Базы данных

Представления и курсоры

Представления

Представление (view) – это псевдотаблица, содержимое которой определяется SELECT-запросом.

Представление – средство реализации ***внешних моделей БД***

Содержимое представления формируется в момент обращения

В запросах таблицы и представления трактуются одинаково, поэтому их имена не должны совпадать

Использование представлений:

- Для ***группировки столбцов*** разных таблиц в виде одного объекта
- Для ***ограничения доступа*** пользователей к определенным ***строкам*** таблиц
- Для ***ограничения доступа*** пользователей к определенным ***столбцам*** таблиц
- Для предоставления результатов ***преобразования данных***

Создание представлений

CREATE VIEW <имя представления>
[(<столбец> [,…n])]
AS <оператор SELECT>

Список столбцов представления (если он есть) должен соответствовать списку столбцов запроса.

В запросе нельзя использовать ORDER BY, но можно GROUP BY

```
CREATE TABLE Employee
(
t_num INT,
name NVARCHAR(30),
position NVARCHAR(30),
salary DECIMAL,
department NVARCHAR(20)
)
```

Примеры представлений

Горизонтальное представление
(столбцы представления определяются
по умолчанию)

```
CREATE VIEW SalesDepartment
AS
SELECT  * FROM Employee
WHERE department = ‘отдел продаж’
```

Использование

```
SELECT name, salary FROM SalesDepartment
```

Примеры представлений

Вертикальное представление

```
CREATE VIEW Tabel
AS
SELECT  t_num, name, position, department
FROM Employee
```

Использование

```
SELECT name, department FROM Tabel
WHERE position = ‘начальник отдела’
```

Многотабличное представление

```
CREATE VIEW Debtors
(isbn, title, reader_id, last_name, home_phone,
    work_phone, date_back)
AS
SELECT p.isbn, title, r.reader_id, last_name, home_phone,
    work_phone, date_out+30
FROM Publications p JOIN Exemplars e ON p.isbn = e.isbn
JOIN Readers r ON e.Reader_id = r.Reader_id
WHERE e.date_out + 30 < GetDate()
```

Использование

```
SELECT DISTINCT last_name FROM Debtors WHERE home_phone IS NULL
```

Преобразование данных

```
CREATE VIEW DeadLines
([name], [count], dead_line)
AS
SELECT last_name + ' ' + first_name, COUNT(*),
   MAX(e.date_out) + 30
FROM Readers r JOIN Exemplars e ON
   r.reader_id = e.reader_id
GROUP BY r.reader_id, last_name, first_name
```

Использование:

```
SELECT * FROM DeadLines WHERE dead_line > GETDATE()
```

Модификация данных

Ограничения:

- В запросе не должно быть DISTINCT
- Изменяться могут поля только одной таблицы
- Каждое имя в списке параметров представления должно быть простой ссылкой на столбец таблицы
- В предложении WHERE не должно быть вложенных запросов
- Не должна использоваться группировка

Ограничения можно обойти, используя триггеры типа INSTEAD OF к представлению

Пример:
UPDATE SalesDepartment SET salary = 1000 where t_num = 2

Управление представлениями

- Изменение представления:
  ALTER VIEW <как в CREATE VIEW>
- Переименование: хранимой процедурой sp_rename, например,
  EXEC sp_rename <OldName>, <NewName>, <OBJECT>
- Удаление:
  DROP VIEW <имя представления>

Курсоры

Курсор – средство работы клиентских приложений с отдельными строками результирующего набора данных

Использование:

- Результирующий набор данных слишком большой для хранения на стороне клиентского приложения
- Требуется обращение к отдельным строкам результирующего набора данных по номеру
- Реализация сложных алгоритмов обработки данных, в которых результаты могут зависеть от нескольких строк

Реализации курсоров:

- Курсоры Transact-SQL: элемент языка Transact-SQL, доступны для использования в скриптах и хранимых процедурах. Реализуются на сервере баз данных
- Курсоры сервера: доступны клиентским приложениям в виде API. Используются реализациями различных технологий, например, ODBC, OLE DB и т.д.
- Курсоры клиента: реализуются самим клиентским приложением для организации доступа к данным, переданным от сервера БД к клиенту.

Типы и поведение курсоров

Типы курсоров:

- Статические (курсор моментального снимка). Результат запроса сохраняется в БД tempdb. Изменение данных не влияет на ранее созданную копию данных, а изменения данных в курсоре не фиксируются в источнике.
- Динамические. Выборка данных выполняется каждый раз при получении строки курсора. Допустимы операции изменения и удаления данных.

