МАТЕМАТИКА, ФИЗИКА И ИНФОРМАТИКА

УДК 519.854.2

В. С. Рублёв, А. Ш. Кайбышев

Организация хранения данных и выполнения запросов в динамической информационной модели DIM

Рассматривается задача организации данных [6] для новой объектной технологии СУБД DIM [2], с помощью которой обеспечивается удобное хранение и изменение данных, а также оптимизация выполнения запросов. Эти решения будут положены в основу реализации СУБД DIM.

Ключевые слова: объектная СУБД, хранение данных, динамика данных, оптимизация выполнения запросов.

V. S. Rublev, A. Sh. Kaibyshev

Organization of Data Storage and Realisation of Inquiries in a Dynamic Information Model DIM

The problem of data organization and organization of object queries execution [6] for new object DBMS technology DIM [7] is considered and that provides good storage and change of data and the optimal computational complexity of queries execution. These solutions will be taken as a basis for realization object DBMS DIM.

Keywords: object DBMS, data organization, dynamic of data, optimization of computational complexity.

1. Постановка задачи

Развитие вычислительной техники в плане роста быстродействия и памяти сделало возможным совершенствование технологий проектирования и использования баз данных. С этой целью были поставлены задачи определения моделей данных для новых типов, интегрирования их с традиционными технологиями, управления потоками данных, автоматизации проектирования и администрирования баз данных [8]−[11]. Новые достижения в научных исследованиях (например, в астрофизике) требуют хранения и обработки колоссальных объемов информации, которые подошли уже к пентабайтам [9]. Потребность в постоянных изменениях не только данных, но и алгоритмов является «неизменной характеристикой современного мира» [7]−[12].

В течение ряда лет мы разрабатываем новый объектный подход к созданию СУБД, который предполагает не только изменение данных объектов, но и возможность изменения типов объектов, то есть схемы базы данных. В этом подходе мы выделили 6 базовых отношений объектов: *наследования, включения, внутреннего наследования, внутреннего включения, истории и взаимодействия* – и 2 дополнительных отношения, и назвали эту СУБД *динамической информационной моделью* (в алгебре система объектов с введенными отношениями называется моделью (DIM)), а также разработали объектный язык запросов ODQL. Отметим, что новыми по сравнению с традиционными технологиями являются следующие отношения объектов: внутреннего наследования, внутреннего включения, взаимодействия, а также отношение включения через объект связи.

Описанный в [6] язык объектно-динамических запросов ODQL для динамической информационной модели DIM, концепции которой приведены в [2], обладает полнотой – любая группа объектов DIM (вместе с любыми их свойствами) может быть выделена ODQL-запросом [4]. Вместе с тем, язык запросов в своем варианте SODQL [4] позволяет особенно просто конструировать запросы за счет достаточности указания лишь списка требуемых свойств объектов. В [5] описана организация выполнения запросов, при которой трудоемкость оптимальна и не уступает традиционным технологиям. Но время выполнения запросов [10] зависит от организации хранения данных и манипулирования ими, а

также от выбора платформы реализации. При этом должны быть учтены вопросы динамического изменения данных. Рассмотрим общие вопросы такой организации, не зависящие от выбора платформы для реализации.

При этом, учитывая, что полнота описания любой дискретной детерминированной модели (см. в [2]) может быть достигнута даже при использовании только базовых отношений, мы в данной работе не будем рассматривать пока *отношения идентификации и выбора.* Сначала рассмотрим базовые отношения, иллюстрируя их основным примером, который будет использован в дальнейшем.

2. Основной пример и базовые отношения классов и объектов

Для иллюстрации выбран пример, отражающий данные производства шин. На рис.1 приведена схема базы данных. Класс **Шина** описывает объекты, являющиеся образцами продукции. Каждый объект-шина этого класса определяется двумя идентификационными параметрами: *Модель* и *Обозначение.* Класс **Шина** связан отношениями включения с классами **Протектор**, **Боковина**, **Ободная лента**, **Брекер**, объекты которых описывают структуру того или иного объекта класса **Шина**. Каждый объект-шина, как правило, включает объект-протектор, а также некоторые из объектов других трех классов (но не все). Так как каждый объект-шина может содержать не более одного объекта из этих включенных классов, то класс связи для описания данного включения не требуется. Связь включения на схеме определяется пунктирной стрелкой, идущей от включенных классов **Протектор**, **Боковина**, **Ободная лента**, **Брекер** к включающему **Шина**.

Требования к продукции описываются классом **Стандарт**, который является родительским классом для классов **Шина**, **Протектор**, **Боковина**, **Ободная лента, Брекер**, поскольку каждый объект этих классов наследует свойства родительского объекта-стандарта такие, как *Наименование* стандарта и все связи включения других классов в класс **Стандарт**. В данном случае класс **Показатели**, включенный в класс **Стандарт**, определяет для каждого объекта-стандарта те объекты-показатели, значения которых могут быть критичными для соответствия продукции стандарту. Поэтому включение класса **Показатели** в класс **Стандарт** идет через класс связи **Нормы показателя** (на схеме класс связи изображается прямоугольником со скругленными углами, который находится в разрыве пунктирной стрелки связи), объект которой для каждой пары объектов стандарт и показатель, определяемой отношением включения, может задавать диапазон ограничений значения показателя через *Нижнюю норму* и *Верхнюю норму.* Таким образом, каждый объект классов **Шина**, **Протектор**, **Боковина**, **Ободная лента**, **Брекер** наследует через свой объект-родитель в классе **Стандарт**, не только наименование этого стандарта, но также и показатели вместе с ограничениями на их значения, которым должна удовлетворять соответствующая продукция.

