Тест №1 Рубежный

Дайте определение следующим понятиям

**1. Определение реляционной базы данных.**

Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном, а столбцы таблицы содержат различные характеристики этих объектов - атрибуты.

Строки таблицы называются записями. Все записи таблицы имеют одинаковую структуру - они состоят из полей (элементов данных), в которых хранятся атрибуты объекта. Каждое поле записи содержит одну характеристику объекта и представляет собой заданный тип данных. Для идентификации записей используется первичный ключ. Первичным ключом называется набор полей таблицы, комбинация значений которых однозначно определяет каждую запись в таблице.

**2. Среды разработки баз данных.**

Workbench, Navicat, PHPMyAdmin, dbForge Studio for MySQL, HeidiSQL, SQL Maestro для MySQL, EMS SQL Manager для MySQL, SQLyog, DBTools Manager, MyDB Studio.

**3. Основные правила создания SQL инструкций.**

Инструкции содержат описание набора данных на языке SQL. Инструкции SQL состоят из предложений (SELECT, FROM, WHERE и т.д.). Предложения на языке SQL состоят из терминов (операторов или команд, идентификаторов, констант и т.д.). Инструкция начинается оператором (одной из команд SELECT, CREATE, INSERT, UPDATE, DELETE и т.д. ) и заканчивается точкой с запятой. Основные операторы SQL: SELECT, FROM и WHERE.

**4. Арифметические операторы SQL.**

Арифметические операторы выполняют математические операции над двумя выражениями одного или различных типов данных из категории числовых типов данных

+ (сложение), - (вычитание), \* (умножение), / (Деление), % (Остаток от деления)

**5. Псевдонимы столбцов.**

Псевдоним столбца:

- переименовывает заголовок столбца

- полезен с вычислениями

- следует сразу за именем столбца (Может также быть необязательное ключевое слово AS между именем столбца и псевдонимом.)

-требует двойных кавычек, если он содержит пробелы или специальные символы, или если он чувствителен к регистру

**6. Оператор конкатенации строк.**

Операторы конкатенации (слияния строковых значений) в запросах Oracle SQL, слияние строковых данных разных типов

В реальной работе очень часто приходится производить слияние строковых значений. Такая операция называется конкатенацией.

В Oracle SQL предусмотрен оператор конкатенации — две вертикальные черты (||). Отметим, что в некоторых версиях сервера Oracle, например, предназначенных для работы на мэйнфреймах IBM, используется другой оператор конкатенации. Поэтому на всякий случай в Oracle SQL предусмотрена еще и встроенная функция CONCAT, которая также производит конкатенацию и работает на любых платформах.

Пример применения оператора конкатенации может выглядеть так:

SELECT first\_name || ' ' || last\_name FROM hr.employees;

В этом примере у нас сливается три значения: имя, пробел (в виде литерала) и фамилия.

Оператор конкатенации можно применять как для строковых значений и значений CLOB (больших строковых значений), так и для чисел и дат (эти значения будут автоматически конвертированы в строковые). При слиянии строкового значения со значением типа NULL Oracle вернет строковое значение (в отличие от поведения по умолчанию SQL Server), однако Oracle предупреждает, что такое поведение в будущих версиях Oracle не гарантируется. Поэтому в таких ситуациях рекомендуется использовать функцию NVL, которая преобразует значения типа NULL в строки нулевой длины.

При выполнении конкатенации рекомендуется подумать над ограничениями типов данных, которые используются для конкатенации. Oracle использует для результата конкатенации больший тип данных из тех значений, которые используются для конкатенации. Например, при слиянии двух значений с типом данных CHAR (максимальный размер — 2000 символов) вернется CHAR, при слиянии CHAR и VARCHAR2 (максимальный размер — 4000 символов) вернется VARCHAR2, а при слиянии любого типа данных с CLOB вернется CLOB.

Пробелы при конкатенации всегда сохраняются.

**7. Приоритет операторов.**

Если сложное выражение содержит несколько операторов, порядок выполнения этих операторов определяется их приоритетом. Порядок исполнения может существенно повлиять на результирующее значение.

Уровни приоритета операторов показаны в следующей таблице. Оператор с более высоким уровнем выполняется прежде, чем оператор с более низким уровнем.

Level Операторы

1 ~ (побитовое НЕ)

2 \* (умножение), / (деление), % (остаток от деления)

3 + (Положительное), - (отрицательное), + (сложение), (+ объединение), - (вычитание), & (побитовое и), ^ (побитовое исключающее или), | (Побитовый оператор или)

4 =, >, <, > =, < = <>,! =,! >,! < (операторы сравнения)

5 NOT

6 и

7 ALL, ANY, BETWEEN, IN, LIKE, OR, SOME

8 = (присваивание)

Если два оператора в выражении имеют один и тот же уровень старшинства, они выполняются в порядке слева направо по мере их появления в выражении.

**8. Классификация одностроковых функций и их назначение.**

Существует несколько типов однострочных функций.

Символьные функции: Принимают на входе символы и могут возвратить, как символьное, так и числовое значение

Числовые функции: Принимают на входе число и возвращают числовые значения

Функции даты: Работают на значениях типа данных DATE (Все функции даты возвращают значение типа данных DATE, кроме функции MONTHS\_BETWEEN, которая возвращает число.)

Функции преобразования: Преобразуют значение из одного типа данных в другой

Общие функции:

-NVL

-NVL2

-NULLIF

-COALESCE

-CASE

-DECODE

**9. Символьные функции. Классификация.**

Символьные однострочные функции принимают символьные данные на входе и могут возвратить как символьные, так и числовые значения.

Символьные функции могут быть разделены на следующие:

Функции замены регистра: UPPER, LOWER, INITCAP

Функции символьного манипулирования:

CONCAT, SUBSTRING, LENGTH, INSTR, LPAD/RPAD, TRIM, REPLACE

Некоторые из функций, которые являются полностью или частично совместимыми с SQL:2003:

-UPPER

-LOWER

-TRIM

-LENGTH

-SUBSTR

-INSTR

**10. Числовые функции.**

В Oracle предусмотрен ряд встроенных функций для работы с числами.

**Функция ABS(n)**

Функция ABS возвращает абсолютное значение числа. Например:

**SELECT ABS(100) XI, ABS(-100) Х2, ABS(-100.2) Х3   
FROM DUAL**

|  |  |  |
| --- | --- | --- |
| X1 | Х2 | ХЗ |
| 100 | 100 | 100,2 |

**Функция CEIL (n)**

Функция CEIL возвращает наименьшее целое, большее или равное пе­реданному в качестве параметра числу n. Например:

**SELECT CEIL(100) X1, CEIL(-IOO) Х2, CEIL(100.2) Х3, CEIL(-100.2) Х4 FROM DUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| XI | X2 | X3 | X4 |
| 100 | -100 | 101 | 100 |

**Функция FLOOR(n)**

Функция FLOOR возвращает наибольшее целое, меньшее или равное переданному в качестве параметра числу n. Например:

**SELECT FL00R(100.22) X1, FL00R(-100.22) Х2, FL00R(100.99) ХЗ, FL00R(100.01) Х4   
FROM DUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | ХЗ | X4 |
| 100 | -101 | 100 | 100 |

**Функция TRUNC(n [,m])**

Функция TRUNC возвращает число n, усеченное до m знаков после десятичной точки. Параметр m может не указываться — в этом случае n усекается до целого.

