CÁCH TẠO PROGRAM TỐT

*Lưu hành nội bộ - AIT*

Hướng dẫn tạo program tốt

Lịch sử thay đổi

|  |  |  |  |
| --- | --- | --- | --- |
| No. (Version) | Ngày | Nội dung chỉnh sửa | Người chỉnh sửa |
| 1.0.0 | 2010/05/08 | Tạo mới | Huongct |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Lời tựa**

Một program tốt phải hội đủ các yếu tố: tính đúng đắn (đúng theo Tài liệu thiết kế), tính dễ bảo trì (rõ ràng, dễ đọc, dễ hiểu, dễ sửa), tính dễ nâng cấp và tốc độ xử lý (response) tối ưu nhất.

Hướng tới mục tiêu: “Tạo ra những program tốt”, tài liệu này được ra đời như một sự chia sẽ kinh nghiệm với các SE mới, đồng thời là lời kêu gọi các SE cùng nhau thống nhất cách coding để tạo nên phong cách code riêng cho AIT.

Rất mong nhận được sự đóng góp ý kiến của các bạn để tài liệu được hoàn thiện và đạt được mục đích đã đề ra.

Nội dung chính của tài liệu gồm:

* Xác định bố cục của program.
* Những qui tắc chung cần tuân thủ.
* Những qui tắc cần tuân thủ khi code Java/VB
* Kết luận

[1. XÁC ĐỊNH BỐ CỤC CỦA PROGRAM 5](#_Toc454974196)

[1.1. Phân tích và hiểu tài liệu thiết kế: 5](#_Toc454974197)

[1.2. Quyết định bố cục của program 6](#_Toc454974198)

[1.2.1. Xác định phần chung 6](#_Toc454974199)

[1.2.2. Xác định kỹ thuật 6](#_Toc454974200)

[1.2.3. Quyết định bố cục của chương trình 6](#_Toc454974201)

[2. TUÂN THỦ QUY TẮC CHUNG KHI CODE 6](#_Toc454974202)

[2.1. Tuân thủ CODE-CONVENTION 6](#_Toc454974203)

[2.1.1. Cách đặt tên 7](#_Toc454974204)

[2.1.2. Không viết dòng code quá dài: 7](#_Toc454974205)

[2.1.3. Không hard-code: 7](#_Toc454974206)

[2.1.4. Không sử dụng goto 8](#_Toc454974207)

[2.1.5. Không thay đổi giá trị của parameter truyền vào: 8](#_Toc454974208)

[2.1.6. Comment đầy đủ, ngắn gọn, rõ ràng: 8](#_Toc454974209)

[2.1.7. Dòng code đơn giản, rõ ràng: 9](#_Toc454974210)

[2.1.8. Tạo sqlString: 9](#_Toc454974211)

[2.1.9. Không để source code thừa: 9](#_Toc454974212)

[2.1.10. Cân nhắc các trường hợp gây exception: 10](#_Toc454974213)

[2.2. Nguyên tắc truy cập DB 10](#_Toc454974214)

[2.2.1. Điều khiển transaction 10](#_Toc454974215)

[2.2.2. Đóng connection 11](#_Toc454974216)

[2.2.3. Nguyên tắc khi tạo SQL 11](#_Toc454974217)

[2.3. Tuân thủ qui trình PDCA (PLAN-DO-Check-ACTION) 13](#_Toc454974218)

[2.3.1. Tuân thủ qui trình PDCA (PLAN-DO-Check-ACTION): 14](#_Toc454974219)

[3. KẾT LUẬN 14](#_Toc454974220)

# XÁC ĐỊNH BỐ CỤC CỦA PROGRAM

## Phân tích và hiểu tài liệu thiết kế:

Vì bị ức chế về thời gian, hoặc một số lý do khác, đa phần các bạn thường bắt tay vào code ngay sau khi đọc tài liệu thiết kế chi tiết mà chưa phân tích và hiểu rõ mình phải làm gì? Vì sao và làm như thế nào? Đây là một thói quen không tốt, nó là nguyên nhân dẫn đến báo cáo tiến độ sai, mất “kiểm soát” đối với PG mình phụ trách, chất lượng PG kém.

