УДК 004.4’244

С.Н. Гриша, О.С. Родичева, Е.О. Роздольский

ПОСТРОЕНИЕ ТЕКСТА SQL ЗАПРОСА ИЗ РЕЛЯЦИОННОГО ДЕРЕВА ЗАПРОСА

Постановка задачи

В статье [1] рассматривался общий подход к автоматизации процесса проектирования систем управления бизнесом. В ней были обоснованы актуальность и необходимость автоматического построения базы данных (БД) и программного обеспечения (ПО) на основе семантического описания предметной области.

При решении задачи автоматического построения БД и ПО необходимо решить ряд технических и теоретических задач. В предложенном подходе аналитик-разработчик описывает систему в терминах показателей. Это означает, что аналитик-разработчик сначала проектирует систему показателей, и после этого для каждого документа системы задает ему запрос на выборку над показателями, используя правила реляционной алгебры. При генерировании БД и ПО специальный алгоритм генерирует БД и при генерировании ПО алгоритм замещает запросы реляционной алгебры над показателями запросами над таблицами, канонизирует полученные выражения и преобразует их в текст SQL. В этой статье решается задача преобразования реляционного дерева запроса над таблицами в текст SQL. Входные данные для задачи – это реляционное дерево запроса (соответствует реляционной алгебре Кода [2]). На основании его необходимо построить SQL запрос стандарта SQL-92.

Обзор публикаций

При изучении публикаций на выбранную тематику было найдено много работ по преобразованию SQL-запросов в реляционное дерево [3], но практически отсутствуют материалы по обратному преобразованию. Есть материалы по преобразованию одной структурированной информации в другую. Так, в [4] описан метод преобразования структурированных документов одной грамматики в другую, причем это должны быть контекстно-независимые грамматики и входные данные – структурированный текст.

 ISSN 1562-9945

Также существуют преобразователи xml-документов, такие как XSLT [5]. Но такие преобразователи тоже не подходят для решения поставленной задачи, так как преобразуют из xml в xml и малопригодны для преобразования в SQL.

Изучение трансформационно-порождающих (ТП) грамматик в литературе показало, что большинство исследований направлено на изучение языковых структур и лингвистических преобразований из одного разговорного языка в другой [6]. Применение же термина "ТП грамматика" не в области лингвистики встречалось крайне редко. Несмотря на это, идея применения ТП грамматики в лингвистике подходит и для преобразования других структурированных языков, но только контекстно-независимых. Для контекстно-зависимых грамматик также описаны алгоритмы, но они неточны и требуют конкретного описания для каждого языка.

Алгоритм

Для преобразования реляционного дерева в SQL запрос применяется трансформационная порождающая грамматика, на вход которой подается реляционное дерево S (запускается выполнение продукций грамматики). Рассмотрена была стандартная грамматика SQL стандарта ANSI [7] в расширенной форме Бэкуса-Наура [8].

На применение порождающей грамматики накладываются такие условия:

1. за один шаг должны быть применена одна продукция, т.е. дерево S уменьшиться на один узел;

2. продукции должны просматриваться в том порядке, в котором они представлены.

Каждый реляционный оператор порождает или терминал(ы) или появление вложенного подзапроса, причем второе случается, если соответствующая позиция в текущем транслированном SQL-выражении уже занята.

Очевидно, грамматика для преобразования контекстно-зависима[9], так как узлам реляционного дерева соответствуют несколько правил вывода, что зависит от текущего вида полученного выражения SQL. Например, рассмотрим такие два реляционных дерева: $Q_1 = \gamma_A(\sigma_C(T))$ и $Q_2 = \sigma_C(\gamma_A(T))$. В первом случае при преобразовании в SQL запрос необходимо в нем сделать подзапрос, во

втором случае не нужно хотя операторы использованы одинаковые. SQL запросы для деревьев Q_1 и Q_2 будут такими:

select(Q_1) = select A from (select * from T where C) group by A / A_{agg}

select(Q_2) = select A from T where C group by A / A_{agg}

где A_{agg} - агрегирующие аргументы оператора группировки.

Таким образом, терминальные символы вместе с нетерминалом будут определять продукцию в трансформационной грамматике. Для упрощения грамматики (уменьшения количества правил) добавим предварительную обработку дерева для его упорядочивания. Сделать это можно после этапа канонизации, добавив в правила канонизации правила упорядочивания структуры дерева.

