### Лекция 15:

1. Понятие доменной структуры хранения информации. Необходимость использования доменов. Примеры доменов.
2. Применение полной иерархии для доменной структуры.
3. Хранение изменений в доменной структуре данных. Организация таблиц логов. Триггеры.

При проектировании баз данных, а именно при создании взаимосвязей между различными сущностями, разработчик часто встречается с проблемами следующего рода:

* Как хранить различные сущности в БД? Можно ли нумеровать объекты из разных сущностей одинаковыми идентификаторами?
* Как устанавливать связи между сущностями наиболее простым и удобным способом?

Ответ на первый вопрос кажется понятным и давно известным. Логично хранить каждую сущность в отдельной таблице. Соответственно экземпляры разных сущностей могут иметь одинаковые идентификаторы.

Ответ на второй вопрос в принципе тоже понятен: мы изучали, что любая связь между объектами – это сущность. То есть все связи должны храниться в отдельной таблице. Но можно ли хранить несколько разнотипных связей в одной таблице и чем это может быть полезно?

Как пример, у Вас в БД N сущностей типа: сотрудники, отделы, права, товары, группы, категории, покупатели, домены покупателей. Если каждую сущность с расшифровкой Вы будете хранить в отдельной таблице, то всегда ли правильно для каждой связи заводить отдельную таблицу?

**Пример 1:** Один и тот же человек может быть сотрудником, должен иметь права на просмотр данных в определенных базах, имеет возможность покупать в магазине товары по оптовой цене, бесплатно обедать в столовой предприятия, может заходить на территорию предприятия через вход №1, но пропуска для входа через проходную №2 у этого сотрудника может не быть.

Как бы Вы хранили в БД все его свойства?

Перед ответом на вопрос, вспомните, что такое НФ, и учтите, что список свойств не обязательно будет ограничиваться описанными.

Рассмотрим структуру для хранения прав на просмотр данных в определенных базах:

У Вас есть отделы: секретариат и охрана. Секретариату необходимо иметь доступ к табельным отметкам сотрудников, а охране необходимо иметь доступ к базе пропусков на территорию предприятия. Информацию о том, кому и куда права нужны, мы хотим сами хранить в своей БД. Структура таблиц (в минимальном виде) следующая:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сотрудники** |  | **Права** |  | **Базы** |
| **Сотрудник\_ID** |  | **Отдел\_ID** |  | **База\_ID** |
| Отдел\_ID |  | База\_ID |  | База |

Как быстро определить, на какие базы есть права у сотрудника с заданным идентификатором?

SELECT \*

FROM Сотрудники

INNER JOIN Права

ON Сотрудники.Отдел\_ID = Права.Отдел\_ID

INNER JOIN Базы

ON Базы.База\_ID = Права.База\_ID

WHERE Сотрудник\_ID = @empl

В чём проблема у такого способа? Здесь Вам бы пришлось помнить о том, что права определяются в зависимости от отдела. Кроме того, если бы Вы решили права раздавать не по отделу, а по части отдела, в этом случае Вам бы пришлось полностью изменять всю структуру данных.

*Упражнение:* Подумайте, что пришлось бы сделать, если сотрудник 1 из секретариата должен уметь видеть и базу пропусков тоже (например, для печати пропусков), а сотрудник 2 из секретариата не должен? Возможно, Ваш способ будет лучше предложенного.

Попробуем реализовать распределение прав другим образом. Зависимость объектов друг от друга показана на рисунке. Видим, что эта зависимость представляет собой граф (не совсем дерево, так как мы решили выше, что права на базу можно выдавать и самому сотруднику напрямую). Как хранить графы мы знаем, единственное отличие заключается в том, что раньше у нас все вершины нумеровались уникальными идентификаторами, а сейчас мы договорились, что объекты различного типа могут нумероваться одинаковым образом. Поэтому необходимо сохранять в структуре для графа в том числе и типы объектов.

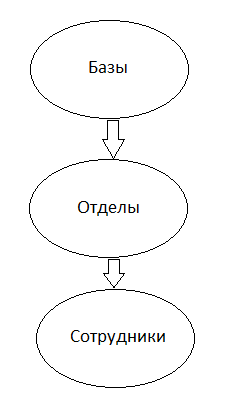


Рис.1. Иерархия сущностей.

