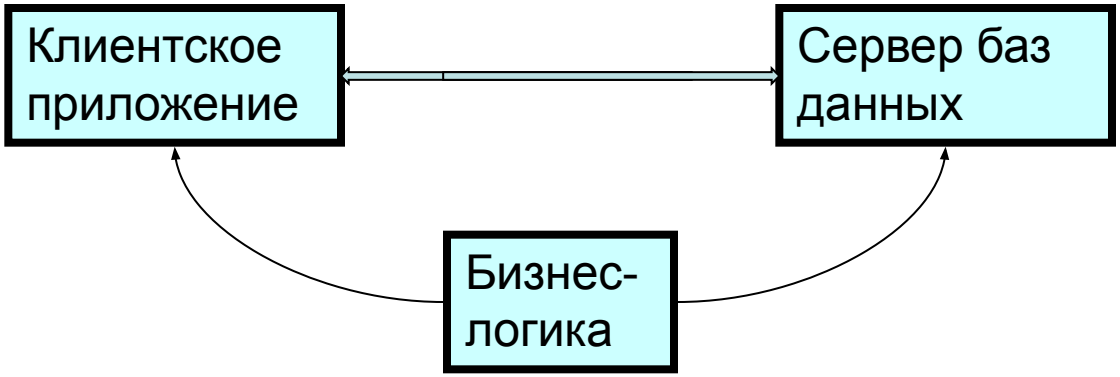
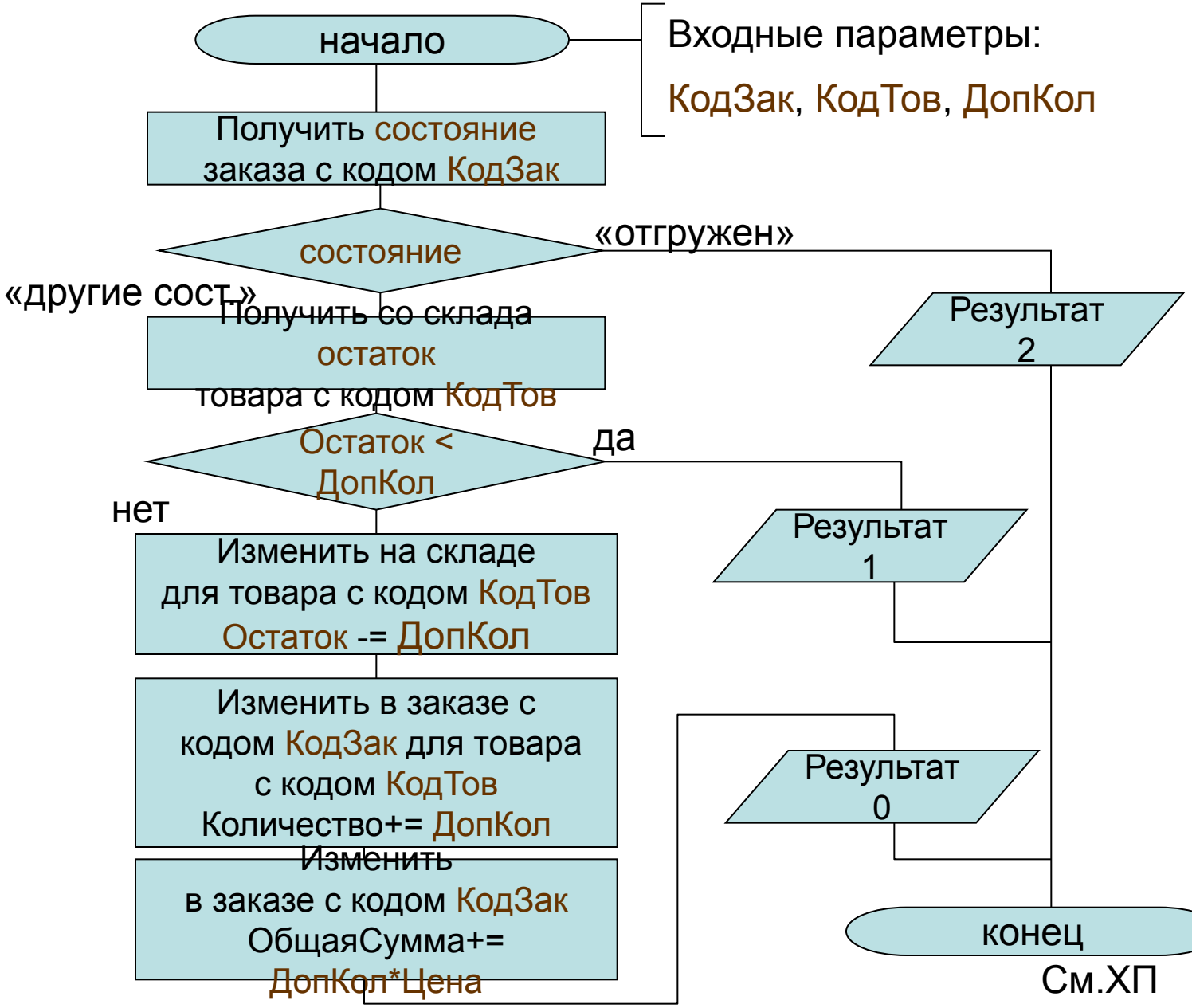
Программирование сервера БД

