Data Warehouse

và cách thiết kế Data Warehouse

Lưu Đức Thắng

Ban BI, phòng Nghiên cứu phát triển – Trung tâm phần mềm viễn thông Viettel

DreamingFighter@gmail.com

thangld2@viettel.com.vn

Contents

Phần 1: Data Warehouse là gì

1.1 Giới thiệu về database

Database đóng vai trò quan trọng bậc nhất trong tất cả các hệ thống thông tin, là phân hệ trung tâm của các hệ thống nghiệp vụ xử lý dữ liệu giao dịch. Database bao gồm một tập hợp dữ liệu có cấu trúc và một bản thông tin mô tả dữ liệu có cấu trúc đó (metadata), được thiết kế cho nhu cầu lưu trữ và xử lý thông tin của tổ chức, doanh nghiệp. Một database thường được cài đặt kèm theo một **hệ quản trị cơ sở dữ liệu** (DBMS – Database Management System), tức là một phần mềm cho phép người dùng định nghĩa, tạo mới, điều khiển và quản trị một database.

Trong ngữ cảnh database trong hệ thống nghiệp vụ, một tác vụ cần truy suất, xử lý dữ liệu gọi là một **giao dịch** (transaction). Để đảm bảo xử lý giao dịch chính xác, thiết kế database cần phải thoả mãn 4 tính chất là Tính nguyên tố (Atomicity), Tính nhất quán (Consistency), Tính tách biệt (Isolation), Tính bền vững (Durability) – ACID.

- **Tính nguyên tố (Atomicity)** quy định rằng nếu một giao dịch có nhiều thao tác xử lý dữ liệu thì hoặc là tất cả các thao tác đó đều thành công, hoặc không có thao tác nào thành công cả. Một hệ thống gọi là có tính nguyên tố khi nó thoả mãn điều kiện trên trong bất kỳ tình huống lỗi nào, bao gồm cả lỗi phần cứng hay phần mềm. Về phía người dùng, một giao dịch thành công được thể hiện như là một thao tác duy nhất, còn một giao dịch thất bại không có bất kỳ tác động nào đến database cả.
- **Tính nhất quán (Consistency)** quy định rằng dữ liệu trong database phải hợp lệ trước và sau mỗi giao dịch. Mỗi thao tác ghi dữ liệu vào database phải thoả mãn các luật quy định trước bao gồm nhưng không gói gọn trong constraint, cascade, trigger… Tính chất này không đảm bảo dữ liệu trong database đúng nghiệp vụ (đó là việc của lập trình viên, database làm hết thì coder làm gì?), nó chỉ giúp hạn chế lỗi (nếu có) trong quá trình phát triển phần mềm.
- **Tính tách biệt (Isolation)** quy định rằng việc thực thi song song nhiều giao dịch một lúc cho ra kết quả tương đương việc thực thi các giao dịch đó một cách tuần tự.
- **Tính bền vững (Durability)** quy định rằng một giao dịch khi đã thành công sẽ được ghi nhận vĩnh viễn ngay cả khi có sự cố về phần cứng hay phần mềm.

Trong hệ thống cơ sở dữ liệu quan hệ, dữ liệu thường được lưu trữ dưới dạng các thực thể. **Thực thể (Entity)** là cách thể hiện một tập các đối tượng thực tế mà hệ thống nghiệp vụ cần phải quản lý, lưu trữ. Về mặt ứng dụng mà nói, các thực thể thuộc cùng một loại có các thuộc tính giống hệt nhau.