Пусть у нас есть следующие сотрудники и отделы:

|  |  |
| --- | --- |
| **Сотрудник\_ID** | **Отдел\_ID** |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |

Пусть отдел 1 – секретариат, 2 – охрана. База табельных отметок – 1, База пропусков – 2. Типы объектов: 1 – Сотрудники, 2 – Отделы, 3 – Базы.

Давайте хранить информацию о правах в следующем виде. Заведем таблицу, в которой есть поля: p\_type, p\_id, ch\_type, ch\_id. Каждая строчка этой базы – это предложение вида «объект типа ch\_type с идентификатором ch\_id входит в объект типа p\_type с идентификатором p\_id».

|  |  |  |  |
| --- | --- | --- | --- |
| **p\_type** | **p\_id** | **ch\_type** | **ch\_id** |
| 2 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 |
| 2 | 2 | 1 | 3 |
| 3 | 1 | 2 | 1 |
| 3 | 2 | 2 | 2 |

Расшифровка полей:

ch\_id – идентификатор экземпляра описываемой сущности (child\_id);

ch\_type – идентификатор описываемой сущности (тип объекта ch\_id);

p\_id – идентификатор экземпляра сущности, находящейся выше на один уровень по иерархии, чем элемент ch\_id;

p\_type – соответственно идентификатор сущности p\_id (родителя – parent, тип объекта p\_id)

Что нужно сделать, чтобы получить полную информацию о каждом сотруднике? (Например, как ответить на вопрос, какие права есть у сотрудника?)

**Ответ: Построить полный граф!**

В чём преимущество этого способа по сравнению с предыдущим?

Например, мы можем выдать только сотруднику 1 дополнительные права следующим образом. Создать новый тип объектов «Роли» с идентификатором 4. Назвать конкретную роль «Выписка пропусков секретариатом», идентификатор роли = 1.

Добавим в нашу таблицу новые строки:

|  |  |  |  |
| --- | --- | --- | --- |
| **p\_type** | **p\_id** | **ch\_type** | **ch\_id** |
| 4 | 1 | 1 | 1 |
| 3 | 2 | 4 | 1 |

Первая строка говорит о том, что Вы сотрудника №1 включили в роль «Выписка пропусков секретариатом». А вторая строка, что этой роли выдали права на базу пропусков.

Недостаток: От отдельных таблиц, содержащих информацию о расшифровке экземпляров сущностей, мы всё равно не избавляемся. Кроме того появляется отдельная таблица, описывающая типы сущностей (объектов): 1 – Сотрудники, 2 – Отделы, 3 – Базы.

Но всё же, вместо N таблиц, описывающих попарные связи между отдельными экземплярами сущностей, мы заводим только 2 таблицы: одна с доменной структурой зависимостей и одна с расшифровкой типов объектов.

*Вопрос: Что бы Вы делали, если пытались реализовать права «Выписка пропусков» в структуре с одной таблицей, реализующей права сотрудников? Возможный вариант: Вам бы пришлось завести еще таблицу Права\_сотрудники {Сотрудник\_ID, База\_ID} и надеяться на то, что Вы не запутаетесь в реализации запросов на получение прав сотрудников.*

*Вопрос: Что нужно изменить/добавить в эту структуру, чтобы уметь по сотруднику определять и другие его возможности из примера?*

*Вопрос: Почему вариант раздавать каждому сотруднику права персонально может быть хуже, чем заведение однотипных ролей и раздача прав по ролям?*

*Упражнение:* Пусть у нас есть контрагенты: Иванов (сотрудник), Петров (сотрудник), Сидоров (оптовый покупатель), Кузнецов (поставщик). Нужно создать доменную структуру для хранения контрагентов, в которой сотрудники могли бы быть покупателями, оптовые покупатели не могли бы быть сотрудниками, а поставщики не были бы ни сотрудниками, ни покупателями, и покупатели не могли быть поставщиками.

