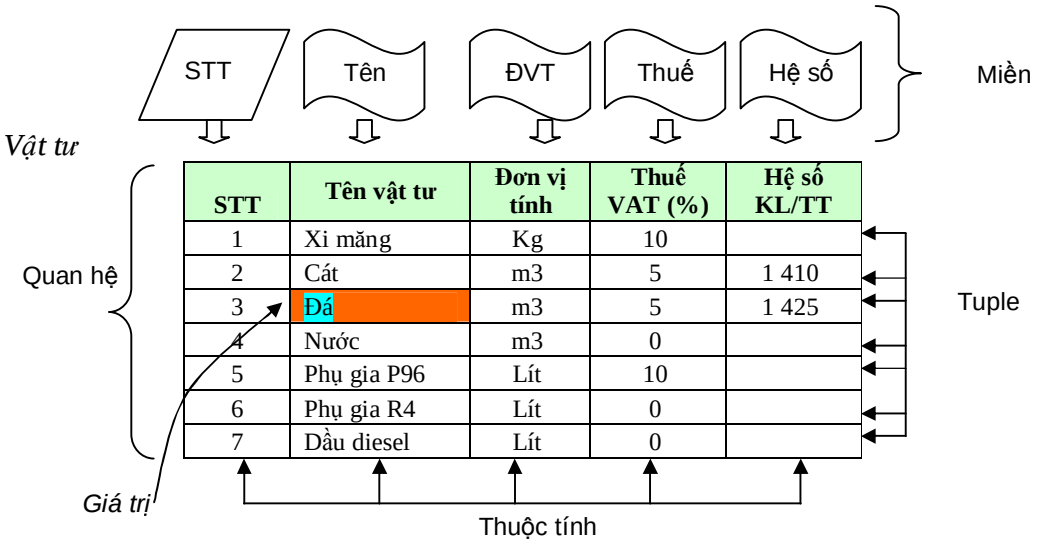
Bài 3

1./ *Miền và quan hệ*

Hệ thống khái niệm:

- Quan hệ: bảng
- Tuple: bản ghi, dòng
- Cardinality: số bản ghi
- Thuộc tính: trường, cột
- Degree: Bậc
- Khóa chính (primary key)
- Miền (domain): là vùng dữ liệu để định nghĩa giá trị các thuộc tính.



STT	Tên vật tư	Đơn vị tính	Thuế VAT (%)	Hệ số KL/TT
1	Xi măng	Kg	10	
2	Cát	m3	5	1 410
3	Đá	m3	5	1 425
4	Nước	m3	0	
5	Phụ gia P96	Lít	10	
6	Phụ gia R4	Lít	0	
7	Dầu diesel	Lít	0	

Hình 1: Quan hệ và các khái niệm

Như đã được đề cập, giao điểm của một hàng và một cột trong một bảng là một *giá trị*. Trong CSDL quan hệ, giá trị là phần tử không thể chia nhỏ hơn được nữa và đó là một giá trị vô hướng. Nghĩa là người ta không chấp nhận một giá trị có thể là tổ hợp của các giá trị khác. Nếu một giá trị trong thực tế lại có thể tổ hợp từ các giá trị khác thì bảng (quan hệ) đó cần được chia nhỏ hơn.

Miền được định nghĩa là tập hợp của các giá trị vô hướng: miền = {v1, v2, ...}. Chúng ta nhìn thấy trong Hình 1 các giá trị của một thuộc tính (cột) thuộc vào một miền giá trị nhất định:

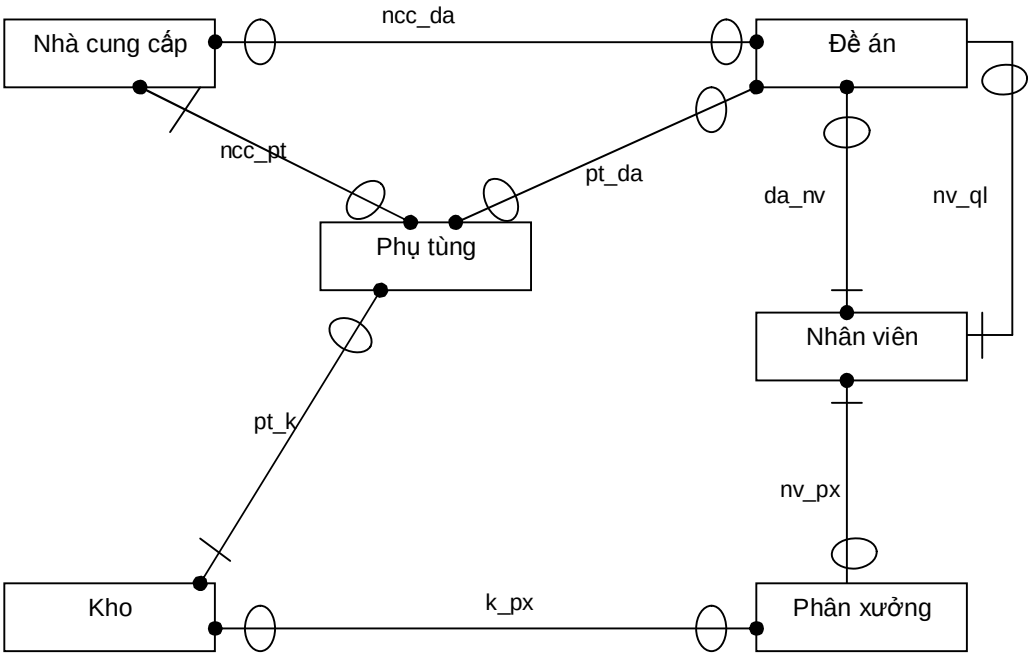
STT: Số nguyên dương
Tên vật tư: Dãy ký tự
Đơn vị tính: Dãy ký tự
Thuế VAT(%): Số trong khoảng [0,100]
Hệ số KL/TT: Số dương (có thể không có giá trị)

Vậy ý nghĩa của miền giá trị là gì?

Chúng ta dễ dàng nhận thấy STT, Thuế VAT, Hệ số KL/TT đều là số nhưng chúng rõ ràng là rất khác nhau. Chẳng hạn Thuế VAT phải là các giá trị được Pháp lệnh thuế chấp nhận (một tập hợp hữu hạn các giá trị. Tương tự, STT và Hệ số KL/TT khác nhau về bản chất. Chúng ta lưu ý thêm là Hệ số KL/TT có các khoảng trống (nghĩa là không có giá trị).

2./ *Xây dựng một CSDL quan hệ*

Để hiểu được vấn đề một cách cụ thể hơn, chúng ta hãy xây dựng một phần của CSDL nhà máy sản xuất ô tô Lạc Hồng (như đã đề cập trong Bài 1). Đó là xây dựng các bảng để lưu thông tin các *nhà cung cấp* và *phụ tùng*.



Hình 2: Sơ đồ quan hệ thực thể của nhà máy sản xuất ô tô Lạc Hồng

Chúng ta lưu ý các thực thể và quan hệ được in đậm gồm **Nhà cung cấp**, **Phụ tùng**, **ncc_pt**. Ta lập bảng `supplier` cho thực thể **Nhà cung cấp**, bảng `part` cho thực thể **Phụ tùng** và bảng `sp` cho quan hệ **ncc_pt**.

Trong thời điểm hiện tại của giáo trình, chúng ta hãy chấp nhận các câu lệnh tạo bảng, chèn dữ liệu sau mà chưa cần các giải thích chi tiết (chúng ta sẽ trở lại).

Tuy vậy, đây là một ví dụ rất tiêu biểu của việc tạo CSDL từ việc quan sát sơ đồ quan hệ thực thể, các thực thể và các quan hệ. Có một điểm đáng lưu ý ngay là ***thực thể*** và ***quan hệ*** đều có thể biến thành ***bảng***.

