### Лекция 11:

1. Понятие графа, иерархического графа. Древовидные структуры. Применение иерархических структур в бизнесе.
2. Разработка структуры БД для хранения древовидной (иерархической) структуры.

**Граф –** некоторая система вершин и соединяющих их дуг. Более строгое определение графа будет дано ниже.

Язык графов оказывается удобным для описания процессов в различных системах.

**1. Применение графов в бизнесе:**

1. *Транспортные задачи*, в которых вершинами графа являются пункты, а ребрами – дороги (автомобильные, железные и др.) и/или другие транспортные (например, авиационные) маршруты. Другим примером могут быть: сети снабжения (энергоснабжения, газоснабжения, снабжения товарами и т.д.), в которых вершинами являются пункты производства и потребления, а ребрами – возможные пути перемещения (линии электропередачи, газопроводы, дороги и т.д.). Соответствующий класс задач оптимизации потоков грузов, размещения пунктов производства и потребления, иногда называются задачами обеспечения или задачами о размещении. Их подклассом являются задачи о грузоперевозках.
2. *«Технологические задачи»*, в которых вершины отражают производственные элементы (заводы, цеха, станки и т.д.), а дуги – потоки сырья, материалов и продукции между ними, заключаются в определении оптимальной загрузки производственных элементов и обеспечивающих эту загрузку потоков.
3. *Обменные схемы*, являющиеся моделями таких явлений как бартер, взаимозачеты и тд. Вершины графа при этом описывают участников обменной схемы (цепочки), а дуги – потоки материальных и финансовых ресурсов между ними. Задача заключается в определении цепочки обменов, оптимальной с точки зрения, например, организатора обмена и согласованной с интересами участников цепочки и существующими ограничениями.
4. *Модели коллективов и групп*, используемые в социологии, основываются на представлении людей или групп в виде вершин, а отношений между ними в виде ребер или дуг. (Граф ваших друзей в соц. сети).
5. *Модели организационных структур*, в которых вершинами являются элементы организационной системы, а ребрами или дугами – связи (информационные, управляющие, технологические и т.д.) между ними.
6. ***Управление проектами:***

*Управление проектами* - раздел теории управления, изучающий методы и механизмы управления изменениями, проектом называется целенаправленное изменение некоторой системы, осуществляемое в рамках ограничений на время и используемые ресурсы.

С точки зрения графов *проект* – совокупность операций и зависимостей между ними (сетевой график).

Совокупность моделей и методов, использующих язык и результаты теории графов и ориентированных на решение задач управления проектами, получила название календарно – сетевого планирования и управления (КСПУ).

В рамках КСПУ решаются задачи выполнения операций и распределения ресурсов между ними, оптимальных с точки зрения тех или иных критериев (времени выполнения проекта, затрат, риска и др.)

В дальнейшем, наибольшее внимание будет уделено именно задачам управления проектами.

Мы будем использовать реляционную СУБД для хранения графов, но имейте в виду, что для графов есть специализированные графовые базы данных. Например, СУБД Neo4j (NOSQL СУБД), с языком Cypher (но есть API с Java). Данные в этой СУБД хранятся в виде упорядоченных графов, а не в виде таблиц, кроме того Neo4j позволяет удобно визуализировать графы.

**2. Определения и способы задания в теории графов.**

*Граф* – пара (V, E), где V – непустое множество объектов, называемых вершинами графа, а E – подмножество двухэлементных подмножеств множества V, называемых ребрами графа. Число вершин в графе – порядок, а число ребер – размер графа.

*Орграф* – ориентированный граф. Орграф – пара (V, A), где V – множество вершин, А – подмножество двухэлементных упорядоченных подмножеств множества V, называемых *дугами* орграфа. Для дуги e = (u, v) вершины u и v называют соответственно началом и концом дуги.

Две концевые вершины одного и того же ребра называются смежными. Точнее, если (i,j) принадлежит графу, то смежная, иначе нет; для ориентированного графа, например, (i,j) принадлежит графу, а (j,i) не принадлежит.

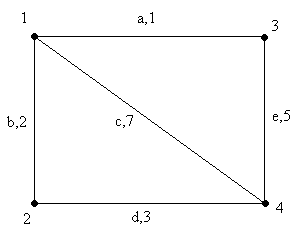


Рис.1. Неориентированный граф. 4 вершины, 5 ребер.