// TODO: phân rõ thực thể mạnh, thực thể yếu, quan hệ giữa các thực thể

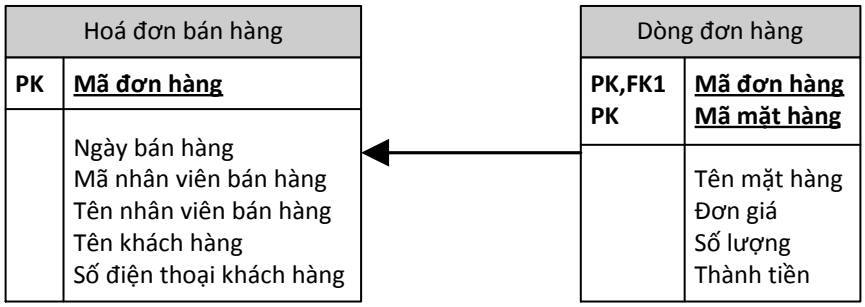
Để thoả mãn tính ACID, thiết kế của một database thường được đưa về dạng chuẩn 3 (3NF – 3th Normal Form), quy định rằng mỗi thực thể trong database phải thoả mãn các điều kiện sau:

- Giá trị của một thuộc tính phải là giá trị nguyên tố, tức là không phải một danh sách các giá trị hoặc giá trị phức hợp (**Chuẩn 1**)
- Các thuộc tính không phải khoá chính chỉ phụ thuộc vào toàn bộ tập khoá chính, không phụ thuộc vào tập con nào của tập khoá chính nói trên (**Chuẩn 2**)
- Các thuộc tính không phải khoá độc lập với nhau (không thể nội suy giá trị một thuộc tính từ các thuộc tính không phải khoá khác) (**Chuẩn 3**)

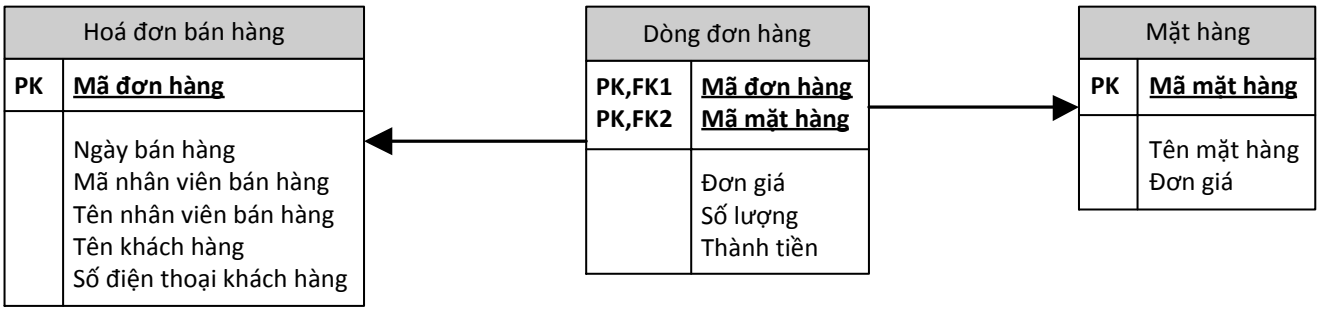
Ví dụ đối tượng Hoá đơn bán hàng gồm các thông tin chủ yếu sau:

Hoá đơn bán hàng	
PK	**Mã đơn hàng**
	Ngày bán hàng Mã nhân viên bán hàng Tên nhân viên bán hàng Tên khách hàng Số điện thoại khách hàng Các dòng đơn hàng - Mã mặt hàng - Tên mặt hàng - Đơn giá - Số lượng - Thành tiền

Thực thể này có thuộc tính "Các dòng đơn hàng" vi phạm quy định giá trị nguyên tố của dạng chuẩn 1. Để đưa về chuẩn 1, ta thêm thực thể "Dòng đơn hàng" vào thiết kế như sau:



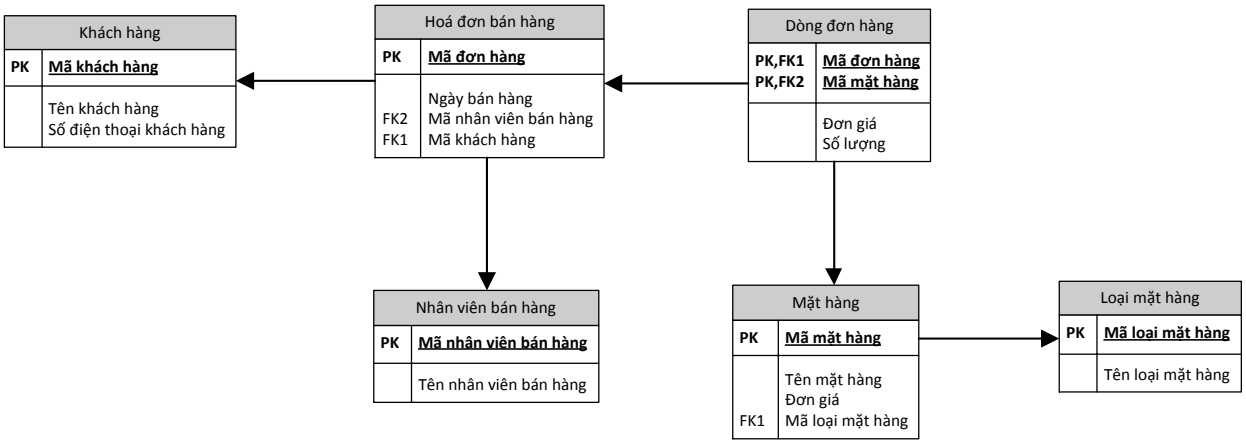
Thiết kế này vi phạm chuẩn 2 ở chỗ thực thể "Dòng đơn hàng" có thuộc tính "Tên mặt hàng" chỉ phụ thuộc vào thuộc tính "Mã mặt hàng" (là tập con của khoá chính) => Sửa bằng cách thêm thực thể "Mặt hàng" lưu các thông tin trên.



Chú ý: thuộc tính "Đơn giá" cũng phụ thuộc "Mã mặt hàng" nhưng do yêu cầu nghiệp vụ, giá trên hoá đơn là bất biến trong khi giá mặt hàng thực tế có thay đổi nên cần lưu thông tin này vào thực thể "Dòng đơn hàng".

Thiết kế này vi phạm chuẩn 3:

- Bảng "Hoá đơn bán hàng" có trường "Tên nhân viên bán hàng", "Số điện thoại khách hàng" không phụ thuộc khoá chính mà phụ thuộc thuộc tính "Mã nhân viên bán hàng" => khắc phục bằng cách tạo thêm bảng "Nhân viên"
- Bảng "Hoá đơn bán hàng" có trường "Số điện thoại khách hàng" không phụ thuộc khoá chính mà phụ thuộc trường "Tên khách hàng" => Khắc phụ bằng cách thêm bảng "Khách hàng"
- Bảng "Dòng đơn hàng" có trường "Thành tiền" có thể nội suy từ 2 trường "Đơn giá" và "Thành tiền" => Khắc phục bằng cách bỏ trường "Thành tiền"



Phần này mô tả qua về database và giới thiệu sơ lược cách thiết kế database trong một hệ thống nghiệp vụ. Trong phần sau chúng ta sẽ nói về data warehouse, các thành phần của data warehouse và thiết kế thường thấy của data warehouse.

1.2 Data Warehouse

Kinh tế khó khăn, đối thủ càng nhiều thì việc phân tích dữ liệu càng trở nên quan trọng đối với doanh nghiệp nhằm hỗ trợ ra quyết định, gia tăng lợi thế cạnh tranh. Tuy nhiên database thông thường lại không thoả mãn các yêu cầu về phân tích dữ liệu, database thông thường chỉ hỗ trợ tốt các nghiệp vụ hàng ngày và điểm mạnh nhất của nó là đảm bảo toàn vẹn dữ liệu, xử lý giao dịch, truy cập song song. Database thông thường đó được gọi là database nghiệp vụ (**operational database**) hoặc hệ thống xử lý giao dịch thời gian thực (online transaction processing – OLTP). Thông thường các database nghiệp vụ chỉ lưu trữ dữ liệu chi tiết cho thời điểm hiện tại, không lưu dữ liệu lịch sử, dữ liệu trong database được thiết kế chuẩn hoá rất cao nên thường có hiệu năng kém khi truy vấn phức tạp (join nhiều bảng dữ liệu với nhau) hoặc khối lượng dữ liệu lớn. Thêm nữa, việc truy vấn dữ liệu từ nhiều nguồn khác nhau là gần như không thể nếu chỉ dùng database nghiệp vụ.