**SELECT TRUNC(100.25678) X1, TRUNC(-100.25678) Х2, TRUNC(100.99) ХЗ, TRUNC(100.25678, 2) Х4   
FROM DUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X3 | X4 |
| 100 | -100 | 100 | 100.25 |

**Функция ROUND(n [,m])**

Функция ROUND возвращает число n, округленное до m знаков после десятичной точки по правилам математического округления. Параметр m может не указываться — в этом случае n округляется до целого.

**SELECT ROUND(100.25678) X1, ROUND(100.5) Х2, ROUND(100.99) ХЗ, ROUND(100.25678, 2) Х4   
FROM DUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | X2 | X3 | X4 |
| 100 | 101 | 101 | 100.26 |

**Функция SIGN(n)**

Функция SIGN определяет знак числа. Если n положительное, то функ­ция возвращает 1. Если отрицательное — возвращается -1. Если равно нулю, то возвращается 0. Например:

**SELECT SIGN(100.22) X1, SIGN(-100.22) Х2, SIGN(O) Х3**  
**FROM DUAL**

|  |  |  |
| --- | --- | --- |
| X1 | Х2 | ХЗ |
| 1 | -1 | 0 |

**Функция MOD(n, m)**

Функция MOD возвращает от деления n на m. Например:

**SELECT MOD(10, 3) X1, MOD(10, 2) X2, MOD(100, 0) X3   
FROM DUAL**

|  |  |  |
| --- | --- | --- |
| X1 | Х2 | ХЗ |
| 1 | 0 | 100 |

Интересной особенностью данной функции является возможность пе­редачи m равного нулю — при этом не возникает ошибки деления на 0.

**Функция POWER(n, m)**

Функция POWER возводит число n в степень m. Степень может быть дробной и отрицательной, что существенно расширяет возможности дан­ной функции.

**SELECT POWER(10, 2) X1, P0WER(100, 1/2) Х2,   
P0WER(1000, 1/3) Х3, P0WER(1000, -1/3) Х4   
FROM DUAL**

|  |  |  |  |
| --- | --- | --- | --- |
| X1 | Х2 | ХЗ | Х4 |
| 100 | 10 | 10 | 0.1 |

В некоторых случаях при вызове данной функции может возникнуть исключительная ситуация. Например:

**SELECT POWER(-100, 1/2) Х2   
FROM DUAL**

В данном случае производится попытка вычисления квадратного корня от отрицательного числа, что приведет к возникновению ошибки ORA- 01428 «Аргумент вне диапазона».

**Функция SQRT(n)**

Данная функция возвращает квадратный корень от числа n  Например:

**SELECT SQRT(1OO) X   
FROM DUAL**

**Функции EXP(n) и LN(n)**

Функция EXP возводит e в степень n, а функция LN вычисляет нату­ральный [логарифм](http://edu.sernam.ru/book_el_math.php?id=27) от n (при этом n должно быть больше нуля). Пример:

**SELECT EXP(2) X1, LN(1) Х2, LN(EXP(2)) Х3 FROM DUAL**

|  |  |  |
| --- | --- | --- |
| X1 | X2 | Х3 |
| 2,71828182845905 | 0 | 2 |

Попытка передать функции LN отрицательное значение приводит к возникновению ошибки ORA-1428.

**Функция LOG (n, m)**

Функция LOG производит вычисление логарифма m по основанию n. Пример:

**SELECT L0G(2, 8) XI, L0G(10, 100) Х2   
FROM DUAL**

|  |  |
| --- | --- |
| X1 | X2 |
| 3 | 2 |

**11. Функции управления датами.**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [SYSDATETIME](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sysdatetime-transact-sql) | SYSDATETIME () | Возвращает datetime2(7)значение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Смещение часового пояса не включается. | datetime2(7) | Недетерминированная |
| [SYSDATETIMEOFFSET](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sysdatetimeoffset-transact-sql) | SYSDATETIMEOFFSET ( ) | Возвращает datetimeoffset(7)значение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Смещение часового пояса включается. | DateTimeOffset(7) | Недетерминированная |
| [SYSUTCDATETIME](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/sysutcdatetime-transact-sql) | SYSUTCDATETIME ( ) | Возвращает datetime2(7)значение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Дата и время возвращаются в формате UTC (время по Гринвичу). | datetime2(7) | Недетерминированная |

#### **Функции даты и времени меньшей точности системы**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [CURRENT\_TIMESTAMP](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/current-timestamp-transact-sql) | CURRENT\_TIMESTAMP | Возвращает datetimeзначение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Смещение часового пояса не включается. | datetime | Недетерминированная |
| [ФУНКЦИЯ GETDATE](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/getdate-transact-sql) | GETDATE ( ) | Возвращает datetimeзначение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Смещение часового пояса не включается. | datetime | Недетерминированная |
| [GETUTCDATE](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/getutcdate-transact-sql) | GETUTCDATE ( ) | Возвращает datetimeзначение, содержащее дату и время компьютера, на котором экземплярSQL Serverзапущена.Дата и время возвращаются в формате UTC (время по Гринвичу). | datetime | Недетерминированная |

### **Функции, получающие компоненты даты и времени**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [ФУНКЦИЯ DATENAME](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datename-transact-sql) | Функция DATENAME ( datepart , даты ) | Возвращает символьную строку, представляющий указанный datepartуказанной даты. | nvarchar | Недетерминированная |
| [DATEPART](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datepart-transact-sql) | DATEPART ( datepart , даты ) | Возвращает целое число, представляющее указанную datepartуказанного даты. | int | Недетерминированная |
| [ДЕНЬ](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/day-transact-sql) | ДЕНЬ ( даты) | Возвращает целое число, представляющее день указанной даты. | int | Детерминированное |
| [МЕСЯЦ](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/month-transact-sql) | МЕСЯЦ ( даты ) | Возвращает целое число, представляющее месяц указанной даты даты. | int | Детерминированное |
| [ГОД](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/year-transact-sql) | ГОД ( даты ) | Возвращает целое число, представляющее год указанной даты даты. | int | Детерминированное |

### **Функции, получающие значения даты и времени из их компонентов**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [DATEFROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datefromparts-transact-sql) | DATEFROMPARTS ( года, месяц, день ) | Возвращает датызначение для заданного года, месяца и дня. | date | Детерминированное |
| [DATETIME2FROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datetime2fromparts-transact-sql) | DATETIME2FROMPARTS ( года, месяц, день, час, минуту, секунд , дроби, точности) | Возвращает datetime2значение для указанной даты и времени и с указанной точностью. | datetime2 (точности ) | Детерминированное |
| [DATETIMEFROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datetimefromparts-transact-sql) | DATETIMEFROMPARTS ( года, месяц, день, час, минуту, секунд , миллисекунд) | Возвращает datetimeзначение для указанной даты и времени. | datetime | Детерминированное |
| [DATETIMEOFFSETFROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datetimeoffsetfromparts-transact-sql) | DATETIMEOFFSETFROMPARTS ( года, месяц, день, час, минуту,секунд, дроби, hour\_offset, minute\_offset, точности) | Возвращает datetimeoffsetзначение для указанной даты и времени и с указанным смещением и точностью. | DateTime (точности ) | Детерминированное |
| [SMALLDATETIMEFROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/smalldatetimefromparts-transact-sql) | SMALLDATETIMEFROMPARTS ( года, месяц, день, час, минуту ) | Возвращает smalldatetimeзначение для указанной даты и времени. | smalldatetime | Детерминированное |
| [TIMEFROMPARTS](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/timefromparts-transact-sql) | TIMEFROMPARTS ( час, минуту, секунд, дроби, точности ) | Возвращает времязначение в течение определенного времени и с указанной точностью. | время (точности ) | Детерминированное |

