Некоторые черты SQL/92 и SQL-3

Оператор выделения памяти под дескриптор

<allocate descriptor statement> ::=

 ALLOCATE DESCRIPTOR <descriptor name>

 [WITH MAX <occurrences>]

<occurences> ::= <simple value specification>

<descriptor name> ::=

 (<scope option>] <simple value specification>

 <scope option> ::= GLOBAL | LOCAL

<simple value specification> ::=

 <parameter name>

 (<embedded variable name>

 (<literal>

*Комментарий:*

Дескриптор, это динамически выделяемая часть памяти прикладной программы, служащая для принятия информации о результате или параметрах динамически подготовленного оператора SQL или задания параметров такого оператора. Смысл того, что для выделения памяти используется оператор SQL, а не просто стандартная функция alloc или какая-нибудь другая функция динамического запроса памяти, состоит в том, что прикладная программа не знает структуры дескриптора и даже его адреса. Это позволяет не привязывать SQL к особенностям какой-либо системы программирования или ОС. Все обмены информацией между собственно прикладной программой и дескрипторами производятся также с помощью специальных операторов SQL (GET и SET, см. ниже).

Второй вопрос: зачем вообще выделять память под дескрипторы динамически. Это нужно потому, что в общем случае прикладная программа, использующая динамический SQL, не знает в статике число одновременно действующих динамических операторов SQL, описание которых может потребоваться. С этим же связано то, что имя дескриптора может задаваться как литеральной строкой символов, так и через строковую переменную включающего языка, т.е. его можно генерировать во время выполнения программы.

В операторе ALLOCATE DESCRIPTOR, помимо прочего, может указываться число описательных элементов, на которое он рассчитан. Если, например, при выделении памяти под дескриптор в разделе WITH MAX указано целое положительное число N, а потом дескриптор используется для описания M (M>N) элементов (например, M столбцов результата запроса), то это приводит к возникновению исключительной ситуации.

Оператор освобождения памяти из-под дескриптора

<deallocate descriptor statement> ::=

 DEALLOCATE DESCRIPTOR <descriptor name>

*Комментарий:*

Выполнение этого оператора приводит к освобождению памяти из-под ранее выделенного дескриптора. После этого использование имени дескриптора незаконно в любом операторе, кроме ALLOCATE DESCRIPTOR.

Оператор получения информации из области дескриптора SQL

<get descriptor statement> ::=

 GET DESCRIPTOR <descriptor name>

 <get descriptor information>

<get descriptor information> ::=

 <get count>

 ( VALUE <item number>

 <get item information>

 ({<comma> <get item information>}...]

<get count> ::=

 <simple target specification 1>

 <equals operator> COUNT

<get item information> ::=

 <simple target specification 2>

 <equals operator>

 <descriptor item name>

<item number> ::= <simple value specification>

<simple target specification 1> ::= <simple target specification>

<simple target specification 2> ::= <simple target specification>

<descriptor item name> ::=

 TYPE

 ( LENGHT

 ( OCTET\_LENGHT

 ( RETURNED\_LENGHT

 ( RETURNED\_OCTET\_LENGHT

 ( PRECISION

 ( SCALE

 ( DATETIME\_INTERVAL\_CODE

 ( DATATIME\_INTERVAL\_PRECISION

 ( NULLABLE

 ( INDICATOR

 ( DATA

 ( NAME

 ( UNNAMED

 ( COLLATION\_CATALOG

 ( COLLATION\_SCHEMA

 ( COLLATION\_NAME

 ( CHARACTER\_SET\_CATALOG

 ( CHARACTER\_SET\_SCHEMA

 ( CHARACTER\_SET\_NAME

<simple target specification> ::=

 <parameter name>

( <embedded variable name>

*Комментарий:*

Оператор GET DESCRIPTOR служит для выборки описательной информации, ранее размещенной в дескрипторе с помощью оператора DESCRIBE. За одно выполнение оператора можно получить либо число заполненных элементов дескриптора (COUNT), либо информацию, содержащуюся в одном из заполненных элементов.

Оператор установки дескриптора

<set descriptor statement> ::=

 SET DESCRIPTOR <descriptor name>

 <set descriptor information>

<set descriptor information> ::=

 <set count>

 ( VALUE <item number> <set item information>

 [{<comma> <set item information>}...]

<set count> ::= COUNT <equals operator> <simple value specification 1>

<set item information> ::=

 <descriptor item name>

 <equals operator>

 <simple value specification 2>

<simple target specification 1> ::= <simple target specification>

<simple target specification 2> ::=<simple target specification>

<item number> ::= <simple value specification>

*Комментарий:*

Оператор SET DESCRIPTOR служит для заполнения элементов дескриптора с целью его использования в разделе USING. За одно выполнение оператора можно поместить значение в поле COUNT (число заполненных элементов), либо частично или полностью сформировать один элемент дескриптора.