Программирование сервера БД



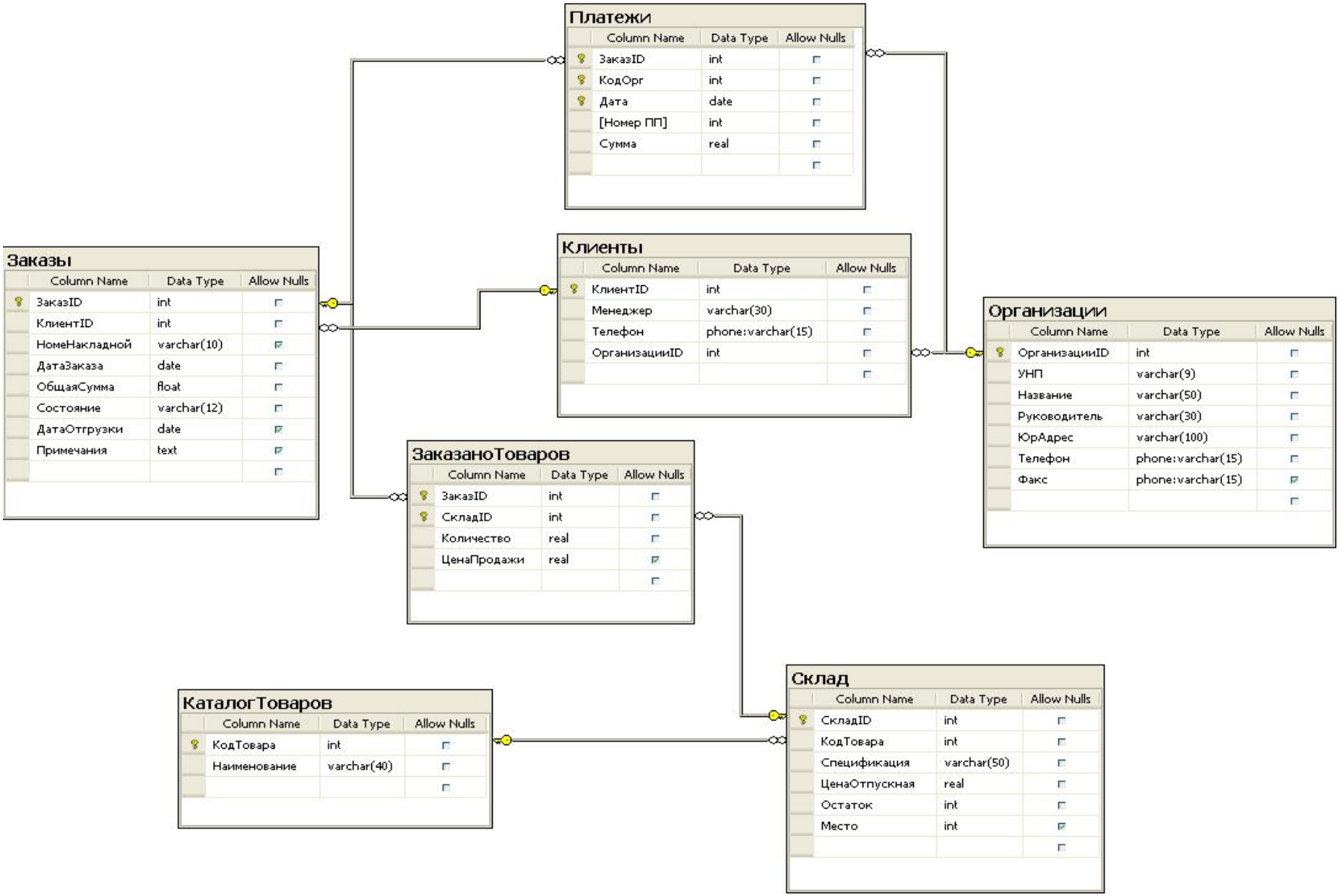
Пример функции бизнес - логики



См.схему БД

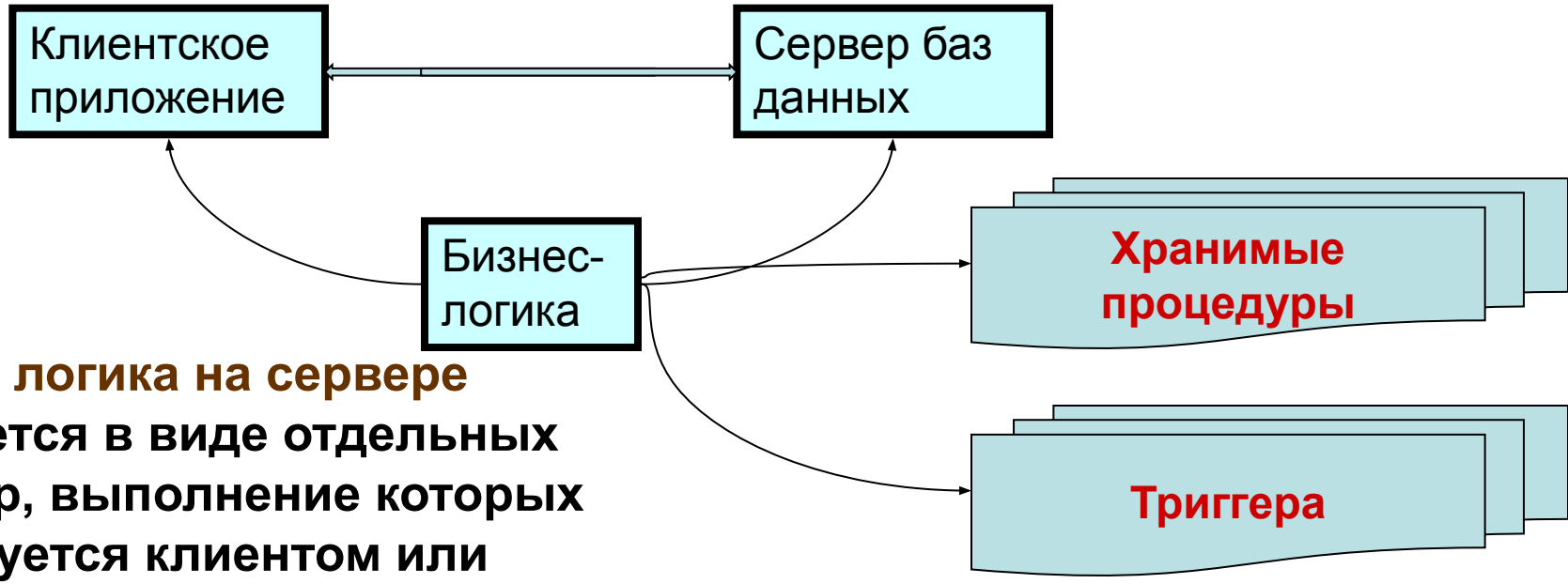
См.ХП

Схема БД «Заказы»



См.схему алгоритма

Реализация бизнес-логики на сервере БД



Бизнес – логика на сервере реализуется в виде отдельных процедур, выполнение которых инициируется клиентом или событиями, происходящими на сервере

Типы этих процедур соответственно
Хранимые процедуры
Триггера

Достоинства:

1. **Уменьшение нагрузки на сеть**
2. **Технологичность разработки программного обеспечения;**
3. **Высокий уровень защиты базы данных**

Язык программирования бизнес-логики сервера БД

Языком программирования бизнес логики серверов баз данных является расширение SQL: Transact-SQL для MS SQL Server, PL\SQL – для Oracle…

Элементы расширения языка SQL

Переменные, константы, типы

Операторы присваивания

Операторы управления вычислительным процессом

Операторы ввода-вывода

Элементы расширения языка T-SQL

Переменные - это дополнительный объект T-SQL, который описывают идентификаторами (как и объекты БД)

В T-SQL идентификаторы переменных начинаются с символа

@ - для локальной переменной

@@ - для глобальной переменной

Для объявления переменной используется оператор

```
DECLARE @name_local_var  type [, …]
```

Элементы расширения языка T-SQL

type - это те же типы, которые используются для описания столбцов таблиц, а также дополнительные типы, используемые только в программном коде:

table – тип таблица (операции такие же как и обычной таблицей)

cursor – тип виртуальной таблицы со структурой полей и данными, получаемыми запросом

Элементы расширения языка T-SQL

Преобразование типов выполняется неявно и явно, используя функции:

data_type - имя типа, в который нужно выполнить преобразование

convert (data_type [(length)], expression [, style])

expression - стиль, определяющий вид преобразования в символьный тип

cast (expression as data_type)