### **Функции, получающие разность даты и времени**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [ФУНКЦИЯ DATEDIFF](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datediff-transact-sql) | DATEDIFF ( datepart , startdate , enddate ) | Возвращает количество даты или времени datepartграницы, пересекаемых между двумя указанными датами. | int | Детерминированное |
| [DATEDIFF\_BIG](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datediff-big-transact-sql) | DATEDIFF\_BIG ( datepart , startdate , enddate ) | Возвращает количество даты или времени datepartграницы, пересекаемых между двумя указанными датами. | bigint | Детерминированное |

### **Функции, которые изменяют значения даты и времени**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [ФУНКЦИЯ DATEADD](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/dateadd-transact-sql) | Функция DATEADD (datepart , номер, даты) | Возвращает новый datetimeзначения путем добавления интервала к указанной datepartуказанного даты. | Тип данных датыаргумент | Детерминированное |
| [EOMONTH](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/eomonth-transact-sql) | EOMONTH ( start\_date [, month\_to\_add ]) | Возвращает последний день месяца, содержащего указанную дату, с необязательным смещением. | Тип возвращаемого значения — это типstart\_dateилидаты. | Детерминированное |
| [SWITCHOFFSET](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/switchoffset-transact-sql) | КОММУТАТОРСМЕЩЕНИЕ(DATETIMEOFFSET , time\_zone) | КОММУТАТОРСМЕЩЕНИЕизменяет смещение часового пояса для значения DATETIMEOFFSET и сохраняет значение UTC. | DateTimeOffset с точностью в доляхDATETIMEOFFSET | Детерминированное |
| [TODATETIMEOFFSET](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/todatetimeoffset-transact-sql) | TODATETIMEOFFSET (выражение, time\_zone) | TODATETIMEOFFSET преобразует значение типа datetime2 в значение типа datetimeoffset.Значение datetime2 преобразуется в местное время для указанного time\_zone. | DateTimeOffset с точностью в долях datetimeаргумент | Детерминированное |

### **Функции, получить или задать формат сеанса**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [@@DATEFIRST](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datefirst-transact-sql) | @@DATEFIRST | Возвращает текущее значение параметра SET DATEFIRST для сеанса. | tinyint | Недетерминированная |
| [SET DATEFIRST](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/set-datefirst-transact-sql) | SET DATEFIRST { номер| @number\_var } | Устанавливает первый день недели в виде числа от 1 до 7. | Неприменимо | Неприменимо |
| [SET DATEFORMAT](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/set-dateformat-transact-sql) | SET DATEFORMAT { формат | @format\_var } | Задает порядок составляющих даты (месяц/день/год) для ввода datetimeили smalldatetimeданных. | Неприменимо | Неприменимо |
| [@@LANGUAGE](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/language-transact-sql) | @@LANGUAGE | Возвращает название используемого в данный момент языка.@@LANGUAGE не является функцией даты или времени.Однако на данные, выводимые функциями даты, могут повлиять настройки языка. | Неприменимо | Неприменимо |
| [УСТАНОВИТЬ ЯЗЫК](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/set-language-transact-sql) | SET LANGUAGE {[N] "язык" | @language\_var} | Устанавливает языковую среду сеанса и системных сообщений.SET LANGUAGE не является функцией даты или времени.Однако на данные, выводимые функциями даты, влияет параметр языка. | Неприменимо | Неприменимо |
| [sp\_helplanguage](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-stored-procedures/sp-helplanguage-transact-sql) | sp\_helplanguage[[@language = ] "язык" ] | Возвращает сведения о формате даты всех поддерживаемых языков.sp\_helplanguageне является дата или время хранимой процедуры.Однако на данные, выводимые функциями даты, влияет параметр языка. | Неприменимо | Неприменимо |

### **Функции, проверяющие значения даты и времени**

| Функция | Синтаксис | Возвращаемое значение | Тип возвращаемых данных | Детерминизм |
| --- | --- | --- | --- | --- |
| [ФУНКЦИЯ ISDATE](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/isdate-transact-sql) | Функция ISDATE ( выражение ) | Определяет, является ли datetimeили smalldatetimeвходное выражение является допустимым значением даты или времени. | int | Функция ISDATE детерминирована, только если используется совместно с функцией CONVERT и если заданный параметр стиля CONVERT не равен 0, 100, 9 или 109. |

**12. Функции преобразования типов данных.**

## **Синтаксис**

Syntax for CAST:

CAST ( expression AS data\_type [ ( length ) ] )

Syntax for CONVERT:

CONVERT ( data\_type [ ( length ) ] , expression [ , style ] )

## **Аргументы**

*expression*

Любое допустимое [выражение](https://msdn.microsoft.com/ru-ru/library/ms190286(v=sql.120).aspx).

*data\_type*

Целевой тип данных. К таким типам данных относятся **xml**, **bigint** и **sql\_variant**. Псевдонимы типов данных недопустимы.

*length*

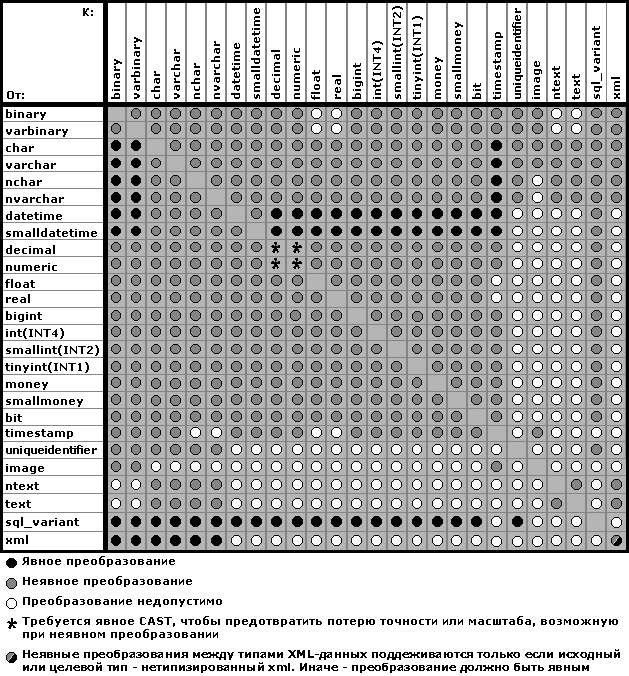
Указываемое дополнительно целое число, обозначающее длину целевого типа данных.Значение по умолчанию равно 30.

*style*

Целочисленное выражение, определяющее, как функция CONVERT преобразует параметр *expression*. Если стиль имеет значение NULL, возвращается NULL. Диапазон определяется параметром *data\_type*. Дополнительные сведения см. в разделе "Заметки".

## **Типы возвращаемых значений**

Возвращает параметр *expression*, преобразованный в тип *data\_type*.



