Иерархия в SQL

Способы представления иерархических данных

- родители-потомки
- тип **hierarchyid**
- XML

Родители-потомки

```
CREATE TABLE Parent_Child (
    Id INT PRIMARY KEY,
    Par_id INT REFERENCES Parent_Child(Id),
    Name Char(20),
    …
    )
```

Найти каждому его руководителя

```
SELECT *
FROM Parent_Child C JOIN Parent_Child P
ON C. Par_id = P.Id
```

Как найти всех подчиненных?

Обобщенные табличные выражения (CTE)

- Обобщенные табличные выражения (CTE) можно представить себе как временные результирующие наборы, определенные в области выполнения единичных инструкций SELECT, INSERT, UPDATE, DELETE или CREATE VIEW.
- CTE не сохраняются в базе данных в виде объектов, время их жизни ограничено продолжительностью запроса.
- CTE могут ссылаться сами на себя, а на них один и тот же запрос может ссылаться несколько раз.

Структура CTE

```
WITH expression_name  ( column_name [,...n] )
AS
( CTE_query_definition )
```

Инструкция для обращения к ОТВ:

```
SELECT <column_list>
FROM expression_name;
```

CTE предназначены для:

- Создания рекурсивных запросов.
- Группирования по столбцу, производного от скалярного подзапроса выборки
- Многократных ссылок на результирующую таблицу из одной и той же инструкции.

Рекурсивное выполнение имеет следующую семантику:

- разбиение CTE на закрепленный и рекурсивный элементы;
- запуск закрепленных элементов с созданием первого вызова или базового результирующего набора (T_0);
- запуск рекурсивных элементов, где T_i — это вход, а T_{i+1} — это выход;
- повторение шага 3 до тех пор, пока не вернется пустой набор;
- возвращение результирующего набора. Результирующий набор получается с помощью инструкции UNION ALL от T_0 до T_n.

Структура CTE

```
WITH expression_name  ( column_name [,...n] )
AS
( CTE_query_definition )
```

Инструкция для обращения к ОТВ:

```
SELECT <column_list>
FROM expression_name;
```

Create Employees table

```
CREATE TABLE Employees
(
   empid   int         NOT NULL
   ,mgrid   int         NULL
   ,empname varchar(25) NOT NULL
   ,salary  money       NOT NULL
   CONSTRAINT PK_Employees PRIMARY KEY(empid)
);
```

Employees table - insert values

```
INSERT INTO Employees VALUES(1 , NULL, 'Nancy'   , $10000.00);
INSERT INTO Employees VALUES(2 , 1   , 'Andrew'  , $5000.00);
INSERT INTO Employees VALUES(3 , 1   , 'Janet'   , $5000.00);
INSERT INTO Employees VALUES(4 , 1   , 'Margaret', $5000.00);
INSERT INTO Employees VALUES(5 , 2   , 'Steven'  , $2500.00);
INSERT INTO Employees VALUES(6 , 2   , 'Michael' , $2500.00);
INSERT INTO Employees VALUES(7 , 3   , 'Robert'  , $2500.00);
INSERT INTO Employees VALUES(8 , 3   , 'Laura'   , $2500.00);
INSERT INTO Employees VALUES(9 , 3   , 'Ann'     , $2500.00);
INSERT INTO Employees VALUES(10, 4   , 'Ina'     , $2500.00);
INSERT INTO Employees VALUES(11, 7   , 'David'   , $2000.00);
INSERT INTO Employees VALUES(12, 7   , 'Ron'     , $2000.00);
INSERT INTO Employees VALUES(13, 7   , 'Dan'     , $2000.00);
INSERT INTO Employees VALUES(14, 11  , 'James'   , $1500.00);
```

Employees

	empid	mgrid	empname	salary
1	1	NULL	Nancy	10000,00
2	2	1	Andrew	5000,00
3	3	1	Janet	5000,00
4	4	1	Margaret	5000,00
5	5	2	Steven	2500,00
6	6	2	Michael	2500,00
7	7	3	Robert	2500,00
8	8	3	Laura	2500,00
9	9	3	Ann	2500,00
10	10	4	Ina	2500,00
11	11	7	David	2000,00
12	12	7	Ron	2000,00
13	13	7	Dan	2000,00
14	14	11	James	1500,00

Все дерево от корня

```
WITH tree1 (manager, employe, employe_name, emp_salary, emp_level)
AS
(SELECT mgrid, empid, empname, salary, 0 FROM Employees
WHERE mgrid IS NULL                    /* закрепленный элемент

UNION ALL
SELECT mgrid, empid, empname, salary, emp_level+1 FROM Employees
JOIN tree1 ON mgrid= employe           /* рекурсивный элемент

)
SELECT * from tree1
ORDER BY manager;
```