выражение, значение которого нужно преобразовать

Элементы расширения языка T-SQL

В T-SQL Часто используемые глобальные переменные

@@ERROR – содержит код ошибки последнего выполненного оператора SQL Server

@@IDENTITY – содержит значение, которое было последний раз помещено в столбец со свойством IDENTITY

@@ROWCOUNT – содержит значение числа строк, которое было обработано последним оператором SQL Server

@@SERVERNAME – содержит имя локального сервера

Элементы расширения языка T-SQL

Команды присваивания значений переменным

SET @name_local_var = <expression>

```
DECLARE @aa int,
@bb nvarchar(20)
SET @aa = 25
SET @bb = ‘База’
```

SELECT @name_local_var = <column | function>[,…] FROM …

Для присваивания результатов запроса

```
DECLARE @aa int
SELECT @aa = SUM(Цена) FROM Склад
```

Элементы расширения языка T-SQL

Команды управления вычислительным процессом управляют порядком выполнения инструкций на языке T-SQL, блоками инструкций, определяемыми пользователем функциями и хранимыми процедурами.

Блок объединяет нескольких инструкций языка T-SQL в логический блок

BEGIN < ***sql_statement*** > […] **END**

задаёт выполнение одной или другой (ELSE) инструкции или блока инструкций в зависимости от заданного условия

Условие

```
IF Boolean_expression
   { sql_statement | statement_block }
[ ELSE
   { sql_statement | statement_block } ]
```

```
IF ( SELECT Состояние  FROM Заказы WHERE КодЗаказа = @КодЗак) =
                                        “Не отгружен”
     SET @result = 2
ELSE
     SELECT @остаток = Остаток FROM СКЛАД WHERE КодСклада = КодТов
```

Элементы расширения языка T-SQL

Команды управления вычислительным процессом

Цикл

```
WHILE Boolean_expression
    { sql_statement | statement_block }
    [ BREAK ]
    { sql_statement | statement_block }
    [ CONTINUE ]
```

Пример. Увеличить цену всех товаров на складе с шагом 10% так, чтобы средняя цена всех товаров была больше 200$

```
WHILE (SELECT avg(Цена) FROM Склад WHERE Остаток > 0) < 200
BEGIN
    UPDATE Склад SET Цена = Цена*1.1 WHERE Остаток > 0
END
```

Элементы расширения языка T-SQL

Команды управления вычислительным процессом

Безусловный переход

GOTO *label*

…

***label* :**

```
GOTO do_update
SELECT * FROM Склад WHERE Остаток > 0
do_update:
    UPDATE Склад SET Цена = Цена*1.1 WHERE Остаток > 0
```

Элементы расширения языка T-SQL

Команды обработки ошибок

Блок TRY / CATCH

```
BEGIN TRY
 < SQL statement (s)  >
END TRY
BEGIN CATCH
< SQL statement (s) >
 END CATCH [;]
```

Начиная с версии SQL Server 2005

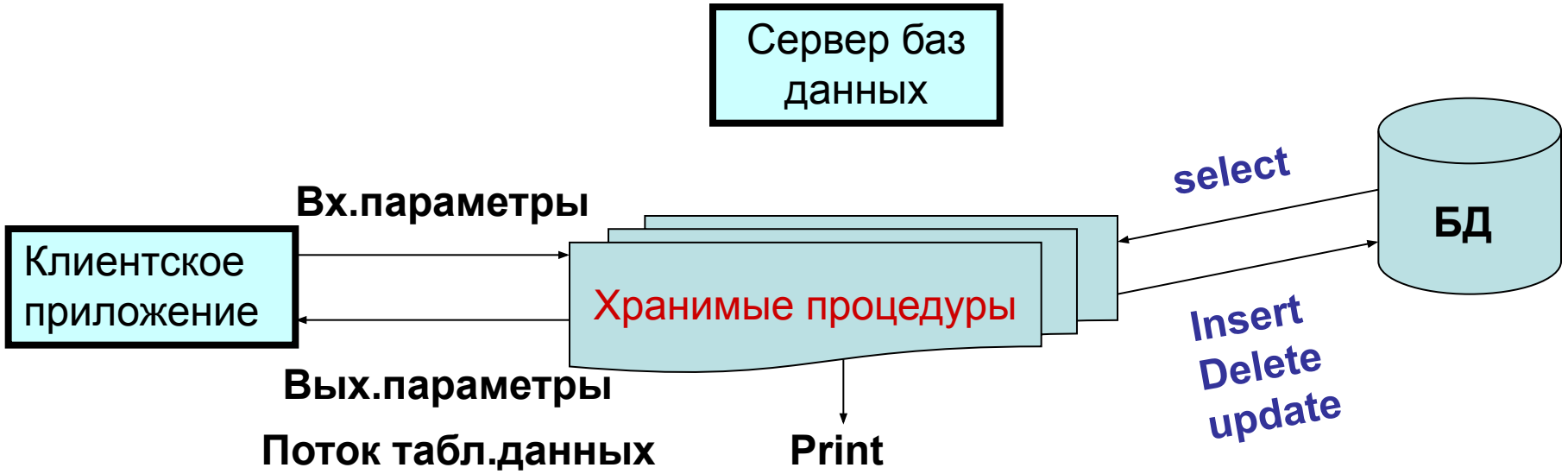
```
BEGIN TRY
   INSERT INTO …
END TRY
BEGIN CATCH
  DECLARE @ErrorNo int
  Set @ErrorNo = ERROR_NUMBER()
  if @ErrorNo = 547
      BEGIN
```

Функции для работы с ошибками:

ERROR_NUMBER()	– возвращает номер ошибки
ERROR_SEVERIRY()	– возвращает номер степени серьёзности ошибки
ERROR_MESSAGE()	– возвращает текст сообщения об ошибке
ERROR_LINE()	– возвращает номер строки, где возникла ошибка

Элементы расширения языка T-SQL

Команды ввода-вывода данных



Сообщения клиенту

Для отправки сообщений из ХП и Триггеров клиенту, используется оператор T-SQL RAISERROR:

RAISERROR (сообщение, уровень, состояние, параметр1, …)

Сообщение – это код или строка, содержащая символа формата подстановки параметров.

Полный формат

RAISERROR (…) [WITH {[SETERROR][,LOG,][NOWAIT]}]

SETERROR - регистрируется на сервере код ошибки независимо от уровня её серьёзности

LOG – запись сообщения в журнал ошибок и сообщений сервера

NOWAIT– отправка клиенту сообщения немедленно

Коды и сообщения всех ошибок находятся в таблице sys.messages системной базы данных master.