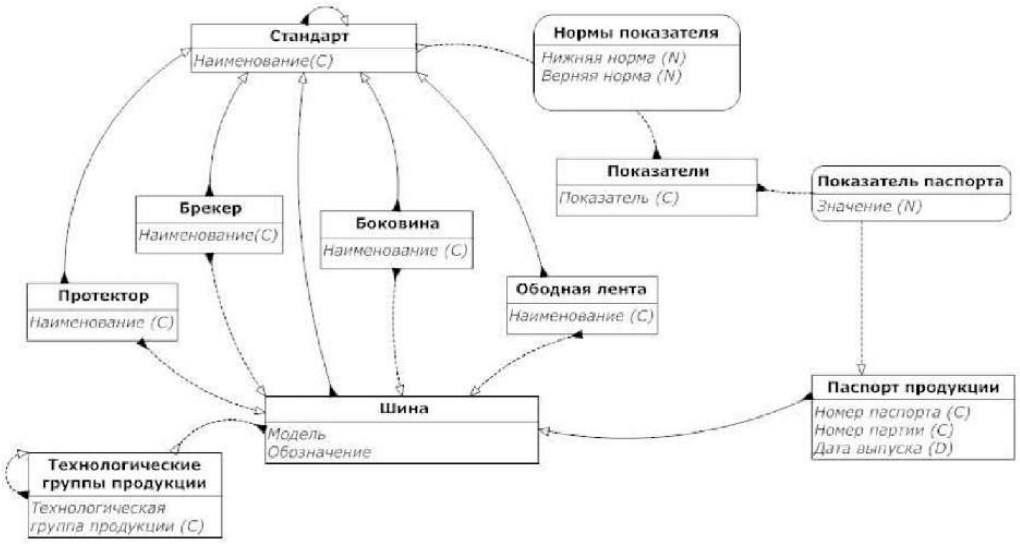

Рис. 1. Основной пример

Ситуация осложняется тем, что для некоторой продукции определяется не один, а несколько стандартов, например, ГОСТ (государственный стандарт), ОСТ (отраслевой стандарт), ТУ (технические условия), Регламент. Поэтому продукция должна удовлетворять сразу нескольким стандартам. Если для нее определяется старший стандарт ГОСТ и более низкий отраслевой ОСТ, то диапазон ограничений на значения показателя для ОСТ не может быть шире, чем для ГОСТ. То есть стандарты находятся в иерархии от старшего ГОСТ до самого младшего Регламент, но при этом не все они могут быть определены. Каждый объект-стандарт может иметь своего родителя в этом же классе **Стандарт**. Такое отношение объектов этого класса называется ***внутренним наследованием***. Оно отмечается стрелкой наследования, идущей из этого класса в него же. При этом правила наследования таковы:

Каждый дочерний объект наследует все свойства его родительского объекта, за исключением тех свойств, которые в нем непосредственно определены. Таким образом, в младшем стандарте могут быть определены некоторые показатели, отсутствующие в старшем стандарте. Но если показатель определен для обоих стандартов, то значение диапазона норм в младшем стандарте не должно быть шире, чем в старшем стандарте.

Заметим, что поскольку мы заинтересованы в возможно меньшем действии при выполнении запросов, то исследование наследования внутреннего включения в данном примере не будет вызывать дополнительных действий, если перенести при внутреннем включении наследуемые свойства включения *в младший* **Стандарт** *из наследуемого старшего* **Стандарта**.

Классы продукции, описанные выше, задают образцы продукции. А реальная продукция должна проходить регулярно испытания на соответствие стандартам. При этом из партии продукции в определенную для контроля дату производятся замеры значений показателей, перечень которых определен стандартом, и составляется паспорт продукции, в который заносятся эти значения и нормы на них, а также вывод ОТК о годности продукции (соответствии стандарту). Поэтому в схеме указан класс **Паспорт продукции** со свойствами *Номер паспорта, Номер партии, Дата выпуска* и связями наследования от класса **Шина** и включения класса **Показатели** через класс связи Показатель паспорта со свойством *Значение.* При этом для каждого объекта-паспорта продукции наследуются показатели через родительские объекты классов **Шина**, **Стандарт,** и только для них может быть определен объект-показатель паспорта.

Наконец, класс **Технологические группы продукции** определяет группирование продукции по разным ее свойствам. Например, такими группами могут быть *Диагональные шины, Радиальные шины, Автомобильные шины, Шины для грузовых машин* и т. д. Некоторые группы могут пересекаться: например, в группу автомобильных шин могут входить и радиальные, и диагональные шины, а в группу диагональных шин могут входить и автомобильные шины, и шины для грузовых машин. Некоторые группы могут целиком входить в другие: например, группа *Радиальные автомобильные шины* входит целиком в группы *Радиальные шины* и *Автомобильные шины.* Для этого используется отношение внутреннего включения объектов, что указывается пунктирной стрелкой, идущей из этого класса в него же.

Еще 2 базовых отношения – это *отношение истории* и *отношение взаимодействия* классов и, соответственно, отношения объектов. Отношение истории определяет динамику классов при изменении параметров классов и их связей с другими классами, а также динамику объектов в тех случаях, когда одни объекты перестают существовать и заменяются другими объектами. Каждый такой класс или объект снабжается двумя параметрами: *Дата рождения* и *Дата смерти.* Рис. 2 иллюстрирует такое изменение класса **Шина**, связанное с введением дополнительного параметра *Нить,* не обязательного для всех объектов класса. При этом определяются класс-предшественник, для которого задана *Дата смерти* в момент изменения класса, и класс-последователь, для которого, помимо главных идентификационных параметров (MIPO) *Обозначение, Модель*, появляется дополнительный параметр (ADO) *Нить,* не обязательный для объектов этого класса, а *Дата рождения* измененного класса получает тоже значение, что и *Дата смерти* для класса-предшественника. Заметим, что в данном случае все объекты класса-предшественника не изменяются и становятся объектами класса-последователя. В общем случае может произойти между несколькими классами перераспределение параметров с добавлением новых или удалением некоторых старых. В подобном случае такое изменение классов влечет за собой изменение всех объектов этих классов.