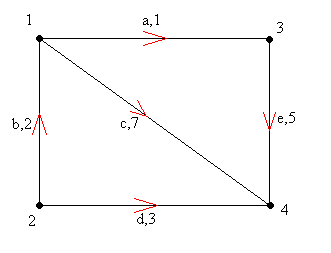


Рис.2. Орграф. Стрелки – направление дуг.

Отношение смежности задается *матрицей смежности A(G).* Это квадратная матрица размером n\*n (n – порядок графа), у которой элемент a(ij) равен 1, если вершины vi и vj смежны, и 0 в противном случае. A(ii) = 0. Матрица смежности определяет граф с точностью до изоморфизма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 1 | 0 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 |

Табл.1. Матрица смежности для неориентированного графа на рис.1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 |

Табл.2. Матрица смежности для ориентированного графа на рис.2.

*Матрица инцидентности:* Если e = (a,b) – ребро в графе G, то можно говорить, что ребро e инцидентно вершине a или вершине b. Аналогично определяется инцидентность вершин ребру. Отношение инцидентности представляется в виде матрицы B(G) инцидентности размером n на m (n – число вершин, m – число ребер), у которой элемент b(ki) = 1, если вершина i инцидентна ребру k. В каждом столбце матрицы всего две единицы, равных столбцов нет. Матрица инцидентности также содержит всю информацию о графе, но ее использование затрудняется большим количеством нулей.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| a | 1 | 0 | 1 | 0 |
| b | 1 | 1 | 0 | 0 |
| c | 1 | 0 | 0 | 1 |
| d | 0 | 1 | 0 | 1 |
| e | 0 | 0 | 1 | 1 |

Табл.3. Матрица инцидентности для неориентированного графа на рис.1.

Матрица инцидентности для ориентированного графа определяется так: элемент равен 1, если ребро исходит из вершины, равно - 1, если заходит, и 0 иначе.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| a | 1 | 0 | -1 | 0 |
| b | -1 | 1 | 0 | 0 |
| c | 1 | 0 | 0 | -1 |
| d | 0 | 1 | 0 | -1 |
| e | 0 | 0 | 1 | -1 |

Табл.4. Матрица инцидентности для ориентированного графа на рис.2.

Пусть v(1)….v(k) – последовательность вершин, такая, что каждая пара соседних вершин v(i), v(i+1) определяет ребро в графе. Тогда данная последовательность называется *маршрутом*. Маршрут называется простым (*или цепью*), если ни одна вершина (*кроме, возможно, начальной и конечной*) не встречается в нем дважды.

Граф называется *связным*, если каждая пара вершин в графе соединена цепью. *Цикл* – цепь в графе, концы которой совпадают.

*Длина цепи* – число ребер, образующих цепь. Если каждому ребру сопоставлено некоторое вещественное положительное число w(e), называемое весом ребра, причем вес цепи, порождаемой ребрами, определяется как сумма весов составляющих ее ребер. Цепь Q(u,v) называется кратчайшей цепью, если ее длина является наименьшей среди длин всех цепей, соединяющих u и v.

На рис.1 от вершины 1 до вершины 4 возможно 3 цепи: 1-4 (сумм длины 7), 1-2-4 (сумм длины 5), 1-3-4 (сумм длины 6). То есть кратчайшей цепью при заданных весах является цепь 1-2-4, несмотря на то, что содержит 2 ребра.

*Путь в орграфе* – последовательность вершин (v1…vk) такая, что каждая пара соседних вершин определяет дугу в графе, дуги ориентированы в направлении движения от v1 до vk.

*Цепь* – последовательность дуг, в которой дуги могут совпадать с направлением от начала цепи к ее концу или не совпадать.

*Кратчайший путь от u к v* – длина пути минимальная среди всех возможных путей от u к v.

Путей от вершины 1 до 4 на рис.2 два: 1-3-4 (сумм длина = 6), 1-4 (сумм длина = 7). Кратчайшим путем является путь 1-3-4. Цепи остались теми же, как и для рисунка 1.

*Дерево –* неориентированный связный граф без циклов. Несвязный неориентированный граф без циклов называется *лесом*. Пример дерева показан на рис.3.

