УДК 681.3.01

А.И. Баранчиков, А.В. Алпатова

РЕИНЖИНИРИНГ КЛЮЧЕЙ В ОТНОШЕНИЯХ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Перечислены причины целесообразности применения реинжиринга реляционных баз данных. Предложены алгоритмы нахождения одиночных и составных ключей отношений баз данных. Проведен расчет оценки временной сложности данных алгоритмов.

Ключевые слова: *реинжиниринг, база данных, реляционная, проектирование, информационная система, ключ, отношение*

Введение

При возникновении необходимости преобразования, модернизации и миграции баз данных применяется несколько подходов и методов.

Объектно-ориентированный метод основывается на представлении информационной системы в виде совокупности объектов, взаимодействующих между собой. В результате применения такого метода модель системы представляется совокупностью диаграмм, которые строятся по определенным правилам. Примером объектно-ориентированных методологий может служить методология UML (Unified Modeling Language).

Структурный метод начинается с общего обзора объекта исследования, а затем предполагает его последовательную детализацию. Модель, построенная с применением структурных методов, представляет собой иерархический набор диаграмм, графически изображающих выполняемые системой функции и взаимосвязи между ними. В составе методологий структурного анализа к наиболее распространенной можно отнести IDEF.

Недостаток этих методов в том, что они предполагают анализ с «нуля», что приводит к большой затрате времени и вероятности упущений, которые возникают вследствие человеческого фактора при взаимодействии внутри рабочей группы. Информация может теряться на пути от заказчика через аналитика к исполнителю.

Поэтому на предварительном этапе целесообразно использовать реинжиниринг ранее существовавших информационных систем, позволяющий извлечь заложенную в них семантику предметной области, которая обеспечит базис для будущего развития этих систем.

Таким образом, реинжиниринг – всего лишь один шаг в жизненном цикле информационных систем. Для трудоемкого восстановления семантики базы данных должны существовать обоснованные причины. Среди таковых можно выделить следующие:

- расширение системы – например добавление нового функционала, или повышение ее надежности;
- интеграция системы – интеграция двух или более систем, приводящая к образованию единственной результирующей системы, которая включает в себя функции и данные предыдущих;
- миграция или модернизация системы – заключается в замещении одной технологии реализации другой. Начало проектам миграции и модернизации баз данных положил переход с настольных и офисных систем на базе некогда популярных dBase, Clipper, FoxPro и Paradox на промышленные серверы баз данных. Сегодня в силу разных обстоятельств все чаще осуществляется переход с одного популярного сервера баз данных на другой. Кроме того, уже встречаются проекты перехода на другую СУБД, подразумевающие организацию поддержки более чем одной платформы [1];
- оценка качества системы, которую можно получить после подробного анализа ее структуры данных;
- приобретение знаний в области разработки систем. В процессе разработки новой системы, одна из ранних фаз состоит в сборе и оформлении требований пользователей из разных источников, таких как беседы с ними и анализ корпоративных документов. Во многих случаях частичная реализация будущей системы может уже существовать, например, в виде небольшой системы, анализ которой может принести первоначальную полезную информацию [2].

Сегодня можно говорить, что эра, когда разработчики информационных систем

приходили в организацию и начинали проекты информатизации «с нуля», прошла. Наступает время проектов по систематической трансформации существующих систем, или эра реинжиниринга. Следствием сложившейся ситуации становится объективная потребность в исследовании, пересмотре и переосмыслении существующих подходов, методологий и технологий разработки информационных систем, что, в свою очередь, может потребовать их модернизации, а возможно, и разработки новых решений [3].

Данная работа посвящена поиску такого решения, применение которого на практике позволило бы уменьшить сроки проекта, улучшить качество системы и расширить ее функциональность, снизить вероятность рисков, значимых для заказчика.

В работе рассматривается реинжиниринг ключей как один из первых этапов реинжиниринга баз данных в целом.

Нахождение и проверка простых ключей

Каждое отношение обязательно имеет одну или несколько комбинаций атрибутов, которая служит ключом. Ее существование гарантируется тем, что отношение - это множество, которое не содержит одинаковых элементов - кортежей.

Мощность отношения определяется как число его кортежей.