Рис. 2. История классов

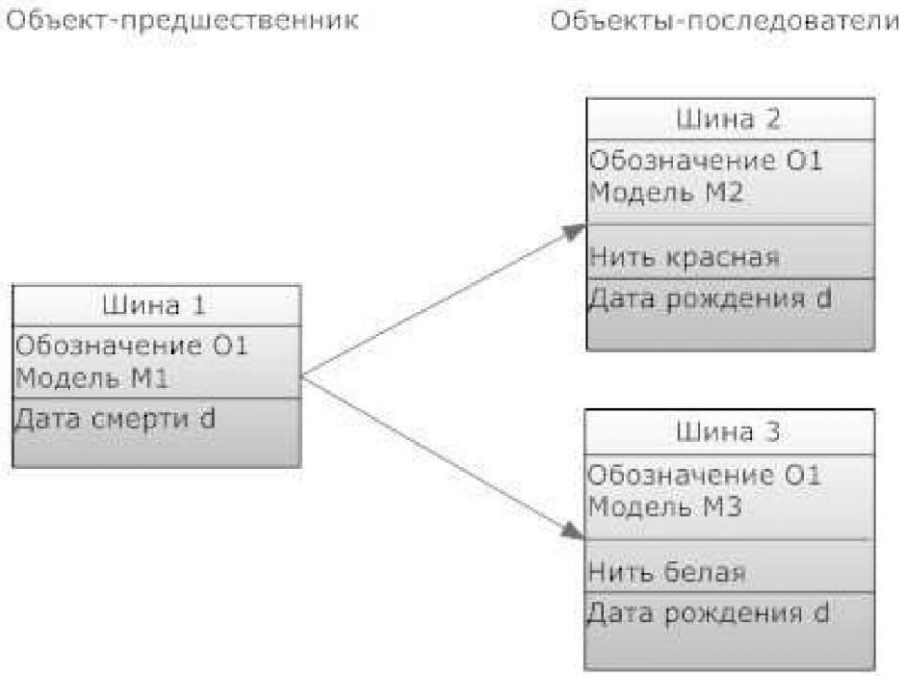

Рис. 3. История объектов

Рис. 3 иллюстрирует изменение объекта класса **Шина**, ставшее возможным после введение дополнительного параметра *Нить.* В таком случае объект **Шина 1**, определенный значениями идентификационных параметров *O1, M1*, получает значение параметра *Дата смерти* в момент изменения, а на смену этому объекту-предшественнику возникают 2 объекта-последователя **Шина 2, Шина 3** с тем же значением параметра *Обозначение,* но с другими значениями параметра *Модель,* а также со значениями дополнительного параметра *Нить.* Момент рождения этих объектов-последователей определяется моментом смерти объекта-предшественника. Отметим, что в общем случае несколько объектов-предшественников могут заменяться некоторым количеством объектов-последователей. Заметим также, что если значение дополнительного параметра определить для всех объектов класса, то такой параметр можно перевести в группу обязательных главных неидентификационных параметров (MNPO) без изменения истории объектов. В этом случае при добавлении нового объекта в класс определение значения этого параметра станет обязательным (для дополнительных параметров это не так). Если же параметр перевести в группу главных идентификационных параметров, то потребуется изменение класса и всех объектов, что также потребует отношения истории.

При помощи *отношения взаимодействия* (см. [1, 3]) определяются все изменения в базе данных. В частности, это изменения, определяющие историю объектов и классов. Рис. 4 иллюстрирует схему *взаимодействия,* в которой участвуют объекты четырех классов:

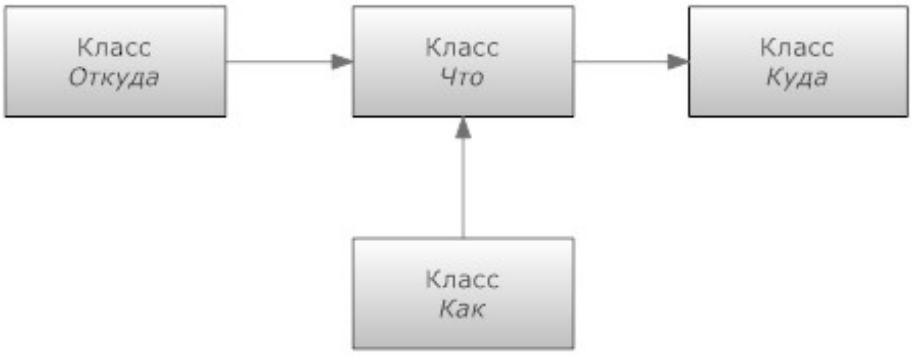

Рис. 4. Схема взаимодействия

- *объект класса* Откуда *имеет роль* источника взаимодействия*;*
- *объект класса* Куда *имеет роль* направления взаимодействия*;*
- объект класса *Что имеет* роль *объекта взаимодействия*;
- *объект класса* Как *имеет роль* процесса взаимодействия.

Рис. 5 иллюстрирует *взаимодействие,* при котором происходит добавление дополнительного параметра *Нить* в класс **Шина**. В этом случае источником взаимодействия является *пользователь,* который вызывает это *взаимодействие,* в качестве направления взаимодействия выступает *Система,* в качестве объекта взаимодействия – класс **Шина**, представляющий собой объект схемы метауровня (см. далее), а в качестве процесса взаимодействия – класс **Изменение параметров класса**.

Рис. 5. Взаимодействие объектов

Отметим, что указанная схема взаимодействия является достаточно общей для описания любых процессов изменения информации в базе данных. Так, оно используется не только для манипулирования всеми данными пользователем, но и для таких процессов, как продажа продукции (источник – продавец, направление – покупатель, объект взаимодействия – продукция, процесс взаимодействия – класс **Продажа**, определяющий программные действия изменения данных) или изготовления детали (источник – сырье, направление – деталь, объект взаимодействия – станок, процесс взаимодействия – класс **Изготовление**, определяющий объект *Рабочий,* изготавливающий деталь и программные действия изменения данных).

3. Описание метауровня хранения данных

Поскольку необходима организация динамической СУБД, в которой могут изменяться не только объекты, но и их типы, определяемые изменением классов и параметров, то для описания классов объектов, параметров объектов и связей (отношений) классов и объектов используется метауровень, который представляет собой реляционную базу данных с набором таблиц, содержащих в себе все нужные значения. При этом выбор в качестве платформы реляционной СУБД может быть сделан

позже на основе анализа удачности выбора той или иной платформы для метауровня. В таблицах метауровня мы как объекты будем представлять информацию о классах как объектах. Поскольку параметры каждого класса могут меняться, то информацию о параметрах классов мы также будем представлять в других таблицах, связывая их с таблицей классов реляционными связями. Объекты каждого класса будем держать в отдельных таблицах, связывая их с классами таблицы классов. Подобным образом поступим и со значениями параметров объектов, помещая их в отдельные таблицы, связанные с соответствующими таблицами параметров и объектов реляционными связями. Отметим, что каждый класс, параметр, объект имеет свой уникальный идентификатор (IdClass, IdParamter, IdObj) для установления упомянутых связей между таблицами, а также 2 атрибута *Дата рождение* и *Дата смерти,* определяющими период жизни соответствующего данного (при изменении класса, параметра, объекта их объектные идентификаторы также изменяются).

На рис. 6 приведена схема организации метауровня DIM. Поясним ее.