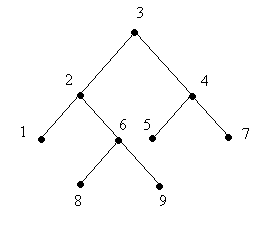


Рис.3. Дерево.

Дерево называют корневым, если в нем выделена вершина r, именуемая корнем.

*Ориентированным деревом с корнем r (ордерево)* называется корневое дерево, в котором каждое ребро заменено дугой, таким образом, что, либо из каждой вершины можно попасть в корень, двигаясь вдоль ориентации дуг (входящее дерево), либо в каждую вершину можно попасть из корня, двигаясь вдоль ориентации дуг (выходящее дерево). Висячие вершины выходящего дерева называются листьями.

Глубина вершины – длина пути из корня в некоторую вершину.

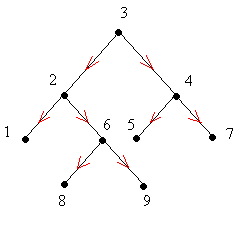


Рис.4. Ор-дерево (выходящее).

Представление деревьев - способ записи информации о нем, однозначно и полностью восстанавливающий структуру дерева.

Вершина v достижима из u, если существует путь из v в u. Отношение достижимости задается либо матрицей достижимости, либо списками достижимости. Матрица достижимости R(D) – квадратная матрица n\*n, в которой элемент r(ij) = 1, если вершина vj достижима из вершины vi, и r(ij) = 0 в противном случае.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 1 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 |

Табл.5. Матрица достижимости для графа на рис.2.

*Упражнение:* Подумайте,если матрицу смежности для графа на рис.2 умножить саму на себя (правило сложения: 1+ 1 = 1, 1 + 0 = 0 + 1 = 1, 0 + 0 = 0, то есть как функция sign), то что будет представлять собой получившаяся матрица? Верно ли, что матрица достижимости равна M + M\*M + M\*M\*M + …., где M – матрица смежности?

**3. Представление графов в БД.**

Так как в задачах бизнеса большую часть занимают задачи, для решения которых удобнее пользоваться ориентированными графами, то и для БД будем рассматривать представление именно ориентированных графов.

По приведенным выше примерам видим, что использование матриц (смежности, достижимости, инцидентности) для описания графов в БД неудобно по нескольким причинам:

* данные матрицы обладают большим количеством нулей, а хранение матрицы в таком виде предполагает использование n\*n либо n\*m ячеек.
* Хранение матриц в БД в виде n – колво полей, n (m) – колво строк противоречит логике БД, так как при таком способе хранения атрибутом будет являться номер столбца в матрице, а по логике реляционных БД атрибуты не упорядочены, точно также, как и кортежи. (А еще в Access существует ограничение на количество вводимых атрибутов).

Кроме того, можно вспомнить задачу из прошлого семестра о способе хранения матриц в БД. Поэтому можно предложить следующий способ хранения ориентированных графов в БД:

*Начальная вершина – конечная вершина – вес ребра.*

Для графа на рис.2 соответствующая таблица будет иметь следующий вид:

|  |  |  |
| --- | --- | --- |
| **начало** | **конец** | **вес** |
| 1 | 3 | 1 |
| 1 | 4 | 7 |
| 2 | 1 | 2 |
| 2 | 4 | 3 |
| 3 | 4 | 5 |

Табл.6. Представление графа на рис.2 с помощью БД

Понятно, что такая таблица однозначно задает граф, не содержит нулей (меньше по размеру) и удобна для обработки средствами БД.

В принципе, неориентированные графы можно хранить точно также, только в таком случае потеряют смысл термины «начало» и «конец». Кроме того, при работе с таким графом нужно договориться, хранить ли ребра по одному разу или дважды (начало-конец, конец-начало).