Có cung thì tất có cầu, ngay từ những năm 70 nhiều công ty đã bán các hệ thống database hỗ trợ phân tích, báo cáo như Teradata, MAPPER, nhưng thuật ngữ "data warehouse" chỉ được sử dụng vào năm 1988 trong một bài báo kỹ thuật của IBM có tiêu đề "Kiến trúc hệ thống thông tin và kinh doanh" (An architecture for a business and information system – http://altaplana.com/ibmsj2701G.pdf). Phần này sẽ dành riêng để nói về khái niệm data warehouse.

Theo wikipedia (http://en.wikipedia.org/wiki/Data_warehouse), data warehouse chính là **database chuyên dùng** cho tạo **báo cáo** và **phân tích dữ liệu**. Nó vừa hỗ trợ các truy vấn phức tạp, vừa là điểm tập trung dữ liệu từ nhiều nguồn khác nhau để có được thông tin phân tích đầy đủ nhất. Theo đó, data warehouse là một tập hợp dữ liệu hướng chủ đề, toàn vẹn, không bị rò rỉ mất mát và có giá trị lịch sử. Cụ thể các tính chất đó như sau:

- **Tính hướng chủ đề (Subject – oriented)** nghĩa là data warehouse tập trung vào việc phân tích các yêu cầu quản lý ở nhiều cấp độ khác nhau trong quy trình ra quyết định. Các yêu cầu phân tích này thường rất cụ thể, và xoay quanh loại hình kinh doanh của doanh nghiệp, ví dụ các công ty phân phối sẽ quan tâm đến tình hình kinh doanh, doanh nghiệp viễn thông quan tâm đến lưu lượng dịch vụ… Tuy nhiên một doanh nghiệp thường quan tâm đến vài chủ đề khác nhau, như công ty phân phối còn phải quan tâm đến kho bãi, chuỗi cung ứng…

- **Tính toàn vẹn (Integrated)** giải quyết các khó khăn trong việc kết hợp dữ liệu từ nhiều nguồn dữ liệu khác nhau, giải quyết các sai khác về tên trường dữ liệu (dữ liệu khác nhau nhưng tên giống nhau), ý nghĩa dữ liệu (tên giống nhau nhưng dữ liệu khác nhau), định dạng dữ liệu (tên và ý nghĩa giống nhau nhưng kiểu dữ liệu khác nhau).
- **Tính bất biến (Nonvolatile)** quy định rằng dữ liệu phải thống nhất theo thời gian (bằng cách hạn chế tối đa sửa đổi hoặc xoá dữ liệu), từ đó làm tăng quy mô dữ liệu lên đáng kể so với hệ thống nghiệp vụ (5-10 năm so với 2 đến 6 tháng như database thông thường)
- **Giá trị lịch sử (time – varying)** nói về khả năng lấy các giá trị khác nhau của cùng một thông tin và thời điểm xảy ra thay đổi. Ví dụ thông tin địa chỉ, email, số điện thoại của khách hàng có thể thay đổi, nhưng việc thay đổi đó không được phép tác động đến giá trị báo cáo, phân tích thực hiện trước khi sự thay đổi xảy ra.