■ Tạo các quan hệ (bảng)

```
CREATE TABLE supplier (
       s_id CHAR(5) NOT NULL,
       sname CHAR(20) NOT NULL,
       status NUMERIC(5) NOT NULL,
       city CHAR(15) NOT NULL,
       PRIMARY KEY (s_id)
);

CREATE TABLE part (
       p_id CHAR(6) NOT NULL,
       pname CHAR(20) NOT NULL,
       color CHAR(6) NOT NULL,
       weight NUMERIC(5) NOT NULL,
       city CHAR(15) NOT NULL,
       PRIMARY KEY (p_id)
);

CREATE TABLE sp(
       s_id CHAR(5) NOT NULL,
       p_id CHAR(6) NOT NULL,
       qty NUMERIC(9) NOT NULL,
       PRIMARY KEY (s_id, p_id)
);
```

■ Nhập dữ liệu

o Cho bảng `supplier`:

```
INSERT INTO supplier (s_id, sname, status, city)
      VALUES ('S1', 'Smith', 20, 'London');
INSERT INTO supplier (s_id, sname, status, city)
      VALUES ('S2', 'Jones', 10, 'Paris');
INSERT INTO supplier (s_id, sname, status, city)
      VALUES ('S3', 'Blake', 30, 'Paris');
INSERT INTO supplier (s_id, sname, status, city)
      VALUES ('S4', 'Clark', 20, 'London');
INSERT INTO supplier (s_id, sname, status, city)
      VALUES ('S5', 'Adams', 30, 'Athens');
```

o Cho bảng `part`:

```
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P1', 'Nut', 'Red', 12, 'London');
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P2', 'Bolt', 'Green', 17, 'Paris');
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P3', 'Screw', 'Blue', 17, 'Rome');
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P4', 'Screw', 'Red', 14, 'London');
```

```
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P5', 'Cam', 'Blue', 12, 'Paris');
INSERT INTO part (p_id, pname, color, weight, city)
      VALUES ('P6', 'Cog', 'Red', 19, 'London');
```

- Cho bảng sp:

```
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P1', 300);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P2', 200);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P3', 400);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P4', 200);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P5', 100);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S1', 'P6', 100);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S2', 'P1', 300);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S2', 'P2', 400);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S3', 'P2', 200);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S4', 'P2', 200);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S4', 'P4', 300);
INSERT INTO sp(s_id, p_id, qty)
      VALUES ('S4', 'P5', 400);
```

■ Kết quả

- Bảng supplier:

s_id	sname	status	city
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

- Bảng `part`:

p_id	pname	color	weight	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

- Bảng `sp`:

s_id	p_id	qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

■ Miền giá trị trong các phép toán so sánh

Ví dụ 1:

```
SELECT part.*, sp.qty
FROM part, sp
WHERE part.p_id = sp.p_id;
```

Trong câu lệnh trên, phép so sánh là hợp lệ vì `p_id` trong part và `sp` đều cùng một miền giá trị.

Ví dụ 2:

```
SELECT part.*, sp.qty
FROM part, sp
WHERE part.weight = sp.qty;
```

Câu lệnh trên so sánh `weight` (trọng lượng) với `qty` (số lượng) không có nghĩa, vì người ta không so sánh hai đại lượng khác nhau. Từ đó, theo suy luận logic, `weight` và `qty` phải thuộc vào hai miền giá trị khác nhau.

Tuy nhiên, thực tế các HQT CSDL, ở thời điểm hiện nay, phần lớn đều không bắt lỗi câu lệnh trên.

- Định nghĩa dữ liệu và miền giá trị

Khi lập CSDL chung cho cơ quan, doanh nghiệp, người ta phải thống nhất lập ra *từ điển dữ liệu*. Ví dụ họ và tên sẽ dùng bao nhiêu ký tự, mức lương phải là các giá trị nào và việc kiểm soát sẽ phải được thực hiện theo một cơ chế nhất định.

(Vì SQL 92 không hỗ trợ khái niệm miền giá trị nên phần lớn các ứng dụng "tạo miền" bằng cách đặt độ dài của loại dữ liệu và kiểm soát miền thông qua các ràng buộc.)

- Miền giá trị và loại dữ liệu

Khi xem các câu lệnh CREATE ở trên, ta nhận thấy không có miền mà chỉ có khai báo loại dữ liệu (CHAR, NUMERIC, ...) và khai báo độ dài của loại dữ liệu đó.

- *Quan hệ* (bảng)

Quan hệ có 2 phần: các trường và các bản ghi. Chúng ta nhận thấy qua ví dụ, người ta tạo ra cấu trúc quan hệ trước (câu lệnh CREATE TABLE) và sau đó tạo và cập nhật các bản ghi (các câu lệnh INSERT, UPDATE, DELETE).