Номера 1- 50000 зарезервированы за системой

Ошибки SQL Server

```
SELECT * from sys.messages
```

Results | Messages

	message_id	language...	sever...	is_event_logg...	text
1794	4030	1033	10	0	The medium on device '%ls' expires on %hs and cannot be overwritten.
1795	4035	1033	10	0	Processed %I64d pages for database '%ls', file '%ls' on file %d.
1796	4037	1033	16	0	The user-specified MEDIANAME "%.*ls" does not match the MEDIANAME "%ls" of the device "%ls".
1797	4038	1033	16	0	Cannot find file ID %d on device '%ls'.
1798	4060	1033	11	0	Cannot open database "%.*ls" requested by the login. The login failed.
1799	4061	1033	11	0	Neither the database "%.*ls" requested by the login nor the user default database could be opened. Th...
1800	4062	1033	11	0	Cannot open user default database. Using master database instead.
1801	4063	1033	11	0	Cannot open database "%.*ls" that was requested by the login. Using the user default database "%.*ls" ...
1802	4064	1033	11	0	Cannot open user default database. Login failed.
1803	4065	1033	16	1	User is trying to use '%.*ls' through ODS, which is not supported any more.
1804	4066	1033	16	0	Type IDs larger than 65535 cannot be sent to clients shipped in SQL Server 2000 or earlier.
1805	4067	1033	16	0	CLR type serialization failed because an invalid cookie was specified.
1806	4068	1033	20	0	sp_resetconnection was sent as part of a remote procedure call (RPC) batch, but it was not the last RP...
1807	4069	1033	16	0	The final value of the output parameter was null, and could not be sent to a 6.5 client expecting the par...
1808	4070	1033	16	0	More than 255 columns were specified in the COMPUTE clause, and this metadata cannot be sent to a ...
1809	4071	1033	10	0	The XP callback function '%.*ls' failed in extended procedure '%.*ls' because it was executed within an l...
1810	4072	1033	10	0	The XP callback function '%.*ls' failed in extended procedure '%.*ls' because the extended procedure is...
1811	4073	1033	16	0	A return value of data type varchar(max), nvarchar(max), varbinary(max), XML or other large object typ...
1812	4074	1033	16	0	Client drivers do not accept result sets that have more than 65,535 columns.
1813	4075	1033	16	0	The USE database statement failed because the database collation %.*ls is not recognized by older cli...
1814	4076	1033	16	0	The ALTER DATABASE statement failed because the database collation %.*ls is not recognized by old...

Ошибки SQL Server

```
SELECT * from sys.messages where language_id = 1049
```

Results | Messages

	message_id	language_id	severity	is_event_logged	text
1785	4016	1049	16	0	Язык, запрашиваемый в попытке входа %1!", не является официальным именем языка в это...
1786	4017	1049	16	0	Ни язык, запрашиваемый в попытке входа "%1!", ни пользовательский язык по умолчанию ...
1787	4018	1049	16	0	Пользовательский язык по умолчанию %1! не является официальным именем языка в этом ...
1788	4019	1049	16	0	Язык, запрашиваемый в попытке входа "%1!", не является официальным именем языка в эт...
1789	4020	1049	16	0	Недопустимый порядок даты по умолчанию "%1!" для языка %2!. Будет использован форма...
1790	4021	1049	16	0	При сбросе соединения выполняется переход в состояние, отличное от исходного входа в с...
1791	4022	1049	16	0	Ожидалась массовая загрузка данных, но они не были отправлены. Выполнение пакета буд...
1792	4027	1049	16	0	Установите ленту %1! для базы данных "%2!" в накопитель на магнитной ленте "%3!".
1793	4028	1049	16	0	Конец ленты. Извлеките ленту "%1!" и установите следующую ленту для %2! базы данных "...
1794	4030	1049	10	0	Носитель на устройстве "%1!" был отключен в %2! и не может быть перезаписан.
1795	4035	1049	10	0	Обработано %1! страниц для базы данных "%2!", файл "%3!" для файла %4!.
1796	4037	1049	16	0	Указанное пользователем имя MEDIANAME "%1!" не совпадает с MEDIANAME "%2!" устройс...
1797	4038	1049	16	0	Не удалось найти идентификатор файла %1! на устройстве "%2!".
1798	4060	1049	11	0	Не удается открыть базу данных "%1!", запрашиваемую именем входа. Не удалось выполни...
1799	4061	1049	11	0	Невозможно открыть ни базу данных "%1!", запрашиваемую именем входа, ни пользователь...
1800	4062	1049	11	0	Невозможно открыть пользовательскую базу данных по умолчанию. Будет использована ба...
1801	4063	1049	11	0	Не удалось открыть базу данных "%1!", запрошенную именем входа. Будет использована по...
1802	4064	1049	11	0	Невозможно открыть пользовательскую базу данных по умолчанию. Не удалось выполнить ...
1803	4065	1049	16	1	Пользователь пытается использовать "%1!" через ODS, которая больше не поддерживается.
1804	4066	1049	16	0	Идентификаторы типов выше 65535 не могут быть переданы на клиенты, входящие в поста...
1805	4067	1049	16	0	Не удалось сериализовать тип CLR, так как был указан недопустимый файл cookie.
1806	4068	1049	20	0	Процедура sp_resetconnection была отправлена как часть пакета удаленного вызова проце...

Пользовательские ошибки SQL Server

Для добавления пользовательских ошибок используется системная ХП
sp_addmessage

Номера зарезервированные за пользователями от 50001 и далее

sp_addmessage
[@msgnum =] *msg_id* , — ***Код ошибки***
[@severity =] *severity* , — **Уровень серьёзности ошибки**
[@msgtext =] '*msg*' — **Текст сообщения**
[, [@lang =] '*language*'] — **Язык сообщения**
[, [@with_log =] '*with_log*'] — **Регистрация в Log Windows NT**
[, [@replace =] '*replace*'] — **Признак замены существующего сообщения или уровня серьёзности ошибки**

Например, добавление сообщения
сначала на английском
sp_addmessage 60001, 11, ‘Error code organization: %d. ' , 'us_english'

потом на русском
sp_addmessage 60001, 11, 'Отсутствует код организации: %1!. ' , 'Russian'

Использование: RAISERROR (60001, 11, @OprID)

Выполнение замены
sp_addmessage 60001, 12, 'Отсутствует код организации: %1! в таблице ‘Организации’ ', NULL, FALSE, REPLACE

Пользовательские ошибки SQL Server

Для удаления пользовательских ошибок используется системная ХП

sp_dropmessage [@msgnum =] *message_number*
[, [@lang =] '*language*']

Хранимые процедуры

ХП- это объект SQL Server, представленный набором откомпилированных операторов T-SQL.

Системные ХП- это ХП, поставляемые SQL Server для выполнения действий по администрированию базы данных или сервера.

Пользовательские ХП - это ХП, разработанные пользователем SQL Server, для конкретной БД.

