# [Bí kíp chân truyền của BA](https://thinhnotes.com/chuyen-nghe-ba/bi-kip-chan-truyen-cua-ba/" \o "Bí kíp chân truyền của BA)

**1. Ở mọi mặt trong đời sống**

Đúng, 2 thứ này tồn tại ở mọi mặt trong đời sống.

Ví dụ.

**Cái bàn** gồm 2 components: ***mặt bàn*** và ***chân bàn***.  
**Cái ghế** cũng gồm 2 components: ***mặt ghế*** và ***chân ghế***.  
**Cái lu** cũng gồm 2 components: ***thân lu*** và ***nắp lu***.

Cả 2 component trong các đối tượng trên đều có relationship chặt chẽ với nhau.

Có mặt bàn, mà không có chân bàn, thì không thể tạo thành cái bàn. Tương tự, có mặt ghế mà không có cái chân ghế thì cũng không thể tạo thành cái ghế.

Nhưng cái lu thì khác…

Có thân lu, mà không có nắp lu, thì vẫn tạo ra được cái lu. Vì có nhiều cái lu không có nắp lu mà vẫn xài tốt được đấy thôi.

Do đó, relationship giữa các component cho thấy được chức năng của đối tượng đó.

Cái bàn **là để** đặt đồ vật lên trên cao, cách xa mặt đất. Không có mặt bàn thì không đặt đồ vật lên được. Và không có chân bàn thì không thể để vật lên trên cao, cách khỏi mặt đất được.

2 component này phải đi liền với nhau thì mơi tạo ra cái bàn có chức năng: *đặt đồ vật lên cao* và *cách xa mặt đất được.*

Tương tự là cái ghế. Phải có chân ghế **thì** mới ngồi lên cao được. Và ghế mà không có mặt ghế, thì cũng chả ai ngồi được.

Nhưng cái lu thì khác…

Chỉ cần thân lu là có thể đựng được nước. Nắp lu chỉ là một ***“tính năng bổ trợ”*** giúp nước trong lu sạch hơn mà thôi. Thiếu cái nắp thì cái lu vẫn đựng được nước, không nhằm nhò gì.

Hoặc một ví dụ khác.

Nhà vệ sinh có 2 gian: 1 gian để tắm và 1 gian để đi wc. Đây cũng chính là 2 components của một nhà vệ sinh.

2 components này quan hệ song song với nhau. Cả 2 cùng tồn tại, và có thể tồn tại độc lập nếu thiếu một trong hai.

Mặc định, chúng ta sẽ nói: **nhà vệ sinh này vừa dùng để tắm, vừa dùng để đi wc**. Đây là 2 chức năng của nhà vệ sinh.

Nhưng nếu bỏ bớt 1 trong 2 component, **chức năng** của nhà vệ sinh sẽ bị *cắt đi 1 nửa.*

.

.

.

Nếu ánh xạ điều này vào thế giới phần mềm thì nó cũng i chang như vậy.

Nói đến lập trình hướng đối tượng. Đối tượng ở đây chính là các component. Và các component này có relationship chặt chẽ với nhau. Đây là 2 yếu tố chính yếu tạo nên bất kỳ phần mềm nào.

**2. Component**

Trong một hệ thống quản lý *whatever*, giả sử chúng ta sẽ lấy khách hàng ra làm trọng tâm. Và xem nó như component đầu tiên trong system của mình.

Khách hàng thì có nhiều dạng. Tổng quát nhất thì có:

* Khách hàng cá nhân
* Khách hàng doanh nghiệp.