Оператор подготовки

<prepare statement> ::=

 PREPARE <SQL statement name>

 FROM <SQL statement variable>

<SQL statement variable> ::= <simple target specification>

<preparable statement> ::=

 <preparable SQL data statement>

| <preparable SQL schema statement>

| <preparable SQL transaction statement>

| <preparable SQL session statement>

| <preparable implementation-defined statement>

<preparable SQL data statement> ::=

 <delete statement: searched>

| <dynamic single row select statement>

| <insert statement>

| <dynamic select statement>

| <update statement: searched>

| <preparable dynamic delete statement: positioned>

| <preparable dynamic update statement: positioned>

<preparable SQL schema statement> ::= <SQL schema statement>

<preparable SQL transaction statement> ::= <SQL transaction statement>

<preparable SQL session statement> ::= <SQL session statement>

<dynamic select statement> ::= <cursor specification>

<dynamic simple row select statement> ::= <query specification>

<SQL statement name> ::= <statement name> | <extended statement name>

<extended statement name> ::= [scope option] <simple value specification>

<cursor specification> ::=

 <query expression> [<order by clause>]

 [<updatability clause>]

<updatability clause> ::=

 FOR { READ ONLY | UPDATE [ OF <column name list> ] }

<query expression> ::=

 <non-join query expression>

| <joined table>

<query specification> ::=

 SELECT [<set quantifier>]

 <select list> <table expression>

<set quantifier> ::= DISTINCT | ALL

*Комментарий:*

Оператор PREPARE вызывает компиляцию и построение плана выполнения заданного в текстовой форме оператора SQL. После успешного выполнения оператора PREPARE с подготовленным оператором связывается указанное (литерально или косвенно) имя этого оператора, которое потом может быть использовано в операторах DESCRIBE, EXECUTE, OPEN CURSOR, ALLOCATE CURSOR и DEALLOCATE PREPARE. Эта связь сохраняется до явного выполнения оператора DEALLOCATE PREPARE.

Оператор отказа от подготовленного оператора

<deallocate prepared statement> ::=

 DEALLOCATE PREPARE <SQL statement name>

*Комментарий:*

Выполнение этого оператора приводит к тому, что ранее подготовленный оператор SQL, связанный с указанным именем оператора, ликвидируется, и, соответственно, имя оператора становится неопределенным. Если подготовленный оператор являлся оператором выборки, и к моменту выполнения оператора DEALLOCATE существовал открытый курсор, связанный с именем подготовленного оператора, то оператор DEALLOCATE возвращает код ошибки. Если же для подготовленного оператора выборки существовал неоткрытый курсор, образованный с помощью оператора ALLOCATE CURSOR, то этот курсор ликвидируется. Если курсор объявлялся оператором DECLARE CURSOR, то такой курсор переходит в состояние, существовавшее до выполнения оператора PREPARE. Если с курсором был связан подготовленный оператор (динамический DELETE или UPDATE), то для этих операторов выполняется неявный оператор DEALLOCATE.

Оператор запроса описания подготовленного оператора

<describe statement> ::=

 <describe input statement>

| <describe output statement>

<describe input statement> ::=

 DESCRIBE INPUT <SQL statement name> <using descriptor>

<describe output statement> ::=

 DESCRIBE [OUTPUT] <SQL statement name> <using descriptor>

<using clause> ::= <using arguments> | <using descriptor>

<using arguments> ::=

 { USING | INTO } <argument> [{<comma> <argument>}...]

<argument> ::= <target specification>

<using descriptor> ::=

 { USING | INTO } SQL DESCRIPTOR <descriptor name>

<target specification> ::=

 <parameter specification>

| <variable specification>

<parameter specification> ::=

 <parameter name> [<indicator parameter>]

<indicator parameter> ::= [INDICATOR] <parameter name>

<variable specification> ::=

 <embedded variable name> [<indicator variable>]

<indicator variable> ::= [INDICATOR] <embedded variable name>

*Комментарий:*

При выполнении оператора DESCRIBE происходит заполнение указанного в операторе дескриптора информацией, описывающей либо результат ранее подготовленного оператора SQL (если это оператор выборки), либо количество и типы параметров подготовленного оператора. В <using descriptor> здесь полагается писать USING SQL DESCRIPTOR.