Một cách hình thức, chúng ta có định nghĩa sau:

Định nghĩa: Quan hệ *R* trên các miền giá trị $D_1, D_2, ..., D_n$, trong đó D_i có thể trùng nhau, gồm 2 phần *Đề mục* và *Thân*. Trong đó:

Đề mục: Là tập hợp các thuộc tính $\{A_1, A_2, ..., A_n\}$ sao cho có sự tương ứng
$\{ <A_1, D_1>, <A_2, D_2>, ..., <A_n, D_n> \}$,
D_i là các miền giá trị nói trên.

Thân: Là tập hợp các n-tuple $\{V_{j1}, V_{j2}, ..., V_{jn}\}$, sao cho có sự tương ứng
$\{ <A_1, V_{j1}>, <A_2, V_{j2}>, ..., <A_n, V_{jn}> \}$
Trong đó $j = 1, 2, ...m$.

Giá trị *n* chính là *degree* và giá trị *m* chính là *cardinality*.

Một cách thực tế:

Tạo cấu trúc quan hệ:

```
CREATE TABLE <tên bảng> (
      attribute-definition comma-separated list
      Candidate key definition
      Foreign key definition
);
```

Nhận xét về câu lệnh tạo cấu trúc quan hệ:
(SV ghi chép tại lớp)

Nhận xét về các bản ghi trong một quan hệ:

- Không có các bản ghi trùng nhau
- Các bản ghi không có thứ tự
- Các trường không có thứ tự
- Các giá trị đều là nguyên tử

Một quan hệ như vậy được gọi là quan hệ đạt chuẩn 1NF (First Normal Form). Chúng ta sẽ trở lại vấn đề chuẩn hóa sau.

- Các loại quan hệ:

Base relation:	quan hệ gốc
Derived relation:	quan hệ chuyển hóa được tạo ra từ các quan hệ gốc
View:	quan hệ chuyển hóa được đặt tên. Còn gọi là bảng ảo.
Snapshot:	quan hệ chuyển hóa được đặt tên, giống như view, nhưng là quan hệ thực.
Query Result:	kết quả truy vấn, quan hệ chuyển hóa không được đặt tên.
Intermediate Result:	quan hệ kết quả trung gian
Stored relation:	quan hệ lưu – là dạng đặc biệt của quan hệ chuyển hóa được lưu trong CSDL để nâng cao tính hiệu dụng

3./ *Tính toàn vẹn của dữ liệu quan hệ*

Chúng ta xây dựng và lưu CSDL để thể hiện một phần của thế giới thực. Các giá trị trong CSDL, đương nhiên là phải thể hiện thế giới thực. Lấy ví dụ, khối lượng và số lượng không thể là các số âm được. Tên của các thành phố phải nằm trong danh mục các thành phố thực trên thế giới, ...

Để có thể đảm bảo các đặc trưng đó luôn luôn được giữ vững, người ta tạo ra các qui tắc để đảm bảo một đặc tính được đặt tên là *toàn vẹn dữ liệu*. Trên thực tế toàn vẹn dữ liệu chính là các ràng buộc giữa các dữ liệu với nhau. Các giá trị trong CSDL phải thỏa mãn các ràng buộc đã được đặt ra.

Trong các CSDL, số lượng các ràng buộc có thể rất lớn. Ví dụ:

- Mã hiệu các nhà cung cấp phải theo qui ước Sxxxx (xxxx là số)
- Mã hiệu phụ tùng phải theo qui ước Pxxxxx (xxxxx là số)
- Tên các thành phố phải được trích xuất từ một danh mục
- Status của các nhà cung cấp phải nằm trong khoảng [0, 100]
- Trọng lượng phải lớn hơn không
- Số lượng phải là bội số của 100
- ....

4./ *Khóa (candidate keys – Khóa đề cử)*

Giả thiết *R* là một quan hệ, *K* là một tập hợp của các thuộc tính của *R* (một tổ hợp của các cột).

