**1.Что такое главные переменные? Как они определяются и используются в программах на языке ESQL/С?**

Выполнение оператора SQL фактически является вызовом сервера БД как отдельной программы. Информация должна передаваться от программы-клиента к программе-сервера и возвращаться обратно. Часть этих взаимодействий осуществляется через т.н. **главные переменные:**

Host-переменные:

Host-переменные - переменные, позволяющие создавать связь между SQL запросами и си-кодом.

**Объявление переменных**

Объявление host-переменных происходит внутри секции SQL:

**exec sql begin declare section;**

...

**exec sql end declare section;**

Сами переменные внутри такой секции описываются на обычном синтаксисе языка си. В качестве типа данных может быть практически всё что угодно:

* Массивы (одномерные либо двумерные);
* Указатели (в основном для работы со строками);
* Переменные;
* Структуры (для создания элементов, совпадающих по полям с полями таблицы);
* Параметры функций (тут уже хз зачем).

При этом нельзя использовать typedef-ы (псевдонимы типов), раскрывающиеся как многомерные массивы, и нельзя использовать union-ы (тип, позволяющий хранить в себе значение любого из заданных типов данных).

Пример из лабораторной работы:

**exec sql begin declare section;**

int reiting, count\_izd, weight, min\_weight, pves;

float mves;

char n\_post[7], name[21], town[21], n\_izd[7], n\_det[7], task\_num;

**exec sql end declare section;**

Так же, host-переменные можно определять без оборачивания в блок, используя знак доллара ($):

$int reiting, count\_izd, weight, min\_weight, pves;

$float mves;

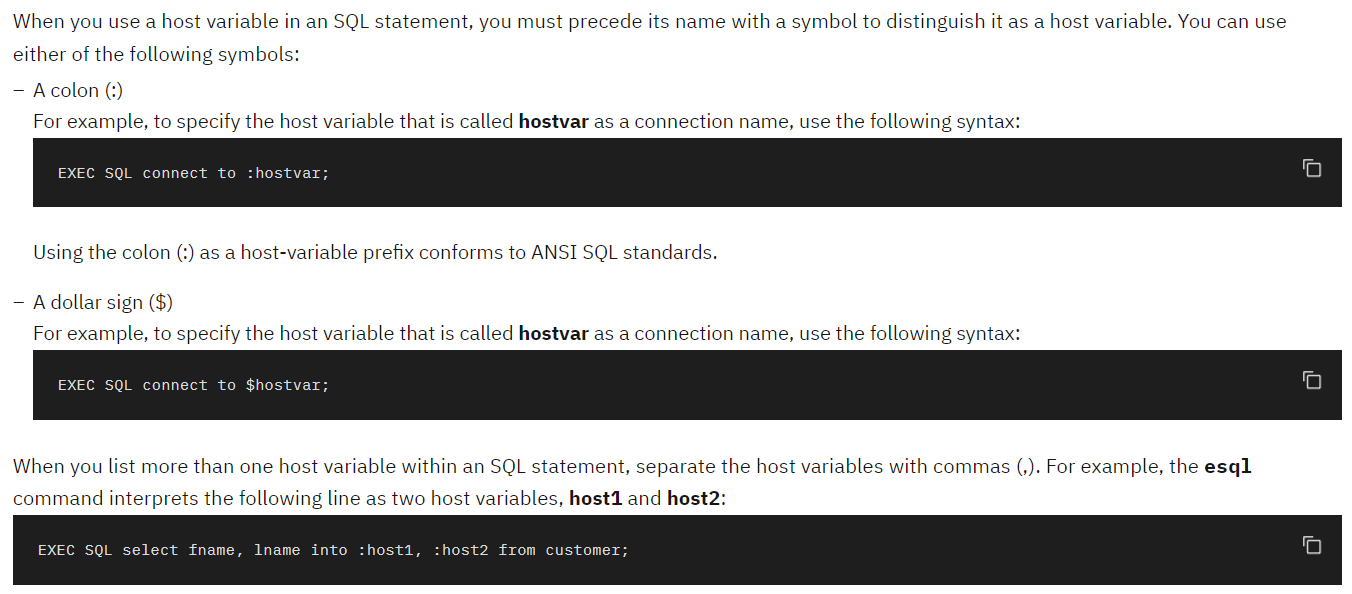
$char n\_post[7], name[21], town[21], n\_izd[7], n\_det[7], task\_num;