Оператор выполнения подготовленного оператора

<execute statement> ::=

 EXECUTE <SQL statement name>

 (<result using clause>]

 (<parameter using clause>]

<result using clause> ::= <using clause>

<parameter using clause> ::= <using clause>

*Комментарий:*

Оператор EXECUTE может быть применен к любому ранее подготовленному оператору SQL, кроме <dynamic select statement>. Если это оператор <dynamic single row select statement>, то оператор EXECUTE должен содержать раздел <result using class> с ключевым словом INTO. В любом случае число фактических параметров, задаваемых через разделы using, должно соответствовать числу формальных параметров, определенных в подготовленном операторе SQL.

Оператор подготовки с немедленным выполнением

<execute immediate statement> ::=

 EXECUTE IMMEDIATE <SQL statement variable>

*Комментарий:*

При выполнении оператора EXECUTE IMMEDIATE производится подготовка и немедленное выполнение заданного в текстовой форме оператора SQL. При этом подготавливаемый оператор не должен быть оператором выборки, не должен содержать формальных параметров и комментариев.

Оператор объявления курсора над динамически подготовленным оператором выборки

<dynamic declare cursor> ::=

 DECLARE <cursor name> [INSENSITIVE] [SCROLL]

 CURSOR FOR <statement name>

*Комментарий:*

Как определяется в новом стандарте, для всех операторов DECLARE CURSOR, курсоры фактически создаются при начале транзакции и уничтожаются при ее завершении. Заметим, что в этом операторе <cursor name> и <statement name> - прямо заданные идентификаторы.

Оператор определения курсора над динамически подготовленным оператором выборки

<allocate cursor statement> ::=

 ALLOCATE <extended cursor name> [INSENSITIVE] [SCROLL]

 CURSOR FOR <extended statement name>

<extended cursor name> ::=

 (<scope option>] <simple value specification>

*Комментарий:*

Курсоры, определяемые с помощью оператора ALLOCATE CURSOR, фактически создаются при выполнении такого оператора и уничтожаются при выполнении оператора DEALLOCATE PREPARE или при завершении транзакции. В этом операторе имена курсора и подготовленного оператора SQL могут задаваться не только в литеральной форме, но и через переменные. <scope option> относится к области видимости имен: в пределах текущего модуля или в пределах текущей сессии.

Оператор открытия курсора, связанного с динамически подготовленным оператором выборки

<dynamic open statement> ::=

 OPEN <dynamic cursor name> [<using clause>]

*Комментарий:*

По сути, оператор открытия курсора, связанного с динамически подготовленным оператором SQL, отличается от статического случая только возможным наличием раздела using, в котором задаются фактические параметры оператора выборки. Кроме того, имя курсора может задаваться через переменную.

Оператор чтения строки по курсору, связанному с динамически подготовленным оператором выборки

<dynamic fetch statement> ::=

 FETCH [[<fetch orientation>] FROM]

 <dynamic cursor name> <using clause>

*Комментарий:*

По сути, оператор чтения по курсору, связанному с динамически подготовленным оператором SQL, отличается от статического случая только возможным наличием раздела using, в котором задается размещение значений текущей строки результирующей таблицы. Кроме того, имя курсора может задаваться через переменную.

Оператор закрытия курсора, связанного с динамически подготовленным оператором выборки

<dynamic close statement> ::=

 CLOSE <dynamic cursor name>

*Комментарий:*

По сути, оператор закрытия курсора, связанного с динамически подготовленным оператором SQL, отличается от статического случая только тем, что имя курсора может задаваться через переменную

Оператор позиционного удаления по курсору, связанному с динамически подготовленным оператором выборки

<dynamic delete statement: positioned> ::=

 DELETE FROM <table name>

 WHERE CURRENT OF <dynamic cursor name>

*Комментарий:*

По сути, оператор позиционного удаления по курсору, связанному с динамически подготовленным оператором SQL, отличается от статического случая только тем, что имя курсора может задаваться через переменную.

Оператор позиционной модификации по курсору, связанному с динамически подготовленным оператором выборки

<dynamic update statement: positioned> ::=

 UPDATE <table name>

 SET <set clause> [{<comma> <set clause>}...]

 WHERE CURRENT OF <dynamic cursor name>

*Комментарий:*

По сути, оператор позиционной модификации по курсору, связанному с динамически подготовленным оператором SQL, отличается от статического случая только тем, что имя курсора может задаваться через переменную.