Задание 1 – добавить в выборку имя менеджера

	manager	employe	employe_name	emp_salary	emp_level
1	1	2	Andrew	5000,00	0
2	1	3	Janet	5000,00	0
3	1	4	Margaret	5000,00	0
4	2	5	Steven	2500,00	1
5	2	6	Michael	2500,00	1
6	3	7	Robert	2500,00	1
7	3	8	Laura	2500,00	1
8	3	9	Ann	2500,00	1
9	4	10	Ina	2500,00	1
10	7	11	David	2000,00	2
11	7	12	Ron	2000,00	2
12	7	13	Dan	2000,00	2
13	11	14	James	1500,00	3

- Задание 2

  Написать функцию, возвращающую таблицу всех подчиненных сотрудников с параметром Id менеджера.

- Задание 3

  Написать функцию, возвращающую сумму зарплаты всех подчиненных сотрудников с параметром Id менеджера.

Функция, возвращающая таблицу

```
CREATE FUNCTION dbo.fn_getsubtree(@empid AS INT)
   RETURNS @TREE TABLE (
   empid   INT NOT NULL
   ,empname VARCHAR(25) NOT NULL
   ,mgrid   INT NULL
   ,lvl     INT NOT NULL
)
AS
BEGIN
…
  INSERT INTO @TREE
    SELECT * FROM Employees_Subtree;
  RETURN
END
```

Departments

	deptid	deptname	deptmgrid
1	1	HR	2
2	2	Marketing	7
3	3	Finance	8
4	4	R&D	9
5	5	Training	4
6	6	Gardening	NULL

Create Departments table and insert values

```
CREATE TABLE Departments
(
   deptid    INT NOT NULL PRIMARY KEY
   ,deptname  VARCHAR(25) NOT NULL
   ,deptmgrid INT NULL REFERENCES Employees
);
GO
INSERT INTO Departments VALUES(1, 'HR',          2);
INSERT INTO Departments VALUES(2, 'Marketing',    7);
INSERT INTO Departments VALUES(3, 'Finance',      8);
INSERT INTO Departments VALUES(4, 'R&D',         9);
INSERT INTO Departments VALUES(5, 'Training',     4);
INSERT INTO Departments VALUES(6, 'Gardening', NULL);
```

Оператор APPLY

- позволяет вызывать возвращающую табличное значение функцию для каждой строки, возвращаемой внешним табличным выражением запроса.

```
SELECT Tl.*, Tr.*
FROM Table AS Tl
    CROSS APPLY function(Tl.field1) AS Tr;
```

Типы оператора APPLY

- CROSS APPLY возвращает только строки из внешней таблицы, которые создает результирующий набор из возвращающего табличное значение функции.
- OUTER APPLY возвращает и строки, которые формируют результирующий набор, и строки, которые этого не делают, со значениями NULL в столбцах, созданных возвращающей табличное значение функцией.

Задание 4.

