modificacion de millones de registros) requiere que los SPLs tengan programada la funcionalidad de hacer commit cada 1000 registros, esto con la intención de poder matar el proceso y volverlo a lanzar sin provocar un rollback masivo, que bloquee los recursos (bd, tablas, indices, memoria) por un tiempo excesivo ya sea en horario de operación de sucursales (OLTP) o de ejecución de cierres de módulos nocturnos (BATCH).

Vistas

Para reemplazar vistas (tabla logica) primero tienen que crear el script para respaldar la misma con la siguiente sintaxis.

El nombre del script debe indicar para que sirve por ejemplo:

resp_view_ *nombre_bd_nombre_vist.sh*

Y debe contener lo siguiente:

dbschema –d *nombre_bd* –t *nombre_vista* –ss *nombre_vista_resp.sql*

Ejemplo:

resp_view_*bdibi_tbl_view_cajeros*.sh

Y contiene dentro:

dbschema -d *bdibi* -t *tbl_view_cajeros* -ss *tbl_view_cajeros_resp*.sql

Crear el archivo script SQL siguiendo la misma logica.

reemp_view_*bdibi_tbl_view_cajeros*.sql

Que contendra las sentencias del DROP VIEW y el CREATE VIEW de la vista a ser reemplazada.

Ejemplo:

```
DROP VIEW "informix".tbl_view_cajeros;


CREATE VIEW "informix".tbl_view_cajeros
(fechatransaccion,fecha,numtarjeta) as
  SELECT x0.fechatransaccion ,x0.fecha ,x0.numtarjeta
from "informix".bi_cajeros x0;
```

MODIFICACION A UN ESQUEMA "ALTER TABLE"

Las sentencias ALTER (para modificar la estructura de una tabla), que vayan dirigidas a la misma base de datos deberán tener siempre al usuario informix entre comillas " ";

```
ALTER TABLE tabla ------| Opciones Básicas |------
```

Opciones Básicas

```
ADD Columna
ADD CONSTRAINT
ADD/DROP Columna
DROP CONSTRAINT
MODIFY
```

Así también deben colocar como primera instrucción **BEGIN;** al comienzo de los bloques y **COMMIT;** al final de los mismos, solo pueden indicar una sentencia alter a la vez.

```
begin;
ALTER TABLE nombre_tabla
ADD (campo1 DECIMAL(6,3) BEFORE campo2);
commit;
```

MODIFY

La sentencia MODIFY debe tener sus consideraciones: Cuando se modifica una columna, se eliminan todos los atributos asociados previamente a la columna (es decir, el valor predeterminado, la restricción de verificación de una columna o la restricción referencial).

Cuando desee que ciertos atributos de la columna permanezcan, como PRIMARY KEY, debe volver a especificar los atributos en la misma cláusula MODIFY.

Siempre debe ser considerado el volumen de la tabla, no se recomienda su uso para grandes cantidades de información (millones de registros).

Solicitar a personal de base de datos la revisión y/o VoBo del script .sql

```
begin;
ALTER TABLE nombre_tabla MODIFY (description LVARCHAR(3072));
commit;
```

Reorganización de Tablas

Cuando una tabla está muy fraccionada en varios segmentos en el mismo dbspace (reposositorio / contenedor de datos) es necesario reorganizarla para distribuirla circularmente en más de un dbspace de datos para mejorar los tiempos / costos de inserción / lectura – acceso la recomendación es que sean Dbspaces nuevos para mantener aislado el acceso hacia estas tablas exclusivamente en esos dbspaces de datos y sacarlas de Dbspaces de datos (datos00, datos01, datos02, datos03) muy accesados (calientes) por la operación diaria.

>=20,000,000 <=49,999,999 registros en al menos 5 Dbspaces
>= 50,000,000 registros en al menos 6 Dbspaces

Y en el caso de los indices crearlos en un Dbspace exclusivo para indices con tamaños de página de 16 Kb.