Bạn chỉ có thể làm tốt một việc nếu như bạn biết rõ việc đó là việc gì? Bạn phải làm gì và làm như thế nào. Để tạo được program tốt, trước tiên bạn phải hiểu được Bản thiết kế yêu cầu bạn những gì, và bạn sẽ thực hiện những yêu cầu đó như thế nào, theo trình tự nào. Như vậy, việc đầu tiên là bạn phải đọc và hiểu được Tài Liệu thiết kế chi tiết (TLTK).

Có một cách để phân tích, hiểu TLTK khá tốt là áp dụng Kỹ thuật tư duy 5W1H. Bằng cách đọc và cố gắng trả lời được các câu hỏi What? Where? When? Why? Who? How?. Quá trình áp dụng cách phân tích 5W1H bao gồm các bước sau:

Bước 1: Phân tích trên cơ sở xem PG như 1 đơn vị xử lý trong toàn hệ thống lớn. Kết hợp TLTK và tài liệu thiết kế cơ bản, hoặc thông qua người quản lý dự án, để tìm ra được mối tương quan giữa PG này trong toàn hệ thống lớn. Qua bước này, bạn cũng sẽ xác định được những tài liệu, những người liên quan (gọi là xác định TeamWork) mà bạn có thể tham khảo để xác định rõ những điểm chưa rõ ràng hoặc chưa được đề cập đến trong TLTK.

* What?: chức năng chính của PG này là gì? Mỗi một PG thường là 1 xử lý trong một hệ thống lớn. Bạn phải xác định được PG mà bạn phụ trách nó có nhiệm vụ xử lý gì? Đăng ký hay modify, hay xóa dữ liệu? import dữ liệu hay xuất report? Hay chỉ đơn giản là check dữ liệu?...
* Where?: Trong sơ đồ toàn hệ thống, thì PG của bạn ở vị trí nào?
* When?: PG này được chạy khi nào? PG nào sẽ kích hoạt PG này? Và sau khi PG này kết thúc, PG nào sẽ được kích hoạt?
* Why?: Tại sao PG này lại được kích hoạt từ PG A mà không phải là PG khác? Tại sao PG này phải được thực hiện trước sau đó PG B mới được gọi?
* How?: Làm thế nào để PG này thực hiện được chức năng chính của nó? Đối tượng dữ liệu IN/OUT như thế nào? Có giải thuật nào đặc biệt được nêu ra trong TLTK không?
* Who?: Đối tượng sử dụng PG này là gì? -> xác định người sử dụng cuối.

Bước 2: Phân tích trên cơ sở chia PG thành những xử lý nhỏ theo sự kiện, chức năng DB…Bạn sẽ phải đọc kỹ tài liệu để hiểu được những xử lý chi tiết bên trong PG này. Ở bước này, bạn sẽ phát hiện ra những điểm sai, những điểm chưa rõ ràng trong TLTK, thông qua TeamWork (đã xác định ở bước 1) để làm sáng tỏ các vấn đề đó, nếu như vẫn không xác định rõ được thông tin cần thiết thì tiến hành đặt QA.

* What?: PG gồm có những xử lý gì? Lấy dữ liệu master? Check dữ liệu nhập? Lưu dữ liệu từ màn hình xuống DB? Có thể list ra danh sách theo trình tự trong tài liệu. Có những TLTK tốt, bạn sẽ dễ dàng trả lời câu hỏi này, tuy nhiên cũng có những tài liệu trình bày khó hiểu, khó năm bắt, đòi hỏi bạn phải đọc kỹ, và tổng hợp lại mới có thể xác định được.
* Where?: Thông tin này có được từ đâu? Điều kiện này được thiết lập ở đâu? Dữ liệu ban đầu lấy từ đâu, từ kết quả xử lý trước nó, hay từ table master?, dữ liệu nhập sẽ đăng ký xuống table nào?
* When?: Xử lý được thực hiện khi nào? Khi khởi động ban đầu, hay ứng với 1 sự kiện, hoặc 1 điều kiện nào đó? Xử lý trước nó là gì? Xử lý sau nó là gì?
* Why?: Tại sao có đều kiện này? tại sao thực hiện xử lý 2.1 trước và thêm điều kiện 2.2 nữa mới thực hiện 2.3? ….
* How? Làm thế nào để xác định thông tin này? Có hàm common nào hỗ trợ? Giải thuật nào được áp dụng cho xử lý này? Làm thế nào kết nối giữa các xử lý?
* Who?: Đối tượng dữ liệu IN/OUT của các xử lý? Xử lý này sẽ làm thay đổi thông tin nào?