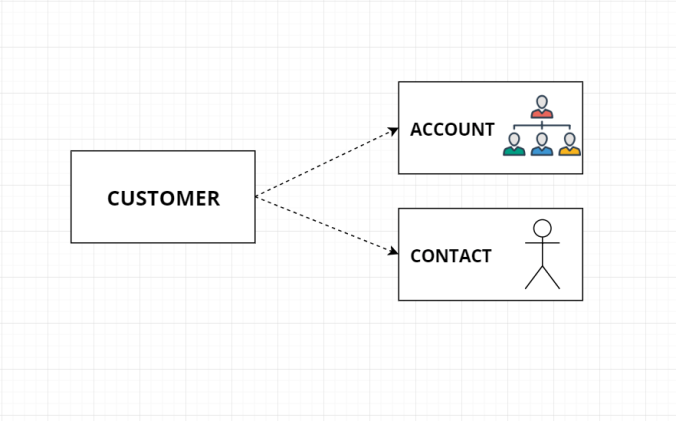
Khách hàng cá nhân đại diện cho 1 cá thể riêng lẻ. Khách hàng doanh nghiệp đại diện cho 1 công ty, hoặc 1 tổ chức abc xyz nào đó.

Nhưng nếu là tổ chức hoặc công ty thì nó cũng chỉ là thứ vô tri vô giác. Trong khi thứ cần quản lý là các mối quan hệ, là các tương tác giữa người với người.

Do đó, anh em cần phải quản lý luôn: ai là người liên lạc, người đại diện cho công ty đó. Tức là có dính tới các cá thể riêng lẻ.

Vậy ở đây, component khách hàng sẽ được mình chia thành 2 component nhỏ:

* Account: là khách hàng doanh nghiệp
* Contact:
  + Là khách hàng cá nhân
  + Hoặc người liên lạc/ người đại diện cho khách hàng doanh nghiệp.



**Đại khái nó sẽ như vầy.**

Giờ chúng ta sẽ xét tới relationship giữa Account và Contact.

**3. Relationship**

Thường thì 1 công ty sẽ có nhiều người liên hệ. Do đó, mối quan hệ đầu tiên ở đây là **1 Account gồm nhiều Contact**, *quan hệ một – nhiều*.

Tiếp theo xét tới Contact.

Một người (người đại diện/ người liên hệ) có thể thuộc nhiều công ty khác nhau hay không? *Tức là tại một thời điểm, người đó có thể cùng làm ở nhiều công ty khác nhau hay không? Hay chỉ có thể làm tại một công ty.*

Điều này sẽ phần nào **nói lên được business** của khách hàng **hoạt động như thế nào**, à há.

Nói lên chỗ nào?

Thực tế một người có làm ở cùng 1 công ty, hay làm ở nhiều công ty khác nhau hay không, không quan trọng. Quan trọng là ở business của khách hàng quy định ra sao.

Nếu 1 Account gồm nhiều Contact khác nhau. Và 1 Contact có thể thuộc nhiều Account khác nhau. Tức **Account – Contact** quan hệ **nhiều – nhiều**. Thì nó cho thấy được đặc điểm của Contact này là: *nó có thể cùng thuộc nhiều công ty, nhiều đơn vị khác nhau.*

Từ đó chúng ta sẽ hiểu được một số **đặc thù về quy trình nghiệp vụ** của khách hàng >> dẫn tới một số **đặc thù về quy trình của hệ thống** >> phần nào nói lên được **chức năng** của hệ thống là gì.

Còn nếu 1 Account gồm nhiều Contact khác nhau. Nhưng 1 Contact chỉ có thể thuộc duy nhất 1 Account. Tức **Account – Contact** quan hệ **một – nhiều**. Thì nó nói lên được: *Contact này chỉ có thể đại diện cho 1 công ty, 1 tổ chức nhất định.*