*Возможный вариант:*

Основные сущности в нашей БД: люди и кем они могут быть (роли).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Контрагенты** | |  | **Роли** |  |
| IDC | Фамилия | | ID | Роль |
| 1 | Иванов |  | 1 | Сотрудники |
| 2 | Петров |  | 2 | Поставщики |
| 3 | Сидоров |  | 3 | Покупатели |
| 4 | Кузнецов | |  |  |

Соответственно необходимо завести таблицу с кодировкой типов этих сущностей:

|  |  |
| --- | --- |
| **Типы** |  |
| 1 | Контрагент |
| 2 | Роль |

Тогда таблица, описывающая структуру в упражнении, была бы следующей:

|  |  |  |  |
| --- | --- | --- | --- |
| p\_type | p\_id | ch\_type | ch\_id |
| 2 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 |
| 2 | 3 | 1 | 3 |
| 2 | 2 | 1 | 4 |
| 2 | 3 | 2 | 1 |

**Проиллюстрируем задание доменов на товарах и свойствах этих товаров.**

**Пример 2:** Товар «Мышь Logitech» обладает свойствами: 4 кнопки, серый цвет, относится к группе «Мыши Logitech», а группа «Мыши Logitech» относится к категории «Мыши». Товар «Монитор Samsung белый 19 дюймов» обладает свойствами: диагональ 19 дюйма, цвет: белый, относится к группе «Мониторы Samsung», которая в свою очередь относится к категории «Мониторы».

Ваша задача: по содержащейся в БД информации отображать на сайте следующие данные: товар по категории, проводить поиск товаров по свойству, отображать для категории наборы свойств, доступных для этой категории. Например, согласитесь, что бесполезно отображать на странице со списком всем доступных мониторов свойство «наличие колесика прокрутки», которое типично для мышей.

Исходно в базе данных содержатся следующие таблицы: таблица «Группы\_товары» содержит соответствие Товар\_ID – Группа\_ID, таблица «Категории\_группы»: Категория\_ID – Группа\_ID. Эти таблицы содержат взаимосвязи между сущностями Товары, Группы и Категории. Так же есть отдельные таблицы с расшифровкой, какие текстовые значения стоят за идентификаторами «Группа\_ID» и «Категория\_ID», таблицы соответственно называются «Категории», «Группы». Расшифровка для идентификатора Товар\_ID пусть лежит в таблице «Товары».

Для каждого товара есть свой список (расширяемый!!!) различных свойств: цвет, размеры, частоты, объем памяти, тип сокета и т.д. Так как список расширяемый, и для каждого типа товаров по сути свой, то хранить свойства как атрибуты сущности «Товары» не является правильным способом, так как для каждого нового свойства нам придется заводить новую колонку в таблице и проставлять значение параметра свойства для всех товаров. Поэтому лучшим способом хранения такой информации станет создание дополнительной сущности «Свойства», содержащейся в отдельной таблице, а также дополнительная таблица «Товары\_свойства», реализующая связь многие-ко-многим для сущностей Товары и Свойства и содержащей значение каждого свойства для данного товара.

Таблица «Товары\_свойства» содержит соответствия: Товар\_ID – Свойство\_ID – Значение. Типы свойств могут быть следующими: количество кнопок, цвет, размеры упаковки, производитель и т.д.

Описанная схема взаимодействия сущностей может быть представлена в следующем виде:

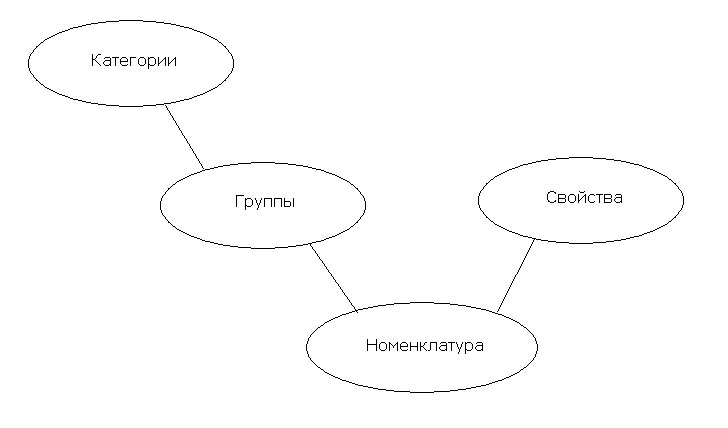


Рис.2. Иерархия сущностей.