## Quyết định bố cục của program

### Xác định phần chung

Sau khi hiểu rõ TLTK, hãy xác định những nội dung xử lý nào được sử dụng nhiều lần, nhiều nơi thì tập hợp lại để common hóa bằng cách tạo các method/function sử dụng chung. Những nội dung xử lý giống nhau nếu để ở nhiều nơi trong program, khi có nhu cầu chỉnh sửa, phải sửa nhiểu nơi dẫn tới tốn nhiều thời gian đồng thời khả năng phát sinh bug level-down rất cao.

### Xác định kỹ thuật

Hầu hết các dự án phát triển ở AIT chỉ dừng lại ở mức áp dụng kỹ thuật dựa trên 1 framework định sẵn, vì thế bạn sẽ không phải lựa chọn kỹ thuật. Tuy nhiên, trong chừng mực có thể, bạn có thể quyết định những nội dung như: xử lý nào sẽ thực hiện ở đâu, xử lý nào sẽ viết ở client (javascript, jsp, aspx..), xử lý nào viết bằng SQL, PL/SQL? Xử lý nào sẽ viết bằng Java/VB? Class nào? Tầng nào?….Việc xác định trước kỹ thuật sẽ góp phần quyết định bố cục của program

.

### Quyết định bố cục của chương trình

Việc xây dựng bố cục program hợp lý, rõ ràng sẽ nâng cao tính chuyên nghiệp, tính bảo trì của PG. Trên cơ sở chia program thành những đơn vị xử lý nhỏ độc lập đến mức có thể, để khi có nhu cầu chỉnh sửa, bổ sung…sẽ thực hiện một cách dễ dàng nhất, không làm mất đi bố cục đã định ban đầu, và không tốn nhiều thời gian.

Bố cục của program thường được thực hiện khi vẽ sequence –chart. Cụ thể, bạn phải quyết định được: sẽ tạo bao nhiêu class? Bao nhiêu trang jsp/aspx? Bao nhiêu form? Bao nhiêu method/function? Tham số truyền nhận giữa các xử lý như thế nào?

Thực tế khi tiến hành PG dựa trên framework cụ thể, bạn không phải quyết định số class, nội dụng class phải tạo, tuy nhiên, bạn thường xuyên phải quyết định: đối với 1 xử lý cụ thể, bạn sẽ phân chia thành bao nhiêu method/function, xử lý nào sẽ là common, vì thế hãy cố gắng suy nghĩ và cân nhắc để phân chia metod/function cách hợp lý nhất.

Thông thường, một xử lý ứng với một bộ dữ liệu IN/OUT nhất định được qui về một method/function. Bạn không nên chia quá chi tiết, quá nhiều method/function sẽ làm cho program của bạn trở nên rối rắm, khó theo dõi, khó kiểm soát và khả năng phát sinh bug cao.

# TUÂN THỦ QUY TẮC CHUNG KHI CODE

## Tuân thủ CODE-CONVENTION

Khi bạn coding Java, VB hay bất cứ ngôn ngữ lập trình nào, bạn cũng phải tuân thủ những qui định chung về cách viết code để đảm bảo tính bảo trì của source-code. Mỗi dự án, thường có những qui định riêng về cách coding mà bạn phải tuân thủ, bên cạnh đó, một số qui định chung sau đây bạn cũng cần phải tuân thủ. Chi tiết ứng với từng ngôn ngữ lập trình thì xem ở mục số 3. QUI TẮC CODE JAVA và 4. QUI TẮC CODE VB.