K được gọi là *Khóa* nếu *K* thỏa mãn 2 tính chất:

1. *Tính duy nhất*: không có 2 bản ghi có cùng giá trị khóa K
2. *Tính tối giản*: K không bao gồm tập hợp con mà tập hợp con đó lại là khóa

Chú ý rằng, đã là một quan hệ thì *R* có ít nhất một khóa. Vì sao vậy? Vì theo định nghĩa, *R* không có 2 bản ghi trùng nhau (1NF).

R có thể có nhiều khóa. Có hai trường hợp xảy ra đối với *R*:

1. Hoặc tập hợp của tất cả các trường của *R* tạo thành một khóa
2. Hoặc có ít nhất một tập hợp con các trường của *R* tạo thành một khóa

Ví dụ:

- Bảng supplier có khóa là trường s_id.
- Bảng part có khóa là trường p_id
- Bảng sp có khóa là tổ hợp (s_id, p_id)

■ Vai trò của khóa:

Khóa giúp ta lấy dữ liệu từ bảng ra một cách duy nhất : { *tên bảng, khóa, tên trường* }

Chú ý: Thực tế, các HQT CSDL đều chấp nhận trường hợp có nhiều bản ghi trùng nhau hoàn toàn mà không hề báo lỗi.

Trong một quan hệ có thể tồn tại nhiều khóa. Do phải khai báo một *khóa chính*, nên người ta gọi tập hợp tất cả các khóa là *khóa đề cử* (*candidate keys*). Nghĩa là các khóa đều có thể là khóa chính.

5./ *Khóa chính, khóa phụ*

Trong số các *khóa đề cử*, người ta lấy ra một *khóa chính(Primary Key)* và các khóa còn lại gọi là *khóa phụ (Alternate Keys)*.

Có thể chúng ta sẽ đặt ra câu hỏi: Có nguyên tắc nào để chọn khóa chính không? Chính thức thì "**Không**".

Thực tế và kinh nghiệm? Theo kinh nghiệm của tác giả, nên có 2 nguyên tắc sau:

Đối với các thực thể: Nên chọn chỉ *một* trường là khóa chính (không phải là tập hợp nhiều trường). Nếu không chọn được một trường nào làm khóa chính thì nên tạo ra một trường tự động làm khóa chính (ta sẽ đề cập đến sau).

Đối với quan hệ: Chỉ nên chọn 2 trường làm khóa chính và 2 trường đó là các khóa ngoại (xem dưới đây)

Chú ý: Đó là các kinh nghiệm của tác giả, không phải là lý thuyết về CSDL quan hệ.

6./ Khóa ngoại (Foreign Keys)

Có thể hiểu một cách đơn giản là tổ hợp của các trường trong quan hệ *R1* lại là khóa của một quan hệ *R2*. Trong ví dụ, bảng `sp` có 2 khóa ngoại là `s_id` và `p_id`, vì `s_id` là khóa của `supplier` và `p_id` là khóa của `part`.

Một cách hình thức:

Cho *R2* là một quan hệ. Một khóa ngoại trong *R2*, gọi là *FK*, nếu:

1. Tồn tại một quan hệ *R1*, sao cho khóa chính của nó *PK* chính là *FK* của *R2*. (Chú ý rằng *R1* có thể trùng với *R2*.)

2. Trong bất cứ thời điểm nào, một bản ghi trong *R2* ứng với tổ hợp trường tạo nên *FK* phải trùng với một giá trị duy nhất của tổ hợp trường tạo nên *PK* trong *R1*.

Lưu ý:

Trong một quan hệ chỉ có một khóa chính nhưng có thể có nhiều khóa ngoại.

Mệnh đề ngược của điểm 2 trong định nghĩa trên đây không đúng. Nghĩa là không phải bất cứ giá trị *PK* nào của *R1* phải trùng với *FK* của *R2*.

FK của *R2* là tổ hợp của nhiều hơn một trường khi và chỉ khi *PK* của *R1* cũng là tổ hợp của nhiều hơn một trường (*PK* và *FK* trong trường hợp này gọi là *khóa tổ hợp*).

7./ Khái niệm về giá trị Null trong CSDL quan hệ

Khái niệm giá trị Null trong CSDL được sinh ra là để giải quyết vấn đề khi một trường nào đó không có giá trị để đưa vào.

