使用 ESQL/C 將 Informix Dynamic Server 中的游標最佳化

高效能技術

IBM® Informix® ESQL/C 具有許多功能，可減少伺服器與用戶端之間的資料流量，從而提高應用程式的效能。請詳閱本文，以瞭解各項功能，以及如何藉此達到最大效能。

前言

每個應用程式的主要目標，都包含達成效能一項。提升效能與最佳化，是所有應用程式的永恆任務，任何應用程式中，效能都扮演重要的角色。提高效能的方法之一，便是減少用戶端與伺服器之間的資料往返。IBM Informix ESQL/C 可協助您將游標最佳化，利用這些功能，達成較佳效能。

您可以使用下列方法，將游標的執行最佳化：

- 增加提取與插入緩衝區的大小
- 自動釋放游標
- 延遲 PREPARE 陳述式，直到執行 OPEN 陳述式爲止
- 使用 OPTOFC

請詳閱後續章節，以瞭解這些技術。

Back to top

增加提取與插入緩衝區的大小

若應用程式日常需要傳送與接收大量資料，則增加游標緩衝區的大小，有助於達成較佳效能，游標緩衝區可同時作爲提取緩衝區與插入緩衝區。

- *提取緩衝區*可保留來自選取或函數游標的資料。

資料庫伺服器從作用中查詢集傳回資料行後，ESQL/C 就會將這些資料行，儲存在提取緩衝區。

- *插入緩衝區*可保留插入游標的資料。

ESQL/C 將要插入的資料行儲存在插入儲存區，然後傳送整個緩衝區至資料庫伺服器，以供插入。

若要增加游標緩衝區的大小，您可以在用戶端環境中設定以下變數：

- 您可以在應用程式執行環境中，設定 BIG_FET_BUF_SIZE 環境變數，以將應用程式中所有游標緩衝區，設定成您所希望的大小。

  例如，以下指令可將 C-shell 環境中的 BIG_FET_BUF_SIZE 環境變數，設定成 40,000 Byte：

```
setenv BIG_FET_BUF_SIZE 40000
```

- 您可以在應用程式中設定 BigFetBufSize 廣域變數，以變更*游標之間 (from cursor to cursor)* 的游標緩衝區。

  每當 ESQL/C 建立游標緩衝區，緩衝區將是 BigFetBufSize 中目前給定的大小（若有設定）。

  舉例而言，以下的 C 程式碼將 BigFetBufSize 設定成 40000 Byte：

  ```
  BigFetBufSize = 40000;
  ```

  此行之後，每一個建立的游標都將爲 40000 Byte 的大小，直到 BigFetBufSize 的值變更爲止。

  BigFetBufSize 廣域變數，會改寫 BIG_FET_BUF_SIZE 環境變數中的所有值。

  BigFetBufSize 廣域變數定義於 ESQL/C sqlhdr.標頭檔，該檔案會自動納入所有 ESQL/C 程式中。

您可爲 BIG_FET_BUF_SIZE 或 BigFetBufSize 設定的最大值是 4194303。若您設定的值低於預設大小，或者高於系統上限值，新值便會忽略。若您未指定緩衝區大小，資料庫伺服器將會使用預設大小。BigFetBufSize 和 FetBufSize 相同，但 BigFetBufSize 游標緩衝區的上限值較高。

Back to top

自動釋放游標

自動釋放功能 (AUTOFREE) 是 ESQL/C 最佳化功能中的一項，可在 ESQL/C 應用程式從使用游標的資料庫伺服器提取資料行時，設法將網路資料流量降至最低。啓用 AUTOFREE 功能後，ESQL/C 可減少訊息要求的來回轉換數目，因爲無須傳送 FREE 陳述式給資料庫伺服器以供執行。相反地，資料庫伺服器在關閉此游標時，會自動釋放該游標。若該游標擁有相關聯的備妥陳述式，資料庫伺服器也會釋放該備妥陳述式。

ESQL/C 應用程式使用游標時，通常會傳送 FREE 陳述式給資料庫伺服器，以便在不需要該游標時，解除配置指派給選取游標的記憶體。此陳述式的執行，涉及在應用程式與資料庫伺服器之間進行訊息要求的來回轉換。