**13. Условный оператор.**

Самым простым в использовании, в **PL/SQL** является оператор **IF**. Его основная логическая форма имеет вид:

IF(некоторое условие справедливо) THEN

-- условие справедливо, выполнять это.

ELSE -- условие не выполняется

-- выполнять оператор в этой части.

END IF; -- конец условного оператора.

Первый блок оператора после **IF-THEN** называется "ПРЕДЫДУЩИМ", а блок следующий за **ELSE**"ПОСЛЕДУЮЩИМ". Каждый оператор **IF-THEN** может содержать обрамляющие блоки **BEGIN - END**, если в этом есть необходимость, так как при записи типа:

IF(некоторое условие справедливо) THEN

-- оператор

END IF; -- конец условного оператора.

Можно уверенно сказать, что все четыре оператора выполнятся наверняка! Но, если применить такую конструкцию:

IF(некоторое условие справедливо) THEN

BEGIN

-- оператор

END;

END IF; -- конец условного оператора.

Код будет более нагляден, хотя это вопрос относится к стилю программирования. Так же считается, хорошим стилем наличие комментария при использовании оператора **IF**. Особенно в случае большого количества вложений. Оператор **IF** можно вкладывать на любую глубину выражения. Таким образом, можно задать достаточно сложную логику выражения. Так же для экономии памяти, возможно использовать конструкцию, типа:

IF(некоторое условие справедливо) THEN -- проверка условия

BEGIN-- условие справедливо, выполнять это.

END;

ELSE -- условие не выполняется

DECLARE

x NUMBER;

BEGIN

END;

END IF; -- конец условного оператора.

В этом случае переменная **x** будет создана, если условие предыдущего блока ложно. Хотя использовать, такие конструкции не всегда оправдано. Так же условные операторы в **PL/SQL** вычисляются с помощью так называемой "сокращенной" оценки. Допустим, есть такое условие:

IF( a AND b ) THEN -- проверка условия

BEGIN

END;

END IF; -- конец условного оператора.

Если выражение **a** равно **FALSE**, то дальнейшее выражение не вычисляется! Все выражения вычисляются слева направо, а выражения в скобках, имеют наивысший приоритет. Например, вот так:

IF( (a OR f) AND (c OR k) ) THEN -- проверка условия

BEGIN

END;

END IF; -- конец условного оператора.

Выражения в скобках вычисляются первыми. Что ж, пришло время вспомнить старого знакомого! Работа с **NULL**. Оператор **IF** работает с **NULL** достаточно просто и эффективно. Если записать:

IF(X = NULL) THEN -- проверка условия

END IF; -- конец условного оператора.

IF(X != NULL) THEN -- проверка условия

END IF; -- конец условного оператора.

Такое условие будет иметь значение **FALSE** и ничего выполняться не будет. Правильная запись будет такая:

IF(X IS NULL) THEN -- проверка условия

END IF; -- конец условного оператора

IF(X IS NOT NULL) THEN -- проверка условия

END IF; -- конец условного оператора.

От троичной логики никуда не деться, по этому запоминайте хорошенько! :) А, что делать, если необходимо проверить одно значение несколько раз? Например, вот так:

IF( val = 1 ) THEN -- проверка условия

END IF; -- конец условного оператора.

IF( val = 9 ) THEN -- проверка условия

END IF; -- конец условного оператора.

Такое количество операторов **IF** конечно можно использовать, но это слишком громоздко и не верно, так как существует конструкция **IF - THEN - ELSIF**. работает она достаточно просто и ее для таких целей по моему опыту вполне достаточно:

IF ( val = 1 ) THEN -- проверка условия

ELSIF ( val = 2 ) THEN

ELSIF ( val = 3 ) THEN

ELSIF ( val = 9 ) THEN

ELSE -- не сработало не одно из условий!

END IF; -- конец условного оператора.

**14. Изменение данных в таблице. Примеры.**

Чтобы изменить значение в поле в таблице БД необходимо выполнить запрос UPDATE.

Общий синтаксис запроса UPDATE

UPDATE [LOW\_PRIORITY] имя\_таблицы

SET название\_поля1=выражение1,название\_поля2=выражение2,...

[WHERE условное\_выражение] [LIMIT количество\_записей]

UPDATE заменит значения текущих полей таблицы на новые значения.  
SET устанавливает какие поля изменять и новые значения, которые нужно присвоить этим полям.  
WHERE (если необходимо) — условие на изменение определенных записей. Если WHERE не указан, изменены будут все записи.  
При указании параметра LOW\_PRIORITY, выполнение UPDATE задержится пока другие клиенты читают таблицу.

Пример запроса на изменение всех записей поля «apple» таблицы «fruit» на определенное значение

UPDATE fruit SET apple=5;

Пример запроса с WHERE который изменяет определенную запись:

UPDATE tovar SET price=500 WHERE id=5

При изменении значения поля можно использовать его текущее значение.  
Пример запроса увеличивающего значение поля price в 2 раза:

UPDATE tovar SET price=price\*2;

SET в UPDATE вычисляет выражения слева направо.  
Пример запроса который удваивает цену (поле price), а потом уменьшает его на 10:

UPDATE tovar SET price=price\*2, price=price-10;

Запрос UPDATE возвратит количество полей, которые были изменены в этом запросе.  
Использование LIMIT позволит изменять заданное количество записей.

**15. Различие между использованием DDL и DML инструкций.**

Триггеры DDL и триггеры DML используются для решения разных задач.

Триггеры DML создаются для инструкций INSERT, UPDATE и DELETE, чтобы обеспечить соблюдение бизнес-правил и целостность данных таблиц или представлений при их изменении.

Триггеры DDL применяются к инструкциям CREATE, ALTER, DROP и некоторым другим инструкциям DDL, а также к хранимым процедурам, которые выполняют схожие с DDL операции. Используют их для выполнения административных задач и гарантии соблюдения бизнес-правил, влияющих на базы данных. Действие этих триггеров распространяется на все команды одного типа во всей базе данных или на всем сервере.

Чтобы создать, изменить или удалить триггеры DML и триггеры DDL, используется похожий синтаксис Transact-SQL, к тому же они имеют и другие похожие характеристики.

Как и триггеры DML, триггеры DDL могут выполнять управляемый код, упакованный в сборку, созданную в среде Microsoft .NET Framework и переданную на SQL Server.

Как и в случае триггеров DML, для одной инструкции Transact-SQL можно создать несколько триггеров DDL. Кроме того, триггер DDL и инструкция, приводящая к его срабатыванию, выполняются в одной транзакции. В триггере можно выполнить откат этой транзакции. Серьезные ошибки могут приводить к автоматическому откату целой транзакции. Срабатывание триггера DDL, выполняемого в пакете и явно включающего инструкцию ROLLBACK TRANSACTION, приводит к отмене целого пакета.