### Cách đặt tên

Cơ bản, tên phải dễ hiểu, dễ nhớ, phản ánh được đầy đủ bản chất của nội dung.

Khi đặt tên cho method/function phải cân nhắc đến tính đối xứng: setXX()/getXX()…

Không đặt tên trùng nhau cho những scope khác nhau. Ví dụ: đặt tên biến local và global trùng nhau…

Đặt tên cho các control/item màn hình theo loại control. Tóm tắt như sau:

|  |  |  |
| --- | --- | --- |
| Loại control/item | Tiền tố | Ví dụ |
| Nút | btn | btnAdd |
| Label | lbl | lblProgress |
| Text | txt | txtRegCode |
| Check box | chk | chkDeleted |
| Nút radio | rdo | rdoRedColor |
| Combo box | cmb | cmbDept |
| List box | lst | lstOders |
| List view | lsv | lsvMembers |

### Không viết dòng code quá dài:

Đối với dòng code quá dài, khi dọc code cần phải kéo scroll của Editor để đọc được toàn bộ dòng, như vậy sẽ tốn thời gian và cũng dễ gây ra nhầm lẫn, và tiềm ẩn nguy cơ phát sinh bug.

Không nên sử dụng quá nhiều câu lệnh if lồng nhau. Trường hợp câu lệnh if mà có nhiều hơn 2 điều kiện else thì nên sử dụng switch hay select…tương ứng

### Không hard-code:

Không code “cứng” trong source code (dù là hằng chuỗi hay hằng số ). Thay vào đó , hãy định nghĩa thành những biến constants , biến static read-only , định nghĩa enumeration/enum , hoặc đọc những giá trị từ những file cấu hình hay những file resource

Ví dụ:

' WindowState

Public Const NORMAL As Integer = 0 ' 0 - ノーマル

Public Const MINIMIZED As Integer = 1 ' 1 - アイコン化

Public Const MAXIMIZED As Integer = 2 ' 2 - 最大表示

'手配区分

Public Const FE\_TEKBN\_PREP As String = "2" '製版

Public Const FE\_TEKBN\_INSA As String = "4" '印刷

Public Const FE\_TEKBN\_KAKO As String = "6" '加工

### Không sử dụng goto

Sử dụng goto làm mất đi bố cục của program, gây khó đọc, khó bảo trì, vì thế tuyệt đối tránh sử dụng.

### Không thay đổi giá trị của parameter truyền vào:

Về cơ bản, parameter truyền có thể đồng thời đóng 2 vai trò IN/OUT luôn, tuy nhiên, nếu sử dụng cùng một biến cho nhiều mục đích khác nhau sẽ dễ gây nhầm lẫn, khó hiểu, đây cũng là tiềm ẩn của bug, vì thế phải tránh.

### Comment đầy đủ, ngắn gọn, rõ ràng:

Bất cứ xử lý nào cũng phải có comment. Nội dung comment phải phản ánh được nội dung trả lời câu hỏi WHAT-WHY-WHEN. Nội dung comment được diễn giải quá dài hoặc quá ngắn đều làm cho comment trở nên khó hiểu, hoặc hiểu sai. Cách tốt nhất là bạn có thể trích dẫn nội dung của tài liệu để đưa vào comment, nhớ ghi luôn số thứ tự hạng mục trong tài liệu luôn.

Ví dụ: //1.3.1 Check right for updating….

Ngoài việc comment nội dung, bạn cũng phải comment thông tin quản lý lịch sử: người tạo, ngày giờ tạo… điều này thể hiện tinh thần trách nhiệm, cũng như một sự chứng nhận về sản phẩm mà bạn tạo ra.