Хранимые процедуры

При создании ХП выполняется действия

1. Лексический анализатор разбивает процедуру на отдельные компоненты

2. Проверяется существование объектов в БД (возможно отложенное существование объектов)

3. В системную таблицу sysobject заносится имя ХП, а в syscomments - её исходный текст

4. Создается предварительный план выполнения запросов (нормализованный план или дерево запроса) и сохраняется в системную таблицу sysprocedure

При выполнении ХП в первый раз

1. Дерево запросов ХП считывается из sysprocedure и окончательно оптимизируется и сохраняется в КЭШ

2. ХП считывается из КЭШ и выполняется

При выполнении ХП в другой раз

1. ХП выполняется из КЭШ

Хранимые процедуры

Создание ХП

CREATE PROC [EDURE] *procedure_name*
[{ **@***parameter* *data_type* } [**VARYING**] [**=** *default*] [**OUT**[**PUT**]]] [**,**...*n*]

[**WITH**
{ **RECOMPILE** | **ENCRYPTION** | **RECOMPILE** , **ENCRYPTION** }]
[**FOR REPLICATION**]

AS

sql_statement **[...*n*]**

Прекращение выполнения кода и возвращение кода выполнения ХП

RETURN [*code_return (int)*]

RETURN *надо использовать для возврата кода выполнения процедуры, который должен анализироваться в клиентском приложении*

Параметры

RECOMPILE – запрещает сохранение плана выполнения ХП В КЭШ

ENCRYPTION– определяет шифрование исходного кода ХП

FOR REPLICATION – может выполняется только при репликациях

Хранимые процедуры

Изменение ХП

ALTER PROC [EDURE] *procedure_name*
[{ **@***parameter* *data_type* } [**VARYING**] [**=** *default*] [**OUT**[**PUT**]]] [**,**...*n*]

[**WITH**
{ **RECOMPILE** | **ENCRYPTION** | **RECOMPILE** , **ENCRYPTION** }]
[**FOR REPLICATION**]

AS

sql_statement **[...*n*]**

Если для модификации процедуры использовать последовательно команды DROP PROC и CREATE PROC вместо ALTER PROC, то достигается тот же эффект, но придется определять пользователям заново все права на эту процедуру

Удаление ХП

DROP PROC [EDURE] *procedure_name*

Хранимые процедуры

Вызов ХП

```
[ EXEC [ UTE ] ]
   {
      [ @return_status = ]
         { procedure_name | @procedure_name_var
   }
   [ [ @parameter = ] { value | @variable [ OUTPUT ] | [ DEFAULT ] ]
      [ ,...n ]
[ WITH RECOMPILE ]
```

Хранимые процедуры

Входные параметры ХП

```
CREATE PROC ВхПарам
   @Имя VARCHAR(30),
   @Всего INT,
   @ТекДата DATATIME
AS ….
```

Передача параметров в виде константы в порядке описания

```
EXEC ВхПарам ‘Иванов’ , 1000, “03/25/2008”
```

Передача параметров в виде переменных в порядке описания

```
EXEC ВхПарам @ТекИмя , @Сумма, @Дата
```

Передача параметров с использованием их описаний в любой последовательности

```
EXEC ВхПарам @Всего = @Сумма, @ТекДата =@Дата, @Имя = @ТекИмя
```

Хранимые процедуры

Входные параметры ХП со значениями по умолчанию

```
CREATE PROC ВхПарам
   @Имя VARCHAR(30),
   @Всего INT=1000,
   @ТекДата DATATIME=GETDATE()
AS ….
```

Передача параметров в виде константы в порядке описания

```
EXEC ВхПарам ‘Иванов’
```

Хранимые процедуры

Выходные параметры ХП

```
CREATE PROC ВыхПарам
   @КодЗаказа INT,
   @Результат INT OUT
AS
...
@Результат = 2
```

Получение результата выходного параметра в QE

```
DECLARE @КодВыполнения INT
EXEC ВыхПарам 1000, @КодВыполнения OUT
PRINT STR(@КодВыполнения)
```

ХП добавления товара в заказ

```
CREATE PROC ДобавитьЗаказКолТовар

@КодЗак  INT, @КодТов INT, @ДопКол INT
AS
DECLARE @Состояние VARCHAR(10), @Остаток INT,  @Цена MONEY
SELECT @Состояние = Состояние FROM Заказы WHERE ЗаказID = @КодЗак

IF @Состояние IS NOT NULL  AND @Состояние <> ‘отгружен’
  BEGIN
     SELECT @Остаток = Остаток, @Цена= ЦенаОтпускная FROM Склад
                    WHERE СкладID = @КодТов
     IF @Остаток >= @ДопКол
       BEGIN

          UPDATE  Склад  SET Остаток = Остаток - @ ДопКол WHERE СкладID = @КодТов
          UPDATE  ЗаказаноТоваров  SET Количество = Количество + @ДопКол
                    WHERE ЗаказID = @КодЗак AND СкладID = @КодТов
         UPDATE  Заказы  SET ОбщаяСумма = ОбщаяСумма + @ДопКол* @Цена
                    WHERE ЗаказID= @КодЗак

           RETURN 0
       END
     ELSE
       RETURN 1
ELSE
       RETURN 2
```

См.схему алгоритма

Пример 2 ХП

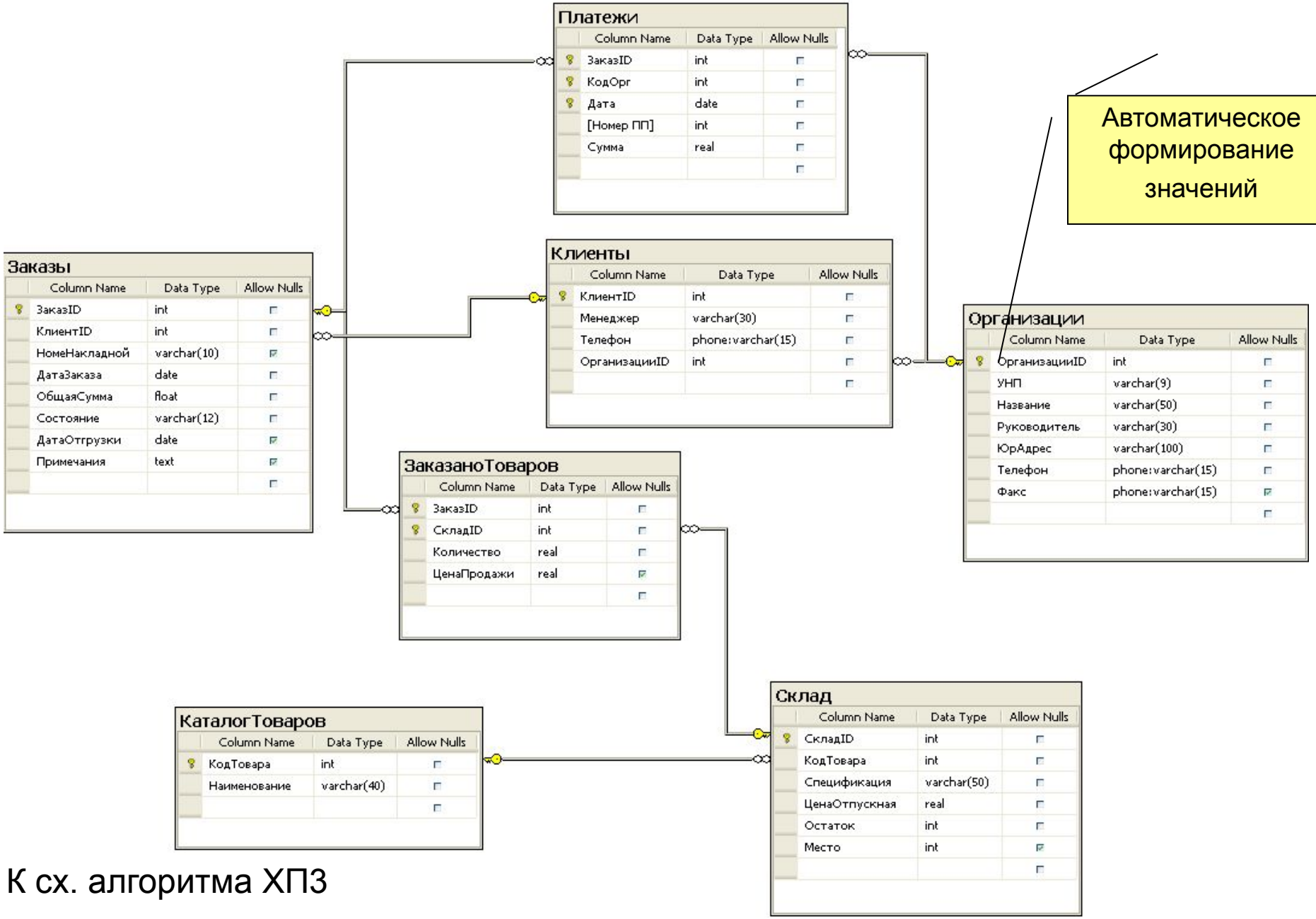
Пример 2 бизнес-логики: добавление нового клиента