- El nombre de este Dbspace lo pueden solicitar mediante una solicitud de VoBo de Base de datos.
  - En caso de no existir, se solicitara la creación del mismo.

a) Crear la nueva tabla en distribución circular.

```
CREATE TABLE my_orders (
   order_num      SERIAL(1001),
   order_date    DATE,
   customer_num   INT,
   ship_instruct  CHAR(40),
   backlog        CHAR(1),
```

```
po_num        CHAR(10),
ship_date     DATE,
ship_weight   DECIMAL(8,2),
ship_charge   MONEY(6),
paid_date     DATE,
PRIMARY KEY (order_num),
FOREIGN KEY (customer_num) REFERENCES customer(customer_num))
FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2, dbspace3
```

b) Crear los índices en un dbspace particular.

create index "informix".idx_nombreIdx on "informix".nombre_tabla
(empresa,cuenta) using btree **in datos01;**

Creación de Bases de Datos

Como parte de los proyectos nuevos donde necesitan aislar los datos / información relativa a todas las tablas, llaves primarias / foráneas, triggers, vistas, índices, SPLs, etc. se debe crear un contenedor lógico de todos estos objetos en la base de datos de administración de la instancia (sysmaster), se recomienda crearla en Dbspaces de datos y Dbspaces de Índices propios para que no compita por los mismos recursos de almacenamiento.

Ejemplo de Sintaxis:

CREATE DATABASE nombre_bd IN nombre_dbspace_datos WITH BUFFERED LOG;

Ejemplo:

CREATE DATABASE vehículos IN datosnvo WITH BUFFERED LOG;

Sentencia UPDATE

La sentencia UPDATE modifica los valores de una o más columnas en las filas seleccionadas de una o varias tablas.

La sintaxis es la siguiente:

```
UPDATE - origen - SET - nbcolumna = expresión
       WHERE condición
```

- Origen.- puede ser un nombre de tabla, un nombre de consulta o una composición de tablas.

- La cláusula SET especifica qué columnas van a modificarse y qué valores asignar a esas columnas.
- nbcolumna, es el nombre de la columna a la cual queremos asignar un nuevo valor por lo tanto debe ser una columna de la tabla origen. El SQL estándar exige nombres sin cualificar pero algunas implementaciones (como por ejemplo el SQL de Microsoft Jet que estamos estudiando) sí lo permiten.
- La expresión en cada asignación debe generar un valor del tipo de dato apropiado para la columna indicada. La expresión debe ser calculable a partir de los valores de la fila que se está actualizando. *Expresión* no puede ser una subconsulta.

Ejemplo

```
UPDATE  si_bpiusuarios
set fecha_registro = '05/20/2009'
where empresa = '001'
and numcte = '0000009992222333'
```

Siempre se debe considerar el volumen de la tabla, NO se debe actualizar tablas con grandes cantidades de información (millones de registros) sin un filtro WHERE optimizado (Rango de información menor). El riesgo de generar una transacción larga es latente y generar problemas.

Descarga de Datos usando tablas externas

Cuando sea necesario realizar respaldo de información en / hacia archivos planos (ASCII) de manera paralela y mas rápido se pueden usar las tablas externas para descargarla.

a) Usar la sentencia CREATE EXTERNA TABLE *nombre_tabla_externa*
Para crear la tabla externa.

b) Usar la sentencia SAMEAS *nombre_tabla_origen_de_los datos*
Para que use la misma estructura (en cuanto al número de columnas separadas por el carácter "|") de la tabla origen de los datos.

c) Usar la sentencia USING
Para indicar el uso de los DATAFILES.

d) Usar la sentencia DATAFILES ("DISK:/ruta_completa/*nombre_archivo_datos*.dat"
Para indicar que se crean los archivos y direcciona la descarga Disco (DISK) usando la Ruta (Filesystem y directorios) y el nombre del archivo que contendrá la información.

e) Usar la sentencia REJECTFILE "/ ruta_completa/*nombre_archivo_datos*.rej"
Para indicar que se crean los archivos y direcciona la descarga Disco (DISK) usando la Ruta (Filesystem y directorios) y el nombre del archivo que contendrá la información rechazada / fallas de conversión / violaciones de contraints (llaves primarias / índices únicos) por el proceso (que no cumpla con el formato).

