**1. Каковы назначение и синтаксис оператора Prepare?**

Оператор prepare выполняет предварительную проверку запроса к базе данных на его корректность, а также позволяет СУБД заранее построить план исполнения данного запроса.

Также prepare позволяет в дальнейшем в удобном формате делать подстановку параметров из пользовательского ввода в код запроса, при этом не меняя сам текст запроса и план его исполнения.

Оператор prepare имеет следующий синтаксис:

PREPARE <имя запроса> FROM <текст запроса>

В качестве <имя запроса> может быть любое уникальное имя, по которому в дальнейшем этот запрос можно идентифицировать в программе. В качестве <текст запроса> должна быть символьная строка char[] либо const char[].

В <текст запроса> некоторые переменные могут быть заменены знаком вопроса (?), тогда эти переменные можно подставить во время выполнения запроса, реализуя тем самым работу с пользовательским вводом и вариативность исполнения программы.

Пример использование PREPARE без курсора и без подстановки значений:

exec sql begin declare section;

char task\_1[] = "SELECT SUM(\*) FROM s WHERE s.n\_post = ‘S1’";

int sum;

exec sql end declare section;

exec sql begin work;

// Подготовка запроса к выполнению

exec sql prepare task\_1 from :task\_1;

exec sql execute task\_1 into :sum;

exec sql commit work;

Пример использование PREPARE без курсора и с подстановкой значений:

exec sql begin declare section;

char task\_1[] = "SELECT SUM(\*) FROM s WHERE s.n\_post = ?";

int sum;

exec sql end declare section;

exec sql begin work;

// Подготовка запроса к выполнению

exec sql prepare task\_1 from :task\_1;

exec sql execute task\_1 into :sum USING “S1”;

exec sql commit work;

Пример использования PREPARE с курсором и без подстановки значений:

exec sql begin declare section;

char task\_1[] = "SELECT kol FROM spj WHERE n\_post = ‘S1’";

int kol;

exec sql end declare section;

exec sql begin work;

// Подготовка запроса к выполнению

exec sql prepare task\_1 from :task\_1;

exec sql declare cursor\_1 cursor for task\_1;

exec sql open cursor\_1;

exec sql fetch cursor\_1 into :kol;

while (sqlca.sqlcode == 0) {

exec sql fetch cursor\_1 into :kol;

}

exec sql commit work;

Пример использования PREPARE с курсором и c подстановкой значений:

exec sql begin declare section;

char task\_1[] = "SELECT kol FROM spj WHERE n\_post = ?";

int kol;

exec sql end declare section;

exec sql begin work;

// Подготовка запроса к выполнению

exec sql prepare task\_1 from :task\_1;

exec sql declare cursor\_1 cursor for task\_1;

exec sql open cursor\_1 USING “S1”;

exec sql fetch cursor\_1 into :kol;

while (sqlca.sqlcode == 0) {

exec sql fetch cursor\_1 into :kol;

}

exec sql commit work;

**2. Каковы назначение и синтаксис оператора Execute?**

Оператор execute предназначен для исполнения предварительно подготовленного командой prepare запроса. Позволяет вернуть только 1 строку из результирующего запроса.

Синтаксис оператора execute:

EXECUTE <имя запроса> INTO <прог. перемен.> [, <прог. перемен.>]\*   
[USING <прог. перемен.> [, <прог. перемен.>\*] ]

<прог. перемен.> - программная переменная (любая переменная из программы, которая объявлена в секции DECLARE).

USING позволяет подставлять реальные значения вместо (?) в запрос.

INTO позволяет записать результат запроса в переменные программы.

Пример использования EXECUTE:

exec sql begin declare section;

char task\_1[] = "SELECT SUM(\*) FROM s WHERE s.n\_post = ?";

int sum;

char n\_post] = “S1”;

exec sql end declare section;

exec sql begin work;

// Подготовка запроса к выполнению

exec sql prepare task\_1 from :task\_1;

exec sql execute task\_1 into :sum USING :n\_post;

exec sql commit work;

**3. Каковы особенности использования динамических операторов SQL?**

Динамические операторы SQL позволяют заранее подготовить сложные запросы (один или несколько подряд), что особенно полезно, если эти запросы будут исполняться несколько раз за время работы программы с разными переменными. В таком случае, СУБД построит план исполнения лишь 1 раз, а затем будет исполнять его с разными переменными, не тратя время и ресурсы на создания нового плана каждый раз.