Tóm lại, khi comment cần lưu ý một số điểm sau :

* Comment nên đơn giản, ngắn gọn, súc tích.
* Phải nêu được chức năng, ý nghĩa của class hay đoạn code
* Các comment phải thống nhất với nhau trong cùng một class, method…
* Comment nên phải viết ở dòng trước đoạn code. Trong trường hợp viết phía sau code thì comment phải nằm trên 1 dòng với code.
* Comment không được quá dài trên 1 dòng(trường hợp comment quá dài thì nên dùng khối comment

### Dòng code đơn giản, rõ ràng:

Trong trường hợp bạn có nhiều lựa chọn ứng với nhiều giải thuật khác nhau thì hãy chọn giải thuật đơn giản, dễ hiểu nhất để code (tất nhiên bạn phải cân nhắc đến response của xử lý).

### Tạo sqlString:

Sử dụng nguyên tắc sau để tạo SQL:

sqlStr = “ ”

sqlStr += “ SELECT field1 ”

sqlStr += “ ,field2”

sqlStr += “ ,field3”

sqlStr += “ ,field4”

…….

sqlStr += “ FROM Table1”

sqlStr += “ ,Table2”

sqlStr += “ ,Table3”

…….

sqlStr += “ WHERE 1 = 1”

sqlStr += “ AND DK1”

sqlStr += “ AND DK2”

……

Lưu ý:

* Các từ khóa SELECT, FROM, WHERE, ORDER BY…phải được canh thẳng cột với nhau.
* Các field select, các table, các điều kiện .. được canh thẳng cột với nhau một cách tương ứng..
* Các từ khóa kết nối điều kiện: AND, OR …canh thẳng cột với nhau

Áp dụng nguyên tắc này làm cho câu SQL dễ đọc, dễ hiểu và sẽ tránh được những lỗi thường xảy ra khi tạo chuỗi SQL, đặc biệt đối với những câu SQL được sinh ra từ nhiều điều kiện phức tạp như là:

* sqlStr được tạo ra sai cú pháp do bị thiếu hoặc thừa dấu “,”
* sqlStr được tạo ra sai cú pháp do không có khoảng trắng giữa các từ khóa SELECT, FROM, WHERE, AND..

Ngoài ra, khi sử dụng câu SQL sinh ra để test bằng PL/SQL Develop bạn cũng sẽ tiết kiệm được thời gian khi thay đổi điều kiện test: chỉ cần comment những điều kiện không cần mà không cần thay đổi gì thêm.

### Không để source code thừa:

Hãy xóa tất cả những đoạn source code không có ý nghĩa, không được sử dụng. Việc comment lại những đoạn source không sử dụng làm cho source trở nên “cũ kỹ”, rối rắm, gây tâm lý khó chịu cho người bảo trì, và nhiều khả năng gây hiểu nhầm, vì thế, phải thực hiện xóa toàn bộ source code không sử dụng.

Ví dụ:

if (returnFlag == true) {

return “a”;

} else {

return “b”;

}

Đoạn code trên nhìn vào thì rất dể hiểu nhưng lại dư thừa. Trường hợp returnFlag=false thì phải mất thêm thao tác kiểm tra với else.

Với đoạn code trên thì nên viết như sau:

if (returnFlag == true) {

return “a”;

}

return “b”;

### Cân nhắc các trường hợp gây exception:

Tất cả các dòng code phải được đặt trong try…catch… và điều khiển exception chủ động (ví dụ write ra console hoặc ghi log…..), việc này giúp ích khi test.

Khi thao tác trên 1 đối tượng dữ liệu, phải cân nhắc đến những tình huống có thể làm cho đối tượng đó bị NULL để thiết lập điều kiện xử lý thích hợp

Ví dụ: Khi lấy giá trị trong object thì kiểm tra xem object đó có NULL hay không:

If(object != null){

Return object.value();

} else {

Return 0;

}

## Nguyên tắc truy cập DB

### Điều khiển transaction

Hầu hết các các dự án, việc điều khiển transaction được thực hiện bởi Framework, tuy nhiên, nếu như Framework không hỗ trợ, bạn phải tuân thủ những điểm sau:

Một transaction bắt đầu với StartTransaction và kết thúc bằng Commit transaction/Rollback transaction.