Определим необходимый порядок для структуры дерева. Пусть это будет такой порядок для операторов с одним аргументом: $\delta, \tau, \pi, \sigma, \gamma$. Операторы с двумя аргументами (операторы соединения и объединения) тут нельзя рассматривать так это относиться к оптимизации запроса, над чем работает алгоритм канонизации. Алгоритм канонизации выражений меняет дерево запроса так, чтобы привести его к оптимальной структуре запроса. Изменения же порядка в дереве необходимо только для упрощения процесса построения текста SQL запроса и не влияет на быстродействие запроса.

Для получения нужного порядка необходимо применить такие правила как показано в таблице Таблица 1. По строкам показан первый оператор, по колонкам – оператор-аргумент. В соответствующих ячейках на пересечении элементов стоит:

– прочерк в том случае, если такой порядок следования операторов корректен, и его не нужно менять;

– "$\rightarrow Q$" в том случае, если соответствующее изменение порядка не может быть сделано (или из-за несоответствия аргументов или из-за несохранения результирующего набора данных);

– формула – это правая часть продукции, которую необходимо применить к соответствующим операторам, чтобы упорядочить дерево; левая часть продукции образуется из значения строки и колонки.

 ISSN 1562-9945

Таблица 1

Правила упорядочивания дерева

	(δ)	$(\pi_{L'})$	$(\sigma_{C'})$	$(\gamma_{A'})$	$(\tau_{M'})$
δ	$\to \delta(*)$	—	—	—	—
π_L	$\to \delta(\pi_L(*))$	$\to \pi_{L(L')}(*)$	—	—	$\to Q$
σ_C	$\to \delta(\sigma_C(*))$	$\to \pi_{L'}(\sigma_C(*))$	$\to \sigma_{C \text{ and } C'}(*)$	—	$\to \tau_M(\sigma_C(*))$
γ_A	$\to Q$	$\to Q$	$\to Q$	$\to Q$	$\to Q$
τ_M	$\to \delta(\tau_M(*))$	—	—	—	$\to \tau_M(*)$

Рассмотрим подробнее таблицу. В ячейках по диагонали (когда оператор и его аргумент одинаковы), происходит слияние для всех операторов кроме группировки γ_A с изменением аргументов. Если аргумент оператора группировки не объединение, соединение или таблица, то этот аргумент порождает вложенный подзапрос. Все аргументы оператора селекции кроме группировки поднимаются вверх по дереву запроса. Оператор группировки Оператор удаления дубликатов всегда необходимо перемещать по дереву вверх. Это возможно во всех случаях кроме группировки, так как это повлияло бы на результирующий набор данных. Оператор проекции можно поднимать вверх по дереву только если он является аргументом селекции и сливать с оператором проекции. Оператор сортировки поднимается вверх по дереву, если он является аргументом селекции, в остальных случаях или порядок правильный, или порядок нельзя менять, так как это приведет к изменению набора данных.

Таким образом, получаем следующий набор продукций грамматики $G_O\{T_O, N_O, P_O, S_O\}$ для упорядочивания дерева:

$T_0 = U \setminus N_0, N = \{\pi_L, \sigma_C, y_A, \delta, \tau_M\}$

P_O:

$\delta(\delta(*)) \to \delta(*)$

$\pi_L(\delta(*)) \to \delta(\pi_L(*))$

$\pi_L(\pi_{L'}(*)) \to \pi_{L(L')}(*)$

$\sigma_C(\delta(*)) \to \delta(\sigma_C(*))$

$\sigma_C(\pi_{L'}(*)) \to \pi_{L'}(\sigma_C(*))$

$\sigma_C(\sigma_{C'}(*)) \to \sigma_{C \text{ and } C'}(*)$

$\sigma_C(\tau_M(*)) \to \tau_M(\sigma_C(*))$

$\tau_M(\delta(*)) \to \delta(\tau_M(*))$

$\tau_M(\tau_{M'}(*)) \to \tau_M(*)$

Этот набор продукций необходимо применить к реляционному дереву перед применением порождающей грамматики. Тогда порождающая грамматика упрощается так и будет обрабатывать операторы по мере их появления. Таким образом, получаем грамматику $G(T,N,P,S)$ такую:

Терминалы грамматики G:

$T = \{select, from, where, left, right, full, natural, join,$
$distinct, order\ by, group\ by, *, dual, t_1, ..., t_n, l_1, ..., l_m\}$

