bite-sized.sql

SQL: INTRO TO CURSORS

Owen Price
linkedin.com/in/owenhprice

If you can avoid it: DON’T USE CURSORS

Owen Price
linkedin.com/in/owenhprice

1

Performance

Cursors can be slower and less efficient than other SQL constructs like set-based operations.

Cursors work by processing each row one by one, which can be resource-intensive and time-consuming, especially if you're dealing with a large dataset.

Owen Price
linkedin.com/in/owenhprice

2

Locking

Cursors can also cause locking issues.

When a cursor is open, the database engine locks the rows that are being processed, which can cause contention with other queries that are trying to access the same rows.

Owen Price
linkedin.com/in/owenhprice

3

Complexity

Cursors can make SQL code more complex and harder to read and maintain.

Code that uses cursors tends to be longer and more convoluted than code that uses set-based operations.

Owen Price
linkedin.com/in/owenhprice

4

Memory usage

Cursors can consume a lot of memory.

Each time you fetch a row from a cursor, the database engine has to allocate memory to hold that row, which can add up quickly if you're processing a lot of rows.

Owen Price
linkedin.com/in/owenhprice

That said, there are always exceptions!

Owen Price
linkedin.com/in/owenhprice

Suppose you want to add a column called ‘added’ to each table in a database schema.

Owen Price
linkedin.com/in/owenhprice

The new column will record the current timestamp when a new row is inserted.

Owen Price
linkedin.com/in/owenhprice

bite-sized.sql

SQL SERVER

Owen Price
linkedin.com/in/owenhprice

bite-sized.sql

E.g. SQL Server

```sql
DECLARE @tableName varchar(255)
DECLARE @sql varchar(max)

DECLARE curTable CURSOR FOR
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
AND TABLE_SCHEMA = 'MySchema';

OPEN curTable
FETCH NEXT FROM curTable INTO @tableName

WHILE @@FETCH_STATUS = 0
BEGIN
  SET @sql = 'ALTER TABLE MySchema.'
             + @tableName
             + ' ADD COLUMN added DATETIME2'
             + ' DEFAULT CURRENT_TIMESTAMP'
  EXEC (@sql)

  FETCH NEXT FROM curTable INTO @tableName
END

CLOSE curTable
DEALLOCATE curTable
```

Owen Price
linkedin.com/in/owenhprice

E.g. SQL Server - 1

```
DECLARE @tableName varchar(255)
DECLARE @sql varchar(max)

DECLARE curTable CURSOR FOR
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
AND TABLE_SCHEMA = 'MySchema';

OPEN curTable
FETCH NEXT FROM curTable INTO @tableName

...
```

Here we declare the cursor by providing it a (small) dataset of table names from the MySchema schema.

We use the cursor to iterate over the rows in the query.

We open the cursor, then fetch the first table name into the @tableName variable

Owen Price
linkedin.com/in/owenhprice

E.g. SQL Server - 2

@@FETCH_STATUS returns the status of the last FETCH statement on the current connection. Zero means the previous fetch statement was successful (so the loop can continue).

We build the ALTER TABLE statement using the @tableName, then execute it.

```
...

WHILE @@FETCH_STATUS = 0
BEGIN
  SET @sql = 'ALTER TABLE MySchema.'
             + @tableName
             + ' ADD added DATETIME2'
             + ' DEFAULT CURRENT_TIMESTAMP'
  EXEC (@sql)

  FETCH NEXT FROM curTable INTO @tableName
END

CLOSE curTable
DEALLOCATE curTable
```

We fetch the next row from the cursor into the @tableName variable. The loop continues at WHILE @@FETCH_STATUS = 0.

When we're finished, we must always be careful to both CLOSE and DEALLOCATE the cursor.

When we've altered the last table, the @@FETCH_STATUS after this FETCH will be -1 (the row was beyond the end of the result set), and the loop will stop.