Вывести названия отделов и всех работников этих отделов

hierarchyid

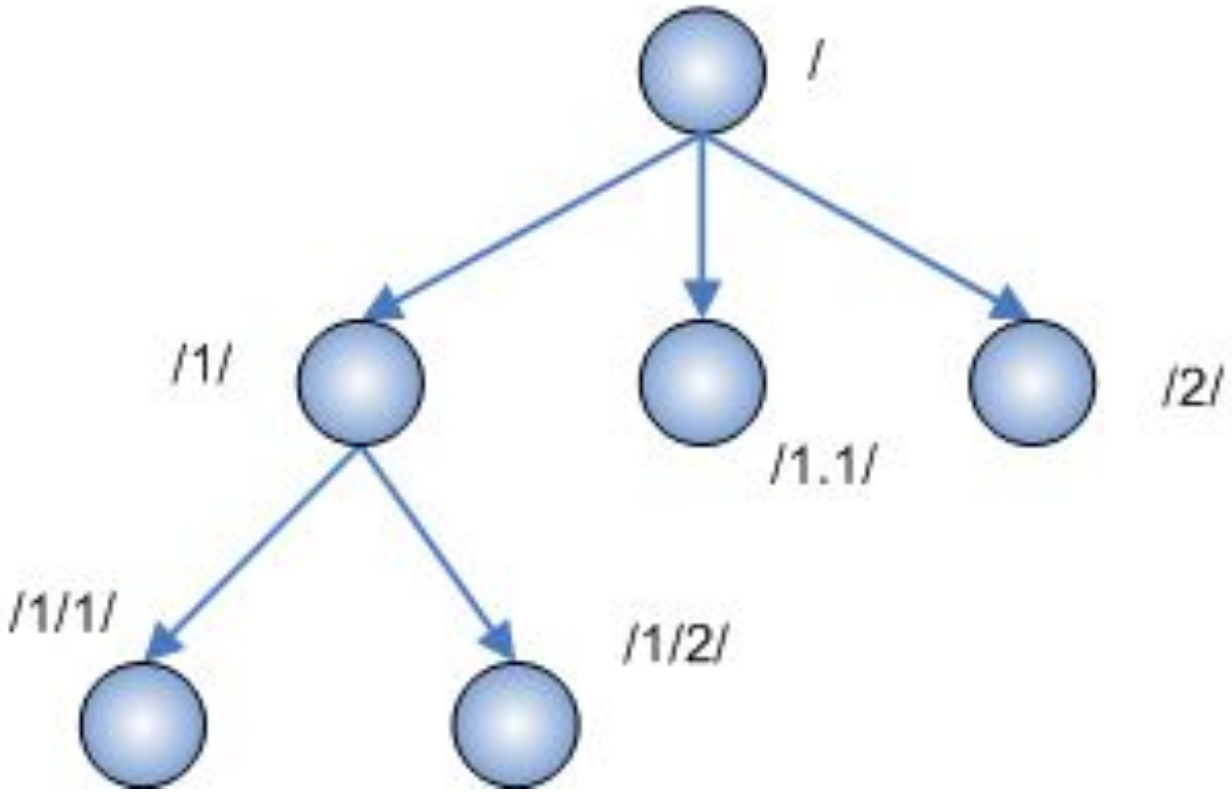

Таблица Employees с полем *hierarchyid*

hierarchyid	empid	empname	salary
/	1	Nancy	10000,00
/1/	2	Andrew	5000,00
/2/	3	Janet	5000,00
/3/	4	Margaret	5000,00
/1/1/	5	Steven	2500,00
/1/2/	6	Michael	2500,00
/2/1/	7	Robert	2500,00
/2/2/	8	Laura	2500,00
/2/3/	9	Ann	2500,00
/3/1/	10	Ina	2500,00
/2/1/1/	11	David	2000,00
/2/1/2/	12	Ron	2000,00
/2/1/3/	13	Dan	2000,00
/2/1/1/1/	14	James	1500,00

```
CREATE TABLE Emp_hierarchy
(
    Id hierarchyid PRIMARY KEY
   ,empid   int         NOT NULL
   ,empname varchar(25) NOT NULL
   ,salary  money       NOT NULL
);
```

Предложение OVER

- Определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции. То есть предложение OVER определяет окно или определяемый пользователем набор строк внутри результирующего набора запроса.
- OVER (

    [<PARTITION BY clause>]

    [<ORDER BY clause>]

)

Рассмотрим пример

```
SELECT
    id, dept, salary
 from Employees
```

id	dept	salary
1	1	100
2	2	150
3	2	110
4	2	100
5	3	200

Сумма нарастающим итогом

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (ORDER BY id)  AS Running_Sum
 from Employees
```

id	dept	salary	Running_Sum
1	1	100	100
2	2	150	250
3	2	110	360
4	2	100	460
5	3	200	660

Сумма с группировкой

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (partition by dept)  AS Dept_Sum
    , AVG(salary) OVER (partition by dept)  AS Dept_AVG
from Employees
```

id	dept	salary	Dept_Sum	Dept_AVG
1	1	100	100	100
2	2	150	360	120
3	2	110	360	120
4	2	100	360	120
5	3	200	200	200

Сумма с группировкой

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (partition by dept ORDER by id)  AS Dept_Sum
    , AVG(salary) OVER (partition by dept)  AS Dept_AVG
 from Employees
```

id	dept	salary	Dept_Run_Sum	Dept_AVG
1	1	100	100	100
2	2	150	150	120
3	2	110	260	120
4	2	100	360	120
5	3	200	200	200

ROW_NUMBER()

```
SELECT
  S.*, ROW_NUMBER() OVER (ORDER BY empName) AS RowNum
FROM Employees S
```

Номер строки