啓用 AUTOFREE 功能後，ESQL/C 可減少訊息要求的來回轉換數目，因爲無須執行 FREE 陳述式。資料庫伺服器關閉選取游標後，會自動釋放爲該選取游標配置的記憶體。假設您爲下列選取游標，啓用 AUTOFREE 功能：

```
/* Select cursor associated with a SELECT statement */
EXEC SQL declare sel_curs cursor for select * from customer;
```

資料庫伺服器關閉 sel_curs 游標後，會自動執行以下 FREE 陳述式的對等項目：

```
FREE sel_curs
```

若游標擁有相關聯的備妥陳述式，資料庫伺服器也會釋放配置給備妥陳述式的記憶體。假設您爲下列選取游標，啓用 AUTOFREE 功能：

```
/* Select cursor associated with a prepared statement */
EXEC SQL prepare sel_stmt 'select * from customer';
EXEC SQL declare sel_curs2 cursor for sel_stmt;
```

資料庫伺服器關閉 sel_curs2 游標後，會自動執行以下 FREE 陳述式的對等項目：

```
FREE sel_curs2;
FREE sel_stmt;
```

開啓或重新開啓游標之前，您必須啓用 AUTOFREE 功能。在啓用或停用這些模式的情況下，執行 SET AUTOFREE 陳述式時，唯有執行 SET AUTOFREE 陳述式，游標已宣告或開啓後，AUTOFREE 功能才會對該游標生效。

一旦對游標啓用 AUTOFREE 功能，便無法再次開啓游標；資料庫伺服器在第一次關閉游標後，就會*自動釋放*該游標。

啓用 AUTOFREE 功能

您可以使用下列方式之一，爲 ESQL/C 應用程式啓用 AUTOFREE 功能：

- 將 IFX_AUTOFREE 環境變數設定爲 1。

  若您使用 IFX_AUTOFREE 環境變數來啓用 AUTOFREE 功能，程式*執行緒中任何游標*關閉後，將自動釋放游標記憶體。
- 執行 SQL 陳述式 SET AUTOFREE。

  有了 SET AUTOFREE 陳述式，您即可爲*特定游標*啓用 AUTOFREE。您也可以爲特定連線或執行緒，啓用或停用此功能。

使用 SET AUTOFREE 陳述式

您可以使用 SQL 陳述式 SET AUTOFREE，來啓用與停用 AUTOFREE 功能。SET AUTOFREE 陳述式，容許您在 ESQL/C 程式中採取下列動作：

- 爲所有游標啓用 AUTOFREE 功能：

  ```
  EXEC SQL set autofree;
  ```

  ```
  EXEC SQL set autofree enabled;
  ```

  些陳述式是等同項目，因爲 SET AUTOFREE 陳述式的預設動作，就是啓用所有游標。
- 爲所有游標停用 AUTOFREE 功能：

  ```
  EXEC SQL set autofree disabled;
  ```
- 爲特定游標 ID 或游標變數，啓用 AUTOFREE 功能：

  ```
  EXEC SQL set autofree for cursor_id;
  ```

```
EXEC SQL set autofree for :cursor_var;
```

SET AUTOFREE 陳述式會改寫 IFX_AUTOFREE 環境變數中的所有值。

清單 1 中的程式碼片段使用 SET AUTOFREE 陳述式的 FOR 子句，以便僅針對 curs1 游標，啓用 AUTOFREE 功能。資料庫伺服器爲 curs1 執行 CLOSE 陳述式後，會自動釋放游標與備妥陳述式。curs2 游標與其備妥陳述式，則不會自動釋放。