Типы (домены) объектов:

|  |  |
| --- | --- |
| Тип\_id | Тип |
| 1 | Товары |
| 2 | Группы |
| 3 | Категории |
| 4 | Свойства |

Пример. Рассмотрим более подробно на следующем примере:

Таблица «Товары»

|  |  |
| --- | --- |
| Товар\_ID | Товар |
| 1 | Мышка Logitech беспроводная |
| 2 | Монитор Samsung 19'' |

Таблица «Группы»

|  |  |
| --- | --- |
| Группа\_ID | Группа |
| 1 | Мыши Logitech |
| 2 | Мониторы Samsung |

Таблица «Категории»

|  |  |
| --- | --- |
| Категория\_ID | Категория |
| 1 | Мыши |
| 2 | Мониторы |

Таблица «Свойства»

|  |  |
| --- | --- |
| Свойство\_ID | Свойство |
| 1 | Цвет |
| 2 | Матрица |

Таблицы «Группы\_товары» и «Категории\_группы».

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа\_ID | Товар\_ID |  | Категория\_ID | Группа\_ID |
| 1 | 1 |  | 1 | 1 |
| 2 | 2 |  | 2 | 2 |

Таблица «Товары\_свойства» (для описания этого нужно заводить отдельные объекты (сущности): цвета, матрицы и т.д.). Значения так же можно было бы завести в виде отдельных сущностей и возможных наборов значений. Здесь значения просто в текстовом виде.

|  |  |  |
| --- | --- | --- |
| Товар\_ID | Свойство\_ID | Значение |
| 1 | 1 | белый |
| 2 | 1 | черный |
| 2 | 2 | PVA |

В результате информацию можем хранить следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| p\_type | p\_id | ch\_type | сh\_id |
| 2 | 1 | 1 | 1 |
| 2 | 2 | 1 | 2 |
| 4 | 1 | 1 | 1 |
| 4 | 1 | 1 | 2 |
| 4 | 2 | 1 | 2 |
| 3 | 1 | 2 | 1 |
| 3 | 2 | 2 | 2 |

Расшифровка полей (на примере двух первых строк):

ch\_id – идентификатор экземпляра описываемой сущности;

(В нашем случае, например, идентификаторы товаров из номенклатуры)

ch\_type – идентификатор описываемой сущности;

(так как считаем, что в ch\_id – экземпляр сущности «товары», то ch\_type – домен (тип) сущности из таблицы равный 1)

p\_id – идентификатор экземпляра сущности, находящейся выше на один уровень по иерархии, чем элемент ch\_id;

(например, товар входит в определенную группу товаров, идентификатор этой группы и следует указать);

p\_type – соответственно идентификатор сущности p\_id (родителя - parent)

(в нашем случае p\_id экземпляр сущности группа, значит цифра 2).

Таблицу назовем «Домены\_элементы».

Первые 2 строки отвечают за нахождение товаров (элементов из сущности Товары) в конкретных группах (элементах из сущности Группы), следующие 3 строки отвечают за обладание товарами некоторых свойств (элементов из сущности «Свойства»), последние 2 строки отвечают за вхождение элементов сущности «Группы» в соответствующие элементы сущности «Категории».

Какие преимущества у данной конструкции?

С помощью следующего запроса:

SELECT \* FROM Домены\_элементы WHERE ch\_id = 1 AND ch\_type = 1

Мы получим всю информацию о товаре 1 (правда, только непосредственную, категорию товара мы пока получить не можем).

Для того чтобы получить полную информацию об элементе, включая не только непосредственно граничащие в иерархии, сущности, достаточно вспомнить материал предыдущих лекций о построении полной иерархии. После построения полного графа мы сможем ответить на вопрос, а какие свойства нужно отображать на странице поиска товаров, если мы открыли категорию «Мониторы»?

Заметим, что такая структура расширяема практически до бесконечности, на текущий момент в нее можно добавить абсолютно любые сущности и взаимосвязи между сущностями.

Проблема: мы не сможем хранить значения свойств в той же таблице.

### Как следует хранить изменения в доменной структуре?

*Во-первых, что понимается под изменениями в доменной структуре?*

Изменения в доменной структуре - перемещение элемента из одного домена в другой, либо добавление элемента в какой-либо новый домен, либо удаление элемента из определенного домена.

Примерами могут быть:

1. Прием и увольнение сотрудника (что будет, если не заводить домены для уволенных сотрудников?).
2. Перевод сотрудника из одного отдела в другой.
3. Добавление нового артикула.
4. Добавление нового свойства у артикула.
5. Перемещение группы товара из одной категории в другую, перемещение товара из одной группы в другую.