Подготавливаемый оператор позиционного удаления

<preparable dynamic delete statement: positioned> ::=

 DELETE [FROM <table name>]

 WHERE CURRENT OF <cursor name>

*Комментарий:*

Основной смысл появления этого и следующего операторов состоит в том, что сочетание курсора, определенного на динамически подготовленном операторе выборки, и статически задаваемых операторов удаления и модификации по этому курсору, выглядит довольно нелепо. Поэтому в стандарте появились динамически подготавливаемые позиционные операторы удаления и вставки. Естественно, что выполняться они должны с помощью оператора EXECUTE.

Подготавливаемый оператор позиционной модификации

<preparable dynamic update statement: positioned> ::=

 UPDATE [<table name>]

 SET <set clause> [{<comma> <set clause>}...]

 WHERE CURRENT OF <cursor name>

*Комментарий:*

Смотри предыдущий пункт.

Если внимательно сравнивать средства динамического SQL СУБД Oracle V.6 и стандарта SQL/92, то видно, что Oracle практически вкладывается в стандарт, если не считать небольших синтаксических различий и (что существенно более важно) разного стиля работы с дескрипторами. Думается, что примерно такая же ситуация имеет место в других СУБД, поддерживающих динамический SQL.

Сводка новых возможностей SQL-3

В стандарте SQL/92 по сравнению со стандартом SQL/89 язык был расширен главным образом количественно, хотя даже этих количественных расширений оказалось достаточно для того, чтобы стандарт SQL/92 не удалось полностью реализовать до сих пор в большинстве коммерческих СУБД. Поскольку SQL/92 не удовлетворял значительной части претензий, исторически предъявляемых к языку SQL, был сформирован новый комитет, который должен выработать стандарт языка с качественными расширениями. Язык SQL-3 пока не сформирован полностью, многие аспекты продолжают обсуждаться. Поэтому к приводимой здесь сводке возможностей нужно относиться как к сугубо предварительной.

17.19.1. Типы данных

Набор встроенных типов данных предполагается расширить типами BOOLEAN и ENUMERATED. Хотя по причине поддержки неопределенных значений языку SQL свойственно применение трехзначной логики, тип BOOLEAN содержит только два возможных значения true и false. Для представления значения unknown рекомендуется использовать NULL, что, конечно, не вполне естественно. Перечисляемый тип ENUMERATED обладает свойствами, подобными свойствам перечисляемых типов в языках программирования.

Расширены возможности работы с неопределенными значениями. Появился новый оператор CREATE NULL CLASS, позволяющий ввести именованный набор именованных неопределенных значений. При определении домена можно явно указать имя класса неопределенных значений, появление которых допустимо в столбцах, связанных с этим доменом. Смысл каждого неопределенного значения интерпретируется на уровне пользователей.

Предполагается включение в язык возможности использования определенных пользователями типов данных. Видимо, будут иметься возможности определения абстрактных типов данных с произвольно сложной внутренней структурой на основе таких традиционных возможностей агрегирования и структуризации как LIST, ARRAY, SET, MULTISET и TUPLE, а также возможности определения объектных типов с соответствующими методами в стиле объектно-ориентированного подхода.

Появляется возможность использования принципов наследования свойств существующей таблицы (супертаблицы) при определении новой таблицы (подтаблицы). Подтаблица наследует от супертаблицы все определения столбцов и первичного ключа. Другая возможность - создать таблицу, "подобную" существующей в том смысле, что в новой таблице наследуются определения некоторых столбцов существующей таблицы.

17.19.2. Некоторые другие свойства SQL-3

Одной из проблем реализации языка SQL всегда являлась проблема распознавания "изменяемости" соединений. Как известно, если представление включает соединение общего вида, то теоретически невозможно определить, можно ли однозначно интерпретировать операции обновления такого представления. Однако существует несколько важных классов соединений, которые заведомо являются изменяемыми. В SQL-3 предполагается выделить эти классы с помощью специальных синтаксических конструкций.

Наконец-то появляется возможность определения триггеров как комбинации спецификаций события и действия. Действие определяется как SQL-процедура, в которой могут использоваться как операторы SQL, так и ряд управляющих конструкций. На самом деле, этот механизм очень близок к тому, который реализован в Oracle V.7.

Что касается управления транзакциями, то происходит возврат к старой идее System R о возможности установки внутри транзакции точек сохранения (savepoints). В операторе ROLLBACK можно указать идентификатор ранее установленной точки сохранения, и тогда будет произведен откат транзакции не к ее началу, а к этой точке сохранения.

Как видно, можно ожидать наличия в SQL-3 многих интересных и полезных возможностей. Однако даже промежуточные проекты стандарта включают почти в два раза больше страниц, чем стандарт SQL/92. Поэтому трудно ожидать быстрой реализации этого стандарта после его принятия (а многие вообще сомневаются, что этот стандарт будет когда-либо реализован).