**16. Типы данных БД Oracle 11g. Классификация и описание типов данных.**

**Символьные типы данных**

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Размер** | **Описание** |
| char(размер) | Максимальный размер 2000 байт. | Где размер — количество символов фиксированной длины. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка. |
| nchar(размер) | Максимальный размер 2000 байт. | Где размер — количество символов фиксированной длины в кодировке Unicode. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка. |
| nvarchar2(размер) | Максимальный размер 4000 байт. | Где размер – количество сохраняемых символов в кодировке Unicode переменной длины. |
| varchar2(размер) | Максимальный размер 4000 байт. Максимальный размер в PLSQL 32KB. | Где размер – количество сохраняемых символов переменной длины. |
| long | Максимальный размер 2GB. | Символьные данные переменной длины. |
| raw | Максимальный размер 2000 байт. | Содержит двоичные данные переменной длины |
| long raw | Максимальный размер 2GB. | Содержит двоичные данные переменной длины |

## Числовые типы данных

|  |  |  |
| --- | --- | --- |
| Типы данных | Размер | Описание |
| number(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. Масштаб может быть в диапазоне от -84 до 127. | Например,number (14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой. |
| numeric(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, numeric(14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой. |
| dec(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, dec (5,2) — это число, которое имеет 3 знака перед запятой и 2 знака после . |
| decimal(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, decimal (5,2) — это число, которое имеет 3 знака перед запятой и 2 знака после . |
| PLS\_INTEGER | Целые числа в диапазоне от -2,147,483,648 до 2,147,483,647 | Значение PLS\_INTEGER требуют меньше памяти и быстрее значений NUMBER |

## Дата/время типы данных

|  |  |  |
| --- | --- | --- |
| date | date может принимать значения от 1 января 4712 года до н.э. до 31 декабря 9999 года нашей эры. |  |

## Большие объекты (LOB) типы данных

|  |  |  |
| --- | --- | --- |
| bfile | Максимальный размер файла 4 ГБ. | Файл locators, указывает на двоичный файл в файловой системе сервера (вне базы данных). |
| blob | Хранит до 4 ГБ двоичных данных. | Хранит неструктурированные двоичные большие объекты. |
| clob | Хранит до 4 ГБ символьных данных. | Хранит однобайтовые и многобайтовые символьные данные. |
| nclob | Хранит до 4 ГБ символьных текстовых данных. | Сохраняет данные в кодировке unicode. |

## Rowid тип данных

|  |  |  |
| --- | --- | --- |
| rowid | Формат строки: BBBBBBB.RRRR.FFFFF, Где BBBBBBB — это блок в файле базы данных; RRRR — строка в блоке;  FFFFF — это файл базы данных. | Двоичные данные фиксированной длины. Каждая запись в базе данных имеет физический адрес или идентификатор строки (rowid). |

## Булевы (BOOLEAN) типы данных

|  |  |  |
| --- | --- | --- |
| BOOLEAN | TRUE или FALSE. Может принимать значение NULL | Хранит логические значения, которые вы можете использовать в логических операциях. |

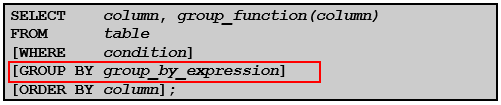
**17. Понятие ограничения целостности данных.**

Ограничения целостности данных позволяют добавить для них требования, дополнительные к соблюдению типа. Заявляемые (схемные, формальные, "декларативные") ограничения целостности записываются ("провозглашаются") в виде условий, которые должны соблюдаться явно как таковые, на уровне схемы данных, и этим отличаются от правил целостности, сформулированных в виде запрограммированных проверок. Поэтому иначе такие ограничения можно называть "явными". Оригинальный термин имеет полное название "integrity data constraints" — "ограничения на значения данных, налагаемые для более точного учета обстоятельств предметной области", но часто сокращается до "integrity constraints" или даже просто "constraints". 

**18. Создание групп данных**

### **СОЗДАНИЕ ГРУПП ДАННЫХ: СИНТАКСИС ПРЕДЛОЖЕНИЯ GROUP BY**

Можно разделить строки таблицы в меньшие группы при использовании предложения GROUP BY.



Можно использовать предложение GROUP BY, чтобы разделить строки таблицы в группы. Можно затем использовать групповые функции, чтобы возвратить итоговую информацию для каждой группы.

В синтаксисе:

|  |  |
| --- | --- |
|  | group\_by\_expression Указывает столбцы, значения которых определяют критерий для того, чтобы сгруппировать строки |

### **ИНСТРУКЦИИ**

* Если Вы включаете групповую функцию в предложение SELECT, невозможно выбрать также и отдельные результаты, если отдельный столбец не появляется в предложении GROUP BY. Вы получаете сообщение об ошибке, если Вы не включаете список столбцов в предложение GROUP BY.
* Используя предложение WHERE, можно исключить строки прежде, чем разделить их на группы.
* Следует включать *столбцы* в предложение GROUP BY.
* Невозможно использовать псевдоним столбца в предложении GROUP BY.

**19. Подзапросы. Типы подзапросов.**

Язык *SQL* разрешает использовать в других операторах языка *DML* *подзапросы*, которые являются внутренними запросами, определяемыми оператором SELECT.

***Подзапрос*** - очень мощное средство языка SQL. Он позволяет строить сложные иерархии запросов, многократно выполняемые в процессе построения результирующего набора или выполнения одного из операторов изменения данных ( *DELETE* , *INSERT* , *UPDATE* ).

Условно *подзапросы* иногда подразделяют на три типа, каждый из которых является сужением предыдущего:

* *табличный подзапрос*, возвращающий набор строк и столбцов;
* *подзапрос строки*, возвращающий только одну строку, но, возможно, несколько столбцов (такие *подзапросы* часто используются во *встроенном SQL*);
* *скалярный подзапрос*, возвращающий значение одного столбца в одной строке.

*Подзапрос* позволяет решать следующие задачи:

* определять набор строк, добавляемый в таблицу на одно выполнение оператора *INSERT* ;
* определять данные, включаемые в представление, создаваемое оператором *CREATE VIEW* ;
* определять значения, *модифицируемые* оператором *UPDATE* ;
* указывать одно или несколько значений во фразах WHERE и HAVING оператора SELECT ;
* определять во фразе FROM таблицу как результат выполнения *подзапроса* ;
* применять *коррелированные подзапросы*. *Подзапрос* называется коррелированным, если запрос, содержащийся в предикате, имеет ссылку на значение из таблицы (внешней к данному запросу), которая проверяется посредством данного предиката.

Hекоторые *СУБД* (например, *СУБД* *Oracle*) позволяют на основе *подзапроса* создавать новые таблицы с помощью оператора *CREATE TABLE*.

Простым примером использования *подзапроса* может служить следующий оператор:

SELECT \* from tbl1

WHERE f2=(SELECT f2 FROM tbl2

WHERE f1=1);

**20. Назначение и классификация распределенных систем.**

Обычно, распределенной считают такую систему, в которой функционирует более одного сервера БД. Это применяется для уменьшения нагрузки на сервер и обеспечения работы территориально удаленных подразделений. Различная сложность создания, модификации, сопровождения, интеграции с другими системами позволяют разделить ИС на классы:  малых, средних и крупных распределенных систем.

**Малые ИС** имеют небольшой жизненный цикл (ЖЦ), ориентацию на массовое использование, невысокую цену, невозможность модификации без участия разработчиков, использующие в основном настольные системы управления базами данных (СУБД), однородное аппаратно-программное обеспечение, не имеющие средств обеспечения безопасности.

**Крупные** корпоративные ИС, системы федерального уровня и другие имеют длительный жизненный цикл, миграцию унаследованных систем, разнообразие аппаратно-программного обеспечения, масштабность и сложность решаемых задач, пересечение множества предметных областей, аналитическую обработку данных, территориальную распределенность компонент

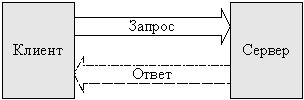
21. Архитектура и основные компоненты распределенных систем.