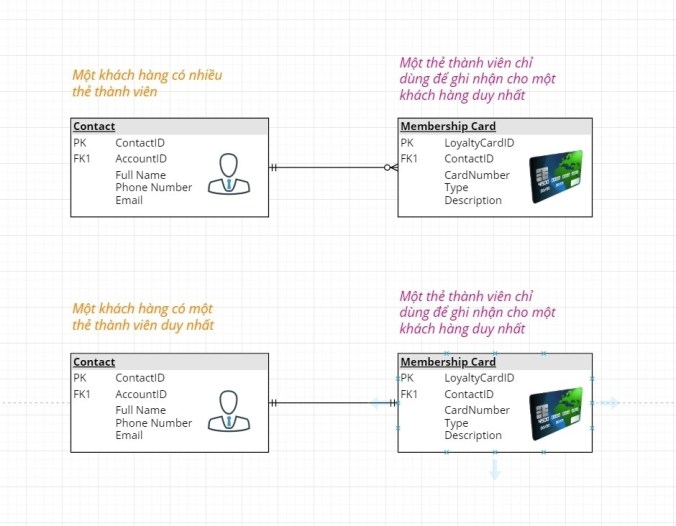
Nếu làm việc với Contact này, thì chúng ta đang làm việc với 1 Account bất kỳ. Nó cũng dẫn tới một số đặc thù về quy trình nghiệp vụ của khách hàng.

Hoặc một ví dụ khác trực quan hơn: ***Contact****và****Membership Card.***

Đây là 2 components trong một hệ thống quản lý khách hàng thân thiết *(loyalty management).*Anh em hãy xét tới quan hệ giữa Contact và Membership Card.

Sẽ có 2 trường hợp xảy ra:

* Một Contact sẽ có duy nhất một Membership Card, tức Contact và Membership Card quan hệ một – một
* Hoặc, một Contact có thể có nhiều Membership Card, tức Contact và Membership Card quan hệ một – nhiều.



**Ví dụ giữa Contact và Membership Card**

2 mối quan hệ khác nhau này dẫn đến 2 trường hợp business khác nhau.

Nếu quan hệ một – nhiều, như hình đầu tiên, thì điều này có nghĩa là: **một khách hàng sẽ có nhiều thẻ thành viên tại một thời điểm**. Và thẻ thành viên này sẽ được đổi qua các năm.

Có thể thẻ thành viên này làm bằng giấy cứng, hoặc thậm chí nhựa. Nhưng không có chức năng tích điểm tự động, chỉ phát hành cho khách hàng để lưu trữ mã số, nhận diện khách hàng, tích lũy các lần sử dụng, để khách hàng được hưởng một số ưu đãi trực tiếp tại cửa hàng.

*Kiểu như phiếu giảm giá… trà sữa vậy. Mua 10 lần được free 1 ly chẳng hạn vậy.*

Sau thời hạn thẻ có giá trị, hoặc hết ưu đãi, khách hàng phải thủ công đổi lại thẻ mới. Và cứ tiếp tục như vậy, 1 khách hàng có cả chục cái thẻ.

Còn nếu quan hệ một – một, tức **một khách hàng chỉ có thể có duy nhất 1 thẻ thành viên tại một thời điểm.**

Điều này nói lên được rằng: thẻ thành viên ở đây được làm bằng nhựa, chống nước, trày xướt. Có tích hợp chip bên trong, có khả năng tích điểm tự động, không cần đổi thẻ định kỳ, abc, xyz…

Đó là sức mạnh của việc: **hiểu rõ được relationship giữa các component**.

.

.

.

Điều mình vừa nói giống như ví dụ cái lu ở phần 1 vậy.

Nếu thân lu và nắp lu có quan hệ mật thiết, không thể tách rời. Thì đây là cái lu có chức năng chứa nước, chức năng đậy nắp và quan trọng là nắp lu được gắn liền với thân lu, tức cái lu có chức năng đóng mở nắp lu.

Còn nếu 2 component này có relationship độc lập, không ăn nhậu gì với nhau, thì đây là cái lu có nắp lu tách rời với thân lu.

Khả năng cao nếu dùng cái lu này thì nước trong lu sẽ bị dơ, nếu vô tình làm… mất nắp lu.

