**Delphi Incomplete reference**

**Ffixed/ffcurency:**

//not this can only be done with a real/float variable

FloattostrF(rNum, ffcurency, 8, 4)

FloattostrF(rNum, Ffixed, 8, 4)

//8 : numbers in front of the comma or fullstop

//4 : decimal places

**Date/Time:**

ShowMessage('Today = '+DateToStr(Date));  
Today = 29/10/2002

**Message dialog:**

Taken off <http://www.delphibasics.co.uk/>

var  
  buttonSelected : Integer;  
begin  
   buttonSelected := messagedlg('Custom dialog',mtCustom, [mbYes,mbAll,mbCancel], 0);   // Show a custom dialog  
   // Show the button type selected  
   if buttonSelected = mrYes    then ShowMessage('Yes pressed');  
   if buttonSelected = mrAll    then ShowMessage('All pressed');  
   if buttonSelected = mrCancel then ShowMessage('Cancel pressed');  
end;

//or:

if messageDlg('Sure you want to delete recod for ' + sVeg +'?', mtWarning,[mbOk, mbCancel], 0) = mrOk

**Passwords:**

Edit2.PasswordChar := '•';

**Random:**

iRandom := RANDOM(10) + 1;

//random(10) will give you a random value from 0 to 9 the plus one will give you 1 to 10

**Jpeg:**

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, ExtCtrls, ComCtrls, **JPEG**;

img.picture.LoadFromFile(‘image.jpg’);

**Text files:**

Reading from:

if FileExists('Names.txt') <> true

ShowMessage('File does not exist');

Exit;

end;

AssignFile(fileRead, 'Names.txt');

Reset(fileRead);

while NOT EoF (fileRead)//while it is not end of file

//only use while loops for looking in text files

ReadLn(fileRead, sLine); //assign sline the current line

iLength := length(sLine);

if SLine = ' ' then delete(sLine,1, iLength);

iPos := POS('#',SLine);

sName := COPY(sLine, 1, iPos - 1);

DELETE(sLine, 1, iPos);

iAge := strtoint(sLine);

end;

Writing to:

AssignFile(myFile, 'Test.txt');  
ReWrite(myFile);

append(myFile);

WriteLn(myFile, sline);

CloseFile(myFile);

Sorting:

for iLoop := 1 to iCount - 1

for iLoop2 := iLoop + 1 to iCount

if arrHeight[iLoop2] < arrHeight[iLoop]

sTemp := inttostr(arrHeight[iLoop]);

arrHeight[iLoop] := arrHeight[iLoop2];

arrHeight[iLoop2] := strtoint(sTemp);

end;end;

end;

**Array:**

Array(2d):

arrTwoD : Array [1..10, 1..10] of integer;

arrTwoD[iRow][ iCol];

**String grid:**

stgDisplay.Cells[iCol, iRow]

stgDisplay.Cells[iCol, iRow]

**Databases not sql:**

Number of records:

iCount := adoFruitandveg.RecordCount;

Displaying a field:

adoFruitandveg.Open;

adofruitandveg.first;

while not adofruitandveg.eof

redOut.Lines.Add(adofruitandveg['FVName']);

adofruitandveg.next;

end;

adofruitandveg.Close;

Searching for the first instance:

adoFruitAndVeg.Open;

if adoFruitAndVeg.Locate('FVname', sVeg, []) = true

redOut.Lines.Add(adoFruitAndVeg['FVName'] + ' found in position ' + inttostr(adoFruitAndVeg['ID']));

end

else

redOut.Lines.Add('not found in database');

end;

adoFruitandveg.Close;

Feld by name

adoQryBookings.FieldByName('StartOfBooking').AsDateTime

Searching for multiple instances:

adoFruitandveg.Open;

adoFruitandveg.First;

while not adoFruitandveg.Eof

if uppercase(adoFruitandveg['colour']) = uppercase(sCol)

sOutput := sOutput + adoFruitandveg['FvName'] + ',';

end;

adoFruitandveg.Next;

end;

adoFruitandveg.Close;

Sorting data:

adoFruitandveg.Open;

adofruitandveg.First;

adofruitandveg.Sort := 'Price DESC';

while not adofruitandveg.Eof

redOut.Lines.Add(floattostrF(adoFruitandveg['Price'], ffCurrency,8,2)

+ #9 + adoFruitandveg['FVName']);

adoFruitandveg.Next;

end;

adoFruitandveg.Close;

Inserting data:

sName := inputBox('','Enter name','');

adoFruitandveg.Insert;

adoFruitandveg['FVName'] := sName;

adoFruitandveg.Post;