*По* утверждению известного специалиста в области информатики Э. Таненбаума, не существует общепринятого и в то же время строгого определения распределенной системы. Некоторые остряки утверждают, что распределенной является такая *вычислительная система*, в которой неисправность компьютера, о существовании которого пользователи ранее даже не подозревали, приводит к остановке всей их работы. Значительная часть распределенных вычислительных систем, к сожалению, удовлетворяют такому определению, однако формально оно относится только к системам с уникальной точкой уязвимости ( *single point of failure* ).

Часто при определении распределенной системы во главу угла ставят разделение ее функций между несколькими компьютерами. При таком подходе распределенной является любая *вычислительная система*, где обработка данных разделена между двумя и более компьютерами. Основываясь на определении Э. Таненбаума, несколько более узко распределенную систему можно определить как набор соединенных каналами связи независимых компьютеров, которые с точки зрения пользователя некоторого программного обеспечения выглядят единым целым.

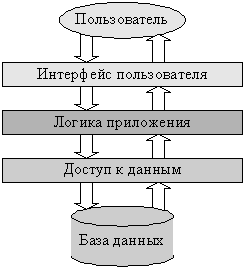
Такой подход к определению распределенной системы имеет свои недостатки. Например, все используемое в такой распределенной системе *программное обеспечение* могло бы работать и на одном единственном компьютере, однако с точки зрения приведенного выше определения такая система уже перестанет быть распределенной. Поэтому понятие распределенной системы, вероятно, должно основываться на анализе образующего такую систему программного обеспечения.

Как основу описания взаимодействия двух сущностей рассмотрим общую модель взаимодействия клиент-*сервер*, в которой одна из сторон (клиент) инициирует обмен данными, посылая *запрос* другой стороне (серверу). *Сервер* обрабатывает *запрос* и при необходимости посылает ответ клиенту ([рис. 1.1](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.1)).



**Рис. 1.1.**Модель взаимодействия клиент сервер

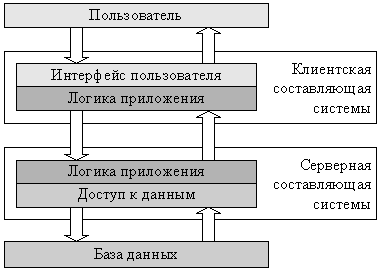
Взаимодействие в рамках модели клиент *сервер* может быть как синхронным, когда клиент ожидает завершения обработки своего запроса сервером, так и асинхронным, при котором клиент посылает серверу *запрос* и продолжает свое выполнение без ожидания ответа сервера. Модель клиента и сервера может использоваться как основа описания различных взаимодействий. Для данного курса важно взаимодействие составных частей программного обеспечения, образующего распределенную систему.



**Рис. 1.2.**Логические уровни приложения

Рассмотрим некое типичное *приложение*, которое в соответствии с современными представлениями может быть разделено на следующие логические уровни ([рис. 1.2](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.2)): *пользовательский интерфейс* (ИП), логика приложения (ЛП) и *доступ* к данным (ДД), работающий с базой данных (*БД*). *Пользователь* системы взаимодействует с ней через *интерфейс* пользователя, *база данных* хранит данные, описывающие предметную область приложения, а уровень логики приложения реализует все алгоритмы, относящиеся к *предметной области*.

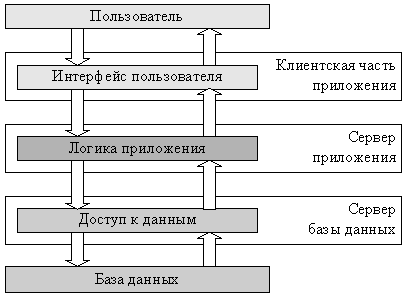
Поскольку на практике разных пользователей системы обычно интересует *доступ* к одним и тем же данным, наиболее простым разнесением функций такой системы между несколькими компьютерами будет разделение логических уровней приложения между одной серверной частью приложения, отвечающим за *доступ* к данным, и находящимися на нескольких компьютерах клиентскими частями, реализующими *интерфейс* пользователя. Логика приложения может быть отнесена к серверу, клиентам, или разделена между ними ([рис. 1.3](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.3)).



**Рис. 1.3.**Двухзвенная архитектура

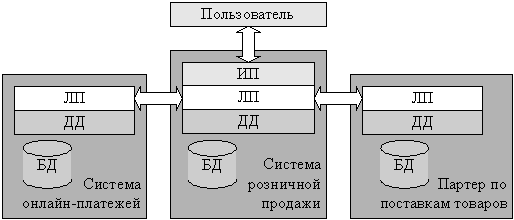
Архитектуру построенных *по* такому принципу приложений называют клиент серверной или двухзвенной. На практике подобные системы часто не относят к классу распределенных, но формально они могут считаться простейшими представителями распределенных систем.

Развитием архитектуры клиент-*сервер* является трехзвенная *архитектура*, в которой *интерфейс* пользователя, логика приложения и *доступ* к данным выделены в самостоятельные составляющие системы, которые могут работать на независимых компьютерах ([рис. 1.4](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.4)).



**Рис. 1.4.**Трехзвенная архитектура

*Запрос* пользователя в подобных системах последовательно обрабатывается клиентской частью системы, сервером логики приложения и сервером баз данных. Однако обычно под распределенной системой понимают системы с более сложной архитектурой, чем трехзвенная.

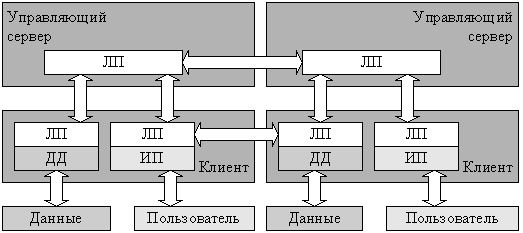


**Рис. 1.5.**Распределенная система розничных продаж

Применительно к приложениям автоматизации деятельности предприятия, распределенными обычно называют системы с логикой приложения, распределенной между несколькими компонентами системы, каждая из которых может выполняться на отдельном компьютере. Например, реализация логики приложения системы розничных продаж должна использовать запросы к логике приложения третьих фирм, таких как поставщики товаров, системы электронных платежей или банки, предоставляющие потребительские кредиты ([рис. 1.5](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.5)).

Таким образом, в обиходе под распределенной системой часто подразумевают рост *многозвенной архитектуры* "в ширину", когда запросы пользователя не проходят последовательно от интерфейса пользователя до единственного сервера баз данных.

В качестве другого примера распределенной системы можно привести сети прямого обмена данными между клиентами ( *peer-to-peer networks* ). Если предыдущий пример имел "древовидную" архитектуру, то сети прямого обмена организованы более сложным образом, [рис. 1.6](http://www.intuit.ru/studies/courses/1115/177/lecture/4778?page=1#image.1.6). Подобные системы являются в настоящий момент, вероятно, одними из крупнейших существующих распределенных систем, объединяющие миллионы компьютеров.



**Рис. 1.6.**Система прямого обмена данными между клиентами

22. Определение функциональных и нефункциональных требований

#### Системные требования и требования к программному обеспечению

Существуют различные трактовки понятия "Системные требования" (*system requirements*).