Алгоритм:

необходимые данные разнести по соответствующим таблицам

См.схему БД

Схема БД «Заказы»



К сх. алгоритма ХПЗ

ХП добавления нового клиента

```
CREATE  PROCEDURE  НовыйКлиент
@УНП varchar(9),
@Наименование varchar(50),
@Руководитель varchar(30),
@ЮрАдрес varchar(100),
@Телефон phone,
@Факс phone = NULL,
@Менеджер varchar(30),
@МТелефон phone = NULL
AS
  DECLARE @ОрганизацияID int

select @ОрганизацияID=ОрганизацииID from Организации
where Название=@НазваниеОрганизации

if @ОрганизацияID = NULL
begin

  INSERT INTO Организации  (УНП, Название,  Руководитель,  ЮрАдрес,  Телефон,  Факс)
  VALUES (@УНП, @Наименование, @Руководитель, @ЮрАдрес, @Телефон, @Факс)
  SET @ОрганизацияID = IDENT_CURRENT('ОрганизацииID')
end

INSERT INTO Клиенты (Менеджер,  Телефон, ОрганизацияID)
VALUES (@Менеджер,  @МТелефон, @ОрганизацияID)
```

Пример 3 ХП

Пример 3 бизнес-логики: аннулирование заказа

Алгоритм:

см. схему алгоритма

Схема алгоритма ХП аннулирования заказа

См.схему БД

ХП аннулирования заказа

```
CREATE  proc АннулированиеЗаказа
@КодЗаказа int
AS
if exists (select * from Заказы where ЗаказID=@КодЗаказа
                                        and Состояние = 'оформление')
begin


-- Возврат кол.товаров в табл. «Склад"
    Update Склад set Остаток = Остаток + Количество
    from ЗаказаноТоваров
    where ЗаказаноТоваров.ЗаказID = @КодЗаказа and
       ЗаказаноТоваров.СкладID = Склад.СкладID
-- Удаление заказанных товаров из табл. "ЗаказаноТоваров" для данного заказа
    delete from ЗаказаноТоваров  where ЗаказID=@КодЗаказа
-- Удаление заказа из табл. "Заказы"
    delete from Заказы where ЗаказID=@КодЗаказа


end
```

Триггера

Триггер - это специальный тип ХП, которая выполняется при наступлении события по изменению данных в таблицах.

Область применения триггеров

1. Обеспечение нестандартной целостности ссылок, поддержание которых обычными средствами SQL Server невозможно.

2. Каскадные изменения в нескольких связанных таблицах.

Не следует применять триггеры – для простых проверок, которые могут быть выполнены с помощью правил или ограничений целостности.

При использовании триггеров – удерживается блокировка на используемые им ресурсы до завершения работы триггера, запрещая обращение к этим ресурсам других пользователей.

События триггеров

Триггеры в SQL Server 2008 могут создаваться на события

модификации данных (DML-триггеры)

модификации модели данных (DDL-триггеры)

Типы и виды DML-триггеров

Типы триггеров

- **INSERT** — **Запускаются при попытке вставки данных**
- **DELETE** — **Запускаются при попытке удаления данных**
- **UPDATE** — **Запускаются при попытке изменения данных**

Виды триггеров

- **AFTER** — **Триггер выполняется после выполнения операторов изменения данных. Если команда не может быть завершена, то и триггер не выполнится!**
- **INSTEAD OF** — **Триггер выполняется вместо выполнения операторов изменения данных. Они могут быть определены и для представлений.**

Создание триггеров

CREATE TRIGGER ***trigger_name***
ON { *table* | *view* }
[WITH ENCRYPTION]
{ FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] [,] [DELETE] }
A

sql_statement **- внутри триггера создаются 2 специальные таблицы inserted и deleted.**

Их структура идентична структуре таблиц, для которой создаётся триггер.