Updating a record:

adoFruitandveg.Open;

adoFruitandveg.First;

while not adoFruitandveg.Eof

adoFruitandveg.edit;

if uppercase(adoFruitandveg['FVName']) = uppercase(sVeg)

adoFruitandveg['Colour'] := sCol;

end;

adoFruitandveg.Post;

adoFruitandveg.Next;

end;

adoFruitandveg.Close;

Deleting a record:

adoFruitandveg.Open;

adoFruitandveg.First;

if AdoFruitandveg.Locate('FVName', sVeg , [])

if messageDlg('Sure you want to delete recod for ' + sVeg +'?', mtWarning,[mbOk, mbCancel], 0) = mrOk

adoFruitandVeg.Delete;

end;

end;

adoFruitandveg.Close;

Filtering:

adoTable.filter(‘string’);

adotable.filtered;//can’t use this in tests

**Methods:**

Searching for a substring:

Containstext(‘help me’, help) //this returns true or false depending on if a the substring is found or not

String and character Methods:

|  |  |  |
| --- | --- | --- |
| CHAR | CHAR(65) (= ’A’) | Converts an ascii value to a character |
| ORD | ORD(‘A’) (= 65) | Finds the ascii value of a character |
| LENGTH | LENTH(‘Sally’) (= 5) | Counts the number of characters in a String |
| UPPERCASE | UPPERCASE(sally) (= SALLY) | Converts the whole String to uppercase |
| POS | POS(‘#’, ‘Sally#18’) (=6) | Finds the position of a substring in a String |
| COPY | COPY(‘Frederick’, 1, 3) (= ‘Fre’) | Copies 3 characters from the existing String from index 1 |
| DELETE | DELETE(‘Fredericko’,5,6) (= ‘Fred’) | Deletes 6 characters from the existing String from index 5 |
| TRIM | TRIM (‘ spaces ‘) (= ‘spaces’) | Cuts off white space before and after the actual characters in the String |

Mathematical Methods:

|  |  |  |
| --- | --- | --- |
| DIV | 10 DIV 5 ( = 2) | Divide integers to get an integer answer |
| MOD | 10 MOD 5 (= 0) | Remainder after dividing |
| INC | INC(2) (= 3) | Adds 1 onto the existing number |
| DEC | DEC(2) (= 1) | Subtracts 1 from the existing number |
| TRUNC | TRUNC(8.7) (= 8) | Cuts off the decimal part and leaves the whole number |
| ROUND | ROUND(8.7) (= 9) | Rounds off the existing number |
| FRAC | FRAC(8.7) (= 7) | Cuts off the whole number and returns the decimal part |
| SQR | SQR(4) (= 16) | Squares the number |
| SQRT | SQRT(16) (= 4) | Finds the square root of the number |
| ABS | ABS (-4) (= 4) | Ensures that the number is positive in all cases |
| POWER | POWER(2,3) (= 8) | Raises the 1st number to the power of the 2nd number |

**Procedures:**

Declaring:

procedure blablabla(sIn: string; var sOut: string);

sOut := sIn;

end;

Calling:

blablabla (sIn, sOut);

**Functions:**

Declaring:

function getString(sAnyString : string) : String;

var

sTheoreticalString : string;

begin

sTheoreticalString := sAnyString

Result := sTheoreticalString;

end;

calling:

sSRealtr := getString(sAnyString);

**Dynamic objects:**

btnTemp: TButton; //step 1 - Declare your global object

procedure TForm1.StaticButton1Click(Sender: TObject);

//step 2 - create your dynamic button

btnTemp := TButton.Create(Form1);

// Step 3 - Set properties of your object

btnTemp.Caption := 'Say hello';

with btnTemp do

Height := StaticButton1.Height; // did this so that the objects could be placed in relation to the other

Width := StaticButton1.Width;

Top := StaticButton1.Top + StaticButton1.Height;

Left := StaticButton1.Left + StaticButton1.Width;

Parent := Form1;

end;

//Step 5 - assign your methods to your object events

btnTemp.OnClick := btnTempwhenClicked;

end;

//Step 4 - create procedure for your methods

procedure TForm1.btnTempwhenClicked(Sender: TObject);

begin

showmessage('Hello!');

end;

More advanced:

for iLoop := 1 to 9 do

arrBtnNumbers[iLoop] := TButton.Create(Form1);

with arrBtnNumbers[iLoop] do

Parent := Form1;

Caption := '+' + inttostr(iLoop);

width := 75;

height := 25;

left := 8;

if iLoop = 1 then

top := StaticBtnReset.height + StaticBtnReset.top + 1

else

top := arrBtnNumbers[iLoop - 1].height + arrBtnNumbers[iLoop - 1].top + 1;

onClick := NumberClick;

end;

end;

procedure TForm1.NumberClick(Sender: TObject);

begin

iSum := iSum + strtoint((Sender as TButton).Caption[2]);

lblSum.Caption := inttostr(iSum);

end;