Главными в модели являются таблица Class классов объектов, таблица объектов соответствующего класса Obj_IdClass, где IdClass − идентификатор класса, Parameters, определяющая параметры объектов, и значение каждого параметра объекта (через таблицу типа Val_IdParameter). При этом каждый класс, объект, параметр определяются уникальным идентификатором соответственно IdClass, IdObject и IdParameter.

Таблица Class классов объектов определяется для каждого класса:

- объектный идентификатор класса;
- имя класса, через поле NameClass;
- его родительские и дочерние классы (через таблицы ClassInheritance наследования – граф наследования с дугами типа TypeGroup);
- классы, которые включены, и классы, в которые включен данный класс (через таблицу ClassInclusion − граф включения классов);
- классы взаимодействия (через таблицу ClassInteraction);
- уровень класса в графе наследования LevelInheritance;
- тип класса TypeClass;
- метка списка помеченных объектов класса; хранится в поле MarkList;
- связи данного класса, хранятся в поле RelRef, представляющий собой объект типа Map, в котором содержатся ключ − идентификатор связанного класса и значение − виды связей;
- параметры класса (через таблицу Parameter, так как все имена параметров уникальны и прикреплены к какому-то классу и через таблицу GroupParam);
- объекты класса (через таблицу Obj_IdClass);
- историю класса (через таблицу ClassHistory).

Отметим, что связи класса с другими классами можно получать как через таблицы классов ClassInheritance, ClassInclusion, ClassInteraction, ClassHistory (основной способ хранения информации о связях класса), так и через поле RelRef (дополнительный способ для ускорения получения этой информации).

Для того чтобы дублирование информации в поле RelRef не привело к коллизиям, перед каждым изменением какой-либо связи в одной из перечисленных четырех таблиц информация в поле RelRef помечается как недействительная, а после изменения обновляется. Такой режим ведет к сохранению целостности базы данных.

Таблица Obj_IdClass объектов класса определяет для каждого объекта:

- идентификатор объекта IdObject;
- поле Mark для пометки объекта, участвующего в выборке, если стоит значение 0 или null – объект не помечен, он не участвует в выборке, если 1, то объект помечен;
- NextIdObject − объектный идентификатор следующего объекта в списке помеченных объектов; через это поле осуществляется альтернативная организация списка, которая обеспечивается внешним индексом по этому полю на ту же таблицу;
- префикс IdClass из названия таблицы является идентификатором класса, объекты которого добавлены в данную таблицу; связь включения между объектами определяется либо в таблице ObjInclusion_IdIncluding_IdIncluded_IdInclusion, либо в таблице ObjInclusion_IdIncluding_IdIncluded, где вместо IdIncluding, IdIncluded, IdInclusion находятся идентификаторы классов: включающего, включения, связи включения;

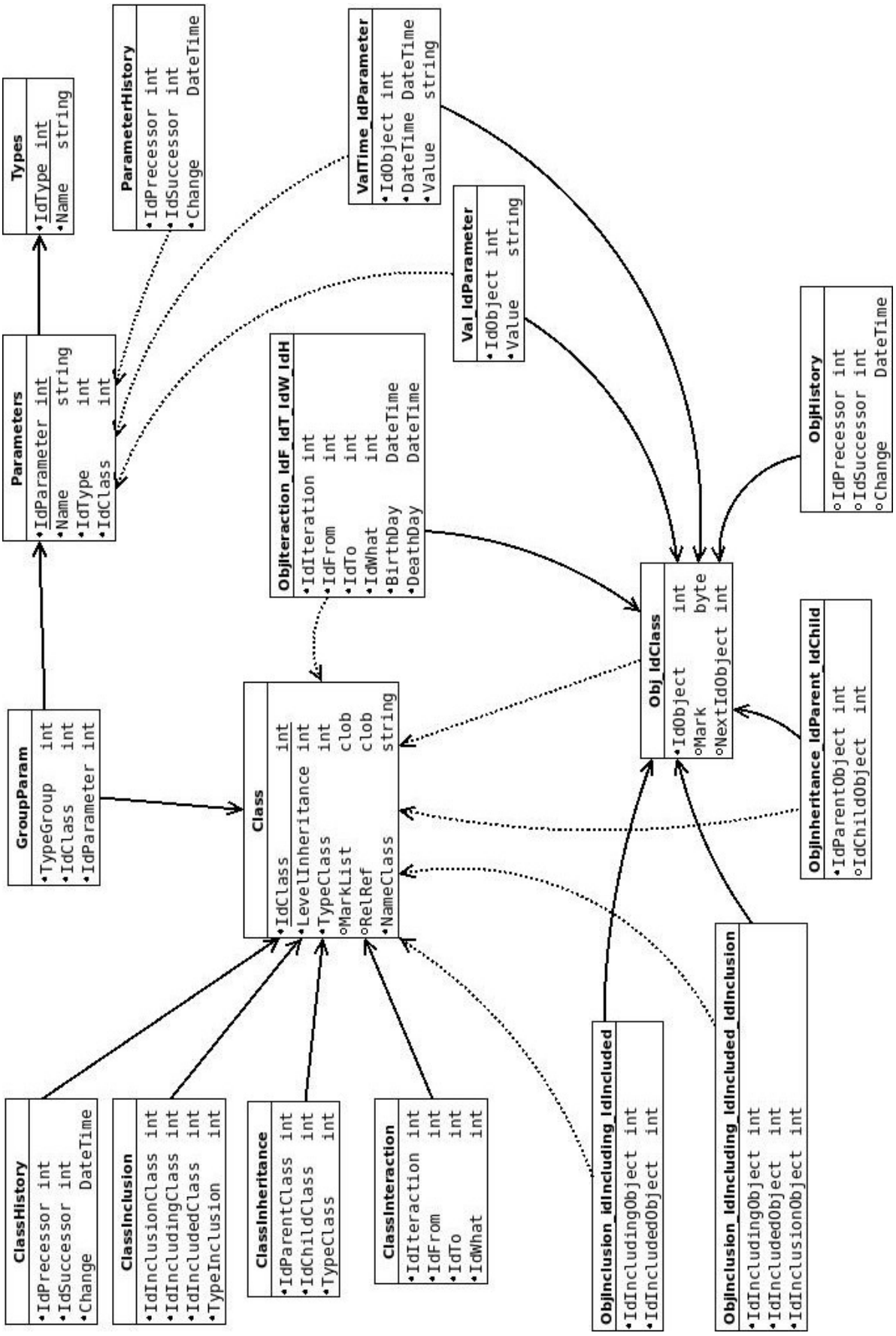

Рис. 6. Схема метауровня

• связь наследования между объектами определяется в следующих таблицах ObjInheritance_IdParent_IdChild, где вместо IdParent, IdChild находятся идентификаторы классов;