**Использование переменных**

Для использования переменных внутри кода си достаточно просто обратиться к ним как к обычной переменной:

printf("\nЧисло изделий: %d\n", count\_izd);

Для использования переменных внутри кода SQL, необходимо поставить перед переменной знак двоеточия (:) либо знак доллара ($):



Пример из нашего кода:

exec sql select count(distinct n\_izd)

into :count\_izd

from spj

join s on s.n\_post = spj.n\_post

join p on p.n\_det = spj.n\_det

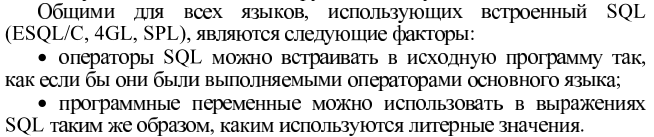
where ves > 12 and s.name in (select name

from s

order by name

limit 1);

**2. Каковы правила использования SQL-описаний в программах на ESQL/C?**



Правила использования SQL-описаний в ESQL/C следующие:

1. Определение переменных для привязки их к SQL-запросам:

Внешние SQL-переменные должны быть объявлены между парой директив **EXEC SQL DECLARE SECTION** и **EXEC SQL END DECLARE SECTION**. Эта секция объявления должна быть помещена перед основным телом функции на языке C.

**exec sql begin declare section;**

int reiting, count\_izd, weight, min\_weight, pves;

float mves;

char n\_post[7], name[21], town[21], n\_izd[7], n\_det[7], task\_num;

**exec sql end declare section;**

1. Набор данных:

SQL-описания используются для определения переменных, которые будут использоваться для хранения результирующего набора или отдельных значений из табличных столбцов в базе данных.

Надо ещё написать примеры таких переменных

1. Привязка переменных:

Переменные, используемые в SQL-операторах, должны быть связаны с SQL-описаниями оператором INTO. Для этого перед именем переменной ставится двоеточие ":", указывая на привязку переменной к SQL-описанию.

Например:

SELECT column1 **INTO :variable** FROM table\_name;

1. Результаты запросов:

Результирующие значения, полученные из SQL-запросов, должны быть присвоены SQL-описаниям с использованием оператора **INTO**. Это делается для сохранения полученных данных в определенной переменной или переменных.

Пример кода:

// запрос

exec sql select count(distinct n\_izd)

**into :count\_izd**

from spj

join s on s.n\_post = spj.n\_post

join p on p.n\_det = spj.n\_det

where ves > 12 and s.name in (select name

from s

order by name

limit name);

1. Компиляция и выполнение кода:

Код, содержащий SQL-описания, должен быть скомпилирован и выполнен вместе с остальной частью программы на языке C. Утилиты компиляции ESQL/C анализируют SQL-описания и генерируют соответствующий код, который будет работать с базой данных.

**3. Как обрабатываются NULL-значения в программах на языке ESQL/C?**

В C нет NULL-типов данных, поэтому для проверки значения переменной на NULL существует 3 способа:

1. Использование в запросах переменных-индикаторов (например, SELECT row INTO :name:nameInd …). В таком случае, если row будет равен NULL, то в nameInd будет значение -1, иначе 0.
2. Проверка переменных функцией risnull(DATATYPE, VAR\_POINTER): функция вернёт -1, если переменная равна NULL, либо 0 в противном случае.
3. Проверка флага sqlca.sqlcode: при включенном параметре компилятора -icheck и возникновении NULL в строке, в sqlca.sqlcode будет отрицательное значение как об ошибке.

Поскольку в языке С нет возможности убедиться, имеет ли элемент таблицы какое-либо значение, ESQL/C делает это для своих главных переменных., называемых переменными-индикаторами. Переменная-индикатор является дополнительной переменной, ассоциированной с главной переменной, в которую могут поступать NULL-значения. Когда сервер БД помещает данные в главную переменную, он также устанавливает специальное значение в переменную-индикатор, которое показывает, не являются ли эти данные NULL-значением.

Переменная-индикатор описывается как обычная главная переменная целого типа, а при использовании отделяется от главной переменной, в которую передаются значения, знаками ":", "$" или словом indicator.