Tính chất	Database nghiệp vụ	Data warehouse
Người dùng	Nhân viên vận hành	Cán bộ quản lý, nhân viên phân tích số liệu
Loại hình sử dụng	Dự đoán được, lặp đi lặp lại	Truy vấn đột xuất, không xác định trước
Dữ liệu	Hiện tại, ở mức chi tiết	Lịch sử, ở mức tổng hợp
Tổ chức dữ liệu	Theo yêu cầu nghiệp vụ	Theo vấn đề cần phân tích
Cấu trúc dữ liệu	Tối ưu cho các giao dịch nhỏ	Tối ưu cho truy vấn phức tạp, trên lượng dữ liệu lớn
Tần suất truy cập	Tần suất cao	Tần suất từ trung bình đến thấp
Loại truy cập	Đọc, ghi, cập nhật, xoá	Đọc, ghi
Số lượng bản ghi mỗi phiên truy cập	Ít	Rất lớn
Thời gian truy cập	Ngắn	Tương đối dài (đến mức phút hoặc tiếng đồng hồ)
Mức độ xử lý song song	Cao, các tác vụ xử lý đồng thời trên một bản ghi nhất định xảy ra thường xuyên	Thấp
Khoá	Cần thiết	Không cần thiết
Tần suất cập nhật dữ liệu	Thường xuyên	Không cập nhật
Dư thừa dữ liệu	Thấp (bảng đã chuẩn hoá)	Cao (bảng dữ liệu thường phi chuẩn)
Mô hình dữ liệu	Mô hình quan hệ thực thể (Entity Relational)	Mô hình dữ liệu đa chiều (multidimensional)
Mô hình triển khai	Toàn bộ hệ thống	Tăng dần theo data mart

Bảng 1.1: So sánh database nghiệp vụ và data warehouse

Data warehouse cho phép người dùng ở mức quản lý, ra quyết định thực hiện các phép phân tích tương tác với data bằng hệ thống **xử lý phân tích trực tuyến (online analytical processing – OLAP)**. Ngoài ra data warehouse cũng được dùng cho báo cáo, data mining và phân tích thống kê. Database và data warehouse, do đó chỉ khác nhau về mặt khái niệm, một database nếu dùng riêng cho các mục đích trên cũng được coi là data warehouse.

Như vậy, nếu như database được ví như cái tủ sách cá nhân, nơi người ta thường xuyên tra cứu, cập nhật, hiệu đính, ghi chú vào lề, thêm mới hoặc chuyển sách đi, thì data warehouse lại được so sánh với thư viện quốc gia, nơi các tài liệu kinh điển được đưa đến liên tục để lưu trữ và tham khảo, không ai sửa chữa hoặc chuyển chúng qua chỗ nào khác cả.

1.3 Mô hình dữ liệu đa chiều

Data warehouse và các hệ thống OLAP được xây dựng dựa vào **mô hình dữ liệu đa chiều (multidimensional model)**. Mô hình này cho hiệu năng tốt trên những phép truy vấn phức tạp và giúp người dùng có thể nhìn dữ liệu theo nhiều khía cạnh khác nhau. Mô hình này hiển thị dữ liệu dưới dạng không gian n-chiều, gọi là **data cube** hoặc **hypercube**.

*Hình 1.1 Một cube 3 chiều hiển thị dữ liệu số lượng bán hàng với 3 chiều **Thị trường** (Store), **Thời gian** (Time), **Sản phẩm** (Product) và chỉ tiêu **Doanh số** (amount)*

Một khối data cube được xác định bằng cắt lớp và tiêu chí. **Cắt lớp (Dimension)** là các thông tin, quan điểm được dùng để phân tích dữ liệu. Ví dụ data cube ở hình 1.1 phân tích số liệu bán hàng, có 3 cắt lớp là ***Thị trường***, ***Thời gian*** và ***Sản phẩm***. Các giá trị trong một cắt lớp gọi là **lớp (dimension member).** Ví dụ Paris, Nice, Rome và Milan