где $t_1, ...t_n$ - названия таблиц, $l_1, ..., l_m$ - названия всех аргументов реляционных операторов кроме таблиц каждого дерева: $t_1, ...t_n, l_1, ..., l_m \notin T \setminus \{t_1, ...t_n, l_1, ..., l_m\}$

Нетерминалы грамматики G:

$N = \{\delta, \tau, \pi, \sigma, \gamma, \times, \triangleright\triangleleft, \underset{C}{\triangleright\triangleleft}, \overset{\circ}{\triangleright\triangleleft}, \overset{\circ}{\triangleright\triangleleft}_L, \overset{\circ}{\triangleright\triangleleft}_R, \underset{C}{\overset{\circ}{\triangleright\triangleleft}}, \cup\}$

Продукции грамматики:

$\varepsilon \to \varepsilon S$

$\varepsilon S \to S(Q)$

$(t_i)(Q) \to t_i$

$S(Q) \to (\text{select S(from)})$

$\delta(S)(\text{from}) \to \text{distinct } S(\text{ from})$

$\tau_{l_i}(S)(\text{from}) \to S(\text{from}) \text{ order by } l_i$

$\pi_{l_i}(S)(\text{from}) \to S(l_i \text{ from})$

$\sigma_{l_i}(S)(l_j \text{ from}) \to S(l_j \text{ from where } l_i)$

$\sigma_{l_i}(S)(\text{from}) \to S(* \text{ from where } l_i)$

$\gamma_{l_i \to l_i'}(S)(\text{from}) \to l_i' \text{ from } S(Q) \text{ group by } l_i$

$\gamma_{l_i \to l_i'}(S)(l_j \text{ from}) \to l_j \text{ from } (\gamma_{l_i \to l_i'}(S)(Q))$

$\gamma_{l_i \to l_i}(S)(l_j \text{ from where } l_k) \to l_j \text{ from } (\gamma_{l_i \to l_i'}(S)(Q)) \text{ where } l_k$

$\gamma_{l_i \to l_i}(S)(* \text{ from where } l_j) \to l_i' \text{ from } (S) \text{ where } l_j \text{ group by } l_i$

$(t_i)(\text{ from}) \to * \text{ from } t_i$

$(t_i)(l_i \text{ from}) \to l_i \text{ from } t_i$

$(t_i)(l_j \text{ from where } l_i) \to l_j \text{ from } t_i \text{ where } l_i$

 ISSN 1562-9945

$(t_i)(*\ \text{from where}\ l_i) \to *\ \text{from}\ t_i\ \text{where}\ l_i$

$(t_i) \to t_i$

$\times(S_1, S_2)(l_i\ \text{from}) \to l_i\ \text{from}\ \times(S_1, S_2)$

$\times(S_1, S_2)(l_j\ \text{from where}\ l_i) \to l_j\ \text{from}\ \times(S_1, S_2)\ \text{where}\ l_i$

$\times(S_1, S_2)(*\ \text{from where}\ l_i) \to *\ \text{from}\ \times(S_1, S_2)\ \text{where}\ l_i$

$\times(S_1, S_2)(\text{from}) \to \text{from}\ \times(S_1, S_2)$

$(\times(S_1, S_2)) \to (S_1)(Q), (S_2)(Q)$

$\triangleright\triangleleft(S_1, S_2)(l_i\ \text{from}) \to l_i\ \text{from}\ \triangleright\triangleleft(S_1, S_2)$

$\triangleright\triangleleft(S_1, S_2)(l_j\ \text{from where}\ l_i) \to l_j\ \text{from}\ \triangleright\triangleleft(S_1, S_2)\ \text{where}\ l_i$

$\triangleright\triangleleft(S_1, S_2)(*\ \text{from where}\ l_i) \to *\ \text{from}\ \triangleright\triangleleft(S_1, S_2)\ \text{where}\ l_i$

$\triangleright\triangleleft(S_1, S_2)(\text{from}) \to \text{from}\ \triangleright\triangleleft(S_1, S_2)$

$(\triangleright\triangleleft(S_1, S_2)) \to (S_1)(Q)\ \text{natural join}\ (S_2)(Q)$

$\underset{l_i}{\triangleright\triangleleft}(S_1, S_2)(l_j\ \text{from}) \to l_j\ \text{from}\ \underset{l_i}{\triangleright\triangleleft}(S_1, S_2)$