清單 1：AUTOFREE 功能

```
EXEC SQL BEGIN DECLARE SECTION;
   int a_value;
EXEC SQL END DECLARE SECTION;

EXEC SQL create database tst_autofree;
EXEC SQL connect to 'tst_autofree';
EXEC SQL create table tab1 (a_col int);
EXEC SQL insert into tab1 values (1);

/* Declare the curs1 cursor for the slct1 prepared
 * statement */
EXEC SQL prepare slct1 from 'select a_col from tab1';
EXEC SQL declare curs1 cursor for slct1;

/* Enable AUTOFREE for cursor curs1 */
EXEC SQL set autofree for curs1;

/* Open the curs1 cursor and fetch the contents */
EXEC SQL open curs1;
while (SQLCODE == 0)
   {
   EXEC SQL fetch curs1 into :a_value;
   printf("Value is: %d\n", a_value);
   }

/* Once the CLOSE completes, the curs1 cursor is freed and
 * cannot be used again. */
EXEC SQL close curs1;
```

```
/* Declare the curs2 cursor for the slct2 prepared
 * statement */
EXEC SQL prepare slct2 from 'select a_col from tab1';
EXEC SQL declare curs2 cursor for slct2;

/* Open the curs2 cursor and fetch the contents */
EXEC SQL open curs2;
while (SQLCODE == 0)
   {
   EXEC SQL fetch curs2 into :a_value;
   printf("Value is: %d\n", a_value);
   }

/* Once this CLOSE completes, the curs2 cursor is still
 * available for use. It has not been automatically freed. */
EXEC SQL close curs2;

/* You must explicitly free the curs2 cursor and slct2
 * prepared statement.  */
EXEC SQL free curs2;
EXEC SQL free slct2;
```

Back to top

延遲 PREPARE 陳述式，直到執行 OPEN 陳述式爲止

ESQL/C 應用程式使用 PREPARE/DECLARE/OPEN 陳述式區塊來執行游標時，每個陳述式會涉及在應用程式與資料庫伺服器之間來回轉換的訊息要求。Deferred-PREPARE（延遲備妥）功能可減少一個來回轉換。啓用 Deferred-PREPARE 功能後，ESQL/C 可減少訊息要求的來回轉換數目，因爲無須傳送個別指令，以執行 PREPARE 陳述式。相對地，資料庫伺服器收到 OPEN 陳述式後，會自動執行 PREPARE 陳述式。

設定 DEFERRED_PREPARE

使用 SET DEFERRED_PREPARE 陳述式，來控制用戶端程序是否要延遲傳送 PREPARE 陳述式給資料庫伺服器，直到傳送 OPEN 或 EXECUTE 陳述式。只有 Informix Dynamic Server 支援此陳述式，此爲 ANSI/ISO SQL 標準的延伸。您只要擁有 ESQL/C，即可使用此陳述式。

清單 2：語法

```
```

```
                                              .-ENABLED------.
>>SET DEFERRED_PREPARE--+-+----------+-+----------------------><
                                               '-DISABLED-'
```

用法

根據預設，SET DEFERRED_PREPARE 陳述式會導致應用程式延遲傳送 PREPARE 陳述式給資料庫伺服器，直到執行 OPEN 或 EXECUTE 陳述式爲止。若要生效，PREPARE 陳述式得搭配其他陳述式，才能在用戶端與伺服器之間只傳送一個訊息來回轉換，而非兩個。此 Deferred-Prepare 功能會影響下列動態 SQL 陳述式系列。

- 搭配 FETCH 或 PUT 陳述式運作的 PREPARE、DECLARE、OPEN 陳述式
- 執行 EXECUTE 或 EXECUTE IMMEDIATE 陳述式後，使用 PREPARE

您可以爲 SET DEFERRED_PREPARE 指定 ENABLED 或 DISABLED 選項。
若您未指定選項，則預設值爲 ENABLED。以下範例依預設啓用 Deferred-Prepare 功能：

```
EXEC SQL set deferred_prepare;
```

ENABLED 選項會啓用應用程式中的 Deferred-Prepare 功能。以下範例明確指定 ENABLED 選項：

```
EXEC SQL set deferred_prepare enabled;
```

應用程式發出 SET DEFERRED_PREPARE ENABLED 後，會爲應用程式中後續的 PREPARE 陳述式，啓用 Deferred-Prepare 功能。接著，應用程式將表現以下行爲：