VD1: Khi lập bảng về nhân sự, chúng ta có trường ghi địa chỉ e-mail. Chúng ta đều biết là không phải ai cũng có địa chỉ e-mail. Vậy đối với trường hợp đó, trong CSDL cần ghi gì để thế vào chỗ trống? Để trắng hay một giá trị nào khác?

VD2: Khi lập bảng về các nhân vật lịch sử, chúng ta có trường năm sinh là một trường số (NUMERIC). Rõ ràng là có nhiều trường hợp chúng ta không rõ năm sinh vì lịch sử không ghi lại hoặc hiện thời còn nhiều tranh cãi về năm sinh. Vậy đối với trường hợp đó, trong CSDL cần ghi gì để thế vào chỗ trống? Để 0 hay một giá trị nào khác? Rõ ràng không thể để là 0 được vì 0 là một giá trị trong miền giá trị số và nếu hiểu năm sinh của nhân vật đó là 0 thì không chấp nhận được.

TS. Codd đã đề nghị đưa vào CSDL một khái niệm gọi là giá trị Null, để đánh dấu là "*không có giá trị thực*" hoặc "*không biết*". Lưu ý rằng Null không đồng nghĩa với trắng trong chuỗi ký tự hay 0 trong miền giá trị là số.

Một số vấn đề liên quan đến NULL

Khi tạo ra bảng (CREATE TABLE) người ta phải chỉ ra là một trường có chấp nhận giá trị NULL hay không.

Khóa chính không chấp nhận là tổ hợp của các trường mà trong đó có trường chấp nhận giá trị NULL.

Có thể chấp nhận khóa ngoại có giá trị NULL. Nếu khóa ngoại không phải là NULL thì nó phải ứng với một giá trị khóa chính của một bảng khác (xem phần khóa ngoại).

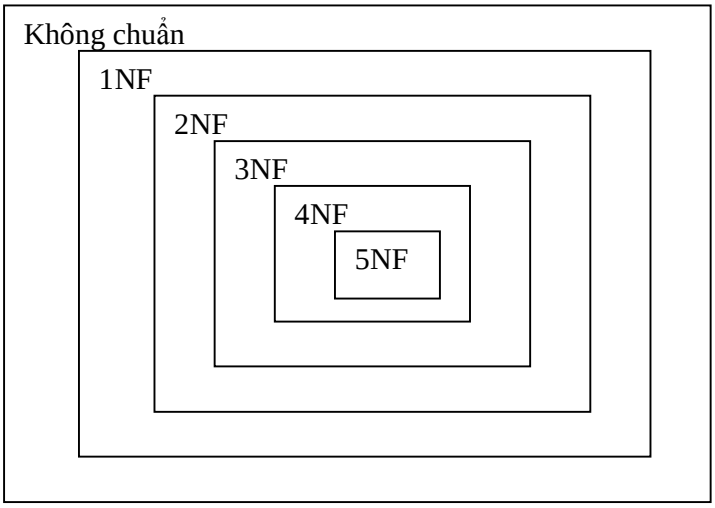
8./ *Các chuẩn quan hệ (Normalization Forms)*

Trong phần này, từ "chuẩn" được dùng thay cho từ "dạng chuẩn hóa". Trên thực tế, chuẩn hóa là một quá trình, và kết quả của quá trình đó là đạt được dạng chuẩn hóa, và tác giả gọi dạng chuẩn hóa đó là một chuẩn, vừa ngắn gọn, vừa dễ hiểu.

■ Chuẩn là gì?

Một bảng được gọi là dạng một chuẩn nào đó, nếu nó thỏa mãn các điều kiện chuẩn đó đề ra.

TS. E.F. Codd đưa ra ba dạng chuẩn đầu tiên (1NF, 2NF, 3NF) và sau đó Fagin đưa ra hai dạng nữa (4NF, 5NF). Các số của dạng chuẩn càng lớn thì dạng chuẩn đó càng chuẩn (nghĩa là càng khó đạt được). Ví dụ, 2NF chuẩn hơn 1NF, vì 2NF chắc chắn thỏa mãn các điều kiện của 1NF. Tương tự như vậy, 3NF chuẩn hơn 2NF, ...