Для дерева на рис.4 (веса на рис.4 не обозначены) таблица будет иметь следующий вид:

|  |  |  |
| --- | --- | --- |
| **начало** | **конец** | **вес** |
| 3 | 2 | 1 |
| 3 | 4 | 1 |
| 2 | 1 | 1 |
| 2 | 6 | 1 |
| 6 | 8 | 1 |
| 6 | 9 | 1 |
| 4 | 5 | 1 |
| 4 | 7 | 1 |

Табл.7. Представление дерева на рис.4 с помощью БД (вес проставлен в 1, но может быть произвольным, в зависимости от задачи)

Возможные применения графов:

**Граф подчиненности:**

Начальник\_ID int

Подчиненный\_ID int

Уровень подчиненности int

**Граф расстояний:**

Из\_ID int

В\_ID int

Расстояние float

**Изделия:**

Изделие\_ID int

Компонент\_изделия\_ID int

Колво\_в\_изделии float

**Практикум:**

Есть древовидная структура данных в БД (выходящее дерево, например рис.4). Необходимо найти:

* корень дерева;
* листовой узел;
* поиск узла с заранее заданной глубиной.

*Решения:*

DECLARE @T TABLE (b int, e int, w float)

INSERT INTO @T VALUES (3, 2, 1), (3, 4, 1), (2, 1, 1), (2, 6, 1), (6, 8, 1), (6, 9, 1), (4, 5, 1), (4, 7, 1)

-- поиск корня: корень - это такое значение в b, которого нет в e (так как корень - это то, у чего есть потомки, но нет исходных)

-- вариантов поиска много, поэтому для поиска корней и листьев используем разные типы запросов, но все эти запросы сработают и на случай леса.

SELECT distinct b

FROM @T

WHERE b not in (SELECT e FROM @T)

-- поиск листьев (те элементы, у которых есть предки, но нет потомков)

SELECT Te.e

FROM @T Te LEFT JOIN @T Tb ON

Te.e = Tb.b

WHERE Tb.b is NULL

-- поиск сотрудников, заданной глубины подчиненности. Здесь предположим, что мы уже точно знаем, что самый главный босс - это номер 3.

-- но правильнее было бы написать подзапрос

-- Сделаем по шагам

-- Шаг 1: посмотрите, что выведется в результате выполнения такого запроса. Подумайте, понятен ли Вам результат? Первый шаг выводит в качестве T2.e - сотрудников второго уровня от босса.

SELECT T1.b, T1.e, T2.b, T2.e

FROM @T T1 LEFT JOIN @T T2 ON

T1.e = T2.b

WHERE T1.b = 3

|  |  |  |  |
| --- | --- | --- | --- |
| T1.b | T1.e | T2.b | T2.e |
| 3 | 2 | 2 | 1 |
| 3 | 2 | 2 | 6 |
| 3 | 4 | 4 | 5 |
| 3 | 4 | 4 | 7 |

-- Шаг 2: согласитесь, что сотрудников 3-его уровня можно получить точно так же.

-- Нужно только определить, кто является потомками сотрудников второго уровня, то есть вывести те вершины e, у которых в качестве вершин b используются вершины T2.e.

SELECT T1.b, T1.e, T2.b, T2.e, T3.b, T3.e

FROM @T T1 LEFT JOIN @T T2 ON

T1.e = T2.b

LEFT JOIN @T T3 ON

T2.e = T3.b

WHERE T1.b = 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T1.b | T1.e | T2.b | T2.e | T3.b | T3.e |
| 3 | 2 | 2 | 1 | NULL | NULL |
| 3 | 2 | 2 | 6 | 6 | 8 |
| 3 | 2 | 2 | 6 | 6 | 9 |
| 3 | 4 | 4 | 5 | NULL | NULL |
| 3 | 4 | 4 | 7 | NULL | NULL |

-- Шаг 3: исправим вывод в SELECT и условие в WHERE так, чтобы выводились только сотрудники 3-его уровня

SELECT distinct T3.e

FROM @T T1 LEFT JOIN @T T2 ON

T1.e = T2.b

LEFT JOIN @T T3 ON

T2.e = T3.b

WHERE T1.b = 3 AND T3.b is not NULL

-- Шаг 4: подумайте, понятно ли Вам, как вывести сотрудников 3-его уровня от любого заданного сотрудника?

Комментарий: вспомните, как Вы делали задачу на перемножение матриц в прошлом семестре? Согласитесь, что решение очень похоже. То есть, мы перемножаем матрицу смежности для нашего графа при помощи соединения таблицы с собой.

