Лекция 4.

Язык SQL - продолжение

Условная логика

Условная логика в SQL

- Стандартный оператор CASE … WHEN … END.
- Встроенная функция, имитирующая if-then-else. Зависит от СУБД, в SQLite функция IIF().

Условный оператор CASE

Проверяет условие и возвращает один из возможных вариантов.

Первая форма:	Вторая форма:

CASE
  WHEN условие_1
  THEN возвращаемое_значение_1
  …
  WHEN условие_N
  THEN возвращаемое_значение_N
  [ELSE возвращаемое_значение]
END
	CASE проверяемое_значение
	  WHEN сравниваемое_значение_1
	  THEN возвращаемое_значение_1
	  …
	  WHEN сравниваемое_значение_N
	  THEN возвращаемое_значение_N
	  [ELSE возвращаемое_значение]
	END

Можно использовать в операторах SELECT, INSERT, UPDATE, DELETE.

SELECT: условный оператор CASE

Проверяет условие и возвращает один из возможных вариантов.

```
SELECT id, name AS 'Фамилия',
  CASE city
    WHEN 'Саранск' THEN 'Да'
    ELSE 'Нет'
  END 'Земляк'
FROM customer;

SELECT id, name AS 'Фамилия',
  CASE
    WHEN rating >= 300 THEN 'Молодец'
    WHEN rating >= 200 THEN 'Неплохо'
    ELSE 'Старайся больше'
  END 'Оценка по рейтингу'
FROM customer;
```

Условный оператор CASE

Возвращаемым значение может быть результат подзапроса.

```
SELECT v.name,
  CASE
    WHEN v.city='Саранск' THEN
      (SELECT count(*) FROM `order` o
       WHERE v.id=o.vendor_id)
    ELSE 'Не из Саранска'
  END sales
FROM vendor v;
```

Аналитические функции

Структура оператора SELECT

SELECT	Определяет, какие столбцы включить в результирующий набор
FROM	Определяет таблицы, из которых выбираются данные, и которые нужно соединить друг с другом
WHERE	Отсеивает ненужные данные
GROUP BY	Используется для группировки строк по общим значениям столбцов
HAVING	Отсеивает ненужные группированные данные
ORDER BY	Сортирует строки результирующего набора по одному или нескольким столбцам

SELECT … FROM … WHERE … GROUP BY … HAVING … ORDER BY …

В SELECT можно включать:

- Столбцы таблиц
- Литералы (числа строки)
- Вычисляемые выражения
- Вызов встроенных и пользовательских функций

SELECT 10 ‘Просто число’, **name** ‘Имя’, **Length(city)** ‘Длина имени города’, **percent || ‘%’** ‘Процент’ **FROM** Vendor

Предложение FROM может отсутствовать

Аналитические запросы

Результирующий набор строк, полученный после выполнения всех шагов запроса (соединение, фильтрация, группировка, сортировка), можно дополнительно анализировать и добавить дополнительные столбцы.

Можно разбить результирующий набор данных на **окна** - набор строк, в которых происходит вычисление функции. Окна данных могут содержать от одной строки до всех строк из результирующего набора данных.

Результаты работы оконных функций добавляются к выборке как еще одно поле.

Оконные функции

функция(выражение) **OVER** ([PARTITION BY выражения] [ORDER BY выражения])

Оконные функции:
ROW_NUMBER(), RANK(), FIRST_VALUE(), LAST_VALUE(), …
Также могут использоваться функции агрегирования SUM(), MIN, MAX, …

```
SELECT *, ROW_NUMBER() OVER () AS number FROM customer;

SELECT *, ROW_NUMBER() OVER (ORDER BY rating DESC) AS rating_place FROM customer
ORDER BY name;

SELECT *, RANK() OVER (ORDER BY rating DESC) AS rank FROM customer ORDER BY name;

SELECT *, MAX(rating) OVER () FROM customer;

SELECT name, city, rating, rating-AVG(rating) OVER () AS delta_rating FROM customer;

SELECT *, COUNT(*) OVER (PARTITION BY city) AS customers_in_city FROM customer;
```

Объединение результатов запросов

UNION

Объединение строк из нескольких результирующих наборов данных.

- Количество столбцов в каждом запросе одно и то же.
- Столбцы в каждом запросе имеют совместимые типы.
- Имена столбцов в первом запросе задают имена для всего объединения.
- ORDER BY применяется ко всему набору.
- GROUP BY и HAVING применяются к каждому подзапросу.

Вывести всех покупателей и продавцов с указанием их роли и города, где они живут. Сортировать по имени города и фамилии человека.

```
SELECT name, city, 'Продавец' AS 'Роль' FROM vendor
UNION
SELECT name, city, 'Покупатель' FROM customer
ORDER BY city, name;
```

UNION и UNION ALL

UNION убирает в результирующем наборе повторяющиеся строки.

UNION ALL оставляет все строки.

```
SELECT 1, 2
  UNION
SELECT 1, 2
  UNION
SELECT 3, 4;
```

```
1 | 2
——-
1 | 2
3 | 4
```

```
SELECT 1, 2
  UNION ALL
SELECT 1, 2
  UNION ALL
SELECT 3, 4;
```

```
1 | 2
——-
1 | 2
1 | 2
3 | 4
```

Представления (view)

Представления

Объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.

- Упрощение запросов.
- Гибкая настройка прав доступа к данным (права даются не на таблицу, а на представление).
- Разделение логики хранения данных и программного обеспечения.

CREATE VIEW имя [столбцы]
AS выборка

```
CREATE VIEW orders_with_names
AS SELECT `order`.order_date, customer.name AS customer_name,
vendor.name AS vendor_name, `order`.summa
FROM `order`, customer, vendor
WHERE `order`.customer_id = customer.id AND `order`.vendor_id =
vendor.id;
```

Обобщенные табличные выражения (CTE)

Обобщенные табличные выражения

Common table expressions, CTE

CTE - именованный подзапрос, указываемый в предложении WITH в верхней части запроса SELECT, INSERT, UPDATE или DELETE.
Действует аналогично временному представлению (view).

- Запрос может содержать несколько CTE.
- CTE может обращаться к другому CTE, определенному над ним.

Обычные и рекурсивные CTE

Обычные CTE действуют как представления (view) с учетом того, что к CTE можно обращаться только в том запросе, где они объявлены. Заменяют подзапросы, позволяя упростить структуру запроса.

Рекурсивные CTE (обращаются сами к себе) позволяют генерировать последовательности значений, выполнять сложные преобразования данных, работать с иерархическими данными.

Рекурсивные CTE

```
WITH [RECURSIVE] cte-table-name AS (
   initial-select
   UNION [ALL]
   recursive-select)
cte-select
```

```
WITH RECURSIVE ten(x) AS (
   SELECT 1
     UNION ALL
   SELECT x+1 FROM ten WHERE x<10)
SELECT * FROM ten;
```

x
1
2
3
5
6
7
8
9
10

Рекурсивные CTE

```
WITH [RECURSIVE] cte-table-name AS (
  initial-select
  UNION [ALL]
  recursive-select)
cte-select
```

1. Выполняется начальная выборка (initial-select), результат помещается в очередь.
2. Пока в очереди есть элементы:
   a. Из очереди извлекается одна строка S.
   b. Строка S добавляется в рекурсивную таблицу.
   c. Выполняется рекурсивная выборка (recursive-select), при этом считается, что рекурсивная таблица состоит из одной строки S. Результат выборки добавляется в очередь.

LIMIT в рекурсивном запросе определяет максимальное число строк, которые добавляются в рекурсивную таблицу на шаге 2b.