• параметры объекта (через таблицы GroupParam и Parameters);

• значение параметров через таблицу с именами Val_IdParameter, где вместо IdParameter находится идентификатор параметра;

• список значений параметра в соответствующий момент времени через таблицу с именем ValTime_IdParameter, где вместо IdParameter находится идентификатор параметра;

• связанные через таблицу ObjInteraction_IdF_IdT_IdW_IdH взаимодействия объекты, где IdF, IdT, IdW, IdH − идентификаторы объектов, имеющих соответственно роли *Откуда, Что, Куда, Как*.

Таблица Parameters определяет параметры объектов и для каждого параметра:

• объектный идентификатор IdParameter;

• его имя;

• тип (через таблицу Types);

• значение каждого параметра IdParameter каждого объекта IdObject через таблицы с именами типа Val_IdParameter;

• список значений динамического параметра IdParameter в определенный момент времени DateTime для каждого объекта IdObject через таблицы с именами типа ValTime_IdParameter.

Поле LevelInheritance класса определяет уровень класса в графе наследования, представляемого таблицей ClassInheritance:

0 − класс является изолированной вершиной графа наследования;

1 − класс является корневым в наследовании;

2 − класс не является ни корневым, ни висячим в графе наследования;

3 − класс является висячим в наследовании.

Поле TypeClass класса определяет тип класса:

0 − класс объектов, не являющихся связями других объектов;

1 − класс объектов, являющихся связями в отношении включения других объектов;

2 − класс объектов, являющихся связями во взаимодействии других объектов.

Каждый класс может содержать 3 группы параметров с именами:

MIPO − Main Identification Parameters of Objects − TypeGroup = 0;

MNPO − Main Nonidentification Parameters of Objects − TypeGroup = 1;

APO − Addinional Parameters of Objects − TypeGroup = 2.

Эти группы параметров определяют спецификацию объектов класса. При этом параметры из списка MIPO являются идентификационными (любые 2 объекта класса не должны иметь одинаковый набор этих значений); параметры из списка MNPO являются другими обязательными параметрами объектов класса, а параметры из списка APO необязательные для того или иного объекта класса. Обязательным является наличие хотя бы одной из групп параметров MIPO или MNPO.

Таблица ClassInheritance описывает связи классов по наследованию (родительский − дочерний) и представляет собой ориентированный граф с определенными компонентами для каждого типа классов. При этом родительский и дочерние классы определяются через IdParentClass и IdChildClass, а тип группы TypeGroup определяет тип параметра дочернего класса.

Таблица ClassInclusion описывает связи включения классов (включающий − включаемый). Параметр IdInclusionClass определяет идентификатор класса связи между включающим классом IdIncludingClass и включенным классом IdIncludedClass (если IdInclusionClass=0, то определяется простое включение объектов включаемого класса в объекты включающего класса). Параметр TypeInclusion определяет тип отношения между включающим и включаемым классом:

0 − многие ко многим;

1 − многие к одному;

2 − один ко многим;

3 − один к одному.

Таблица ClassInteraction описывает классы взаимодействия, определяющие взаимодействие потока объектов класса IdWhat от объекта класса IdFrom к объекту класса IdTo.

Для описания отношения включения объектов используется одна из таблиц ObjInclusion_IdIncluding_IdIncluded_IdInclusion или ObjInclusion_IdIncluding_IdIncluded. Они представляют собой ациклический ориентированный граф с навешанными на дуги параметрами вхождения включаемого объекта во включающий. Параметр IdIncludingObject задает включающий объект класса IdIncluding из названия таблицы. Параметр IdIncludedObject задает включаемый объект класса IdIncluded из названия таблицы. Параметр IdInclusionObject задает объект связи между включающим и включенным объектами класса IdInclusion из названия таблицы.

Таблица ObjInheritance_IdParent_IdChild, где IdParent и IdChild – идентификаторы классов, для каждого объекта IdChildObject класса IdChild из названия таблицы задает объект IdParentObject класса IdParent из названия таблицы.

Таблицы ClassHistory, ParameterHistory и ObjHistory_IdClass описывают динамические связи классов, объектов, параметров соответственно (предшественники − наследники), определяют время преобразования (классов, объектов, параметров) и представляют собой ациклические ориентированные графы.

Таблица ObjInteraction_IdF_IdT_IdW_IdH описывает взаимодействия объектов и дает представление ориентированного графа с потоками (объектов IdWhat, IdInteraction), навешанными на дуги, идущими из объектов IdFrom в объект IdTo.

4. Алгоритм выполнения запроса

Ведущую роль в алгоритме играет схема слоев классов запроса, описанная в [4] (см. рис. 7). Напомним [6], что ODQL-запрос (запрос нижнего уровня) состоит из фраз:

- **select**, определяющей выборку данных запроса;
- **from**, определяющей классы объектов, где находятся эти данные; первый класс этой фразы определяет базовый класс, для объектов которого определяется запрос;
- **for**, ограничивающей условиями на данные каждого из некоторых отдельных классов;
- **links**, определяющей связи классов;
- **where**, ограничивающей условиями на данные разных классов.

Упрощенный язык SODQL-запросов (верхнего уровня) опускает фразу **links** и может не содержать фразы **from**. В этом случае базовый класс определяется по первому данному фразы **select**, а все остальные данные ODQL-запроса получаются при построении схемы слоев. Схема слоев определяется алгоритмом, приведенным в [4] по классам данных фразы **select** и связям этих классов друг с другом последовательно, начиная от базового класса. При этом строятся фразы **from** и **links**, фиксирующие порядок классов и связей в схеме слоев. Поэтому можно считать, что схема слоев запроса записана в получаемом или преобразованном ODQL-запросе.