Пример объявления переменной и индикатора (на этом этапе они между собой не связаны):

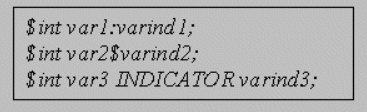
**exec sql begin declare section;**

int var1

int varind1;

**exec sql end declare section;**

Примеры равнозначных использований переменных-индикаторов:



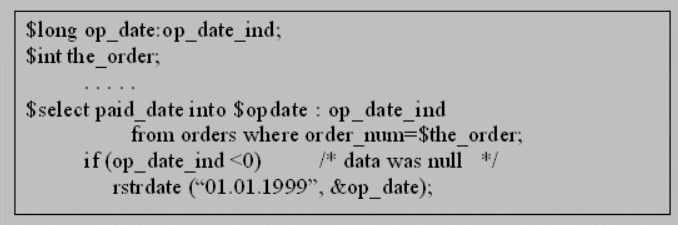
Пример реального использования из документации:

**$char name[16];**  
**$char comp[20];**  
**$short nameind;**  
**$short compind;**

**$select lname, company**  
 **into $name$nameind, $comp$compind**  
 **from customer**  
 **where customer\_num = 105;**

При выборе оператором Select NULL-значения переменная-индикатор (если она используется) получает значение **-1**. В случае нормального возврата переменной индикатор равен **0**. В случае, если индикатор не используется, то результат зависит от режима генерации программы:

Ниже приведен фрагмент программы, использующей переменные-индикаторы для обработки NULL-значений.



Если не использовать переменные-индикаторы и программа компилировалась с флагом -icheck, ESQL/C генерирует ошибку и установит sqlca.sqlcode в отрицательное значение при возврате NULL-значения;   
если программа компилировалась без указанного флага, при отсутствии индикатора ошибка не генерируется и тогда проверку переменной на NULL можно осуществить вызовом функции **int risnull(DATATYPE, VAR\_POINTER)** - функция вернёт 1, если число NULL, и 0 в противном случае.

**4. Каково назначение заголовочных файлов?**

**datetime.h** - описывает структуру для типа данных **DATETIME** и **INTERVAL**;

**decimal.h** - описывает структуру для типа данных **decimal (числа с фиксированной точкой)**;

**locator.h** - описывает структуру для **blobs-данных (**бинарные объекты**), DECIMAL, BYTE** и **TEXT**;

**varchar.h** - описывает структуру для типа данных **varchar**;

**sqlhdr.h** - описывает прототипы функций библиотеки SQL/C;

**sqltype.h, sqltypes.h** - структуры для работы с динамическими главными переменными.

**5. Что такое курсор? В чем отличие последовательного и скроллирующего курсоров по описанию и по использованию?**

**Курсор (CURSOR)** – объект данных, с помощью которого осуществляется обработка многострочного запроса.

(Это набор строк, возвращаемых запросом, может состоять из нуля, одной или нескольких строк, в зависимости от того, сколько строк соответствует вашим критериям поиска. Когда запрос возвращает несколько строк, вы можете явно объявить курсор для обработки строк.)

Последовательный курсор позволяет просматривать активное множество только в последовательном порядке, а также используется при модификации и удалении строк из активного множества.

**DECLARE имя\_курсора CURSOR FOR запрос**

Скроллирующий курсор позволяет просматривать строки из активного множества в произвольном порядке.

**DECLARE имя\_курсора SCROLL CURSOR FOR запрос**

**6. Каковы назначение и синтаксис операторов Declare, Open, Fetch, Close?**

Основные операции при работе с курсором:

• объявление курсора, выполняемое оператором **Declare**:

**DECLARE имя\_курсора CURSOR FOR запрос**

**DECLARE имя\_курсора SCROLL CURSOR FOR запрос**

• открытие курсора, выполняемое оператором **Open**:

**OPEN имя\_курсора;**

• выборка по курсору очередной строки запроса в главные переменные, выполняемая оператором **Fetch (вытягивает данные)**:

**FETCH имя\_курсора [INTO variable\_list];**

variable\_list список переменных, разделенных запятыми, в которые вы хотите сохранить результирующий набор курсора (если таковых не было в самом запросе, иначе можно опустить).

• закрытие курсора, выполняемое оператором **Close:**

**CLOSE имя\_курсора;**

Пример работы с курсором:

exec sql begin work;

// запрос

**exec sql declare cursor\_3 cursor for**

select spj.n\_izd, spj.kol\*p.ves ves\_post, new.min\_ves

into :n\_izd, :pves, :mves

from spj

...