Операция проекции дает «вертикальный срез» отношения, в котором удалены все возникшие дубликаты кортежей.

Пусть дано отношение *r(R)* с множеством атрибутов $R = \{A_1, A_2, ..., A_n\}$. Если мощность проекции данного отношения по атрибуту A_i ($A_i \in R$, i = 1,2,...,n) равна мощности самого отношения, то можно говорить о том, что атрибут A_i является ключом:

$$\left|\pi_{A_i}\right| = |r|. \qquad (1)$$

Поскольку в этом случае среди значений атрибута A_i нет дубликатов, каждому кортежу отношения *r(R)* соответствует в точности одно уникальное значение атрибута A_i.

Алгоритм нахождения одиночных ключей отношения (FindKey) можно описать следующим образом.

Алгоритм 1. FindKey

Вход: отношение r с множеством атрибутов $R = \{A_1, A_2, ..., A_n\}$.

Выход: множество одиночных ключей $K = \{K_1, K_2, ..., K_m\}$, $K \in R$.

begin

$K := \varnothing$;

рассчитать мощность отношения $|r|$;

for каждый $A_i \in R$ (i = 1,2,...,n) do

begin

рассчитать $\left|\pi_{A_i}\right|$;

if $\left|\pi_{A_i}\right| = |r|$ then

$K := K \cup A_i$;

end;

return (K);

end.

Ключ отношения r со схемой R является подмножеством $K \subseteq R$ таким, что для любых различных кортежей t_1 и t_2 из r выполняется $t_1(K) \neq t_2(K)$ и ни одно собственное подмножество $K' \subset K$ не обладает этим свойством [4].

Отсюда следует, что в отношении не может быть двух различных кортежей с одинаковым значением атрибута K. Следовательно, алгоритм FindKey действительно позволяет однозначно определить одиночные ключи отношения по формальным признакам.

Например, у нас есть отношение r(НОМЕР, ПОСТАВЩИК, ГОРОД) с информацией о поставщиках (табл. 1):

Таблица 1

НОМЕР	ПОСТАВЩИК	ГОРОД
1	Иванов	Уфа
2	Петров	Москва
3	Сидоров	Москва
4	Сидоров	Москва

Мощность отношения $|r| = 4$.

Используя формулу (1), определяем мощности проекций отношения r по его атрибутам A_i (i = 1,2,3):

$$\left|\pi_{A_1}\right| = 4;\ \left|\pi_{A_2}\right| = 3;\ \left|\pi_{A_3}\right| = 3,$$

где A_1=НОМЕР, A_2=ПОСТАВЩИК, A_3=ГОРОД.

Таким образом, A_1 – ключ отношения r (K={НОМЕР}), поскольку только этот атрибут удовлетворяет определению ключа.

Составные ключи

После выявления всех простых ключей (состоящих из одного атрибута) переходим к поиску составных ключей, то есть состоящих из двух и более атрибутов.

Алгоритм 2. FindCompoundKey

Вход: отношение r с множеством атрибутов $R=\{A_1, A_2,..., A_n\}$, множество простых ключей $K=\{K_1, K_2,..., K_m\}$.

Выход: множество составных ключей $K_c=\{C_1, C_2,..., C_k\}$, элементами которого являются сочетания атрибутов $C_i=\{A_i,..., A_j\}$, входящих во множество $R \setminus K$.

begin

$K_c := \varnothing$;

Рассчитать мощность отношения $|r|$.

Определить множество неключевых атрибутов $\overline{K}=R \setminus K$, а также множество всех возможных сочетаний неключевых атрибутов $\overline{K}_c$;

for каждое сочетание $C_i \in \overline{K}_c$, (i = 1,2,...,m) do

begin

$\overline{K}_c = \overline{K}_c \setminus C_i$;

if $|\pi_{C_i}| = |r|$ then

$K_c = K_c \cup C_i$;

$\overline{K}_c = \overline{K}_c \setminus C$, где C – множество сочетаний C_j, входящих во множество $\overline{K}_c$, в которые входят все элементы сочетания C_i. Если $Ci=\{A_k,..., A_l\}$, то $A_k,..., A_l \subset C_j$;

end;

return (K_c);

end.

