Notes

richard cattermole

Notes

richard cattermole

This book is for sale at http://leanpub.com/alphanotes

This version was published on 2016-07-27

Contents

Mathmatics

Mathmatics is the study of the universal laws fundamental concepts.

Square roots

Has two kinds of forms $\sqrt[y]{x}$ and $x^{1/y}$ when y is not provided, it is 2.

Powers and exponentials

Powers are the multiplication constant of a base value by itself. So a value of x^2 becomes $x \cdot x$ or x^3 as $x \cdot x \cdot x \cdot x$.

The transformation formats are:

- $a^4 = a \cdot a \cdot a \cdot a$.
- $a^x \cdot b^x = (a \cdot b)^x$.
- $a^x \cdot a^y = a^{x+y}$.
- $a^x - a^y = a^{x-y}$.
- $(a^x)^y = a^(x \cdot y)$.
- $a^1 = a$ evaluated as $a = a$.
- $a^0 = 1$ evaluated as $a/a = 1$.
- $a^{-x} = 1/a^x$.
- $a^{y/x} = (\sqrt[x]{a})^y$ or $a^{a/x} = \sqrt[x]{a}$.

  The n^{th} root of a.
- $1^x = 1$ evaluated as $1_0 \cdot 1_n$.
- $(-a)^x$ is evaluated as $-(a^x)$.
- $a^x = y$ can be rearranged to $y \cdot log_n = x$.

Logarithms

These statements are all correct.

- $log_a \cdot b^y = y \cdot log_a \cdot b$.
- $a \cdot_a = 1$.
- $1 \cdot log_a = 0$.
- $log_a \cdot 0 = \pm\infty (a \lesseqgtr 1)$.

Transcendental number (e)

A very common base seen in derivative mathmatics because the power of it $y = e^x$ is equal to its derivative $\frac{dy}{dx} = e^x$ value.

Derivatives

A derivative is a function based upon the output of another function.

Common forms:

- $\frac{df}{dx}$.
  - d = ignore this.
  - f = function value.
  - x = other value.
- $\frac{dx}{dt}$.
  - d = ignore this.
  - x = a unit.
  - t = time unit.

Where `d` can be replaced with `∆`.

Transformations: - $x^2 = 2 \cdot x$.

Dimensions graph (e.g. 2d)

The form used is $\frac{\Delta y}{\Delta x}$ otherwise known as the *slope*. So the full formulas is of the form:

$$\frac{\Delta y}{\Delta x} = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Common derivatives

Programming

Coding styles

MFC

| Prefix | Meaning | Example | |———|———|———| | C | Class | CConnectionPoint | | I | Interface | IConnectionPoint | | m_ | Member variable | `bool m_bSinging;` | | s_ | Static member variable | `static int s_iBudgies;` | | g_ | Global variable | `int[100] g_Budgies;` |

Hungarian convention

| Prefix | Meaning | Example | |———|———|———| | p | Pointer | `int* pCount;` | | pI | Pointer to interface | `IBird* pIBird;` ` | | `b` | Boolean |bool bBird;| | `i` | Integer |int iNumberOf-Birds;| | `dw` | Unsigned 32-bit integer |uint dwBird;| | `c` | Count |uint cBirdies;| | `sz` | 8-bit string |string szName = “Birdies”;| | `wsz` | 32-bit string |wstring wszName = “My Birdies”w;“ |

COM Introduction

COM components are not classes. Many classes or even none! Can implement a single component. A component can be represented by many interfaces. Each interface should represent behaviour not features.

No interacting with properties of a COM class. Only access via functions.

Swapping around

Rules for how to swap components:

1. Dynamically linked
2. One instance of a component implementation per process
3. Interfaces must not change between implementations/versions If an interface changed, all clients who use that interface must change also. Although this is considered not ok. They should not change.
4. New versions of a component must not break old clients
5. The location of a component does not matter, must be treated as such as part of linking process

ABI Calling convention

COM uses the standard calling convention otherwise noted as being `__stdcall` within C/C++ code. The standard calling convention is a variation of the Pascal calling convention where by the callee cleans up the stack and pushes arguments to functions from left to right. The difference between the two is that the standard calling convention pushes arugments rught to left.

Memory layout

The memory layout of a COM class is:

Offset	Purpose
0	VTBL
WORD	Data

Where VTBL is the virtual table containing pointers to each method.