Ngoại trừ rollback transaction có chủ đích, việc rollback transaction được thự hiện khi phát sinh lỗi, vì thế câu lệnh rollback luôn được đặt trong catch(exception)

Việc commit transaction chỉ được thực hiện khi tất cả các xử lý trong cùng transaction thành công, vì thế sẽ được thực thiện sau cùng.

Try

//start transaction ở đây

…

// Thực hiện các xử lý trong cùng transaction

//trước catch thì thực hiện commit

Catch

//thực hiện rollback

Finally

End Try

### Đóng connection

Khi bạn không còn nhu cầu sử dụng connection nữa, hãy thực hiện close() connection đó đi.

Tuy nhiên, nhớ rằng, bất cứ truy cập DB nào sau câu lệnh đóng connection sẽ gây lỗi, vì thế, cách an toàn nhất, hãy thực hiện lệnh đóng connection trong Finally

### Nguyên tắc khi tạo SQL

#### Trình tự các table trong mệnh đề FROM

Trình tự Oracle xử lý các table được chỉ ra trong mệnh đề FROM là từ Phải qua Trái (RIGHT-LEFT). Vì vậy, hãy đặt những table tồn tại dữ liệu ít hơn theo trình tự từ phải qua Trái.

Ví dụ: Table TAB1: 16,384 records

Table TAB2: 1 record

SELECT COUNT(\*) FROM TAB1, TAB2 -> 0.96 giây

SELECT COUNT(\*) FROM TAB2, TAB1 -> 26.09 giây

#### Trình tự các điều kiện trong mệnh đề WHERE

Trình tự Oracle xử lý các điều kiện được thiết lập trong mệnh đề WHERE là từ Dưới lên Trên (BOTTOM-TOP). vì vậy, những điều kiện trích xuất ra tập dữ liệu lớn thì hãy để trước.

Ví dụ:

a/ Cách viết không tối ưu:

SELECT ….

FROM EMP E

WHERE SAL > 2000

AND JOB = ‘MANAGER’

AND 25 < ( SELECT COUNT(\*)

FROM EMP

WHERE MGR = E.EMPNO)

b/ Cách viết tối ưu:

SELECT ….

FROM EMP E

WHERE 1 = 1

AND 25 < ( SELECT COUNT(\*)

FROM EMP

WHERE MGR = E.EMPNO)

AND SAL > 2000

AND JOB = ‘MANAGER’

#### Luôn luôn viết câu SQL có ORDER BY

Ở hệ QTCLSL Oracle, khi viết câu SQL không có dòng lệnh ORDER BY thì mặc nhiên hệ quản trị Oracle đã mặc định là hạng mục ID số tăng tự động

Đối với hệ QT CLDL khác nhu PostGres thì không tự động ORDER BY như trên. Vậy phải luôn viết ORDER BY để khi gặp các trạng thái phải chuyển đổi hệ QTCSDL thì không phải chỉnh sửa câu SQL

#### Không dùng GetRS List cho trường hợp Câu SQL lấy 1 dòng trên danh sách

Để tránh sự dư thừa gây lãng phí, trường hợp chỉ lấy dòng đầu tiên của danh sách có thể bổ sung thêm điều kiện vào SQL hoặc là sử dụng các hỗ trợ như TOP, FIRST, MAX phù hợp

#### Sử dụng hữu hiệu các hàm số cột

Sử dụng hữu hiệu các hàm số kết hợp như là ＭＡＸ, ＭＩＮ, ＡＶＧ, ＣＯＵＮＴ, ＳＵＭ

#### Chỉ định rõ tên cột một cách tường minh

Chỉ định rõ tên cột khi thực hiện select/insert/order by, khi table layout thay đổi sẽ dễ dàng cập nhật.