В алгоритме FindCompoundKey m - число рассматриваемых комбинаций атрибутов, которое меньше или равно максимальному числу комбинаций $m \le \max$. m зависит от числа найденных в процессе выполнения алгоритма FindCompoundKey составных ключей.

Согласно формуле комбинаторики [5] число сочетания без повторений рассчитывается следующим образом:

$$C_n^m = \frac{n!}{m!(n-m)!},$$

где C_n^m - число всех сочетаний, или неупорядоченных выборок различных элементов из множества $M=\{a_1, a_2,..., a_n\}$, содержащих k элементов.

Итого, максимальное число комбинаций атрибутов max можно рассчитать по формуле:

$$\max = \sum_{m=2}^{n} C_n^m = \sum_{m=2}^{n} \frac{n!}{m!(n-m)!}, \qquad (2)$$

где n - общее число атрибутов за вычетом уже найденных одиночных ключей.

Например, у нас есть отношение r(НОМЕР, ПОСТАВЩИК, ГОРОД, АДРЕС) с информацией о поставщиках (табл. 2):

Таблица 2

НОМЕР	ПОСТАВЩИК	ГОРОД	АДРЕС
1	Иванов	Уфа	Садовая, д.42
2	Петров	Москва	Радиальная, д.32
3	Сидоров	Москва	Костычева, д.3
4	Сидоров	Москва	Пушкина, д.11

Мощность отношения $|r|=4$.

По алгоритму FindKey множество простых ключей K={НОМЕР}.

Используя алгоритм FindCompoundKey, находим множество составных ключей.

Исключим из множества всех атрибутов отношения найденные простые ключи.

$\overline{K}=R \setminus K$ ={ПОСТАВЩИК, ГОРОД, АДРЕС}.

Рассчитаем максимальное число комбинаций атрибутов max, используя формулу (2).

В отношении r общее количество атрибутов равно 4, найденный простой ключ один, значит,

$$\max = \frac{3!}{2!\cdot 1!} + \frac{3!}{3!\cdot 0!} = 4.$$

Рассмотрим сочетания $C_i \in \overline{K}_c$ (i = 1,2,...,m), где $m \le 4$.

$\overline{K}_c = \{C_1, C_2, C_3, C_4\}$.

Итерация 1: C_1={ПОСТАВЩИК, ГОРОД};

$\overline{K}_c = \overline{K}_c \setminus C_1 = \{C_2, C_3, C_4\}$;

$|\pi_{c_1}| = 3 \ne |r|$.

Итерация 2: C_2={ГОРОД, АДРЕС};

$\overline{K}_c = \overline{K}_c \setminus C_2 = \{C_3, C_4\}$;

$|\pi_{c_2}| = 4 = |r|$;

$K_c = C_2$;

$\overline{K}_c = \overline{K}_c \setminus C = \overline{K}_c \setminus C_4 = \{C_3\}$.

Итерация 3: C_3={ПОСТАВЩИК, АДРЕС};

$\overline{K}_c = \overline{K}_c \setminus C_3 = \varnothing$;

$|\pi_{c_2}| = 4 = |r|$;

$K_c = K_c \cup C_3 = \{C_2, C_3\}$.

В результате выполнения алгоритма получаем множество возможных составных ключей $K_c=\{C_2, C_3\}$, где $C_2=\{A_3, A_4\}$, $C_3=\{A_2, A_4\}$.

Оценка временной сложности

Время, которое компьютер расходует на решение задачи, пропорционально числу двоичных операций [6]. Константа пропорциональности может зависеть от технических характеристик компьютера, например времени выполнения одной двоичной операции или времени доступа к памяти.

В оценках времени работы алгоритма, то есть оценке числа двоичных операций, можно пренебречь временем, расходуемым на логические шаги, отличные от двоичных операций.

Для оценки в качестве элементарной операции возьмем операцию побитового сравнения. При побитовом сравнении *k*-разрядного и *l*-разрядного чисел ($k < l$) простая оценка операции будет равна:

$$O \le l. \qquad (3)$$

Итак, пусть имеем отношение *r* с мощностью *p* и количеством атрибутов *n*.