Поведение курсоров:

- Последовательные (forward-only): доступ к данным только последовательно от первой стоки – к последней. Частный случай динамического курсора.
- Прокручиваемые (scrollable): допускается переход к любой строке в любом направлении.

Управление курсорами

- Создание (объявление) курсора. В памяти создается объект (переменная) заданного типа.
- Открытие курсора. Курсор наполняется данными, которые хранятся в tempdb.
- Выборка и изменение данных при помощи курсора.
- Закрытие курсора. Освобождается tempdb от данных.
- Освобождение (уничтожение) курсора. Объект курсора удаляется из памяти и повторно не может быть использован.

Действия с курсорами аналогичны действиям с файловыми переменными в алгоритмических языках

Создание (объявление) курсора

Вариант стандарта ANSI SQL-92

DECLARE <имя курсора>
[INSENSITIVE] [SCROLL]
CURSOR FOR <оператор SELECT>
[FOR { READ ONLY | UPDATE
[OF <имя столбца> [,...n]] }]

INSENSITIVE – статический (копия в tempdb), иначе – динамический

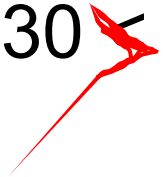
SCROLL – прокручиваемый

READ ONLY – только для чтения

UPDATE – разрешается изменение либо всех столбцов, либо только явно указанных

Создание (объявление) курсора

```
DECLARE DebtorCursor
   INSENSITIVE CURSOR FOR
SELECT p.title, r.reader_id, r.last_name
FROM dbo.Publications AS p INNER JOIN
  dbo.Exemplars AS e ON p.isbn = e.isbn INNER
    JOIN
  dbo.Readers AS r ON e.reader_id = r.reader_id
WHERE (e.date_out + 30 < GETDATE())
FOR READ ONLY
```



Открытие курсора

OPEN [GLOBAL] <имя курсора>
глобальные курсоры – для обмена данными

OPEN DebtorCursor

Считывание данных

Упрощенный синтаксис

FETCH [[NEXT | PRIOR | FIRST | LAST | ABSOLUTE *n* | RELATIVE *n*]

FROM] { [GLOBAL] <*имя курсора*> }

[INTO **@***variable_name* [,...*n*]]

Функция @@FETCH_STATUS возвращает статус выполнения последней операции FETCH

Закрытие и удаление курсора

- CLOSE <имя курсора>
  После закрытия курсор можно повторно использовать (после OPEN), но данные будет другие
- DEALLOCATE <имя курсора>

Пример считывания данных

```
DECLARE DebtorCursor
        INSENSITIVE CURSOR FOR
SELECT p.title, r.reader_id, r.last_name
FROM dbo.Publications AS p INNER JOIN
  dbo.Exemplars AS e ON p.isbn = e.isbn INNER JOIN
  dbo.Readers AS r ON e.reader_id = r.reader_id
WHERE     (e.date_out < GETDATE())
FOR READ ONLY

DECLARE @title NVARCHAR(30), @readerId INT, @lastName NVARCHAR(30)

OPEN DebtorCursor

FETCH NEXT FROM DebtorCursor INTO @title, @readerId, @lastName

WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT @title + ' ' + STR(@readerId) + ' ' + @lastName
  FETCH NEXT FROM DebtorCursor INTO @title, @readerId, @lastName
END

CLOSE DebtorCursor
DEALLOCATE DebtorCursor
```

Изменение данных

Для изменения и удаления данных используются особый синтаксис команд **UPDATE** и **DELETE**, у которых в части WHERE указывается курсор

Ограничения:

- запрос в курсоре должен быть только из одной таблицы
- в запросе не должно быть ORDER BY, DISTINCT, GROUP BY
- в UPDATE могут изменяться только те столбцы, которые были указаны в объявлении курсора

Пример изменения данных

```
DECLARE ShortTimeCursor CURSOR FOR
SELECT date_out
FROM Exemplars
WHERE     (date_out < GETDATE() -1)
FOR UPDATE

OPEN ShortTimeCursor

FETCH NEXT FROM ShortTimeCursor

WHILE @@FETCH_STATUS = 0
BEGIN
  UPDATE Exemplars SET date_out = date_out + 1
      WHERE CURRENT OF ShortTimeCursor
  FETCH NEXT FROM ShortTimeCursor
END

CLOSE ShortTimeCursor
DEALLOCATE ShortTimeCursor
```

Удаление данных

```
DELETE Exemplars
    WHERE CURRENT OF ShortTimeCursor
```