**exec sql open cursor\_3;**

if (sqlca.sqlcode < 0) {

error\_msg("Ошибка при открытии курсора (OPEN).");

exec sql close cursor\_3;

exec sql rollback work;

break;

}

**exec sql fetch cursor\_3;**

if (sqlca.sqlcode < 0) {

error\_msg("Ошибка при чтении курсора (FETCH).");

exec sql close cursor\_3;

exec sql rollback work;

break;

} else if (sqlca.sqlcode == 100)

printf("\nНет данных.\n");

else {

...

while (sqlca.sqlcode == 0) {

**exec sql fetch cursor\_3;**

if (sqlca.sqlcode < 0) {

error\_msg("Ошибка при чтении курсора (FETCH).");

exec sql close cursor\_3;

exec sql rollback work;

break;

} else if (sqlca.sqlcode == 0)

printf("%s\t\t%d\t\t%f\n", n\_izd, pves, mves);

}

}

**exec sql close cursor\_3;**

exec sql commit work;

**7. По какой из команд сервер выделяет память под курсор?**

Сервер выделяет память под курсор после вызова команды **DECLARE.**

**8. По какой из команд сервер начинает поиск строк запроса?**

Сервер начинает поиск строк запроса в трёх случаях:

* После отправки SQL-запроса внутри блока **BEGIN WORK...END WORK**;
* После отправки команды **EXECUTE** для запуска предварительно обработанного командой **PREPARE** SQL-запроса;
* После отправки команды **OPEN** открытия курсора для получения данных с сервера.

**9. Чем заканчивается работа оператора Open и Fetch?**

При вызове оператора **OPEN** сервер:

* открывает данный курсор для дальнейшей с ним работы;
* начинает исполнение пользовательского запроса.

В случае ошибки открытия курсора (например, он уже открыт), сервер возвращает соответствующую ошибку.

При вызове оператора **FETCH** возможны следующие варианты:

* Если в буфере курсора есть строки, то он записывает данные из одной из них в переменные, используемые для запроса через **INTO**;
* Если в буфере нет строк, то программа отправляет на сервер размер буфера, а сервер:
  + Либо возвращает столько строк, сколько влезет в буфер, и программа продолжит работу,
  + Либо вернёт ошибку (отрицательное число) или код окончания данных запроса (положительное).

**10. Каковы назначение и синтаксис операторов Prepare, Execute?**

(подготавливаемый запрос)

Оператор **PREPARE** позволяет серверу заранее обработать предстоящий запрос, проверить корректность его синтаксиса, а также составить план исполнения этого запроса. Выглядит оператор следующим образом:

PREPARE <имя запроса> FROM “<текст запроса>”;

PREPARE <имя запроса> FROM :<имя переменной со строкой запроса>;

Оператор **EXECUTE** говорит серверу о том, что пора бы уже и исполнить подготовленный запрос. Синтаксис прост:

EXECUTE <имя запроса>;

Пример запроса с подготовкой:

**EXEC SQL PREPARE del\_1 FROM**   
 **“DELETE FROM customer WHERE customer\_num = 119”;**  
**EXEC SQL EXECUTE del\_1;**

**ДОПОЛНИТЕЛЬНЫЕ БЛЯТЬ ЕГО ВОПРОСЫ**

**11. Что такое транзакция?**

**Транзакция** — это логическая единица работы.

Можно сказать, что транзакция является набором действий, совершаемых над базой данных в рамках одной задачи.

Например, задача перевода денежных средств с одного счёта на другой включает в себя две операции: снятие денег с одного счёта и пополнение другого счёта. Выполняя эти действия по отдельности, можно столкнуться с проблемой, когда деньги с одного счёта уже исчезли, а на другом они ещё не появились. Если в этот момент начнут выполняться другие операции, например снятие средств со второго счёта, то мы можем столкнуться с проблемой недостатка средств на счёте. Поэтому логично требовать, чтобы операции по переводу денежных средств осуществлялись внутри одной группы, и такая группа и есть - транзакция.

В esql/c транзакции используются следующими образом:

1. Начало транзакции

Для начала транзакции используется команда **BEGIN WORK:**

**EXEC SQL BEGIN WORK;**

EXEC SQL SELECT …..;

...

1. Завершение (принятие) транзакции

Для принятия успешной транзакции (когда все команды успешно выполнены) используется команда **COMMIT WORK**:

If (sqlca.sqlcode == 0)

**EXEC SQL COMMIT WORK;**

1. Откат транзакции

Для отката неудачной транзакции (когда какая-либо из команд не является выполнимой и требуется отменить изменения всех предыдущих команд) используется команда ROLLBACK WORK:

If (sqlca.sqlcode < 0)

**EXEC SQL ROLLBACK WORK;**