- 循序的 PREPARE、DECLARE 以及 OPEN，傳送 PREPARE 陳述式連同 OPEN 陳述式給資料庫伺服器。若備妥陳述式發生語法錯誤，在應用程式爲備妥陳述式宣告游標並開啓游標之前，不會傳回錯誤訊息。
- 循序的 PREPARE 以及 EXECUTE 會傳送 PREPARE 陳述式，連同 EXECUTE 陳述式給資料庫伺服器。若備妥陳述式包含語法錯誤，在應用程式試圖執行備妥陳述式之前，資料庫伺服器不會傳回錯誤訊息。

若在包含 DESCRIBE 陳述式的 PREPARE、DECLARE 以及 OPEN 陳述式區塊中啓用 Deferred-Prepare，則 DESCRIBE 陳述式必須跟在 OPEN 陳述式之後，而不是 PREPARE 陳述式之後。若 DESCRIBE 跟在 PREPARE 之後，DESCRIBE 陳述式將發生錯誤。
使用 DISABLED 選項，以停用應用程式中的 Deferred-Prepare 功能。以下範例指定 DISABLED 選項：

```
EXEC SQL set deferred_prepare disabled;
```

若您指定 DISABLED 選項，則執行 PREPARE 陳述式時，應用程式會將每一個 PREPARE 陳述式傳送給資料

庫伺服器。

SET DEFERRED_PREPARE 範例

以下程式碼片段（參閱清單 3），顯示 SET DEFERRED_PREPARE 陳述式與 PREPARE 以及 EXECUTE 陳述式區塊。在此情況下，資料庫伺服器將一併執行 PREPARE 與 EXECUTE 陳述式：

清單 3：SET DEFERRED_PREPARE 陳述式

```
main()
{

EXEC SQL BEGIN DECLARE SECTION;
    int a;
EXEC SQL END DECLARE SECTION;

EXEC SQL create database 'defered_prep';
printf("database : SQLCODE is %d\n", SQLCODE);

EXEC SQL create temp table x (a int);
printf("table : SQLCODE is %d\n", SQLCODE);

/* Enable Deferred-Prepare feature */
EXEC SQL set deferred_prepare enabled;

/* Prepare an INSERT statement */
EXEC SQL prepare ins_stmt from 'insert into x values (?)';
printf("prepare : SQLCODE is %d\n", SQLCODE);

a = 2;
EXEC SQL EXECUTE ins_stmt using :a;

if (SQLCODE)
    printf("EXECUTE : SQLCODE is %d\n", SQLCODE);

}
```

以下 sqliprint 輸出，就用戶端與伺服器之間的來回轉換，展現 deferred_prepare 所造成的差異。

清單 4：未設定 deferred_prepare 的 sqliprint 輸出

```
C->S (36)
         SQ_PREPARE
                  # values: 1
                  CMD.....: "insert into x values (?)" [24]
         SQ_NDESCRIBE
         SQ_WANTDONE
         SQ_EOT

S->C (70)
         SQ_DESCRIBE
                  Stmt Type...........: 6
                  Server Stmt Id......: 0
                  Estimated Cost......: 0
                  Size of output tuple: 4
                  # output fields.....: 1
                  Size of string table: 2
                  0) Field 'a'
                           Index into string table: 0
                           Starting offset in tuple: 0
                           Type...................: INT; NULLABLE
                           Length : 4 (0x4)
         SQ_INSERTDONE
                  0
         SQ_DONE
                  Warning..: 0x0
                  # rows...: 0
                  rowid....: 0
                  serial id: 0
         SQ_COST
                  estimated #rows: 1
                  estimated I/O..: 1
         SQ_EOT

C->S (22)
         SQ_ID
                  0
         SQ_BIND
                  # values: 1
```

```
                        0) Type.....: INT; NULLABLE
                        Indicator: NOT NULL
                        Precision: 0xa00
                        Data.....: 2

        SQ_EXECUTE
        SQ_EOT

S->C (40)
        SQ_INSERTDONE
                0
        SQ_DONE
                Warning..: 0x0
                # rows...: 1
                rowid....: 257
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT
C->S (4)
        SQ_TXSTATE
        SQ_EOT

C->S (8)
        SQ_DISCONNECT
                CURRENT
```