## Задача расчета состава изделия в исходных компонентах:

***Постановка:***

Задана таблица «Номенклатура\_изделий» следующей структуры:

Изделие\_ID int артикул изделия;

Компонент\_ID int артикул компоненты изделия (может также быть изделием);

Колво\_в\_изделии float колво компоненты в изделии;

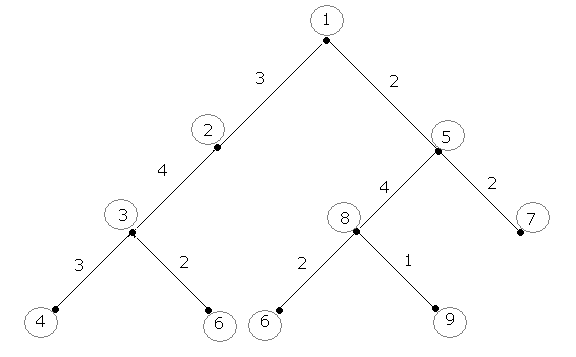
Заранее известно, что глубина вложения не более 3-х.

Напишите запрос, который по артикулу изделия, задаваемому параметром, выдает состав этого изделия в товарах, не являющихся изделиями.

***Решение:***

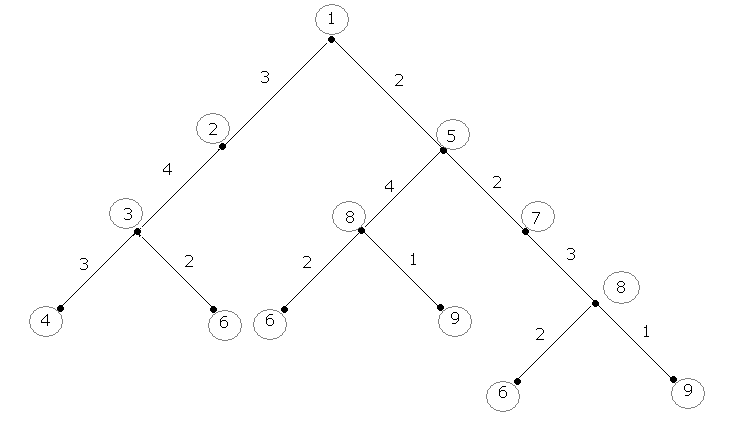
Считаем, что в таблице «Номенклатура\_изделий» представлены только прямые связи: то есть от изделия к его непосредственной составной части.

Понятно, что такую таблицу всегда можно представить в виде дерева (либо множества деревьев), при этом, если одна вершина (компонента) встречается на нескольких «ветках», то ее следует «расщепить», например, так:



*Рис. 1. На графе вершина 6 является потомком и у вершины 3, и у вершины 8, но при этом логически вершины можно разделить (так как дерево направленное).*

Если бы дублирующаяся вершина не являлась листовой, то принципиально ничего бы не изменилось, то есть «расщепление» следовало бы проводить так же, только при «расщеплении» дублировать не только вершину, но и всех ее потомков с ценами ребер, как показано на рис.2.



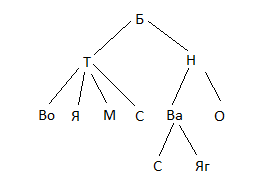
*Рис.2. Например, нелистовая вершина 8 является потомком вершин 5 и 7.*

Понятно, что для целей построения состава изделия, такое «раздвоение» вершины допустимо (потому что хранение в базе ничем отличаться не будет, что в случае раздвоения, что без). При реализации других задач следует понимать, как такое преобразование может повлиять на конечный результат.

|  |  |  |
| --- | --- | --- |
| Изделие\_ID | Компонент\_ID | Колво\_в\_изделии |
| 1 | 2 | 3 |
| 1 | 5 | 2 |
| 2 | 3 | 4 |
| 3 | 4 | 3 |
| 3 | 6 | 2 |
| 5 | 7 | 2 |
| 5 | 8 | 4 |
| 8 | 6 | 2 |
| 8 | 9 | 1 |

Таблица 1. Представление графа с рис.1 при помощи БД

В качестве примера рассмотрим следующий практический пример (но запросы подходят и для примеров выше, значение имеет только глубина вложения).



Изделие Б «Булочка» состоит из компонент. Вы знаете, что булочка на 50% состоит из Теста, а на 50% из Начинки. При этом массовые доли компонент при приготовлении Теста равны: Вода – 30%, Яйца – 20%, Мука – 40%, Сахар – 20%. Начинка состоит на 50% из Орехов, а на 50% из Варенья. Варенье на 80% состоит из Ягод, а на 20% из Сахара.