Owen Price
linkedin.com/in/owenhprice

bite-sized.sql

PostgreSQL

Owen Price
linkedin.com/in/owenhprice

E.g. PostgreSQL

```
DO $$
DECLARE
  a_table text;
  sql_text text;
BEGIN
  FOR a_table IN
    SELECT TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_TYPE = 'BASE TABLE'
    AND TABLE_SCHEMA = 'my_schema'
  LOOP
    sql_text := 'ALTER TABLE my_schema.'
                || a_table
                || ' ADD COLUMN added TIMESTAMP'
                || ' DEFAULT CURRENT_TIMESTAMP';
    EXECUTE sql_text;
  END LOOP;
END;
$$;
```

The *FOR variable IN query* syntax is simpler, but it performs the same purpose.

Notably, there are no FETCH statements in pl/pgsql, just statements within *LOOP…END LOOP;*

Owen Price
linkedin.com/in/owenhprice

bite-sized.sql

MySQL

Owen Price
linkedin.com/in/owenhprice

bite-sized.sql

E.g. MySQL

```sql
DROP PROCEDURE IF EXISTS temp_proc;
DELIMITER $$
CREATE PROCEDURE temp_proc()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a_table VARCHAR(255);
  DECLARE cur CURSOR FOR
    SELECT TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_TYPE = 'BASE TABLE'
    AND TABLE_SCHEMA = 'mydatabase';
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;

  read_loop: LOOP
    FETCH cur INTO a_table;
    IF done THEN
      LEAVE read_loop;
    END IF;
    SET @sql = CONCAT('ALTER TABLE mydatabase.',
                      a_table,
                      ' ADD COLUMN added TIMESTAMP',
                      ' DEFAULT CURRENT_TIMESTAMP');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
  END LOOP;

  CLOSE cur;
END$$
DELIMITER ;

CALL temp_proc();
DROP PROCEDURE IF EXISTS temp_proc;
```

Owen Price
linkedin.com/in/owenhprice

E.g. MySQL - 1

MySQL doesn't allow cursors outside of stored procedures, so we must wrap it in a stored procedure.

```
DROP PROCEDURE IF EXISTS temp_proc;
DELIMITER $$
CREATE PROCEDURE temp_proc()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a_table VARCHAR(255);
  DECLARE cur CURSOR FOR
    SELECT TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_TYPE = 'BASE TABLE'
    AND TABLE_SCHEMA = 'mydatabase';
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
...
```

MySQL databases are a single schema

This statement says that if a FETCH statement can't find another row in the cursor, set the done variable to TRUE.

This will be used to exit the loop when all the work is done, as you'll see on the next page.

Owen Price
linkedin.com/in/owenhprice

E.g. MySQL - 2

The loop is labelled read_loop. It begins with LOOP and ends with END LOOP;

Each iteration begins with trying to fetch another row into the a_table variable.
If there are no more rows, the continue handler sets done to TRUE and the loop exits.

```
  read_loop: LOOP
    FETCH cur INTO a_table;
    IF done THEN
      LEAVE read_loop;
    END IF;
    SET @sql = CONCAT('ALTER TABLE mydatabase.',
                      a_table,
                      ' ADD COLUMN added TIMESTAMP',
                      ' DEFAULT CURRENT_TIMESTAMP');
    PREPARE stmt FROM @sql;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
  END LOOP;

  CLOSE cur;
END$$
DELIMITER ;

CALL temp_proc();
DROP PROCEDURE IF EXISTS temp_proc;
```

In MySQL, dynamic SQL statements must first be prepared before they are executed.

The last statement in the procedure is to close the cursor.

After the procedure is created, it’s executed with the keyword and then dropped when the work is done.

Owen Price
linkedin.com/in/owenhprice

CURSORS:

1. Avoid if possible – use set-based logic instead

2. Can be useful for iterating over database objects

3. Be sure you understand how to exit and deallocate the cursor before using – test with a PRINT operation first!

Owen Price
linkedin.com/in/owenhprice