$\underset{l_i}{\triangleright\triangleleft}(S_1, S_2)(l_j\ \text{from where}\ l_k) \to l_j\ \text{from}\ \underset{l_i}{\triangleright\triangleleft}(S_1, S_2)\ \text{where}\ l_k$

$\underset{l_i}{\triangleright\triangleleft}(S_1, S_2)(*\ \text{from where}\ l_j) \to *\ \text{from}\ \underset{l_i}{\triangleright\triangleleft}(S_1, S_2)\ \text{where}\ l_j$

$\underset{l_i}{\triangleright\triangleleft}(S_1, S_2)(\text{from}) \to \text{from}\ \underset{l_i}{\triangleright\triangleleft}(S_1, S_2)$

$(\underset{l_i}{\triangleright\triangleleft}(S_1, S_2)) \to (S_1)(Q)\ \text{join}\ (S_2)(Q)\ \text{on}\ l_i$

$\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)(l_j\ \text{from}) \to l_j\ \text{from}\ \overset{\circ}{\triangleright\triangleleft}(S_1, S_2)$

$\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)(l_j\ \text{from where}\ l_k) \to l_j\ \text{from}\ \overset{\circ}{\triangleright\triangleleft}(S_1, S_2)\ \text{where}\ l_k$

$\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)(*\ \text{from where}\ l_j) \to *\ \text{from}\ \overset{\circ}{\triangleright\triangleleft}(S_1, S_2)\ \text{where}\ l_j$

$\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)(\text{from}) \to \text{from}\ \overset{\circ}{\triangleright\triangleleft}(S_1, S_2)$

$(\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)) \to (S_1)(Q)\ \text{full join}\ (S_2)(Q)$

$\underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)(l_j\ \text{from}) \to l_j\ \text{from}\ \underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)$

$\underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)(l_j\ \text{from where}\ l_k) \to l_j\ \text{from}\ \underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)\ \text{where}\ l_k$

$\underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)(*\ \text{from where}\ l_j) \to *\ \text{from}\ \underset{l_i}{\overset{\circ}{\triangleright\triangleleft}}_X(S_1, S_2)\ \text{where}\ l_j$

$\overset{\circ}{\triangleright\triangleleft}(S_1, S_2)(\text{from}) \to \text{from}\ \overset{\circ}{\triangleright\triangleleft}(S_1, S_2)$

$(\overset{\circ}{\underset{l_i}{\triangleright\triangleleft}}_X(S_1,S_2)) \to (S_1)(Q)$ full join $(S_2)(Q)$ on l_i

$\cup(S_1,S_2)(l_j \text{ from}) \to l_j$ from $\cup(S_1,S_2)$

$\cup(S_1,S_2)(l_j$ from where $l_k) \to l_j$ from $\cup(S_1,S_2)$ where l_k

$\cup(S_1,S_2)(*$ from where $l_j) \to *$ from $\cup(S_1,S_2)$ where l_j

$\cup(S_1,S_2)(\text{from}) \to$ from $\cup(S_1,S_2)$

$\cup(S_1,S_2) \to (S_1)(Q)$ union $((S_2)(Q))$

Граф переходов автомата, соответствующего грамматике G, показан на рисунке Рисунок 1. Для уменьшения количества узлов, соответствующих операторам соединения и объединения сделана такая замена:

$$\otimes = \{\times, \triangleright\triangleleft, \underset{l_i}{\triangleright\triangleleft}, \overset{\circ}{\triangleright\triangleleft}, \overset{\circ}{\underset{l_i}{\triangleright\triangleleft}}_X, \cup\}.$$

Таким образом, получили алгоритм преобразования реляционного дерева в SQL запрос:

1. подготовить дерево: построить соответствие между всеми конечными аргументами, кроме таблиц, всех операторов и обозначениями $l_1,...,l_m$.

2. применить упорядочивающую грамматику G_O к реляционному дереву S;

3. применить порождающую грамматику G к упорядоченному дереву – результату применения продукций G_O к реляционному выражению S и получить терминальную цепочку;

4. в полученной терминальной цепочке заменить все $l_i, i = \overline{1,m}$ на соответствующие им значения аргументов.

Оба описанных алгоритма являются модификациями одного и того же подхода – применения трансформационной порождающей контекстно-зависимой грамматики к реляционному дереву запроса.