```
SELECT
    id, dept, salary
    , ROW_NUMBER() OVER (ORDER BY id) AS RowNum
 from Employees
```

id	dept	salary	RowNum
1	1	100	1
2	2	150	2
3	2	110	3
4	2	100	4
5	3	200	5

ROW_NUMBER() + PARTITION

```
SELECT
  S.*, ROW_NUMBER() OVER (PARTITION BY S.mgrid
ORDER BY S.empName) AS LocalRowNum
FROM Employees S
```

Номер строки с группировкой

```
SELECT
    id, dept, salary
    , ROW_NUMBER() OVER (PARTITION BY dept ORDER BY id) AS RowNum
 from Employees
```

id	dept	salary	RowNum
1	1	100	1
2	2	150	1
3	2	110	2
4	2	100	3
5	3	200	1

RANK ( ) / DENSE_RANK ( )

- Rank - возвращает ранг каждой строки в секции результирующего набора. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.(1, 1, 1, 4)
- Dense_rank - возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Ранг строки равен количеству различных значений рангов, предшествующих строке, увеличенному на единицу. (1, 1, 2)

RANK ( ) OVER (ORDER by smth)

- Распределяет строки упорядоченной секции в заданное количество групп.

```
SELECT
  S.*
  , RANK (  ) OVER (ORDER by salary desc) AS Gr
FROM Employees S
```

NTILE (N)

- Распределяет строки упорядоченной секции в заданное количество групп.

```
SELECT
  S.*
  , NTILE(3) OVER (ORDER BY S.salary) AS Gr
FROM Employees S
```

```
SELECT
  S.*
  , ROW_NUMBER() OVER
     (PARTITION BY S.mgrid ORDER BY S.empName)  AS
LocalRowNum
  , RANK() OVER (ORDER BY S.salary) AS Rank
  , COUNT(*) OVER
     (PARTITION BY S.mgrid ) AS Amount
FROM Employees S
```

Структура CTE

```
WITH expression_name  ( column_name [,...n] )
AS
( CTE_query_definition )
```

Инструкция для обращения к ОТВ:

```
SELECT <column_list>
FROM expression_name;
```

ROW_NUMBER() + PARTITION

empid	mgrid	empname	salary	LocalRowNum
1	NULL	Nancy	10000,00	1
2	1	Andrew	5000,00	1
3	1	Janet	5000,00	2
4	1	Margaret	5000,00	3
6	2	Michael	2500,00	1
5	2	Steven	2500,00	2
9	3	Ann	2500,00	1
8	3	Laura	2500,00	2
7	3	Robert	2500,00	3
10	4	Ina	2500,00	1
13	7	Dan	2000,00	1
11	7	David	2000,00	2
12	7	Ron	2000,00	3
14	11	James	1500,00	1

Перенос данных из Emloyees в Emp_hierarchy

```
WITH paths(path, EmployeeID)
AS (
-- This section provides the value for the root of the hierarchy
SELECT hierarchyid::GetRoot() AS OrgNode, empid
FROM Employees AS C
WHERE …

UNION ALL
-- This section provides values for all nodes except the root
SELECT
CAST(p.path.ToString() + CAST(( ROW_NUMBER() OVER (PARTITION BY mgrid ORDER BY mgrid) ) AS varchar(30)) + '/' AS hierarchyid),
C.empid
FROM Employees AS C
JOIN paths AS p
  ON …
)
…
```

Обход дерева

```
select s.*
from Emp_hierarchy s
```

Id	empid	empname	salary
0x	1	Nancy	10000,00
0x58	2	Andrew	5000,00
0x5AC0	5	Steven	2500,00
0x5B40	6	Michael	2500,00
0x68	3	Janet	5000,00
0x6AC0	7	Robert	2500,00
0x6AD6	11	David	2000,00
0x6AD6B0	14	James	1500,00
0x6ADA	12	Ron	2000,00
0x6ADE	13	Dan	2000,00
0x6B40	8	Laura	2500,00
0x6BC0	9	Ann	2500,00
0x78	4	Margaret	5000,00
0x7AC0	10	Ina	2500,00

Обход дерева с путем и уровнями