Запрос с применением динамического SQL формируется в 3 этапа:

1. Программа собирает текст SQL запроса в виде символьной строки;
2. Программа вызывает оператор PREPARE для данного SQL запроса, позволяя СУБД подготовить план исполнения этого запроса, а также проверить корректность такого запроса;
3. Программа вызывает оператор EXECUTE либо OPEN для подготовленного запроса, после которых СУБД исполняет заранее подготовленный план исполнения, подставляя переданные в запрос значения.

При использовании динамического оператора SELECT нельзя использовать оператор INTO, позволяющему записать полученные значения в переменные программы. Вместо этого везде, где могут встречаться переменные, куда нужно записать значения, нужно использовать знак (?) и затем эти переменные должны подставляться оператором USING. Помимо этого, можно использовать оператор INTO с командами EXECUTE либо FETCH, тогда всё тоже work.

**4. Что такое динамические главные переменные? Каково их назначение?**

[<i>Informix Guide to SQL: Syntax</i> (oninit.com)](https://www.oninit.com/manual/informix/english/docs/dbdk/infoshelf/sqls/01prep.fm.html)

**Динамические главные переменные** – переменные, выделенные в коде в динамической памяти после предварительной обработки запроса.

После успешного выполнения оператора PREPARE, мы можем выполнить оператор DESCRIBE, который вернёт нам информацию запросе, в том числе и о выходных параметрах запросов: число возвращаемых столбцов, названия столбцов, их типы данных.

За счёт полученной информации

Вызов DESCRIBE происходит следующим образом:

struct sqlda \*desc;

EXEC SQL BEGIN DECLARE SECTION;

char demoselect[200] = "select \* from table";

EXEC SQL END DECLARE SECTION;

EXEC SQL prepare iid from :demoselect;

EXEC SQL describe iid into desc;

// sqlda.sqld - содержит число столбцов, возвращаемых запросом

// sqlda.sqlvar[] - содержат столбцы запроса

// sqlda.sqlvar[].sqltype & SQLTYPE - позволяет получить типы данных структур

// sqlda.sqlvar[].sqldata - указатель на данные, выделяемые нами в программе средствами си (через malloc) (куда можно записывать данные из запроса) (это и есть сами динамические главные переменные)

Затем, чтобы записывать туда данные, используется следующая команда:

EXEC SQL fetch d\_curs using descriptor demodesc;

После неё данные запишутся в поля desc.sqlvar[].sqldata.

В конце использования переменных нужно освободить память, под них выделенную, средствами си (через free)

**5. С какими операторами связано использование динамических главных переменных?**

С DESCRIBE из прошлого пункта

**6. Каково назначение оператора Execute Immediate?**

Оператор EXECUTE IMMEDIATE говорит программе выполнить операторы PREPARE, EXECUTE и FREE (освобождение памяти после оператора PREPARE) за одно действие.

Главное отличие от оператора EXEC SQL <текст запроса> в том, что данный оператор позволяет исполнять запросы только статически, на этапе компиляции программы. Оператор EXECUTE IMMEDIATE позволяет исполнять запросы, динамически формируемые в программе (либо получаемые извне). При этом, в отличие от использования отдельно команд PREPARE, EXECUTE, FREE, эта команда позволяет выполнить динамический запрос в одно действие.

Но, в отличие от prepare, execute, free, при использовании EXECUTE IMMEDIATE план исполнения запроса будет сразу “удалён”, поэтому, если нужно вызывать один запрос несколько раз, лучше всё равно использовать prepare и execute.

Также важное ограничение EXECUTE IMMEDIATE: она запрещает выполнять множество разных команд, например, нельзя использовать команды, возвращающие значения.

[<img src="02exec6.gif" height="49" width="100"> EXECUTE IMMEDIATE (oninit.com)](https://www.oninit.com/manual/informix/english/docs/dbdk/is40/sqls/02exec4.html)