То есть ситуации, когда вместо строки

|  |  |  |  |
| --- | --- | --- | --- |
| p\_type | p\_ID | ch\_type | ch\_ID |
| 2 | 1 | 1 | 1 |

Возникает строка:

|  |  |  |  |
| --- | --- | --- | --- |
| p\_type | p\_ID | ch\_type | ch\_ID |
| 2 | 2 | 1 | 1 |

(Например, сущность (тип) «2» - отделы, сущность (тип) «1» - сотрудники. Здесь сотрудника перевели из 1-ого отдела во 2-й.)

Либо когда такая строка вообще исчезает (у нас нет отдела уволенных, а мы увольняем сотрудника), либо когда появляется новая строка такого типа (сотрудника впервые приняли на работу).

Понятно, что перемещение элемента из одного домена в другой – это удаление элемента из одного домена и заведение элемента в другом. Таким образом, нужно уметь описывать только явления добавления и удаления элементов из домена.

*Во-вторых, как хранить изменения таким образом, чтобы можно было восстановить ситуацию на каждый момент времени достаточно простым запросом?*

*Упражнение 1. Прежде чем приступать к прочтению следующего материала, подумайте, как хранить изменения (какие поля необходимы), чтобы можно было восстановить ситуацию на любой момент времени. Может быть, Ваша версия окажется лучше?*

Так как все изменения, как оказалось, в результате сводятся к исчезновению (возникновению) строчек вида

|  |  |  |  |
| --- | --- | --- | --- |
| p\_type | p\_ID | ch\_type | ch\_ID |

То, соответственно, имеет смысл хранить эту строку, + дату изменения, + сторону изменения (то есть, удалили или добавили).

И тогда структура таблицы для хранения логов будет следующей:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Дата | Оператор | Действие | p\_type | p\_ID | ch\_type | ch\_ID |

А данные, по приведенным выше примерам, такие

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Дата | Оператор | Действие | p\_type | p\_ID | ch\_type | ch\_ID |
| 10:00 | 1 | -1 | 2 | 1 | 1 | 1 |
| 10:01 | 1 | 1 | 2 | 2 | 1 | 1 |

Где Дата – дата изменения, Действие – что сделали со строкой: «-1» – удалили, «1» – добавили.

Таблицу будем называть «Лог».

*Как писать запросы, используя данную структуру?*

*Задача: Пусть перед нами стоит задача: получить всех сотрудников заданного параметром отдела на определенную дату (также заданную параметром).*

*Вариант 1:*

Предположим, что информация в таблице "Лог” полная, то есть лог заполнялся с самого начала существования таблицы.

Пример такого запроса:

Пусть сущность (домен/тип) сотрудников – 1, сущность (домен/тип) отделов – 2.

Если таблица логов велась с самого начала существования таблицы с данными по сотрудникам и отделам, то, если сотрудник на дату расчета есть в отделе, то колво состояний его по этому отделу будет нечетным, если нет, то четным.

SELECT ch\_id, SUM(Действие) Наличие

FROM Лог

WHERE Дата < @Дата\_Расчета AND p\_ID = @Нужный\_отдел AND p\_type = 2 AND ch\_type = 1

GROUP BY ch\_ID

HAVING SUM(Действие) != 0

(Так же можно решать задачу про получение отдела конкретного сотрудника на дату, заданную параметром)

*Вариант 2:*

Получить всех сотрудников определенного отдела на определенную дату, учитывая то, что таблица логов была заведена после того, как некоторое время работали (вносили изменения: добавляли, удаляли сотрудников, переносили из отдела в отдел) с таблицей сотрудников. При заведении таблицы логов в нее НЕ залили состояние доменов (распределение сотрудников по отделам) на момент создания (то есть в таблице с логом только изменения, которые проводились после создания таблицы логов).

В чем отличие данной постановки от постановки в случае одновременного заведения обеих таблиц?