清單 5：設定 deferred_prepare 的 sqliprint 輸出

```
C->S (52)
        SQ_PREPARE
                # values: 1
                CMD.....: "insert into x values (?)" [24]
        SQ_NDESC_ID
        SQ_WANTDONE
        SQ_BIND
                # values: 1
```

```
                        0) Type.....: INT; NULLABLE
                        Indicator: NOT NULL
                        Precision: 0xa00
                        Data.....: 2

        SQ_EXECUTE
        SQ_EOT

S->C (108)
        SQ_DESCRIBE
                Stmt Type...........: 6
                Server Stmt Id......: 0
                Estimated Cost......: 0
                Size of output tuple: 4
                # output fields.....: 1
                Size of string table: 2
                0) Field 'a'
                        Index into string table: 0
                        Starting offset in tuple: 0
                        Type...................: INT; NULLABLE
                        Length : 4 (0x4)
        SQ_INSERTDONE
                0
        SQ_DONE
                Warning..: 0x0
                # rows...: 0
                rowid....: 0
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_INSERTDONE
                0
        SQ_DONE
                Warning..: 0x0
                # rows...: 1
                rowid....: 257
                serial id: 0
```

```
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT

C->S (4)
        SQ_TXSTATE
        SQ_EOT

C->S (8)
        SQ_DISCONNECT
                CURRENT
```

比較以上兩個 sqliprint 輸出，您可以看到啓用 deferred_prepare 時，會少一個來回轉換。若啓用 deferred_prepare，來到 PREPARE 陳述式時，用戶端不會傳送備妥陳述式，而是延遲傳送 PREPARE 陳述式，直到執行 EXECUTE 陳述式爲止。隨同 EXECUTE 陳述式傳送 PREPARE 陳述式，可以省下一次來回轉換。可以減少一個 C->S 與一個 S->C 訊息來回轉換。在 C->S (52) 中啓用延遲備妥（參閱清單 5），系統會傳送 PREPARE 與 EXECUTE。若未啓用延遲備妥，C->S (36) PREPARE 與 C->S (22) EXECUTE 的傳送將產生兩個來回轉換。

Back to top

使用 OPTOFC

ESQL/C 應用程式使用 DECLARE 與 OPEN 陳述式來執行游標時，每個陳述式會涉及在應用程式與資料庫伺服器之間來回轉換的訊息要求。最佳化—開啓—提取—關閉 (Optimize-OPEN-FETCH-CLOSE) 功能 (OPTOFC)，可減少兩個來回轉換。

OPTOFC 功能啓用後：

- ESQL/C 可以減少一個來回轉換，因爲不用透過獨立指令傳送 OPEN 陳述式。
  ESQL/C 執行 OPEN 陳述式後，並不會開啓游標，而會儲存 OPEN 陳述式中 USING 子句的所有輸入值。ESQL/C 執行初始 FETCH 陳述式後，就會隨同 FETCH 陳述式傳送輸入值。資料庫伺服器會開啓游標，並傳回該游標中的第一個值。
- ESQL/C 可以省下第二個來回轉換，因爲不用透過獨立指令傳送 CLOSE 陳述式。
  等資料庫伺服器觸及開啓游標的最後一個值，會在傳送最後一個值給用戶端應用程式後，自動關閉游標。因此，ESQL/C 不需要傳送 CLOSE 陳述式給資料庫伺服器。

啓用 OPTOFC 功能

OPTOFC 環境變數可啓用 OPTOFC 功能。您可以指派以下的值，給 OPTOFC 環境變數：

表 2：可能指派給 OPTOFC 環境變數的值

值	說明
1	這個值可啓用 OPTOFC 功能。若指定這個值，即可爲應用程式中所有執行緒的所有游標，啓用 OPTOFC 功能。
0	這個值可爲應用程式中的所有執行緒，停用 OPTOFC 功能。

OPTOFC 環境變數的預設值爲 0（零）。若您是從 Shell 設定這個環境變數，請務必在啓動 ESQL/C 應用程式之前進行設定。

在 UNIX 中

在 UNIX 作業系統上，您可以使用 putenv() 系統通話，來設定應用程式中的 OPTOFC（只要您的系統支援 putenv() 功能）。例如，以下對 putenv() 呼叫可啓用 OPTOFC 功能：

```
putenv("OPTOFC=1");
```