Chuỗi chữ phải được để trong dấu “’”

Khi thiết lập điều kiện ứng với chuỗi chữ, bạn phải để chuỗi chữ trong dấu nháy (‘). Điều này cũng giống như xác định ép kiểu dữ liệu cách tường minh khi select. Tránh phát sinh những lỗi chỉ phát sinh khi thực thi câu SQL.

Ví dụ: Xét ví dụ sau, với MSTREQUEST.CODE\_KBN có kiể dữ liệu là VARCHAR(1)

SELECT

MREQ.REQUEST\_DATA\_NAME

,MREQ.ACCOUNT\_NO

,MREQ.ACCOUNT\_SBT

,MREQ.BANK\_CD

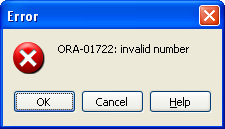
FROM MSTREQUEST MREQ

WHERE MREQ.DELETE\_YMD IS NULL

AND MREQ.CODE\_KBN = 1

Nếu như dữ liệu thực tế của cột MSTREQUEST.CODE\_KBN chỉ là các ký số: 1, 2, 0,… thì khi thực hiện câu SQL trên sẽ không có vấn đề.

Tuy nhiên, nếu như dữ liệu thực tế có tồn tại giá trị = blank hoặc ký tự không phải ký tự số, khi thực hiện câu SQL trên sẽ bị lỗi:



#### Sử dụng hữu hiệu index để tăng response

Khi tạo SQL hãy cân nhắc để sử dụng hiệu quả index của DB, nhằm tăng response.Xem xét một số ví dụ sau về tính hữu hiệu hay vô hiệu của index khi thiết lập điều kiện WHERE như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Tình huống sử dụng | Không phát huy vai trò index | Phát huy index, tăng response |
| 1 | Cột chỉ định trong  biểu thức tính toán | Select ..  From emp  Where sal\*1.5 > 3000 ; | Select ..  From emp  Where sal > 3000/1.5 ; |
| 2 | Search NULL | Select ename  From emp  Where sal is not null ; | Select ename  From emp  Where sal >= 0 ; |
| 3 | Not Equal | Select ename  From emp  Where sal != 20 ; | Select ename  From emp  Where sal >= 21 ; |
| 4 | SubStr() | Select ecode  ,ename  From emp  Where substr(ename, 1, 1) = 'C' ; | Select ename  From emp  Where ename like 'C%' ;  ※ Tuy nhiên: like ‘%C%’ thì  index không hữu hiệu nữa. |
| 5 | Not in, in | Not in, In | Not Exist, Exist |
| 6 | >= | Select ename  From emp  Where sal > 20 ; | Select ename  From emp  Where sal >= 21 ; |
| 7 | || | Select acc\_name  From Account  Where  Acc\_Sbt || Acc\_kbn = ‘0012’ | Select acc\_name  From Account  Where  Acc\_Sbt = ‘001’  And Acc\_kbn = ‘2’ |
| 8 | OR | Select acc\_name  From Account  Where  Acc\_Sbt = ‘001’ Or Acc\_Sbt = ‘002’ Or Acc\_Sbt = ‘003’ | Select acc\_name  From Account  Where  Acc\_Sbt In (‘001’, ‘002’,’003’) |

Tài liệu tham khảo: Reference\ How to write efficient SQL Queries.pdf

#### Không lấy giá trị MAX bằng ORDER BY ASC

Vì giá trị null được xử lý như là giá trị tối đa khi order by nên cần chú ý, cách lấy giá trị lớn nhất của một cột chính là sử dụng hàm MAX, chứ không phải ORDER BY ASC

## Tuân thủ qui trình PDCA (PLAN-DO-Check-ACTION)

Những điểm không tốt đối với thói quen viết code xong toàn bộ sau đó mới chạy thử, test:

Phát sinh rất nhiều bug khi chạy thử, khi test unit. Vì tất cả source đều chưa được chạy thử, việc phát sinh nhiều lỗi là điều hiển nhiên.

Tốn nhiều thời gian để tìm lỗi, sửa lỗi. Cùng một lỗi có thể sẽ phải sửa nhiều nơi, nhiều chỗ và khả năng leveldown cũng rất cao.