Do đó, COMPONENT và RELATIONSHIP nó quan trọng ở chỗ:

* **Nói lên được đặc thù của business:** *ví dụ anh này chỉ có thể liên kết với 1 mình anh này, anh kia thì có thể thuộc nhiều đơn vị, tổ chức khác nhau.*
* **Nó nói lên được chức năng của hệ thống:** *làm được cái gì, không làm được cái gì?*

Từ đó, anh em có thể ***hiểu và nắm được business*** của khách hàng, một cách khoa học nhất, dựa trên các component có trong hệ thống, và relationship giữa chúng với nhau.

Nói không quá thì…

*Dưới góc nhìn của BA, phần mềm chỉ là tập hợp các****component****và****relationship****với nhau*

**4. Không lo sót dữ liệu**

Một điểm nữa cũng lợi hại không kém, đó là việc xác định rõ component và relationship giữa các component còn giúp anh em đảm bảo được mình **không bị sót dữ liệu.**

Ví dụ trong một hệ thống quản lý nhà trọ, anh em xác định được có 8 components như sau:

1. Người dùng hệ thống: *Admin quản trị hệ thống hoặc các quản lý nhà trọ.*
2. Người thuê trọ: *tức khách thuê trọ*
3. Nhà trọ:*hệ thống có thể có nhiều nhà trọ, và mỗi nhà trọ này đều là những thực thể riêng biệt.*
4. Phòng trọ: *một nhà trọ gồm nhiều phòng trọ*
5. Giao dịch: *giúp lưu trữ việc đóng tiền thuê hàng tháng, tháng đó tổng cộng bao nhiêu tiền.*
6. Giao dịch chi tiết:*một giao dịch gồm những line chi tiết nào, ví dụ: tiền phòng, tiện điện, nước, wifi, rác… Mỗi line là bao nhiêu tiền.*
7. Dịch vụ: *như trên, dịch vụ điện, nước, wifi, rác, giữ xe, dọn vệ sinh…*
8. Hợp đồng cho thuê phòng: *quản lý luôn các loại hợp đồng, đặt cọc các thứ trên hệ thống, mà không cần quản lý giấy tờ tùm lum tùm la nữa.*

Sau khi xác định được một đống thứ phía trên, anh em có thể tuyên bố chắc chắn như đinh đóng cột là: **data input vào hệ thống quản lý trọ này, cũng như các data output từ hệ thống sẽ chỉ xoay quanh 8 đối tượng trên.**

Không hơn, không kém.

Từ đó anh em có thể dễ dàng confirm với khách hàng về bộ khung của hệ thống:

*“…à, ứng dụng của mình xoay quanh 8 đối tượng này nhé chị. Từ người dùng hệ thống, chúng ta login vào, tạo nhà trọ, phòng trọ, người thuê trọ, rồi liên kết các đối tượng này với nhau. Cuối tháng mình sẽ tạo các giao dịch để quản lý nó, phục vụ báo cáo và kiểm kê.  À mà giao dịch thì phải có luôn quản lý các service mà trọ mình có đúng không chị, ví dụ như: dịch vụ wifi, dọn vệ sinh, gửi xe, rửa xe các kiểu. Với lại hợp đồng nữa, từ giờ không cần quản lý giấy tờ tùm lum tùm la nữa, mà quản lý trên hệ thống luôn…. Chị ngó qua 1 lượt giúp em xem thử có còn thiếu****cái gì****nữa không chị nha”*

Điều này giúp anh em có cái nhìn cực kỳ high level cho toàn bộ solution của mình.

Giả sử nhà trọ này quá khủng, quá bự, quá to, quá nhiều phòng, và luôn trong tình trạng có phòng trống. Nhu cầu đặt ra: **cần lấp đầy các phòng trống này nhanh nhất có thể.**

Do đó khách hàng đòi quản lý thêm tình hình kinh doanh, tức Sales.