К. Вигерс формулирует данный термин, как "высокоуровневые требования к продукту, которые содержат многие подсистемы, то есть системе" [[2.2]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.2). При этом под системой понимается программная, программно-аппаратная, либо *человеко-машинная система*. Данная система является сложной, структурированной системой и системные требования являются подмножеством функциональных требований к продукту. В данное подмножество целесообразно относить наиболее важные, существенные требования, которые относятся в целом к системе и не содержат избыточной детализации.

INCOSE (International Council on *Systems Engineering*) дает более детальное определение системы: "комбинация взаимодействующих элементов, созданная для достижения определенных целей; может включать аппаратные средства, программное обеспечение, встроенное ПО, другие средства, людей, информацию, техники (подходы), службы и другие поддерживающие элементы". Таким образом, происходит разделение между системными требованиями, как обобщающему понятию и требованиями к программному обеспечению, как выделенному подмножеству системных требований, направленных исключительно на программные компоненты системы. Этот же подход прослеживается в стандарте ГОСТ Р ИСО/МЭК 12207/99 [[2.8]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.8): работы, связанные с системой в целом и с программным обеспечением выделяются в отдельные группы в целях удобства оперирования.

***INCOSE*** (International Council on Systems Engineering) - Международный совет по системной инженерии, [www.incose.org](http://www.incose.org/) — некоммерческая организация, ставящая своей целью развитие системной инженерии и профессиональный рост системных инженеров. В настоящее время насчитывает более 8000 членов. Под эгидой INCOSE разработан ряд международных стандартов в области системной инженерии.

В практике компьютерной инженерии бытует другой, более узкий контекст использования данного понятия: под системными требованиями в узком смысле понимаются требования, выдвигаемые прикладной программной системой (в частности - информационной) к среде своего функционирования (системной, аппаратной). Пример таких требований - тактовая частота процессора, объем памяти, требования к выбору операционной системы.

#### Функциональные, нефункциональные требования и характеристики продукта

Функциональные требования регламентируют функционирование или поведение системы (behavioral requirements). Функциональные требования отвечают на вопрос "что должна делать система" в тех или иных ситуациях. Функциональные требования определяют основной "фронт работ" Разработчика, и устанавливают цели, задачи и сервисы, предоставляемые системой Заказчику.

Функциональные требования записываются, как правило, при посредстве предписывающих правил: "система должна позволять кладовщику формировать приходные и *расходные накладные*". Другим способом являются так называемые варианты использования (users cases) - популярный и весьма продуктивный способ *представления требований*.

***Use case*** - вариант использования, прецедент. Данный термин был введен в обиход программной инженерии шведским учёным Айваром Якобсоном (Ivar Hjalmar Jacobson) и по сей день является одной из наиболее позитивных абстракций в области создания требований к программному обеспечению. Согласно нотации UML 2.4.1 (<http://www.omg.org/spec/UML/2.4.1/>), прецеденты являются средством для определения требуемых использований системы. Как правило, они применяются для извлечения требований к системе, то есть, того, что система предполагает делать. Основными понятиями, связанными с вариантами использования являются акторы, прецеденты, и объект. Объектом является рассматриваемая система, в которой применяются прецеденты. Пользователи и любые другие системы, которые могут взаимодействовать с объектом, представлены в качестве акторов. Акторы всегда моделируют сущности, находящиеся за пределами системы. Требуемое поведение объекта задается одним или несколькими вариантами использования, которые определяются в соответствии с потребностями акторов. Строго говоря, термин "вариант использования" относится к типу прецедента. Экземпляр прецедента относится к проявлению поведения, соответствующего типу прецедента. Такие случаи, как правило, описывается через спецификацию взаимодействий.

Это - основной, определяющий вид требований, который будет рассматриваться на протяжении всего лекционного курса.

Нефункциональные требования, соответственно, регламентируют внутренние и внешние условия или атрибуты функционирования системы. К. Вигерс [[2.2]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.2) выделяет следующие основные группы нефункциональных требований:

* Внешние интерфейсы (*External Interfaces*),
* Атрибуты качества (*Quality Attributes*),
* Ограничения (Constraints).

Среди внешних интерфейсов в большинстве современных АИС наиболее важным является интерфейс пользователя (User Interface, UI). Кроме того, выделяются интерфейсы с внешними устройствами (аппаратные интерфейсы), программные интерфейсы и интерфейсы передачи информации (коммуникационные интерфейсы).

Основные атрибуты качества:

* Применимость,
* Надежность,
* Производительность,
* Эксплуатационная пригодность,

достаточно хорошо раскрыты в модели FURPS (см. ниже).

Ограничения [[2.2]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.2) - формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации. Выбор платформы реализации и/или развертывания (протоколы, серверы приложений, баз данных, ...), которые, в свою очередь, могут относиться, например, к внешним интерфейсам (конец цитаты).

Характеристики продукта. К.Вигерс [[2.2]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.2) формулирует характеристику, "фичу" (feature), как набор логически связанных функциональных требований, которые обеспечивают возможности пользователя и удовлетворяют *бизнес-цели*.

Существует и более общий взгляд на данное понятие [[2.9]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.9): "features могут быть как относящимися к функциональным, так и к нефункциональным требованиям и могут изменяться от версии к версии продукта".

С.Орлик в [[2.6]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.6) отмечает, что "с точки зрения инженерии требований, features являются самостоятельным артефактом, который может быть соотнесен как с функциональными требованиями, так и с нефункциональными".

Роль характеристик проявляется в первую очередь в области маркетинга: не всякий потенциальный потребитель продукта станет читать его функциональные описания, а набор ключевых характеристик, характеризующих конкурентные преимущества, можно сделать лаконичным и уместить на одной странице рекламной листовки, либо напечатать на компакт-диске.

#### Классификация RUP

В спецификациях Rational Unified Process при классификации требований используется модель FURPS+ со ссылкой на стандарт *IEEE* Std 610.12.1990 [[2.1]](http://www.intuit.ru/studies/courses/2188/174/literature#literature.2.1).

Акроним FURPS обозначает следующие категории требований:

* Functionality (Функциональность)
* Usability (Применимость)
* Reliability (Надежность)
* Performance (Производительность)
* Supportability (эксплуатационная пригодность).

Символ "+" расширяет FURPS-модель, добавляя к ней:

* ограничения проекта,
* требования выполнения,
* требования к интерфейсу,
* физические требования,

часть из которых уже была рассмотрена выше.

Кроме того, в спецификациях RUP выделяются такие категории требований, как

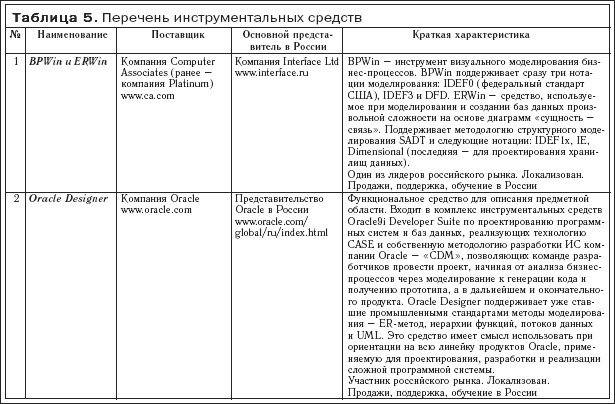
* требования, указывающие на необходимость согласованности с некоторыми юридическими и нормативными актами;
* требования к лицензированию,
* требования к документированию.