 ISSN 1562-9945

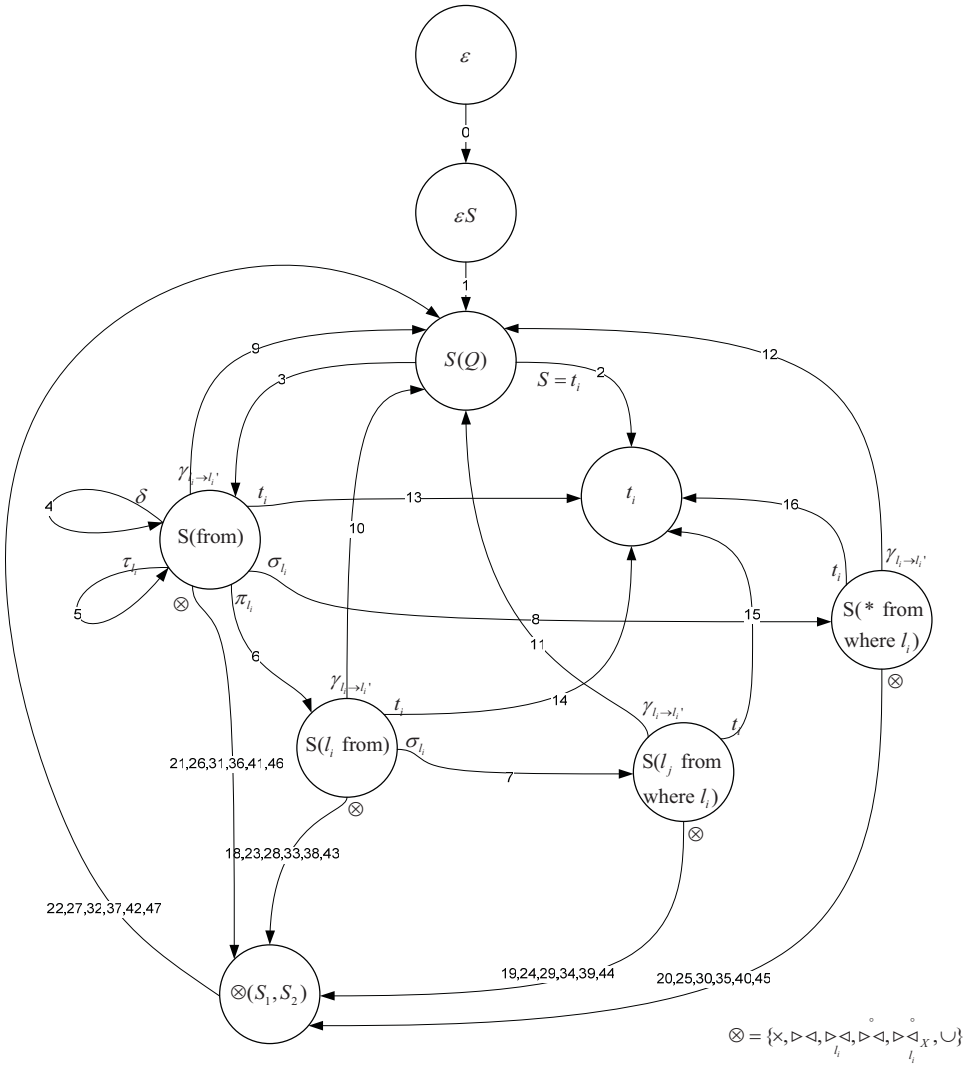

Рисунок 1 – Граф переходов автомата

Примеры преобразования

Рассмотрим пример преобразования дерева $S = \sigma_C(\pi_L(\times(t_1, t_2)))$

Применим алгоритм к заданному примеру.

Шаг 1. Упорядочивание дерева с помощью G_O.

Таблица 2

Упорядочивание дерева G_O

1) $\sigma_C(\pi_L(\times(t_1, t_2))) \to \pi_L(\sigma_C(\times(t_1, t_2)))$	Правило 5