Hình 3: Phân lớp các chuẩn quan hệ

Trên thực tế khi thiết kế CSDL, người ta chỉ quan tâm 3 chuẩn đầu. Trong các phần lớn các trường hợp, người ta chỉ cần đưa các bảng về chuẩn 3NF là đủ.

■ Vì sao lại cần chuẩn?

Như chúng ta đã đề cập trong các bài trước, một trong những mục tiêu của hệ thống CSDL là tránh dữ liệu dư thừa và đảm bảo tính toàn vẹn của chúng. Các chuẩn sinh ra để làm cho các công việc đó được thực hiện một cách dễ dàng hơn.

Theo định nghĩa về quan hệ thì các bảng không có các bản ghi trùng nhau và các giá trị đều là các giá trị đơn (nguyên tử), không phải là các đa trị (giá trị gộp). Do vậy, các bảng tự chúng đã là 1NF.

Định nghĩa 1NF: Các bảng được gọi là thuộc dạng chuẩn 1NF khi và chỉ khi miền giá trị của các trường trong bảng chỉ chứa các giá trị đơn (nguyên tử).

Để hiểu được định nghĩa của các dạng chuẩn khác, chúng ta cần hiểu thêm khái niệm *phụ thuộc hàm*.

■ Tính phụ thuộc hàm (*Functional Dependencies*)

Trong các bảng, giá trị của một số trường có thể suy ra được từ các trường khác. Ví dụ, trong bảng `supplier`, ta có thể lấy được giá trị của trường `city` từ trường `s_id`.

Người ta gọi `city` phụ thuộc hàm `s_id`. Ta viết tắt phụ thuộc hàm là *FD*.

Định nghĩa: Cho R là một quan hệ và X, Y là các tập bất kỳ chứa các thuộc tính của R. Y được gọi là phụ thuộc hàm X, khi và chỉ khi, các giá trị tương ứng với Y được suy ra một cách duy nhất từ các giá trị tương ứng với X.
Ký hiệu: $X \rightarrow Y$

Ví dụ 1:

{s_id} → {city} trong bảng supplier.

Ví dụ 2:

Trong bảng mot_nf sau đây:

mot_nf

s_id	city	p_id	qty
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
S1	London	P4	200
S1	London	P5	100
S1	London	P6	100
S2	Paris	P1	300
S2	Paris	P2	400
S3	Paris	P2	200
S4	London	P2	200
S4	London	P4	300
S4	London	P5	400

Ta có:

```
{s_id, p_id} → {qty}
{s_id, p_id} → {city}
{s_id, p_id} → {city, qty}
{s_id, p_id} → {s_id}
{s_id, p_id} → {s_id, p_id, city, qty}
```

Phía trái của mũi tên gọi là *định thức*, phía phải của mũi tên gọi là *phụ thuộc*.

Chú ý rằng, nếu *X* là khóa của quan hệ *R*, thì *X* sẽ suy ra được tất cả các tổ hợp trường của *R*.

Một cách tinh tế hơn, chúng ta nhận thấy, nếu quan hệ R có A → B và A không phải là khóa thì R sẽ tồn tại dư thừa dữ liệu. Thử quan sát bảng mot_nf và ta sẽ dễ dàng nhận ra s_id và city xuất hiện nhiều lần giống nhau. Đó chính là hiện tượng dư thừa dữ liệu.

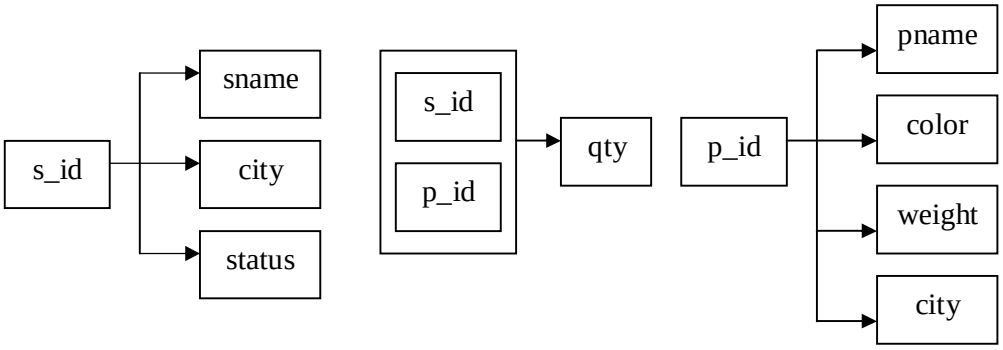
Người ta nhận thấy quan hệ phụ thuộc hàm như

```
{s_id, p_id} → {s_id}
```

không thật sự có ý nghĩa (chúng được gọi là các quan hệ phụ thuộc hàm tầm thường) và chúng ta sẽ không quan tâm tới chúng.