若要設定爲環境變數：

在 CSH Shell 中，setenv OPTOFC 1

在 KSH Shell 中，export OPTOFC=1

在 Windows 中

在 Windows 環境中，您可以使用 ifx_putenv() 函數。

您可以使用 putenv() 或 ifx_putenv()，爲每一個連線或執行緒，啓動或取消啓動 OPTOFC 功能。您必須在建立連線之前，先呼叫 putenv() 或 ifx_putenv() 函數。

範例：

清單 6：建立連線

```
main()
{

EXEC SQL BEGIN DECLARE SECTION;
    int a_value;
EXEC SQL END DECLARE SECTION;

EXEC SQL create database test;
EXEC SQL connect to 'test';
printf("sqlcode=%d\n",SQLCODE);

EXEC SQL create temp table tab1 (a_col int);
EXEC SQL insert into tab1 values (1);

EXEC SQL prepare slct1 from 'select a_col from tab1';
```

```
EXEC SQL declare curs1 cursor for slct1;
printf("sqlcode=%d\n",SQLCODE);

EXEC SQL open curs1;
printf("sqlcode=%d\n",SQLCODE);

while (SQLCODE == 0)
    {
    EXEC SQL fetch curs1 into :a_value;
    printf("Value is: %d\n", a_value);
    }

EXEC SQL close curs1;
printf("sqlcode=%d\n",SQLCODE);

EXEC SQL free slct1;
printf("sqlcode=%d\n",SQLCODE);

}
```

清單 7：未建立 OPTOFC 功能的 Sqliprint 輸出

```
. . . . . .
C->S (34)
         SQ_INFO
                  INFO_ENV
                           Name Length = 12
                           Value Length = 2
                           "SUBQCACHESZ"="10"
                  INFO_DONE
         SQ_EOT

S->C (2)
         SQ_EOT

. . . . . .

C->S (34)
```

```
        SQ_PREPARE
                # values: 0
                CMD.....: "select a_col from tab1" [22]
        SQ_NDESCRIBE
        SQ_WANTDONE
        SQ_EOT

S->C (62)
        SQ_DESCRIBE
                Stmt Type...........: 2
                Server Stmt Id......: 0
                Estimated Cost......: 0
                Size of output tuple: 4
                # output fields.....: 1
                Size of string table: 6
                0) Field 'a_col'
                        Index into string table: 0
                        Starting offset in tuple: 0
                        Type...................: INT; NULLABLE
                        Length : 4 (0x4)
        SQ_DONE
                Warning..: 0x0
                # rows...: 0
                rowid....: 0
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT

C->S (18)
        SQ_ID
                0
        SQ_CURNAME
                 "curs1" [5]
        SQ_OPEN
        SQ_EOT
```

```
S->C (2)
        SQ_EOT

C->S (12)
        SQ_ID
                0
        SQ_NFETCH
                Tuple buffer size: 4096
                Fetch Array  size: 0
        SQ_EOT

S->C (40)
        SQ_TUPLE
                # Warnings..: 0
                Tuple length: 4
        SQ_DONE
                Warning..: 0x0
                # rows...: 1
                rowid....: 257
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT

C->S (8)
        SQ_ID
                0
        SQ_CLOSE
        SQ_EOT

S->C (2)
        SQ_EOT

C->S (4)
        SQ_TXSTATE
        SQ_EOT
```

```
C->S (8)
         SQ_DISCONNECT
                  CURRENT
```