Headers

```
extern(System):

enum {
        S_OK = 0x00000000,
        E_NOINTERFACE = 0x80004002
}

struct GUID {
align(1):

        uint Data1;
        ushort Data2;
        ushort Data3;
        ubyte[8] Data4;
}

alias HRESULT = uint;
alias IID = GUID;

alias LPVOID = void*;
alias CLSID = GUID;
alias REFCLSID = CLSID*;
alias LPCLSID = REFCLSID;
```

```
alias LPCOLESTR = char*;

// TODO: SUCCEEDED, FAILED

interface IUnknown {
        extern(System):

        // with the help of TypeInfo_Class it can be generated
        HRESULT QueryInterface(ref IID iid, out void* ppv);

        // this can be wrapped in a struct with postblit + destructor
        uint AddRef();
        uint Release();
}

// mine
IUnknown* CreateInsance();
HRESULT DllRegisterServer();
HRESULT DllUnregisterServer();

// not mine
extern {
        HRESULT CoCreateInstance(REFCLSID, rclsid, LPUNKOWN pUnkOuter, DWORD dwCLSConte\
xt, ref IID riid, LPVOID* ppv);
        HRESULT CLSIDFromProgID(LPCOLESTR lpszProgID, LPCLSID lpclsid);
        HRESULT ProgIDFromCLSID(REFCLSID clsid, LPOLESTR* lplpszProgID);
}
```

Registry

COM uses the registry to centrally store all shared library to its IID. These are found under *HKEY_CLASSES_ROOT*, *CLSID* keys. The default value for any key under this is the more human readable form. There is one sub key *InProcServer32* which has as the default value the location of the library.

For a more human readable naming and usage, under the components entry under *HKEY_CLASSES_ROOT*, *CLSID* there will be a ProgID sub key that will have a default value. That default value normally by convention will follow a `Program.Component.Version` format.

Reference counting

For in parameters to functions, no calling of AddRef or Release is needed as it already has been dealt with. Same with local variables.

For out parameters to functions or return values, must call AddRef before doing so.

For in + out parameters to functions, it must call Release on the input value before setting and then calling AddRef to the new value.

If unsure, add a AddRef and Release pair around the code block using it.

SQL

Some things to remember: * use database; where database is the name of the database to use * Use - for comments

Data Modeling Language (DML): drop database if exists drop database VideoDatabase if exists; create database VideoDatabase; use VideoDatabase; drop table if exists drop table if exists Video; create table Video (values) engine = InnoDB;

```
For values

ID INT not null primary key
        Add a comma if more proceed
Text VARCHAR(50) not null
        TEXT is a string of length 50
primary key (Director, publisher)
        Director and publisher are columns defined
foreign key (PersonID) references Person(ID)
```

Data Definition Language (DDL): For adding rows of data insert into table values() Grouping of rows group by table.column Functions can be used within e.g. select or where SELECT COUNT(*) from table LIKE, compare a string against another using vlaidation where column like “%has%” For updating rows update table set column=value, column2=value2 where table.id = ‘value’;

Types: All can have a number assciated with it. Which is its length e.g. * INTEGER INTEGER(1) is a boolean * VARCHAR * DECIMAL Decimals bracket takes two numbers (before period and after). The first number is the whole number part. The second is the part number. DECIMAL(3, 1) = yyy.y * DateTime Is in the format of ‘YYYY-MM-DD HH:MM:SS’ can be compared using <> * Date Is in the format of ‘YYYY-MM-DD’ can be compared using <> * Text Is any form of text, basically varchar in this case.

Indexes: An index allows for optimisations on columns to make it faster to search a table. Speed for create, update and delete is slower because it has to update the index table as well.

CREATE INDEX index_name **ON** table_name(column)name); **DROP INDEX** index_name **ON** table_name; **SHOW INDEX FROM** table_name; - shows the indexes on a table

Primary keys generally will always be an index.

Querying the database

Use database, selects a database to use from this point onwards. CREATE **VIEW** name AS, is a **sub query** that can be imported to others. **SELECT** statement is meant to gain information from the database.

Joins: * **Inner** * Inner join is a join between two tables where the reference exists beween the two tables. * **Left** * Left join is a join between two tables where a reference to the 'left' table is optional to the right table. * Will display all rows from the 'left' table. * **Right** * Right join is a join between two tables where a reference to the 'right' table is optional to the right table. * Will display all the rows for the 'right' table. * **Full** * Full join is a join between two tables where a reference between either table is optional. * Will display all rows from both 'left' and 'right' tables which if missing one will display null fields.

WHERE is a conditional statement which will filter out what is to be selected. **SORT BY** sorts the output data. It can be ascending or descending. **GROUP BY** groups the output data before processing. This is used for when e.g. counting or summing of data. **HAVING** is a clause after **GROUP BY** that enables a condition to be placed on a group of data which as been gouped. **LIMIT** number, limits the number of output for the query. **OFFSET** number, starts the query at the given number.