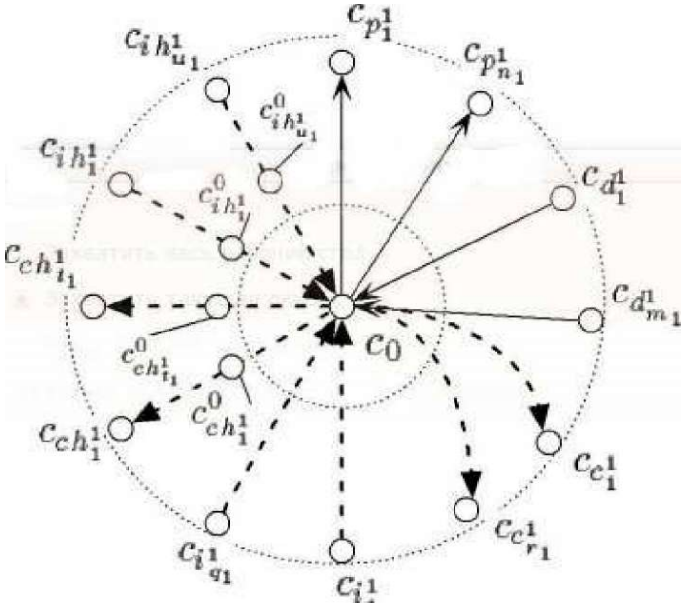

Рис. 7. Схема слоев (представлены 2 слоя: верхний с базовым классом и следующий)

Идея алгоритма выполнения запроса [5] состоит в том, что для каждого класса, участвующего в запросе, создается *индекс-выборка* [5]. Затем, двигаясь по *схеме слоев* [4] от слоя с максимальным уровнем к нулевому (к базовому классу), мы устанавливаем *индекс-выборки отношений классов* [5].

Реализация индекс-выборок с помощью динамической памяти невозможна, так как ее указатели должны быть связаны с узлами индексных деревьев, находящихся во внешней памяти. Поэтому для реализации индекс-выборки класса используются поля таблицы объектов класса: Mark (для пометки объекта), NextldObject с внешним индексом к этой же таблице (для организации списка индекс-выборки), а также поле MarkList записи класса в таблице Class (для пометки списка индекс выборки). Для реализации индекс-выборки отношения объектов классов в таблице связи объектов этих классов используются связанные с ней внешние индексы на соответствующие объекты таблиц объектов обоих классов, что дает возможность организовать алгоритм коррекции индекс-выборок классов.

Алгоритм использования индекс-выборок связан с начальным установлением индекс-выборок классов по фразе **for** запроса [6], [5]. Затем при движении по схеме слоев классов от самого нижнего слоя к самому верхнему (базовому классу запроса) по индекс-выборкам отношений более нижнего в схеме класса со связанным с ним более верхним классом схемы корректируется индекс-выборка последнего класса. Трудоемкость этого алгоритма, как показано в [5], минимально возможная (эвристически).

Третий этап алгоритма выполнения запроса состоит в получении «деревьев» (это может быть ациклический граф) объектов запроса, в каждом из которых для помеченного объекта базового класса запроса строится дерево его данных, определяемых фразой **select** запроса и удовлетворяющих условиям фразы **where** запроса. Реализация этого происходит при движении вниз от каждого из объектов базового класса схемы слоев с проверкой условий фразы **where** запроса.

Сделаем еще 2 замечания.

1. Этап коррекции индекс-выборок классов мы можем начинать только от тех классов, находящихся в нижних слоях схемы, которые содержат не все помеченные объекты. Для этого используется пометка списка ***MarkList*** индекс-выборки класса (ее значение, согласно [5], равно 1).

2. Схема слоев является ациклическим графом, который может не быть деревом. То есть может существовать в схеме слоев класс в одном из нижних слоев, который имеет несколько путей связей с базовым классом (например, это может возникнуть в результате множественного наследования через классы в более верхних слоях или в результате нескольких включений в другой класс в более верхнем слое схемы). При выполнении второго этапа (коррекции индекс-выборок) проблем не возникнет. Но при выполнении третьего этапа (построение «деревьев» данных объектов) порядок обхода «веток» получающегося цикла (циклов) может быть лучшим (меньший перебор), если сначала выбираются «ветки», имеющие меньшее число связей включения и наследования (ниже в одном из примеров этот случай будет рассмотрен).

5. Примеры

Рассмотрим несколько примеров, описывающих выполнение ODQL-запроса и схемы к этим запросам, базирующиеся на основном примере в разделе 2.

Запрос 1. Для каждой Шины, отвечающей группе «Диагональные шины», определить стандарт, название, обозначение, модель и показатели с их нормами (см. рис. 8).

Рис. 8. Схема слоев запроса 1

На языке SODQL запрос выглядит следующим образом:

for ТехнологическаяГруппа = 'Диагональныешины'
select НаименованиеШины, ОбозначениеШины, МодельШины,
НаименованиеСтандарта, НаименованиеПоказателя, НижняяНорма, ВерхняяНорма

После преобразования запроса в язык нижнего уровня ODQL получим:

for НаименованиеТехнологическойГруппы = 'Диагональные шины'
select НаименованиеШины, ОбозначениеШины, МодельШины,
НаименованиеСтандарта, НаименованиеПоказателя, НижняяНорма, ВерхняяНорма
from Шина, Стандарт Ст, Показатель Пок, НормыПродукции Норм,
ТехнологическиеГруппы ТГ links ТГ contains Шина, Ст parent Шина, Ст **contains** (Норм) Пок

Выполнение этого запроса определяется следующими действиями:

1. Проводим начальную установку индекс-выборки для класса **Технологические группы** по ограничениям фразы **for**. При этом используем внутреннее включение для определения всех групп, вложенных в группу, заданную условием фразы **for**. Для остальных классов **Шина**, **Стандарт**, **Показатели**, **Нормы продукции** устанавливаем пометку **MarkList** всех выбранных объектов.

2. Приступаем к коррекции пометок объектов классов. Поскольку фраза for пометила объекты единственного класса и он не является базовым, то по фразе **links** определяем связанный с этим классом отношением включения включающий класс **Шина**. Помечаем объекты класса **Шина** для всех помеченных объектов класса **Технологические группы**. На этом этап коррекции заканчивается.

3. Начинаем обход схемы слоев с базового класса **Шина**. Выбираем по очереди его помеченные объекты.

4. Для текущего выбранного объекта класса **Шина** выбираем значения полей *НаименованиеШины, ОбозначениеШины, МодельШины* и заносим в текущий буфер.