Основные проблемы, которых раньше не было (почему нельзя использовать предыдущий запрос), это:

* если сотрудник был переведен в отдел до заведения таблицы логов и больше никуда не переводился, то его в этой таблице не окажется;
* если сотрудник был переведен в отдел до заведения таблицы логов и переводился в какой-либо отдел после нашей даты расчетов, то в выборке его опять же не окажется (то есть предыдущий запрос использовать нельзя);
* дата расчета может быть как раньше, так и позже чем дата заведения таблицы логов (это просто добавляет сложности к интерпретации результатов);

Как в этом случае поступать?

Предположения:

* Считаем, что добавление сотрудника в новый отдел и удаление из старого происходят одновременно (*что может произойти с результатом запроса, если на самом деле это не так? Влияет ли это на предыдущий запрос?*).
* Если по сотруднику в момент времени только одна запись, то: если это добавление – сотрудника только что приняли на работу, если удаление – то сотрудника уволили (нужен ли домен уволенных сотрудников?).
* Текущее состояние сотрудника обязательно есть в таблице «Сотрудники».

Примеры, которые должны помочь с пониманием:

Нам нужно состояние отдела 1 на 01.01.21. У нас есть информация, что лог ведется с 01.01.20

Пример 1 (содержимое лога):

|  |  |  |  |
| --- | --- | --- | --- |
| **Дата** | **Сотрудник** | **Отдел** | **Действие** |
| 01.03.21 | 1 | 1 | -1 |
| 01.03.21 | 1 | 2 | 1 |

Содержимое таблицы Сотрудники:

|  |  |
| --- | --- |
| **Сотрудник** | **Отдел** |
| 1 | 2 |

Очевидно ли, что сотрудник 1 был в отделе 1 на 01.01.21?

Пример 2: в логах нет данных по сотруднику 2

В таблице Сотрудники:

|  |  |
| --- | --- |
| **Сотрудник** | **Отдел** |
| 2 | 1 |

Делаем вывод, что на 01.01.21 сотрудник 2 был в отделе 1.

Пример 3:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Дата** | **Сотрудник** | | | **Отдел** | **Действие** |
| 01.12.20 | 3 | | | 2 | -1 |
| 01.12.20 | 3 | | | 1 | 1 |
| **Сотрудник** | | **Отдел** |
| 3 | | 1 |

На 01.01.21 сотрудник 3 был в отделе1.

*Пример запроса (напишите свой, возможно он будет лучше):*

Получим по всем работникам состояние на нужную нам дату, а затем выберем

SELECT ch\_ID, p\_ID

FROM (

-- все изменения с расчетного момента в таблице Логов

SELECT ch\_ID, Дата, Действие, p\_ID

FROM Логи

WHERE Дата > @Дата\_Расчета AND p\_type = 2 AND ch\_type = 1

UNION

-- текущее состояние со знаком минус

SELECT ch\_ID, Getdate(), -1, p\_ID

FROM Сотрудники

WHERE p\_type = 2 AND ch\_type = 1

) Итог

GROUP BY ch\_ID, p\_ID

HAVING SUM(Действие) <> 0 AND p\_ID = @Нужный\_отдел

Здесь: отдел без пары будет в том случае, если сотрудника из него исключили после даты расчета, а, значит, на дату расчета он был в этом отделе.

Вторая версия годится и для случая, когда у нас есть полный лог, но работать будет медленнее, так как есть UNION.

**Триггеры:**

Триггеры – специальные процедуры, которые выполняются автоматически. Можно использовать триггеры для случаев изменения данных в таблицах или изменения схемы данных. Триггеры могут срабатывать и до, и после инструкции, которая изменяет состояние базы.

Как создать Триггер. Пример:

Есть таблица Данные (поля id и val) и таблица Данные\_лог (поля Дата, id, val, action). В свойствах для таблицы Данные (разверните дерево у таблицы) найдите пункт «Triggers». Выберите: создать новый триггер.

CREATE TRIGGER [dbo].[Лог\_insert]

ON [dbo].[Данные]

FOR INSERT

AS

BEGIN

SET NOCOUNT ON;

-- Insert statements for trigger here

INSERT INTO Данные\_лог (Дата, id, val, Action)

SELECT GETDATE(), id, val, 1

FROM INSERTED

END

После создания такого триггера, при добавлении данных в таблицу Данные в таблице Данные\_лог будет храниться лог изменений: дата, когда добавили строку, сама добавленная строка с действием = 1, обозначающим добавление.

Аналогично Вы можете добавить триггеры на изменение значения, на удаление строки и т.д.