Необходимо рассчитать, сколько % от массы Булочки составляет каждая компонента-лист (что поможет сосчитать сколько нужно взять компонент по массе, чтобы испечь 1 кг булочек).

Таблица для хранения булочки в БД следующая («Номенклатура\_изделий»):

|  |  |  |
| --- | --- | --- |
| **b** | **e** | **w** |
| Б | Т | 0.5 |
| Б | Н | 0.5 |
| Т | Во | 0.3 |
| Т | Я | 0.1 |
| Т | М | 0.4 |
| Т | С | 0.2 |
| Н | Ва | 0.5 |
| Н | О | 0.5 |
| Ва | С | 0.2 |
| Ва | Я | 0.8 |

На первом этапе выполним такой запрос:

SELECT T1.Изделие\_ID, T1.Компонент\_ID AS Комп1, T1.Колво\_в\_изделии AS Колво1, T2.Компонент\_ID AS Комп2, T2.Колво\_в\_изделии AS Колво2,

T3.Компонент\_ID AS Комп3, T3.Колво\_в\_изделии AS Колво3

FROM (

Номенклатура\_изделий AS T1 LEFT JOIN

Номенклатура\_изделий AS T2 ON

T1.Компонент\_ID = T2.Изделие\_ID

) LEFT JOIN

Номенклатура\_изделий AS T3 ON

T2.Компонент\_ID = T3.Изделие\_ID

WHERE T1.Изделие\_ID = ‘Б’;

(Далее к этому запросу будем относиться как к подзапросу под псевдонимом «Создание\_иерархии»).

Результатом выполнения будет следующая таблица, представляющая собой цепочку от изделия до его составных частей (пустые значение = NULL):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Изделие\_ID** | **Комп1** | **Колво1** | **Комп2** | **Колво2** | **Комп3** | **Колво3** |
| Б | Т | 0.5 | Во | 0.3 | NULL | NULL |
| Б | Т | 0.5 | Я | 0.1 | NULL | NULL |
| Б | Т | 0.5 | М | 0.4 | NULL | NULL |
| Б | Т | 0.5 | С | 0.2 | NULL | NULL |
| Б | Н | 0.5 | Ва | 0.5 | С | 0.2 |
| Б | Н | 0.5 | Ва | 0.5 | Яг | 0.8 |
| Б | Н | 0.5 | О | 0.5 | NULL | NULL |

Таблица 2. Результат выполнения запроса

На втором этапе остается необходимость по каждому изделию получить конечный ненулевой компонент и его результирующее количество компоненты в этом изделии.

Вариант для T-SQL:

SELECT Изделие\_ID,

isNULL([Комп3], isNULL ([Комп2], [Комп1])) AS Компонента,

SUM(ISNULL([Колво3],1)\*ISNULL([Колво2],1)\*ISNULL([Колво1],1) AS Колво

FROM Создание\_иерархии

GROUP BY Изделие\_ID, isNULL([Комп3], isNULL([Комп2],[Комп1]));

Вариант для Access:

Например, так (если колво нецелое число, то приводить не к типу целого числа, а к типу дробного числа (Cdbl())):

SELECT Изделие\_ID,

CInt(nz([Комп3], CInt(nz([Комп2], [Комп1])))) AS Компонента, Sum(Cint(nz([Колво3],1))\*Cint(nz([Колво2],1))\*Cint(nz([Колво1],1))) AS Колво

FROM Создание\_иерархии

GROUP BY Изделие\_ID, CInt(nz([Комп3],CInt(nz([Комп2],[Комп1]))));

***Комментарий:***

*Аналогично можно построить граф полной подчиненности для задачи «начальник-подчиненный» в случае заранее известной глубины подчиненности.*

*Но понятно, что для большей глубины дерева, написание такого количества левых соединений, достаточно трудоемко, кроме того, при неизвестной глубине дерева, неизвестно, сколько левых соединений нужно сделать. Обобщением такой задачи мы будем заниматься на следующем занятии.*

**Литература:**

1. В.Н.Касьянов, В.А.Евстигнеев «Графы в программировании: обработка, визуализация и применение»
2. В.Н.Бурков, Д.А.Новиков «Элементы теории графов»
3. Джо Селко «SQL для профессионалов»