5. Выбираем первый класс, связанный с базовым классом, по фразе **links**. Это класс **Технологические группы**. Поскольку данные объектов этого класса не входят в выбираемые запросом данные, то мы переходим к следующей связи фразы **links**. Это связь наследования с родительским классом **Стандарт**.

6. Для текущего объекта класса **Шина** выбираем его родительский объект в классе **Стандарт**. Так как класс **Стандарт** имеет внутреннее наследование, то по этому отношению помечается также вся иерархия родительских объектов этого класса.

7. Выбираем текущий объект класса **Стандарт**.

8. Выбираем следующие классы, связанные отношением фразы **links**. Это классы **Показатели**, **НормыПродукции**.

9. Для текущего объекта класса **Стандарт** помечаем все объекты связанных с ним объектов этих классов и выводим в текущий буфер *Наименование Стандарта.*

10. Для каждого помеченного объекта класса **Показатели** и связанного с ним объекта класса **Нормы продукции** выводим в буфер значения полей *НаименованиеПоказателя, НижняяНорма, ВерхняяНорма,* если в текущем буфере нет этого показателя.

11. Выбираем в качестве текущего следующий помеченный объект класса **Стандарт**, если он есть, и переходим к п. 9.

12. Так как результат отвечает всем условиям запроса, то он переносится в буфер вывода, а текущий буфер очищается.

13. Выбираем следующий помеченный объект класса **Шина**. Если он есть, то делаем его текущим и переходим к п. 6.

14. Если его нет, то выполнение запроса завершено. Результаты запроса находятся в буфере вывода.

Запрос 2. Для продукции, входящей в группу «Диагональные шины», определить наименование, обозначение, модель, стандарт и значение одного из показателей, по которому определен брак (см. рис. 9).

Рис. 9. Схема слоев запроса 2

Заметим, что для записи во фразе **where** условия запроса «*значение одного из показателей, по которому определен брак*» надо не только определить условие на данные объектов классов **Нормы**, **ПоказателиПаспорта**, соответствующие объекту класса **Показатели**, но и указать, что условие выполняется, если найдется такой показатель. Это требует проверки условия до тех пор, пока не найдется такой показатель. Поэтому в запросе вводится конструкция введения *квантора* exist:

where exist (Показатель) **with**

ВерхняяНорма < ЗначениеВПаспорте | ЗначениеВПаспорте < НижняяНорма

В общем случае конструкция фразы **where**, добавляемая в языки запросов ODQL и SODQL выгладит следующим образом:

where... **exist**(<список объектов, для которых определяется выполнение условия>)

with <условие для данных, связанных с данными списка>

Условие **where** считается выполненным, если найдется такой вычисляемый в запросе список объектов запроса, для которых условие **with** выполняется. В этом случае проверка условия для других объектов, определяющих этот список данных, отменяется. Условие считается невыполненным, если ни для какого вычисляемого в запросе списка объектов оно не выполняется.

Аналогично квантору **exist** вводится квантор **all**:

where... **all**(<список объектов, для которых определяется выполнение условия>)

with <условие для данных, связанных с данными списка>

В этом случае условие **where** считается выполненным, если для любого вычисляемого в запросе списка объектов запроса условие **with** выполняется, и считается невыполненным, если найдется такой вычисляемый в запросе список объектов, для которого оно не выполняется. Нетрудно видеть, что эти действия соответствуют алгебре предикатов.

В списке фразы **select** выбора данных по умолчанию определяется квантор **all**, согласно которому выбираются данные всех объектов, для которых выполнено условие запроса. Если же требуется вывод только тех данных, для которых выполняется условие запроса, то для выбора данных употребляется квантор **exist**. Эта конструкция добавляется в языки запросов.

В рассматриваемом примере во фразе **select** для выбора только того **Показателя**, для которого выполняется условие, записывается квантор **exist**:

select... **exist** (НаименованиеПоказателя)

На языке SODQL запрос выглядит следующим образом:

for НаименованиеТехнологическойГруппы = 'Диагональные шины',

ДатаВыпуска = 01.01.2010 : 31.12.2010

select НаименованиеШины, ОбозначениеШины, МодельШины, НаименованиеСтандарта, **exist** (НаименованиеПоказателя), ЗначениеВПаспорте, НижняяНорма, ВерхняяНорма

where exist (Показатели) **with** ВерхняяНорма < ЗначениеВПаспорте | ЗначениеВПаспорте < НижняяНорма

Учитывая, что в схеме класс *Показатели* входит в две цепочки, идущие от базового класса *Шина,* необходимо составить два коррелированных подзапроса для этого класса и соединить их пересечением. После преобразования запроса в язык нижнего уровня ODQL получим:

for НаименованиеТехнологическойГруппы = 'Диагональные шины',
ДатаВыпуска = 01.01.2010 : 31.12.2010
select НаименованиеШины, ОбозначениеШины, МодельШины,
НомерПартии, НаименованиеСтандарта, exist (НаименованиеПоказателя), ЗначениеВПаспорте, НижняяНорма, ВерхняяНорма
from Шина, ТехнологическаяГруппа ТГ, Паспорт Пас, Стандарт Ст,
(**select object from** Показатели Пк, ПоказательПаспорта ПокПас
links Пас **contains** (ПокПас) Пк)
intersection
(**select object from** Показатели Пк, Нормы Норма
links Ст **contains** (Норм) Пк)
Пок
links ТГ **contains** Шина, Шина **parent** Пас, Ст **parent** Шина,
Пас **contains** (ПокПас) Пок, Ст **contains** (Норм) Пок
where exist (Пок) **with** ВерхняяНорма < ЗначениеВПаспорте |
ЗначениеВПаспорте < НижняяНорма

Выполнение этого запроса определяется следующими действиями:

1. Выбираем все объекты, отвечающие ограничению из **for** для классов **Технологические группы** и **Паспорт**. При этом для класса **Технологические группы** используем внутреннее включение для определения всех групп, вложенных в группу, которая задана условием фразы **for**. Для остальных классов **Шина**, **Стандарт**, **Показатели**, **Показатель Паспорта**, **Нормы продукции** устанавливаем пометку **MarkList** всех выбранных объектов.

2. Приступаем к коррекции пометок объектов классов. По фразе **links** определяем связанный с этим классом отношением включения включающий класс **Шина**. Идем снизу вверх от объектов классов **Технологические группы** и **Паспорт** к объектам класса **Шина** и помечаем те, которые связаны с уже помеченными объектами, то есть отвечающими обоим условиям из фразы **for**. На этом этап коррекции заканчивается.