清單 8：已建立 OPTOFC 功能的 Sqliprint 輸出

```
. . . . . .
C->S (46)
         SQ_INFO
                  INFO_ENV
                           Name Length = 12
                           Value Length = 2
                           "SUBQCACHESZ"="10"
                           "OPTOFC"="1"
                  INFO_DONE
         SQ_EOT

S->C (2)
         SQ_EOT

. . . . . .

C->S (34)
         SQ_PREPARE
                  # values: 0
                  CMD.....: "select a_col from tab1" [22]
         SQ_NDESCRIBE
         SQ_WANTDONE
         SQ_EOT

S->C (62)
         SQ_DESCRIBE
                  Stmt Type...........: 2
                  Server Stmt Id......: 0
                  Estimated Cost......: 0
                  Size of output tuple: 4
                  # output fields.....: 1
                  Size of string table: 6
```

```
                0) Field 'a_col'
                        Index into string table: 0
                        Starting offset in tuple: 0
                        Type...................: INT; NULLABLE
                        Length : 4 (0x4)
        SQ_DONE
                Warning..: 0x0
                # rows...: 0
                rowid....: 0
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_EOT

C->S (28)
        SQ_ID
                0
        SQ_CURNAME
                 "curs1" [5]
        SQ_OPEN
        SQ_ID
                0
        SQ_NFETCH
                Tuple buffer size: 4096
                Fetch Array  size: 0
        SQ_EOT

S->C (42)
        SQ_TUPLE
                # Warnings..: 0
                Tuple length: 4
        SQ_DONE
                Warning..: 0x0
                # rows...: 1
                rowid....: 257
                serial id: 0
        SQ_COST
```

```
                estimated #rows: 1
                estimated I/O..: 1
        SQ_CLOSE
        SQ_EOT

C->S (4)
        SQ_TXSTATE
        SQ_EOT

C->S (8)
        SQ_DISCONNECT
                CURRENT
```

一併使用 OPTOFC 與 Deferred-PREPARE

若要在用戶端應用程式與資料庫伺服器之間，達到訊息數目最佳化，請同時使用 Optimize-OPEN-FETCH-CLOSE 功能與 Deferred-PREPARE 功能。OPTOFC 功能會延遲傳送 OPEN 訊息給資料庫伺服器，直到執行 FETCH 訊息傳送爲止。若您同時啓用 Deferred-Prepare 與 OPTOFC 功能，將會發生下列情況：

- 若備妥陳述式的文字包含語法錯誤，則在執行第一個 FETCH 陳述式之前，系統不會把錯誤訊息傳回應用程式。

請參閱清單 6 中範例 sqliprint 輸出：

清單 9：同時啓用 OPTOFC 與延遲備妥的 sqliprint 輸出

```
. . . . . .
C->S (46)
        SQ_INFO
                INFO_ENV
                        Name Length = 12
                        Value Length = 2
                        "SUBQCACHESZ"="10"
                        "OPTOFC"="1"
                INFO_DONE
        SQ_EOT

S->C (2)
        SQ_EOT
```

```
. . . . . .
C->S (52)
         SQ_PREPARE
                  # values: 0
                  CMD.....: "select a_col from tab1" [22]
         SQ_NDESC_ID
         SQ_WANTDONE
         SQ_CURNAME
                   "curs1" [5]
         SQ_OPEN
         SQ_NFETCH
                  Tuple buffer size: 4096
                  Fetch Array  size: 0
         SQ_EOT

S->C (102)
         SQ_DESCRIBE
                  Stmt Type...........: 2
                  Server Stmt Id......: 0
                  Estimated Cost......: 0
                  Size of output tuple: 4
                  # output fields.....: 1
                  Size of string table: 6
                  0) Field 'a_col'
                           Index into string table: 0
                           Starting offset in tuple: 0
                           Type...................: INT; NULLABLE
                           Length : 4 (0x4)
         SQ_DONE
                  Warning..: 0x0
                  # rows...: 0
                  rowid....: 0
                  serial id: 0
         SQ_COST
                  estimated #rows: 1
                  estimated I/O..: 1
         SQ_TUPLE
```

```
                # Warnings..: 0
                Tuple length: 4
        SQ_DONE
                Warning..: 0x0
                # rows...: 1
                rowid....: 257
                serial id: 0
        SQ_COST
                estimated #rows: 1
                estimated I/O..: 1
        SQ_CLOSE
        SQ_EOT

C->S (4)
        SQ_TXSTATE
        SQ_EOT

C->S (8)
        SQ_DISCONNECT
                CURRENT
```