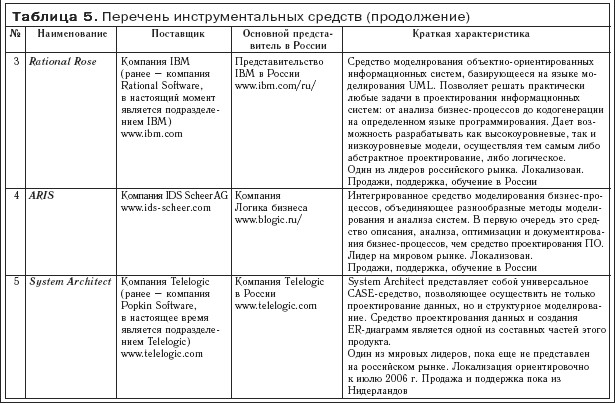
***FURPS*** (Functionality Usability Reliability Performance Supportability: функциональность, удобство использования, надежность, производительность, удобство сопровождения)— классификация требований к программным системам, разработанная в Hewlett-Packard. Согласно классификации, все требования подразделяются на 5 категорий, непосредственно следующих из составляющих наименования классификации. В настоящее время используется, как составная часть более общей классификации FURPS+.

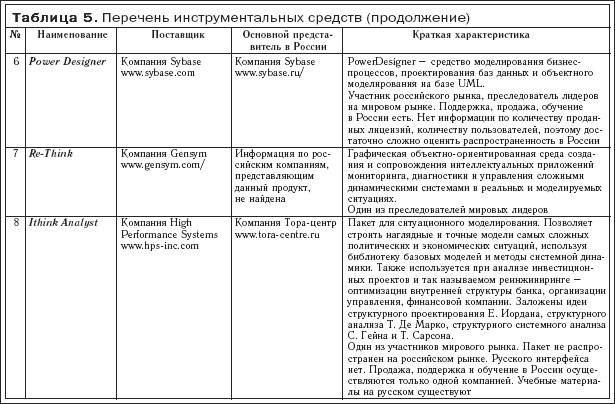
***FURPS+*** (Functionality Usability Reliability Performance Supportability +: функциональность, удобство использования, надежность, производительность, удобство сопровождения, дополнительные требования) — расширенная версия классификации требований FURPS. Дополнительно включает в себя ограничения, разделенные на следующие группы требований:

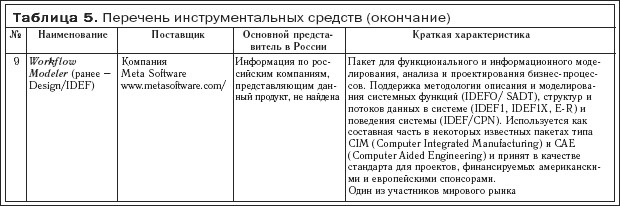
* ограничения проектирования (design);
* ограничения разработки (implementation);
* ограничения на интерфейсы (interface);
* физические ограничения (physical).

**23. Средства анализа бизнес-процессов.**









**24. Определение сущностей и необходимых атрибутов**

Предметом информационной модели является абстрагирование объектов или явлений реального мира в рамках предметной области, в результате которого выявляются сущности (entity) предметной области. Как правило, они обозначаются именем существительным естественного языка.

Сущность описывается с помощью данных, именуемых свойствами или *атрибутами (attributes) сущности*. Как правило, атрибуты являются определениями в высказывании о сущности и обозначаются именами существительными естественного языка. Сущности вступают в связи друг с другом через свои атрибуты. Каждая группа атрибутов, описывающих одно реальное проявление сущности, представляет собой экземпляр (instance) сущности. Иными словами, *экземпляры сущности* - это реализации сущности, отличающиеся друг от друга и допускающие однозначную идентификацию.

Внимание! При представлении сущности в базе данных хранятся только ее атрибуты.

Одним из основных компьютерных способов распознавания сущностей в базе данных является присвоение сущностям идентификаторов (Entity identifier). Часто *идентификатор сущности* называют ключом. Задача выбора *идентификатора сущности* является семантически субъективной задачей. Поскольку сущность определяется набором своих атрибутов, то для каждой сущности целесообразно выделить такое подмножество атрибутов, которое однозначно идентифицирует данную сущность.

Некоторые сущности имеют естественные идентификаторы. Например, естественным идентификатором *счета-фактуры* является его номер. *Идентификаторы сущности* могут быть составными - составленными из нескольких атрибутов и атомарными - составленными из одного атрибута сущности.

Идентификация сущностей проводится аналитиками. Однако чаще всего их решение не является окончательным! Задача проектировщика баз данных - обеспечить при сохранении *экземпляров сущности* в базе данных наличие у каждого ее нового экземпляра уникального идентификатора. *Уникальный идентификатор сущности* - это *атрибут сущности*, позволяющий отличать одну сущность от другой. Если сущность имеет несколько уникальных идентификаторов, так называемых *возможных ключей*, то проектировщик должен выбрать первичный ключ сущности.

Различают однозначные и многозначные атрибуты. Однозначными являются атрибуты, которые в пределах конкретного экземпляра сущности имеют только одно значение. В противном случае они считаются многозначными.

Важным моментом изучения информационной модели проектировщиком является выделение многозначных *атрибутов сущности*. Это связано с тем, что реляционная модель базы данных не поддерживает многозначных атрибутов, и они должны быть разрешены на последующих стадиях проектирования.

Каждый *атрибут сущности* имеет домен (domain). Домен - это выражение, определяющее значения, разрешенные для данного атрибута. Иными словами, домен - это область значений атрибута. Проектировщик базы данных должен проконтролировать, чтобы в *информационной модели предметной области* для каждого *атрибута сущностей* был определен домен.

На уровне информационного моделирования данных назначение домена атрибуту носит общий характер. Например, атрибут текстовый, числовой, бинарный, дата или "не определен". В последнем случае аналитик должен дать описание домена. На последующих стадиях тип домена конкретизируется, смысл понятия домена в логической и физической моделях базы данных уже, чем его может понимать аналитик. Это связано с тем, что в рамках физической модели базы данных домен реализуется посредством механизма ограничения домена, СУБД не понимает неопределенных доменов.

**25. Определение атрибутов, которые будут являться уникальными идентификаторами для каждой сущности.**

Как правило, атрибуты являются определениями в высказывании о сущности и обозначаются именами существительными естественного языка.

Одним из основных компьютерных способов распознавания сущностей в базе данных является присвоение сущностям идентификаторов (Entity identifier). Часто *идентификатор сущности* называют ключом. Задача выбора *идентификатора сущности* является семантически субъективной задачей. Поскольку сущность определяется набором своих атрибутов, то для каждой сущности целесообразно выделить такое подмножество атрибутов, которое однозначно идентифицирует данную сущность.

Некоторые сущности имеют естественные идентификаторы. Например, естественным идентификатором *счета-фактуры* является его номер. *Идентификаторы сущности* могут быть составными - составленными из нескольких атрибутов и атомарными - составленными из одного атрибута сущности.

*Уникальный идентификатор сущности* - это *атрибут сущности*, позволяющий отличать одну сущность от другой. Если сущность имеет несколько уникальных идентификаторов, так называемых *возможных ключей*, то проектировщик должен выбрать первичный ключ сущности.