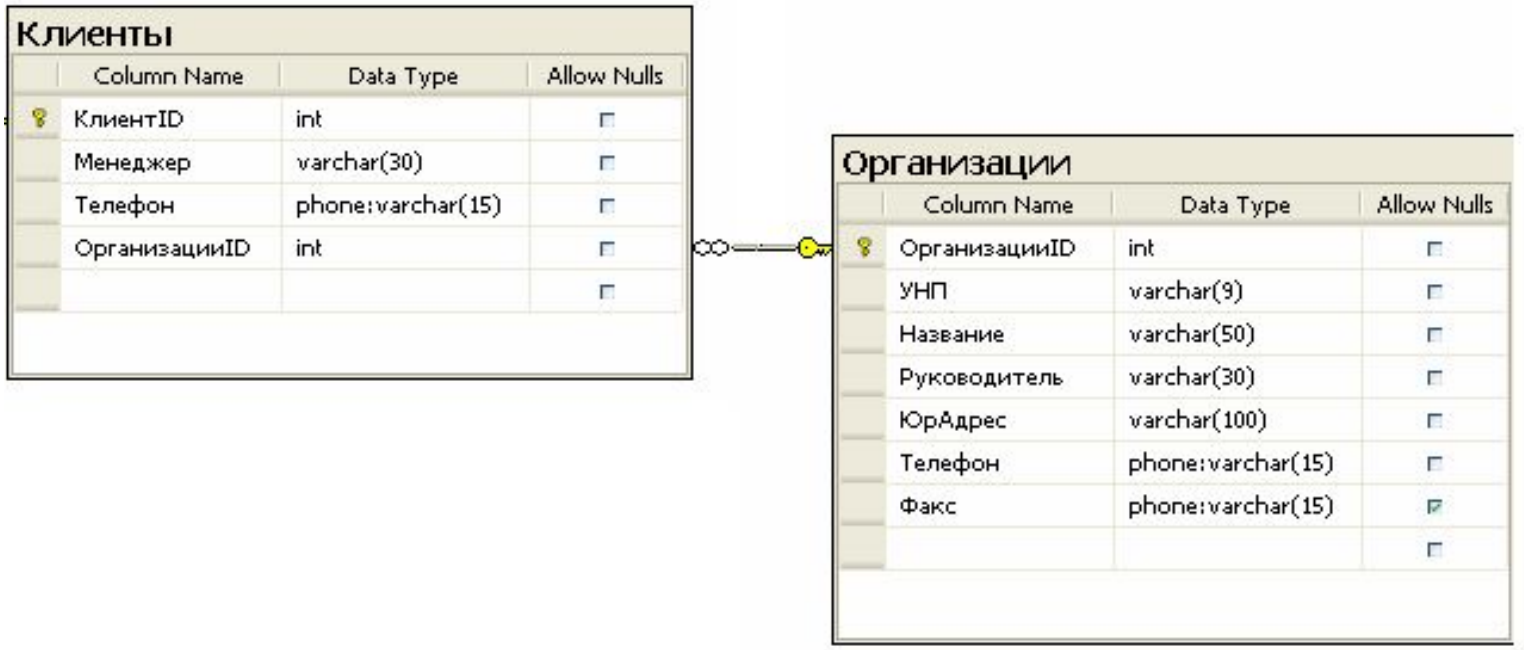
Для каждого триггера создается свой комплект **inserted** и **deleted**, поэтому никакой другой триггер не сможет получить к ним доступ.

Содержимое таблиц **inserted** и **deleted** при выполнении:
команды **INSERT** – в таблице **inserted** содержатся все строки, которые вставляются в таблицу; в таблице **deleted** - нет строк;
команда **DELETE** – в таблице **deleted** будут содержаться все строки, которые пользователь попытается удалить; в таблице **inserted** нет строк;
команда **UPDATE** –в таблице **deleted** находятся старые значения строк; в таблице **inserted** - новые значения строк.

Пример триггера

Проверить наличие организации при добавлении нового клиента-менеджера

Фрагмент схемы БД «Заказы»



К тексту Т2

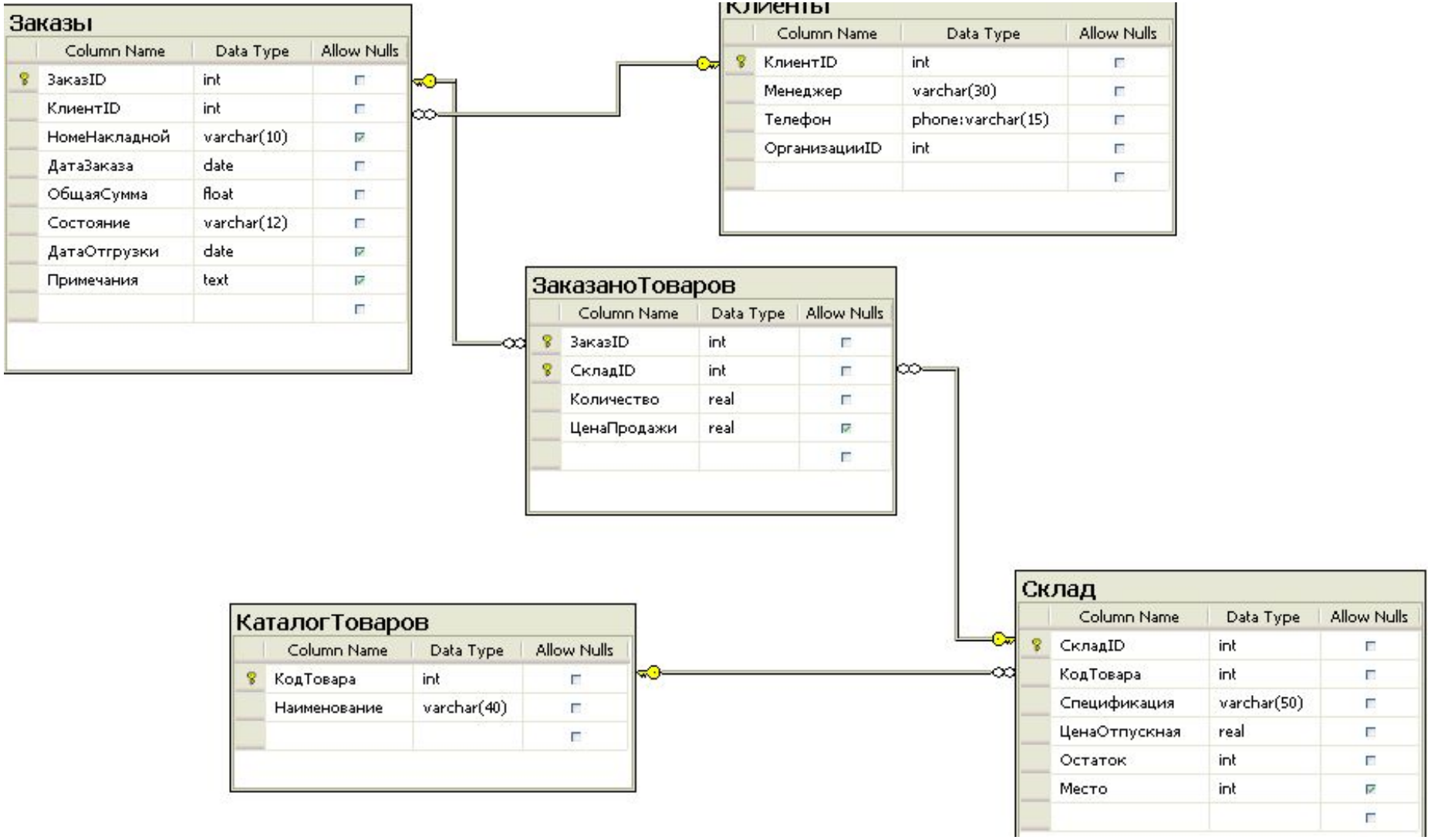
Пример триггера

```
CREATE  TRIGGER Add_Клиенты
ON Клиенты
FOR INSERT
AS
PRINT 'Выполнение триггера';
DECLARE @КлиентID int, @ОрганизID int

SELECT @КлиентID=ОрганизацииID FROM INSERTED

SELECT @ОрганизID=ОрганизацииID FROM Организации
WHERE ОрганизацииID=@КлиентID

IF  @ОрганизID IS NULL
    BEGIN
      PRINT 'нет организации'
      -- отменить вставку записи
          Delete from Клиенты where ОрганизацииID= @КлиентID
END
ELSE
      BEGIN
            PRINT 'Клиент вставлен'
      END
```