*Cụ thể chủ trọ cần biết được:****hôm đó có bao nhiêu người vô hỏi phòng.****Các đồng chí này quê quán ở đâu, học trường gì, làm nghề gì, số điện thoại bao nhiêu, nói chung là thông tin cơ bản.*

*Chủ trọ cần biết được****tiến độ cho mấy đồng chí này thuê phòng đến đâu rồi.****Ví dụ: đã dẫn đi xem phòng chưa, đặt cọc chưa, hay đang deal giá, vâng vâng và mây mây.*

Lúc này anh em cần xem thử: 8 đối tượng ở trên ***có đủ để cover, quản lý được hết những thông tin này hay không?***

Nếu là mình, mình sẽ suy nghĩ: *người thuê trọ thì đã là khách hàng của mình rồi, không thể gọp chung người thuê trọ với người hỏi trọ được…*

Do đó, mình sẽ phải cần 1 component khác thể hiện được: **người hỏi trọ**, khác với component người thuê trọ.

Còn tiến độ cho thuê, mình cũng phải cần đến 1 component khác để thể hiện, đó có thể là component: **tiến độ cho thuê**. Thể hiện tiến độ cho thuê trọ của những người hỏi trọ, từ lúc hỏi, cho tới lúc đóng tiền cọc và tiền phòng tháng đầu tiên.

Chốt lại, mình sẽ cần thêm tối thiểu 2 components để thực hiện được yêu cầu này. Từ đó, bức tranh tổng quan của mình sẽ được nới rộng và hệ thống sẽ có nhiều chức năng hơn.

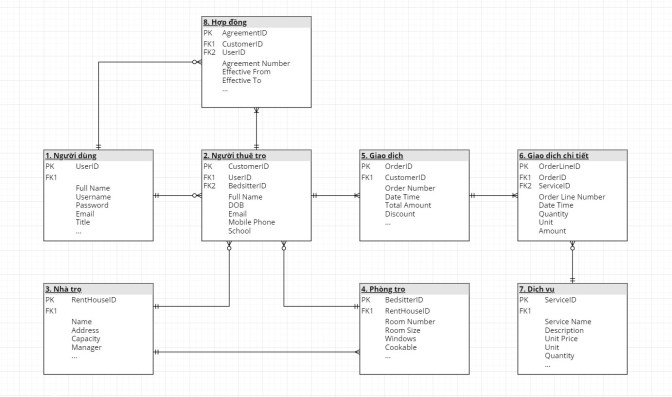
Do đó, đối với bất kỳ yêu cầu nào từ khách hàng. Điều đầu tiên anh em cần làm đó là: LIỆT KÊ RA CÁC COMPONENT có thể có, và các RELATIONSHIP giữa các component này với nhau.

Nó sẽ giúp anh em đảm bảo bám sát yêu cầu, và không bị sót dữ liệu một cách đáng kể!

**5. ERD**

Component và Relationship là cách nói đơn giản dễ hiểu của một sơ đồ cực kỳ, cực kỳ quan trọng, đó là: **Entity Relationship Diagram** – ERD

Mình sẽ kết thúc bài này bằng việc phác thảo nhanh sơ đồ ERD cho mô hình quản lý nhà trọ bên trên.

[](https://thinhnotes.com/chuyen-nghe-ba/bi-kip-chan-truyen-cua-ba/attachment/bi_kiep_chan_truyen_cua_ba_jpg2/)

Dĩ nhiên là lớp behind the scenes phía dưới còn rất nhiều component nữa để hệ thống thực sự vận hành. Nhưng đó là về mặt technical. Nếu có khả năng, rất khuyến khích anh em cover luôn phần này.

Còn đối với BA chúng ta, chỉ cần phác thảo ra được những Component và Relationship thể hiện được business của khách hàng như thế này là đủ

Về sơ đồ ERD thì mình sẽ có riêng 1 bài hướng dẫn chi tiết khác. Anh em cùng chờ nhé, nhanh thì tầm tháng sau, chậm thì năm sau :))

.

.

.