Для расчета простой и удобной оценки предположим, что имеем дело с «наихудшим случаем». Поэтому не будем учитывать число простых ключей, которые были найдены по алгоритму FindKey (очевидно, что чем больше их было найдено, тем меньшее значение примет оценка временной сложности алгоритма FindCompoundKey). Также не будем учитывать длины атрибутов. Удобнее пользоваться простой равномерной оценкой, не зависящей от конкретных значений битов.

Из формул (2) и (3) получаем формульное выражение, обозначающее время выполнения алгоритма FindCompoundKey.

Обозначим его Q:

$$Q = p \cdot \sum_{m=2}^{n} \left(\frac{n!}{m! \cdot (n-m)!} \right). \qquad (4)$$

Число разрядов целого числа по основанию *b* можно записать с помощью логарифмов(«[]» обозначает целую часть числа):

$$k = [\log_b n] + 1. \qquad (5)$$

Обозначим число двоичных операций, необходимых для выполнения процедуры *A*, через *O(A)*.

Используя функции (4) и (5), рассчитаем верхнюю границу для числа двоичных операций, необходимых для вычисления *Q*, иными словами - временную оценку сложности алгоритма FindCompoundKey.

Так как всего число сложений *n*-1 и сложение двух *l*-разрядных двоичных чисел требует *l* двоичных операций, то сложение (*n*-1) *l*-разрядных двоичных чисел требует $(n-2) \cdot l$ двоичных операций.

$$O(FindCompoundKey) = (\log_2 p + 1) \cdot (n-2) \cdot \left(n^2 (\log_2 n)^2 \cdot m^2 (\log_2 m)^2 \cdot (n^2 (\log_2 n)^2 + \log_2 n + 1) \right).$$

Пренебрегая единицами, а также упрощая формулу, получаем менее строгую, но более компактную оценку:

$$O(FindCompoundKey) = \log_2 p \cdot n \cdot \left(n^2 (\log_2 n)^2 \cdot m^2 (\log_2 m)^2 \cdot (n^2 (\log_2 n)^2 + \log_2 n) \right) =$$
$$= \log_2 p \cdot n^3 \cdot m^2 \cdot (\log_2 n)^3 \cdot (\log_2 m)^2 \cdot \left(n^2 (\log_2 n) + 1 \right) = \log_2 p \cdot n^5 \cdot m^2 \cdot (\log_2 n)^3 \cdot$$
$$\cdot (\log_2 m)^3.$$

Т.к. *m* и *n* - величины одного порядка, причем $m \le n$, упростим оценку, заменив *m* на *n*, получив при этом оценку сверху:

$$O(FindCompoundKey) = \log_2 p \cdot n^5 \cdot m^2 \cdot \cdot (\log_2 n)^3 \cdot (\log_2 n)^3 = \log_2 p \cdot n^7 \cdot (\log_2 n)^5. \qquad (6)$$

Полученная оценка позволяет сделать вывод, что предложенный алгоритм решает поставленную задачу в приемлемое время.

По приблизительным оценкам применение предложенного решения позволяет улучшить количественные показатели, а именно сокращение продолжительности выполнения начальных циклов проекта примерно на 30% за счет уменьшения сроков исследования предметной области.

Заключение

В работе предложены алгоритмы выявления ключей отношений схемы реляционных баз данных для этапа изучения предметной области в том случае, когда существуют прототипы проектируемой информационной системы, что позволит сократить время разработки информационной системы и снизить вероятность возникновения ошибок на данном этапе, наиболее важном и критичном из всех.

Библиографический список

1. *Оганесян А.* Реинжиниринг баз данных// Открытые системы, 2004.
2. *Hainaut Jean-Luc.* Introduction to Database Reverse Engineering.- Belgium, LIBD, Institut d'Informatique - University of Namur, 2002.
3. *Ахтырченко К., Сороквашa Т.* Методы и технологии реинжиниринга ИС// Труды Института системного программирования РАН, 2003.
4. *Мейер Д.* Теория реляционных баз данных.- М.: Мир, 1987.
5. *Мальцев Ю.*, *Петров Е.* Введение в дискретную математику. Барнаул: Изд-во Алтайского государственного университета, 1997.
6. *Коблиц Н.* Курс теории чисел и криптографии.- М.: Научное изд-во ТВП, 2001.