Пример 2 триггера

Обеспечить логику первичного ключа таблицы «ЗаказаноТоваров» при добавлении товара в заказ

Фрагмент схемы БД «Заказы»



К тексту Т2

Триггер на вставку

```
CREATE  TRIGGER Add_ЗаказТовар  ON  ЗаказаноТоваров
INSTEAD OF INSERT
AS
DECLARE @НовыйЗаказ int, @НовыйТовар int,  @Кол_во real
DECLARE @ЦенаПродажи real

SELECT @НовыйЗаказ = ЗаказID, @НовыйТовар = СкладID,
          @Кол_во =  Количество, @ЦенаПродажи = ЦенаПродажи
     FROM INSERTED

IF EXISTS (SELECT * FROM ЗаказаноТоваров
          WHERE ЗаказID=@НовыйЗаказ AND СкладID=@НовыйТовар)

   UPDATE ЗаказаноТоваров SET Количество=Количество+@Кол_во
             WHERE ЗаказID=@НовыйЗаказ AND СкладID = @НовыйТовар
ELSE
    INSERT INTO ЗаказаноТоваров
     VALUES (@НовыйЗаказ, @НовыйТовар, @Кол_во, @ЦенаПродажи )
```

DDL-триггеры

Типы триггеров на события (event_type)

- ALTER_<object>
- CREATE_<object>
- DROP_<object>
- DENY_DATABASE
- GRANT_DATABASE
- REVOKE_DATABASE

Например, alter_index, alter_table …

Например, create_index, create_table …

Например, drop_index, drop_table …

здесь <object> - имя объекта базы данных или сервера

```
CREATE TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH ENCRYPTION ]
  {{ FOR | AFTER } event_type, … }
  AS
        sql_statement [ ...n ]
```

Функции пользователя

Функции - это именованная часть бизнес – логики, реализованной и используемой только на сервере базы данных.

Функции могут быть

1. Системными (встроенными) – встроены в язык программирования.

2. Пользовательскими – создаваемые пользователями базы данных.

Пользовательские функции не доступны для клиентских приложений.

Они могут использоваться только в ХП и триггерах или в других пользовательских функциях.

Пользовательские функции не должны изменять внешние источники данных (таблицы) и не должны выполнять системные функции, изменяющие внешние источники.

Типы функций пользователя

Скалярные

- возвращают скалярные значения любого типа данных (исключая, timestamp, text, ntext, image, table, cursor)

Однострочные

- содержат одну команду – SELECT, возвращающей набор данных типа table.

Многострочные

- содержат много команд и возвращают набор данных типа table.

Тип table

Table – это тип для описания виртуальных таблиц (т.е. таблиц в ОП)

Формат описания типа

DECLARE @local_var TABLE *имя_таблицы*

(<*описание_элемента_таблицы*>[,…])

где элемент_таблицы тоже, что и в в операторе создания таблицы:

1) столбец,

2) ограничение целостности таблицы:

а) первичный ключ Primary key …

б) вторичный ключ Foreign key…

в) условие уникальности Unique …

г) условие проверки границ Check

Тип table

Пример создания переменной типа таблицы *КЛИЕНТЫ*

```
DECLARE @КЛИЕНТЫ TABLE
(Код integer not null Primary key,
Фирма varchar(40) not null,
КодМен integer not null,
МинКредит money default 10000 not null,
Check(МинКредит >=5000)
```

Описание функций пользователя

Описание скалярной функции

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
RETURNS scalar_return_data_type
[ WITH [ENCRYPTION] [,] [SCHEMABINDING ] ]
[ AS ]
BEGIN
   function_body
   RETURN scalar_expression
END
```

Возвращает значение выражения этого типа

Запрещает изменение исходного кода функции

Сохранение кода функции в шифрованном виде

Описание функций пользователя

Пример скалярной функции, возвращающей последний день месяца

```
CREATE  FUNCTION ПоследнийДеньМесяца (@текДата Datetime)
RETURNS Datetime
AS
BEGIN
   DECLARE @мес int, @год int, @прДата Datetime, @стрДаты varchar(10)
   Set @мес = datepart (Month, @текДата )
   Set @год = datepart (Year, @текДата )
   If @мес = 12
     Begin
       Set @мес = 1
       Set @год = @год +1
     End
  Else
     Set @мес = @мес +1
  Select @стрДаты=convert (varchar(2), @мес )+'01'+convert(varchar(4), @год )
  Set @прДата = convert (Datetime, @стрДаты )
  Set @прДата = dateadd (Day, -1, @прДата )
  RETURN @прДата
END
```

Описание функций пользователя

Описание однострочной функции

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
RETURNS TABLE
[ WITH [ENCRYPTION] [,] [SCHEMABINDING ] ]
[ AS ]
   RETURN select_operator
```

Структура таблицы определяется по полям оператора SELECT

Описание функций пользователя

Пример однострочной функции, возвращающей таблицу заказанных товаров по заказу с заданным кодом

```
CREATE  FUNCTION ЗаказаноТоваровВЗаказе (@ЗаказID)
RETURNS TABLE
AS
RETURN Select Название, Количество, Сумма
            FROM Товары INNER JOIN ЗаказаноТоваров A ON
                        Товары.ТоварID = A.ТоварID.
       WHERE A.ЗаказID = @ЗаказID
```

Использование в другой ХП для отправки клиенту набора записей

```
…
SELECT * FROM ЗаказаноТоваровВЗаказе (300)
          ODER BY Название
```

Описание функций пользователя

Описание многострочной функции

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ = default ] } [ ,...n ] ] )
RETURNS @return_var  TABLE <table_type_definition>
[ WITH [ENCRYPTION] [,] [SCHEMABINDING ] ]
[ AS ]
   BEGIN
    function_body
    RETURN
END
```

Возвращает значение этой переменной

Описание функций пользователя

Пример многострочной функции, возвращающей таблицу слов, из которых состоит входная строка

```
CREATE  FUNCTION ПолучитьТаблицуСлов (@Строка nvarchar(500))
RETURNS @СтрокаСлов TABLE ( Номер int IDENTITY (1,1) NOT NULL,
AS                                                  Слова nvarchar(30) )
BEGIN
   DECLARE @стр nvarchar(500), @поз int
   Set @стр = @Строка
   WHILE 1>0
     Begin
       Set @поз = Charindex (“ “, @стр)
       if @поз > 0
          Begin
             INSERT INTO @СтрокаСлов VALUES (substring (@стр, 1, @поз))
             Set @стр = substring (@стр, @поз +1, 500)
         End
        Else
         Begin
             INSERT INTO @СтрокаСлов VALUES ( @стр)
             BREAK
         End
     End
 RETURN
END
```