For selecting a column use the format of either **table.column** or **column** if joins are being used. If you wish to select only the unique values ibn a column use **DISTINCT** before it. A condition is in the form of **comparison operator comparison**. A comparison is either a value or a **column**. They can include **and**'s and **or**'s to join them. A comparison can also be a **function**. **MAX**, **SUM**, **MIN**, **COUNT**, **AVG**(average) are all available. But the yrequire a value within its brackets. For aggregate functions (that take a bunch of rows and do math on it) they can take a * for all. In a count a * refers to all rows. A **date** is in the **format** 'YYYY-MM-DD'. It is a string constant essentially. Remember single vs double quotes matter! **Mathematical oeprations** can occur at a select on a column or in a comparison. These can apply to **dates** where more then refers to after a date. **LIKE** can be used as an operator. This is a **regex comaprison** where it takes a % for 0 or more and * for 1 or more and . for optional character. **BETWEEN** enables a comparison of a range. This enables instead of writing column > x1 and column < x2. Becomes column between x1 and x2. An **alias** is of the form **column as newname** this can be of a **string** or a **value**. It also works for tables in the same form. **CONCAT** manipulates text into one value. It is a **function**.

```
- Select all people and output their name as one field and sort by their city.
A nonsense query to demonstrate this stuff:
SELECT CONCAT(p.firstname, " ", p.lastname) AS name, p.city FROM persons
AS p
SORT BY p.city;

- Get the average of all peoples balances per representatives and give us how ma\
ny customers they have.
- Display all representatives regardless of it they have any customers.
```

```
SELECT AVG(p.balance), COUNT(p.id), r.id FROM REPRESENTATIVES AS r
RIGHT JOIN persons as p ON p.repid = r.id
GROUP BY r.id;

- To get all tables in a database
USE database;
SHOW TABLES:
```

Sub queries

- **IN** is the same as = any. When used with not (not in) then it is the same as <> all.
- **ANY** is a value matched against any of the columns of the sub queries return.
- **SOME** is the same as <> any.

Example: `SQL - Gets where a person has a pllied for both cs and ee as their major. SELECT sID FROM apply WHERE major="cs" and sID = any( SELECT sID from apply where major="ee" );`

Manipulating the database

USE database, selects a database to use from this point onwards. **DROP TABLE** name **IF EXISTS**. **CREATE TABLE** name (name type(size) **NOT NULL AUTO_INCREMENT**, NOT NULL and AUTO_INCREMENT are both optional **PRIMARY KEY**(column_name), **FOREIGN KEY**(foreign_local_name) **references** foreign_table_name(foreign_remote_name)) **engine=InnoDB**

USE database, selects a database to use from this point onwards. **INSERT INTO** table (table_columns) **values** (values);

UPDATE table **VALUES**(values) **WHERE** condition; is a condtional statement which will filter out what is to be selected.

ALTER TABLE name **ADD** column_name type; or **DROP** column_name; or **ALTER** column_name type;

DELETE FROM table_name; - deletes all from table

Normalisation

Partial dependency: an attribute depends upon a primary key. **Transient dependency**: an attribute does not depend upon a primary key that can be known by it.

First normal form: 1. Remove duplicate attribute collumns 2. Remove composite attributes, make into simple attributes (can't be broken down more) 3. Create group of attribute columns aka tables 4. Set which attributes are unique and are the primary key or create one

Second normal form: 1. Create relationships between tables, using foreign keys 2. No partial dependencies (make them into tables)

Third normal form: 1. No transient dependencies (make them into tables) 2. Add foreign keys to add relationships between tables

Dependency diagram

- Straight lines from primary keys
- Arrow lines to attributes from primary keys
- Underline the attribute for primary keys
- Add P to a line to represent a partial relationship
- Add T to a line to represent a transient relationship

DBA scripting

To get the last inserted row id use: `SQL SELECT last_index_id() + (if (row_count() > 1, row_count() - 1, 0)); - last_insert_id() gets the first id in the insert statement used.`

Session variables: SET @name = value; **SELECT** @name;

If statements: `SQL IF search_condition THEN statement_list [ELSEIF search_condition THEN statement_list] [ELSE statement_list] END IF`

Triggers

NEW prefixed to a column name enables to get the new/updated value. **OLD** prefixed to a column name enables to get the original value.

```
delimiter $$;
drop trigger if exists name$$
create trigger name
before/after insert/delete/update on table_name
for each row
begin
        insert table_name
        - can do session variables here ext.
        - when using a set here @ represents session variable and a non prefixed is for\
 setting the new value.
end$$
```

Routines

```
delimiter $$;

drop procedure if exists name;
create procedure name (in name type, out ret type, ...)
begin
        select count* into ret from table_name;
end$$

delimiter ;
call proc(@var);
select @var;
```