Gây tâm lý mệt mỏi, đuối sức. Có những lỗi có nguyên nhân từ một lỗi khác, việc truy tìm đâu là nguyên nhân cội rễ sẽ mất nhiều sức lực và tâm trí hơn, vì thế dễ dẫn đến sự mệt mỏi, đuối sức.

### Tuân thủ qui trình PDCA (PLAN-DO-Check-ACTION):

Nguyên tắc cơ bản: Hãy kiểm soát tốt từng dòng code bạn vừa viết.

Thời gian để tìm kiếm và sửa lỗi trên phạm vi hẹp thì nhanh, hiệu quả hơn nhiều so với tìm kiếm và sửa lỗi trên phạm vi rộng hơn.

Trên nguyên tắc này, hãy áp dụng PDCA cho từng đơn vị xử lý nhỏ (method/function..) :

* PLAN : Bạn định viết gì? Chức năng của đoạn code, của method/function là gì? Nó sẽ xử lý nội dung gì?
* DO: Tiến hành code, cụ thể hóa nội dung xử lý bằng dòng những dòng code.
* CHECK: check xem đoạn code/ method/function vừa code có đáp ứng yêu cầu đưa ra ở bước PLAN chưa? Check cú pháp bằng cách compile, và check logic tương đối bằng cách chạy thử.
* ACTION: tìm nguyên nhân và sửa những lỗi đã phát hiện ở bước CHECK. Thời điểm này bạn cũng sẽ tiếp tục phát hiện ra nhiều vấn đề chưa hợp lý của tài liệu và đặt QA.

Thực hiện qui trình trên triệt để sẽ phát hiện và sửa lỗi đúng lúc, bạn có thể rút kinh nghiệm trên từng dòng code, và hạn chế các lỗi phát sinh lại, giảm thiểu thời gian sữa lỗi.

Khi hoàn tất cả program, công việc còn lại được tập trung để test chức năng chính của program. Đây chính là ý nghĩa thực sự của UNIT TEST.

# KẾT LUẬN

Một program tốt là một program hội đủ các yếu tố: đúng đắn, dễ bảo trì, dễ nâng cấp.

Để đảm bảo tính đúng đắn của Program, không gì khác hơn phải hiểu TLTK một cách đầy đủ, rõ ràng và chính xác nhất. Hãy giành thời gian phân tích TLTK để biết bạn phải làm gì, làm như thế nào.

Để đảm bảo tính Dễ Bảo Trì, bạn phải đảm bảo: bất cứ ai cũng có thể đọc hiểu và sửa source code của bạn, thời gian để chỉnh sửa là thấp nhất. Vì thế, bạn phải tuân thủ các qui ước chung khi code.

Để đảm bảo tính Dễ Nâng Cấp, hãy thiết kế bố cục program ở tầm nhìn xa hơn, đảm bảo khi có nhu cầu thay đổi, thêm chức năng thì program của bạn vẫn trong sáng, rõ ràng và chi phí nâng cấp là ít nhất. Cố gắng phân tính và phân chia các đơn vị xử lý để đảm bảo tính độc lập tương đối.

Một thao tác quan trọng không thể thiếu để tạo ra một program tốt là: TEST. Hãy tuân thủ qui trình PDCA ngay từ những dòng code đầu tiên để đảm bảo tính đúng đắn của từng dòng code đồng thời có thể phát hiện và xử lý được những vấn đề kịp thời và ít tốn kém nhất.

Sẽ có nhiều cách khác nhau để tạo ra những program tốt, tuy nhiên để tạo nên “Program tốt” như là một phong cách riêng AIT, hướng đến sự thỏa mãn của khách hàng, đòi hỏi sự hợp tác và nỗ lực của tất cả tập thể.

Kinh nghiệm của cá nhân người viết rất hạn hẹp….vì thế rất mong nhận được sự đóng góp, chia sẽ kinh nghiệm từ các bạn. Chân thành cảm ơn!.