Tính bắc cầu: Nếu A → B và B → C thì A → C

Sơ đồ phụ thuộc hàm (sơ đồ FD):



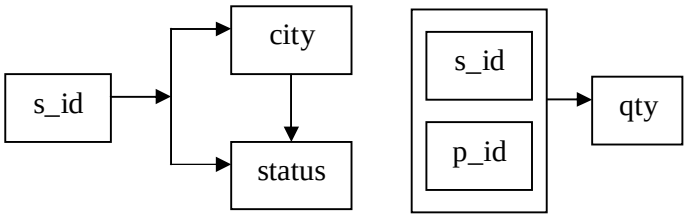
Phụ thuộc hàm đầy đủ (Fully Functional Dependence): Trong quan hệ R, Y được gọi là phụ thuộc hàm đầy dủ vào X nếu Y không phụ thuộc hàm vào một tập con nào của X.

Phụ thuộc hàm đầy đủ còn gọi là phụ thuộc hàm tối giản.

Định nghĩa 2NF: Bảng *R* được gọi là thuộc dạng chuẩn 2NF khi và chỉ khi R là 1NF và các trường không phải là khóa phải là phụ thuộc hàm đầy đủ (phụ thuộc hàm tối giản) vào khóa chính.

Chú ý rằng bảng `mot_nf` không phải thuộc dạng 2NF. Lý do: `s_id` → `city`.

Ta tiến hành tách bảng `mot_nf` thành `hai_nf` và `sp`:



`hai_nf`

s_id	city	status
S1	London	20
S2	Paris	30
S3	Paris	30
S4	London	20
S5	Athens	30

`sp`

s_id	p_id	qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Định nghĩa 3NF: Bảng *R* được gọi là thuộc dạng chuẩn 3NF khi và chỉ khi *R* là 2NF và các trường không phải là khóa không phụ thuộc hàm một cách bắc cầu vào khóa chính.

Chú ý rằng bảng `sp` thuộc dạng 3NF, còn `hai_nf` không phải là 3NF. Lý do bảng `hai_nf` không phải là 3NF: `s_id` → `city` → `status`.

9./ *Bài tập*

Bài tập 3.1 - Giải thích các thuật ngữ:

- Miền
- Khóa
- Khóa chính
- Khóa ngoại
- FD
- Chuẩn quan hệ

Bài tập 3.2 - Thiết kế bảng nhân sự nguoi (nguoi_id, ten, tuoi, dien_thoai, e_mail, que_quan)

Bài tập 3.3 - Tái tạo các bảng `supplier`, `part`, `sp` nhưng thay bằng các tên mới `nha_cung_cap`, `phu_tung`, `ncc_pt`

Bài tập 3.4 - Khi khởi tạo một bảng, chúng ta cần phải định nghĩa những phần nào?

Bài tập 3.5 - Bảng dạng 2NF tránh được dữ liệu dư thừa kiểu nào, cho một ví dụ.

Bài tập 3.6 - Bảng dạng 3NF tránh được dữ liệu dư thừa kiểu nào, cho một ví dụ.

Bài tập 3.7 - Nêu một điểm bất lợi khi các bảng đều thuộc dạng 3NF.

Bài tập 3.8 - Câu lệnh sau đây tái tạo bảng nào trong bài giảng, giải thích.

```
SELECT sp.s_id, supplier.city, sp.p_id, sp.qty
FROM supplier, sp
WHERE supplier.s_id = sp.s_id
```

Bài tập 3.9 - Khóa ngoại liên quan với khóa chính như thế nào? Đặt tên cho tính liên quan này.

Bài tập 3.10 - Nếu ta xóa bản ghi ứng với khóa P1 trong bảng `part`, chuyện gì sẽ xảy ra?