3. Начинаем обход схемы слоев с базового класса **Шина**. Выбираем по очереди его помеченные объекты.

4. Для текущего выбранного объекта класса **Шина** выбираем значения полей *НаименованиеШины, ОбозначениеШины, МодельШины* и заносим в буфер 1.

5. Для текущего помеченного объекта класса **Шина** по следующей связи фразы **links** выбираем связанные с ним объекты класса **Паспорт**.

6. Для текущего помеченного объекта класса **Шина** по следующей связи фразы **links** выбираем связанный с ним объект класса **Стандарт**.

7. Выбираем текущий объект класса **Стандарт**. Для текущего выбранного объекта класса **Стандарт** выбираем значения поля *НаименованиеСтандарта* и заносим в буфер 1.

8. Выбираем текущий объект класса **Паспорт**. Если его нет, то очищаем буфер 1, выбираем следующий текущий объект класса **Шина** и, если он есть, переходим к п. 4, а если его нет, то завершаем выполнение алгоритма.

9. Следующие части фразы **links** связывают с текущим объектом класса **Паспорт** объекты подмножества *Пок* класса **Показатели** и с текущим объектом **Стандарт** также объекты подмножества *Пок* класса **Показатели**. Поэтому образуем по первому коррелированному подзапросу подмножество, используя алгоритм коррекции, а затем, используя алгоритм коррекции для второго подзапроса, и получаем пересечение подмножеств двух коррелированных подзапросов *Пок.*

10. По следующей связи фразы **links** выбираем связанные с текущим объектом класса **Паспорт** объекты множества *Пок* класса **Показатели** и класса **ПоказательПаспорта** и делаем их текущими. Если их нет, то очищаем буфер 1 и переходим к п. 8.

11. Заносим для этих объектов в буфер 2 значения полей *НаименованиеПоказателя, ЗначениеВ-Паспорте.*

12. Используя последнюю связь фразы **links** между классами **Стандарт** и подмножеством *Пок* класса **Показатели,** по текущим объектам классов **Стандарт**, **Показатели** ищем объект класса **Нормы Продукции**. Так как он есть, то заносим в буфер 2 его поля *НижняяНорма, ВерхняяНорма.*

13. По значениям буфера 2 проверяем условие фразы where. Если оно не выполнено, то очищаем буфер 2, выбираем текущими следующие объекты классов Показатели, Показатель Паспорта и переходим к п.11.

14. Если условие выполнено, то переносим буферы 1 и 2 в буфер вывода, очищаем их и переходим к п. 8.

Рассмотренные примеры демонстрируют возможность разработки алгоритма третьего этапа по тексту ODQL-запроса с введением стека для получения всех результатов запроса (см. [5]). Анализ показывает, что трудоемкость этого этапа соответствует оценке $\theta(mk)$, где m – количество классов запроса, а k – ожидаемое количество результирующих кортежей запроса.

6. Заключение

Введенная организация данных с помощью метауровня описания сущностей DIM позволяет не только осуществить доступ к данным и манипуляции с ними, но и реализовать динамику данных и типов. К тому же, организация внешних индексов некоторых таблиц метауровня реализует введенные для вычисления запросов индекс-выборки классов и отношений классов DIM. Это приводит не только к удобному, эффективному способу конструирования запросов, но и достаточно эффективному способу вычисления запросов. Конечно, по сравнению с реляционной организацией данных время вычисления запросов несколько увеличится, но по порядку вычислительной сложности запросы не будут менее эффективны. К тому же подход DIM позволяет создать качества баз данных, которые не имели реляционные СУБД. За новые качества надо платить. Для получения коэффициента ухудшения времени выполнения запросов в настоящее время проводится статистическое исследование для разных платформ реализации.

Библиографический список

1. Писаренко, Д. С., Рублев, В. С. Концепции взаимодействия Динамической информационной модели DIM [Текст] / Д. С. Писаренко, В. С. Рублев // Актуальные проблемы математики и информатики. – Ярославль : ЯрГУ, 2007. − С. 75−80.

2. Писаренко, Д. С., Рублев, В. С. Объектная СУБД Динамическая информационная модель DIM и ее основные концепции [Текст] / Д. С. Писаренко, В. С. Рублев // Моделирование и анализ информационных систем − 2009. – Т. 16, № 1. − С. 62−91.

3. Писаренко, Д. С., Рублев, В. С. Синтез аппарата взаимодействий системы управления базами данных DIM [Текст] / Д. С. Писаренко, В. С. Рублев // Материалы XVII международной школы-семинара «Синтез и сложность управляющих систем» имени академика О. Б. Лупанова. − Новосибирск : изд. ИМ СО РАН, 2008. − С. 130−135.

4. Рублев, В. С. Запросная полнота языка ODQL динамической информационной модели DIM [Текст] / В. С. Рублев // Ярославский педагогический вестник. Естественные науки. – 2011. – № 1. − С. 69−75.

5.Рублев, В. С. Организация выполнения объектных запросов в динамической информационной модели DIM [Текст] / В. С. Рублев // Моделирование и анализ информационных систем. – 2011. – Т. 18, № 2. – С. 39–51.

6. Рублев, В. С. Язык объектных запросов динамической информационной модели DIM [Текст] / В. С. Рублев // Моделирование и анализ информационных систем. – 2010. – Т. 17, № 3. − С. 144−161.

7. Сивцов, А. Шесть компонентов успешных проектов на примере DW/BV. Корпоративные базы данных − 2011 [Текст] / А. Сивцов // Материалы 16-й ежегодной технической конференции. – 2011. − CitForum.ru

8. Gray J. Data Management: Past, Prezent and Future // IEEE Computer 1996, October. V.29, no 10.

9.Gray J., Liu D.T., Nieto-Santisteban M., Szalay A., Dewitt D.J., Heber G. Sientific Data Management in the Coming Deacade // SIGMOD Recjrd 2005, December. V. 34, no 4.

10. Jarke M., Koch J., Query Optimization in Database Systems // Computing Surveys 1984, June. Vol.16, No. 2.

11. Silbershatz A., Zdonik S. Database Systems Breaking Out of the Box // ACM Computing Surveys 1996, December. V.28, no 4.

12. Stonebraker M. My Top 10 Assertions About Data Warehouses // BLOG@CACM, August 26, 2010.