**Object orientated programming:**

Separate unit:

TMatricDance = class // data type

private

fBoy: string;

fGirl: string;

public

constructor create; overload; // allways create

constructor create(sB, sG: string);overload;

//mutators

procedure setBoyName(sB: string); // always set

procedure setGirlName(sG: string);

// accessors

function getBoyName: string;

function getGirlName: string;// always get

// auxillary

function toString: string;

primary unit:

var

Form1: TForm1;

couple: TMatricDance;

constructing in primary unit:

procedure TForm1.Button3Click(Sender: TObject);

var

sBoy: string;

sGirl: string;

begin

sBoy := inputbox('', 'Boy''s name', '');

sGirl := inputbox('', 'Girl''s name', '');

couple := TMatricDance.create(sBoy, sGirl);

RichEdit1.lines.add(couple2.toString);

end;

Mutators:

procedure TMatricDance.setBoyName(sB: string);

begin

fBoy := sB;

end;

Accessors:

function TMatricDance.getBoyName: string;

begin

result := fBoy;

end;

**Databases sql: refer to online resources for more sql stuff**

|  |  |  |
| --- | --- | --- |
| The SELECT clause specifies the **fields** or **calculation results** (new temporary field) to display | | |
| SELECT | SELECT field\_name1, field\_name2, calculation AS alias FROM table\_name | Displays the fields and/or calculations mentioned in the list |
| SELECT \* | SELECT \*  FROM Customers | Displays all fields of the selected records |
| SELECT DISTINCT | SELECT DISTINCT Country  FROM Customers; |  |
| SELECT TOP | SELECT TOP 3 \*  FROM Customers WHERE Country='Germany'; |  |

|  |  |  |
| --- | --- | --- |
| ALIAS (caption) for a field name - existing field or calculated field | | |
| AS (alias) | SELECT field\_name AS field\_alias FROM table\_name  SELECT calculated\_value AS field\_alias FROM table\_name | The alias appears instead of the field name as the column heading.  Calculated fields with no alias appear as the heading Expression1 |

|  |  |  |
| --- | --- | --- |
| SQL CLAUSES that **modify the contents** (records) of the table | | |
| INSERT INTO table\_name  Note that the table name comes first for all three | INSERT INTO table\_name VALUES (value1, value2, value3 ...)  *or*  INSERT INTO table\_name (field1, field2, field3,...) VALUES (value1, value2, value3,....) | An autonumber Primary Key can only be added by the DBMS.  Skip (ignore) this field when adding a new record. |
| DELETE FROM table\_name | DELETE FROM table\_name WHERE some\_field=some\_value  or  DELETE FROM table\_name (**Note:**Deletes the entire table!!) | Note:  DELETE \* FROM is acceptable but unnecessary. If a record is deleted, all fields in that record are also deleted.  An autonumber PK is never re-used. |
| UPDATE table\_name | UPDATE table\_name SET field1=value, field2=value,... WHERE some\_field=some\_value | An autonumber Primary Key cannot be modified.  Skip (ignore) this field when editing a record. |

|  |  |  |
| --- | --- | --- |
| WHERE | SELECT field\_name(s) FROM t1, t2  WHERE t1.PK = t2.FK  AND field\_name operator value | The WHERE clause specifies the **conditions** for selecting (filtering) **records** to display |
| HAVING | SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country HAVING COUNT(CustomerID) > 5; | The HAVING clause was added because the WHERE keyword could not be used with **aggregate functions**. |
|  | SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders, Employees WHERE Orders.EmployeeID = Employees.EmployeeID GROUP BY LastName HAVING COUNT(Orders.OrderID) > 10; |  |
| ORDER BY | SELECT field\_name(s)  FROM table\_name  WHERE …  ORDER BY field\_name [ASC|DESC] | Note:  ORDER BY comes after the WHERE clause  (ASC/DESC must be capitals) |
| GROUP BY  (used with aggregate functions - COUNT, SUM, ) | SELECT field\_name, aggregate\_function(field\_name) AS aggregate \_alias FROM table\_name WHERE field\_name operator value GROUP BY field\_name |  |

|  |  |  |
| --- | --- | --- |
| LOGICAL OPERATORS | | |
| AND / OR | SELECT field\_name(s) FROM table\_name WHERE … condition AND condition  condition OR condition  NOT condition  SELECT SupplierID, Price FROM Products WHERE Price BETWEEN 4 AND 500; | TRUE if **all** the conditions separated by **AND** is TRUE  TRUE if **any** of the conditions separated by **OR** is TRUE  Displays a record if the condition is **NOT** TRUE |
| BETWEEN … AND | SELECT field\_name(s) FROM table\_name WHERE field\_name BETWEEN value1 AND value2 | TRUE if the operand is within the range of comparisons |
| LIKE | SELECT field\_name(s) FROM table\_name WHERE field\_name LIKE pattern | TRUE if the operand matches a pattern |
| IN | SELECT field\_name(s) FROM table\_name WHERE field\_name IN (value1,value2,..) | TRUE if the operand is equal to one of a list of expressions |