Шаг 2. Применение грамматики G

Таблица 3

Применение грамматики G

1) $\varepsilon \to \varepsilon\pi_L(\sigma_C(\times(t_1,t_2)))$	Правило 0
2) $\varepsilon\pi_L(\sigma_C(\times(t_1,t_2))) \to \pi_L(\sigma_C(\times(t_1,t_2)))(Q)$	Правило 1
3) $\pi_L(\sigma_C(\times(t_1,t_2)))(Q) \to (\text{select } \pi_L(\sigma_C(\times(t_1,t_2)))(\text{from}))$	Правило 3
4) $(\text{select } \pi_L(\sigma_C(\times(t_1,t_2)))(\text{from})) \to (\text{select } \sigma_C(\times(t_1,t_2))(l_1 \text{ from}))$	Правило 6, $L = l_1$
5) $(\text{select } \sigma_C(\times(t_1,t_2))(l_1 \text{ from})) \to (\text{select } \times(t_1,t_2)(l_1 \text{ from where } l_2))$	Правило 7, $C = l_2$
6) $(\text{select } \times(t_1,t_2)(l_1 \text{ from where } l_2)) \to (\text{select } l_1 \text{ from } \times(t_1,t_2) \text{ where } l_2)$	Правило 19
7) $(\text{select } l_1 \text{ from } \times(t_1,t_2) \text{ where } l_2) \to (\text{select } l_1 \text{ from } t_1(Q),t_2(Q) \text{ where } l_2)$	Правило 22
8) $(\text{select } l_1 \text{ from } t_1(Q),t_2(Q) \text{ where } l_2) \to (\text{select } l_1 \text{ from } t_1,t_2 \text{ where } l_2)$	Правило 2

Таким образом, получили

$S = \sigma_C(\pi_L(\times(t_1,t_2))) \to (\text{select } l_1 \text{ from } t_1,t_2 \text{ where } l_2)$

Осталось заменить $l_i, i = \overline{1,2}$ на соответствующие значения аргументов реляционных операторов и текст SQL запроса готов.

Программные реализация преобразования реляционного дерева в текст SQL запроса

Текст SQL запроса можно генерировать из реляционного дерева, представленного в виде:

1. xml-файла;
2. объектной программной модели;
3. таблицы базы данных.

Все три варианта программного представления реляционного дерева взаимозаменяемы, и при наличии одного из них это представление легко преобразовать в другой.

 ISSN 1562-9945

В зависимости от вида входных данных можно применить такие программные подходы к реализации генеративной грамматики:

1. из xml-описания дерева с использованием сгенерированного с помощью генератора интерпретатора (например модификации Coco/R или ANTLR [10]) парсера xml;

2. также с использованием генератора интерпретатора (например, модификации Coco/R или ANTLR), но при этом алгоритм сразу преобразует входной xml-файл в sql запрос, используя порождающую грамматику;

3. из объектной программной модели: для этого необходимо осуществить обход дерева объектов в глубину с использованием описанных правил грамматики.

В нашей реализации выбран метод обхода объектного дерева, так как реляционное дерево представлено в программной реализации системы в виде объектной модели.

Выводы

В статье рассмотрена и решена задача преобразования реляционного дерева в текст SQL запроса. Сделан обзор существующих публикаций на данную тематику. Показано алгоритм преобразования с использованием порождающей трансформационной грамматики. Показаны примеры применения грамматики к конкретным примерам реляционных деревьев и приведено описание программной реализации алгоритма.

Эти результаты являются частью технологии интеллектуализированного генерирования базы данных и программного обеспечения "КИТ-XXI", основные концепции которой описаны в [1].

ЛИТЕРАТУРА

1. Гриша С.М., Родічева О.С., Приліпко Д.І. Технологічно інтелектуалізовані інформаційні системи для управління бізнесом (ІСУБ) на сонові алгебри показників // Вісник Національного Технічного Університету «ХПІ» – 2008. – №5. – С.123-133.
2. Codd, E.F. *The Relational Model For Database Management Version 2*. Reading, Mass.: Addison-Wesley, 1990. ISBN 0-201-14192-2.
3. Гарсиа-Молина Г., Ульман Д., Уидом Д. Системы баз данных. Полный курс. – М.: Вильямс, 2002. – 1088 с.

4. Eila Kuikka, Martti Penttonen, Transformation of structured documents with the use of grammar. Electronic publishing, VOL. 6(4), 373–383 (december 1993).
5. M. Kay. XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation 3th November 2005. November 1999.
6. Касевич В. Б. Элементы общей лингвистики. М.: Наука, 1977.
7. Электронный ресурс: http://savage.net.au/SQL/sql-92.bnf
8. Электронный ресурс: ISO/IEC 14977 : 1996(E)
9. Chomsky, Noam (1956). "Three models for the description of language". IRE Transactions on Information Theory (2): 113–124
10. Электронный ресурс: The Catalog of Compiler Construction Tools: http://catalog.compilertools.net/

Получено 05.02.2009г.

 ISSN 1562-9945