以上 sqliprint 輸出可以看到，備妥、開啓與提取等三個陳述式，透過一個封包/訊息一併傳送給伺服器，如此可減少兩個來回轉換。

- 若陳述式文字中存在語法錯誤，在應用程式執行 FETCH 之前，資料庫伺服器不會將錯誤傳回應用程式。ESQL/C 執行 FETCH 陳述式之前，不會傳送 PREPARE、DECLARE 以及 OPEN 陳述式給資料庫伺服器。因此，在資料庫伺服器執行 FETCH 陳述式之前，這些陳述式所產生的任何錯誤並不存在。
- 您必須使用特殊的 GET DESCRIPTOR 陳述式，爲備妥陳述式取得 DESCRIBE 資訊。
  DESCRIBE 陳述式的一般用法，是在 PREPARE 之後執行，以判定備妥陳述式的相關資訊。然而，在同時啓用 OPTOFC 與 Deferred-PREPARE 功能的情況下，ESQL/C 觸及 FETCH 陳述式之前，不會傳送 DESCRIBE 陳述式給資料庫。爲了取得備妥陳述式的相關資訊，ESQL/C 會執行類似於 SET DESCRIPTOR 陳述式的陳述式，以取得資料類型、長度與其他系統描述子欄位，供備妥陳述式使用。接著，您便可在 FETCH 後面使用 GET DESCRIPTOR 陳述式，以取得這類資訊。
  此外，若資料類型爲內建資料類型，SQL/C 只能在 GET DESCRIPTOR 陳述式中的主變數上執行資料轉換。若爲不透明資料類型與複式資料類型（收集與行類型），則資料庫伺服器一律使用原生格式，將資料傳回用戶端應用程式。接下來，您可以在 GET DESCRIPTOR 陳述式後面，執行此資料的資料轉換。例如，資料庫伺服器使用其內部（二進位）格式，從不透明類型欄傳回資料。因此，您的 ESQL/C 程式必須在執行 GET DESCRIPTOR 陳述式時，將欄資料放入變動二進位（或固定二進位）主變數中。變動二進位與固定二進位資料類型，會使用其內部格式來保留不透明類型資料。您無法使用 lvarchar 主

變數，因爲 ESQL/C 無法將不透明類型資料，從其內部格式（即從資料庫伺服器接收的格式），轉換成外部 (lvarchar) 格式。

- 同時啓用 Deferred-PREPARE 與 OPTOFC 功能時，FetArrSize 功能將無法運作。因爲這兩項功能同時啓用時，ESQL/C 要等完成 FETCH 之後，才會知道資料行的大小。但到那時才使用 FetArrSize 值來調整提取緩衝區，爲時已晚。

Back to top

總結

閱畢本文後，您將能執行以下作業：

- 修改應用程式層次與廣域層次的緩衝區大小
- 配置、計劃與使用 IFX_AUTOFREE、BIG_FET_BUF_SIZE 與 OPTOFC 等參數
- 爲單一與多重游標啓用應用程式中的 AUTOFREE 功能
- 爲單一與多重游標啓用應用程式中的 DEFERRED_PREPARE 功能
- 使用與配置 OPTOFC 變數以提升效能
- 同時使用延遲備妥與 OPTOFC 來執行效能最佳化

資源

學習

- developerWorks Informix 專區：尋找文章與指導教學，並連接其他資源以拓展您的 Informix 技能。
- developerWorks Information Management 專區：進一步瞭解 Information Management。尋找技術文件、入門文章、教育、下載、產品資訊和更多內容。
- 透過 developerWorks 技術活動與網路廣播隨時獲得最新消息。
- 技術書店：瀏覽這些與其他技術主題的相關書籍。

取得產品與技術

- Informix Dynamic Server：下載免費的 IDS 試用版。
- 使用 IBM 試用軟體來建置您的下一個開發專案，您可以從 developerWorks 直接下載。

討論

- 參加討論區。
- 參與 developerWorks 部落格以及 developerWorks 社群。

關於作者

Kalyani Thummapudi 是 IBM 印度軟體實驗室的認證 Infomix 專家，專責處理 Informix 用戶端元件 ESQL/C。