|  |  |  |
| --- | --- | --- |
| STRING | | |
| Concat With & | SELECT Address & ", " & PostalCode & ", " & City AS Address FROM Customers; | The & operator allows you to add two or more strings together. |
| Format() | SELECT Format(Price, "Currency") AS FormattedPrice FROM Products;  See table below for other format names | The Format() function formats a value with the specified format. |
| Left() | SELECT Left(CustomerName, 5) AS ExtractString FROM Customers; | The Left() function extracts a number of characters from a string (starting from left). |
| Right() | SELECT Right(CustomerName, 5) AS ExtractString FROM Customers; | The Right() function extracts a number of characters from a string (starting from right). |
| Mid() | SELECT Mid(CustomerName, 4, 6) AS ExtractString FROM Customers; (4=start, 6=number of characters)  Third parameter is optional. The number of characters to extract. If omitted, this function returns all characters from the *start* position | The Mid() function extracts some characters from a string (starting at any position). |
| Len() | SELECT Len(CustomerName) AS LengthOfString FROM Customers; | The Len() function returns the length of a string. |
| UCase()  *LCase()* | SELECT UCASE(CustomerName) AS UppercaseCustomerName FROM Customers; | The UCase() function converts a string to upper-case. |

|  |  |  |
| --- | --- | --- |
| AGGREGATES - for **all records** NB. Only ONE field in the SELECT clause, except when grouping | | |
| Avg()  In sub-query | SELECT Avg(Price) AS AveragePrice FROM Products;  *Note: Format(****AVG(Price)****,"Currency") - format after calculation*  SELECT \* FROM Products WHERE Price > (SELECT Avg(Price) FROM Products); | The Avg() function returns the average value of an expression |
| Count() | SELECT Count(ProductID) AS NumberOfProducts  FROM Products;  **Note:** NULL values are not counted.  Use Count(\*) to count all records | The Count() function returns the number of records returned by a select query. |
| Sum() | SELECT Sum(Quantity) AS TotalItemsOrdered  FROM OrderDetails; | The Sum() function calculates the sum of a set of values. |
| Max() | SELECT Max(Price) AS LargestPrice FROM Products; | Returns the maximum value in a set of values |
| Min() | SELECT Min(Price) AS SmallestPrice FROM Products; | Returns the minimum value in a set of values |
| *For grouping* | SELECT group\_field, **aggregate\_function(field\_name)** FROM table\_name GROUP BY group\_field |  |

|  |  |  |
| --- | --- | --- |
| NUMERIC - for **each record** | | |
| Format() | See STRING functions above and table below |  |
| Round() | SELECT ProductName, Price, Round(Price, 1) AS RoundedPrice FROM Products;  **Note:** If the expression ends with a 5, this function rounds so that the last digit is an even number.  Round(34.55, 1) - Result: 34.6 (rounds up) Round(34.65, 1) - Result: 34.6 (rounds down) | The Round() function rounds a number to a specified number of decimal places. |
| Int() | SELECT Int(Price) AS IntNum FROM Products; | The Int() function returns the integer part of a number. |
| Sqr() | SELECT Sqr(64) AS SqrNum; | The Sqr() function returns the square root of a number. |
| Abs() | SELECT Abs(-243.5) AS AbsNum; | The Abs() function returns the absolute (positive) value of a number. |

|  |  |  |
| --- | --- | --- |
| DATE | | |
| Format() | SELECT Format(BirthDate, "Long Date") AS FormattedBirthDate FROM Employees;  Also see the | The Format() function formats a date value with the specified format. |
| Date() | SELECT CustomerName, Date() AS CurrentDate FROM Customers; | Returns the current system date |
| Day() | SELECT Day(#05/17/2017#);  SELECT Day(Date());  SELECT Day(BirthDate) FROM Employees; | Returns the day of the month for a given date |
| Month() | SELECT Month(BirthDate) FROM Employees; | Returns the month part of a given date |
| Year() | SELECT Year(BirthDate) FROM Employees; | Return the year part of a specified date |
| Now() | SELECT CustomerName, Now() AS CurrentDateTime FROM Customers; | Returns the current date and time based on the computer's system date and time |
| Time() | SELECT CustomerName, Time() AS CurrentTime FROM Customers; | Returns the current system time |
| Hour() | SELECT Hour(Now()); | Returns the hour part of a time/datetime |
| Minute() | SELECT Minute(Now()); | Returns the minute part of a time/datetime |
| Second() | SELECT Second(Now()); | Returns the seconds part of a time/datetime |