f) Usar la sentencia DELUXE
Para indicar que valide los registros, los contraints, que la tabla sea accesible por otros usuarios / procesos

g) Usar la sentencia INSERT INTO *nombre_tabla_externa* SELECT * FROM *nombre_tabla_origen*
Para realizar la selección de la tabla origen de los datos e insértala en la externa recién definida / creada.

h) Usar la sentencia DROP TABLE *nombre_tabla_externa*
Para indicar que se borre la referencia de esta tabla externa en la instancia

Ejemplo

```
CREATE EXTERNAL TABLE sd_movhis_ext
SAMEAS sd_movhis
USING (
  DATAFILES ("DISK:/resplogifx/external_table/sd_movhis_ext1.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext2.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext3.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext4.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext5.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext6.dat"),
  REJECTFILE "/resplogifx/external_table/sd_movhis_ext.rej",
  DELUXE
   );
INSERT INTO sd_movhis_ext SELECT * FROM sd_movhis;

DROP TABLE sd_movhis_ext;
```

NOTA: Solo como dato informativo descargo 100,305,223 de registros distribuidos en 6 archivos de datos en 8 minutos aprox (esto puede variar en relación al número de DATAFILES, cantidad total de

registros, la longitud / tamaño de los mismos, que el filesystem de descarga tenga habilitados Concurrente I/O, Directivo I/O a nivel S.O).

```
16716816 sd_movhis_ext1.dat
 16716096 sd_movhis_ext2.dat
 16718209 sd_movhis_ext3.dat
 16716512 sd_movhis_ext4.dat
 16719566 sd_movhis_ext5.dat
 16718024 sd_movhis_ext6.dat
 100305223 total
```

Para cargar estos datos la sentencia es similar solo cambia, la adición de la sentencia LOCK TABLE *nombre_table_destino_datos* IN SHARE MODE; (modo compartido) y el orden / ubicación del nombre de las tablas de origen y destino de los datos.

Ejemplo:

```
LOCK TABLE employee IN SHARE MODE;

CREATE EXTERNAL TABLE sd_movhis_ext
SAMEAS sd_movhis
USING (
  DATAFILES ("DISK:/resplogifx/external_table/sd_movhis_ext1.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext2.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext3.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext4.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext5.dat",
        "DISK:/resplogifx/external_table/sd_movhis_ext6.dat"),
  REJECTFILE "/resplogifx/external_table/sd_movhis_ext.rej",
  DELUXE
   );
INSERT INTO sd_movhis  SELECT * FROM sd_movhis_ext;

DROP TABLE sd_movhis_ext;
```

ARCHIVOS COMPRIMIDOS (A SER DESCOMPACTADOS EN AIX):

No se pueden recibir archivos comprimidos en formato ZIP ya que este tipo de formato es propio del sistema operativo WINDOWS para el sistema operativo AIX solo se pueden recibir archivos comprimidos en formatos .tar, .gzip. bzip2, compress

- Para esto tienen instalada en su PC la aplicación IZARC (o al menos WINRAR) que les permite elegir este tipo de formatos.
    - O comprimir el/los archivo(s) en sus ambientes de desarrollo / testing AIX y transferirlos a su carpeta de VERSIONES.

CREACION DE DIRECTORIOS, TRANSFERENCIAS / REEMPLAZO DE ARCHIVOS, CAMBIO DE PERMISOS, CREACION DE RELACIONES DE CONFIANZA, AUTORIZACION DE ACCESOS DE IP DE LAS PCS DE USUARIO VIA IPSEC, RESETEO DE PASSWORDS, VERIFICACION CUENTAS DE USUARIO Y PROCESOS CRON, NORMALIZACION DE MENUS, SHELLS, SCRIPTS SQL EN SERVIDORES AIX CENTRALES, DESARROLLO Y MAQUETAS:

Deben ser tramitados con el área de soporte técnico S.O. para AIX,