```
select s.*, Id.ToString() AS Path,
    Id.GetLevel() AS Level
from Emp_hierarchy s
```

Id	empid	empname	salary	Path	Level
0x	1	Nancy	10000,00	/	0
0x58	2	Andrew	5000,00	/1/	1
0x5AC0	5	Steven	2500,00	/1/1/	2
0x5B40	6	Michael	2500,00	/1/2/	2
0x68	3	Janet	5000,00	/2/	1
0x6AC0	7	Robert	2500,00	/2/1/	2
0x6AD6	11	David	2000,00	/2/1/1/	3
0x6AD6B0	14	James	1500,00	/2/1/1/1/	4
0x6ADA	12	Ron	2000,00	/2/1/2/	3
0x6ADE	13	Dan	2000,00	/2/1/3/	3
0x6B40	8	Laura	2500,00	/2/2/	2
0x6BC0	9	Ann	2500,00	/2/3/	2
0x78	4	Margaret	5000,00	/3/	1
0x7AC0	10	Ina	2500,00	/3/1/	2

Id.GetAncestor(n int)

Res Hierarchyid
/*найти детей '/2/‘ */
select *
FROM Emp_hierarchy s
WHERE s.id.GetAncestor(1)='/2/'

/*найти внуков '/2/‘ */
select *
FROM Emp_hierarchy s
WHERE s.id.GetAncestor(2)='/2/'

parent.GetDescendant (child1 , child2)

- Res Hierarchyid
- Для генерации кодов дочерних узлов предназначен метод GetDescendant. У него есть два параметра, определяющих, между какими двумя узлами следует поместить новый узел (любой из параметров может быть равен null). Если это первый дочерний узел, то оба этих параметра должны быть равны null:
- SET @new_node = @node.GetDescendant(@max_child_node, null);

GetRoot

- hierarchyid::GetRoot ( )

```
insert into Emp_hierarchy
(Id, empid, empname, salary)
values
(hierarchyid::GetRoot(), 1, ‘Anna-Maria', 10000)
```

id.GetLevel

- Res smallint
- Возвращает целое число, представляющее глубину *этого* узла в дереве.

child. IsDescendantOf (parent)

- Res true|false
- Возвращает значение true, если объект *this* является потомком объекта parent.

node. GetReparentedValue (oldRoot, newRoot)

- Возвращаемый тип данных SQL Server: hierarchyid
- Переносит ветку дерева

```
UPDATE Employees
SET id = id.GetReparentedValue(@old_node, @new_node)
WHERE employee_hid.IsDescendantOf(@old_node) = 1;
```

Id.ToString()

- преобразование из типа hierarchyid в строковый тип

- 0x5AC0 /1/1/

Parse

- преобразование из строкового типа тип в hierarchyid

- hierarchyid::Parse(@StringValue)
- /1/1/   0x5AC0

Обход поддерева
Выборка всех потомков

```
DECLARE @parent_hid HIERARCHYID;

SELECT @parent_hid = id
FROM Emp_hierarchy
 WHERE empname = ‘Laura’

select s.*, Id.ToString() AS [Path],
    Id.GetLevel() AS [Level]
FROM Emp_hierarchy s
WHERE Id.IsDescendantOf(@parent_hid) = 1;
```

Обход дерева
с суммой зарплаты по всей ветке

```
select s.*, Id.ToString() AS [Path],
    Id.GetLevel() AS [Level] , (select sum(salary) from Emp_hierarchy
where Id.IsDescendantOf(S.id)=1) as Total
from Emp_hierarchy s
```

- Добавить для Robert нового подчиненного Boris между Ron и Dan
- Добавить ему двух любых подчиненных
- Отправить Margaret в подчинение Janet

```
DECLARE
   @reparented_node AS HIERARCHYID, -- Код узла, который мы хотим переподчинить со всеми его потомками
   @new_parent_node AS HIERARCHYID, -- Код узла нового родителя
   @max_child_node AS HIERARCHYID,  -- Код узла максимального потомка нового родителя
   @new_child_node AS HIERARCHYID;  -- Код узла для нового потомка нового родителя

-- Получаем код узла, который хотим переподчинить со всеми его потомками
SELECT @reparented_node = id
FROM Emp_hierarchy
WHERE empid = 3; -- employee_id Janet
 -- Получаем код узла нового родителя
SELECT @new_parent_node = id
FROM Emp_hierarchy
WHERE empid = 4; -- employee_id Margaret
 -- Получаем код узла максимального потомка нового родителя
SELECT @max_child_node = MAX(id)
FROM Emp_hierarchy
WHERE id.GetAncestor(1) = @new_parent_node;
 -- Получаем код узла для нового потомка нового родителя
SET @new_child_node = @new_parent_node.GetDescendant(@max_child_node, null);
 -- Переподчиняем нужный нам узел вместе со всеми его потомками
UPDATE Emp_hierarchy
SET id = id.GetReparentedValue(@reparented_node, @new_child_node)
WHERE id.IsDescendantOf(@reparented_node) = 1;
```