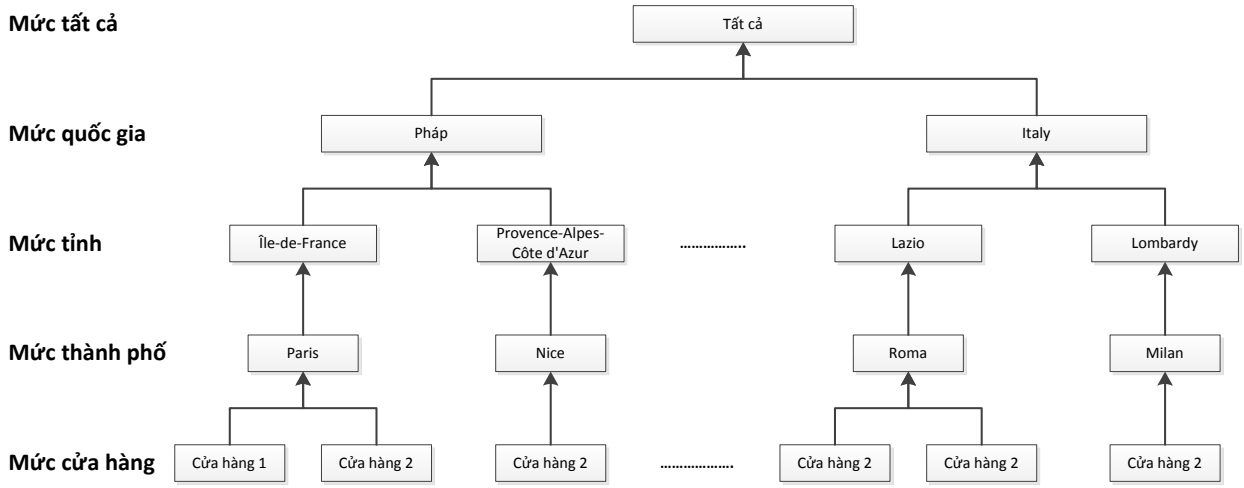
là các lớp của cắt lớp ***Thị trường***. Các cắt lớp thường có thêm các **thuộc tính (attribute)** mô tả thêm thông tin cho nó. Ví dụ cắt lớp ***Sản phẩm*** có thể chứa các thuộc tính như Mã sản phẩm, Tên sản phẩm, Mô tả, Kích thước…, tuy nhiên các thuộc tính này không được thể hiện trong hình trên.

Cùng với cắt lớp, các ô (cell) của một cube chứa các giá trị dạng số và được gọi là **tiêu chí (measure)**. Mô hình đa chiều yêu cầu việc thực hiện các phép toán số học (cộng, trừ, nhân, chia) trên các tiêu chí này mà ý nghĩa của số liệu vẫn chính xác. Ví dụ trong hình 1.1 trên, khối cube có 1 tiêu chí là ***Doanh số***. Thông thường một cube sẽ có nhiều tiêu chí khác nhau. Khối cube ở hình 1.1 mặc dù không hiển thị nhưng có thể có tiêu chí ***Số lượng*** (số sản phẩm bán ra) nữa.

1.3.1 Cây phân cấp

Mức độ chi tiết của các tiêu chí thể hiện cho người dùng được gọi là **mức dữ liệu (data granularity)**, được quyết định bằng việc kết hợp các mức dữ liệu của từng cắt lớp. Ví dụ trong hình 1.1 mức độ chi tiết là: mức thành phố với cắt lớp ***Thị trường***, mức quý với cắt lớp ***Thời gian***, mức loại hàng trong cắt lớp ***Hàng hoá***.

Để đúc rút ra tri thức từ dữ liệu, người dùng cần quan sát cube dưới nhiều mức chi tiết khác nhau. Vẫn ví dụ 1.1 trên, người dùng có thể muốn biết các tiêu chí bán hàng ở mức chi tiết hơn như mức cửa hàng, hoặc mức cao hơn như mức quốc gia chẳng hạn. Tính chất **cây phân cấp (hierrarchie)** của OLAP cho phép thực hiện điều này bằng cách định nghĩa ra một cấu trúc hình cây các mức độ chi tiết khác nhau của một cắt lớp. Với 2 mức độ liền nhau trong một cây, mức thấp hơn gọi là **mức con (child level)**, mức cao hơn gọi là **mức cha (parent level)**. Hình 1.2 bên dưới ví dụ các mức của cắt lớp ***Thị trường***, trong đó từng cửa hàng có thể được gán cho một thành phố, thành phố gán đến tỉnh, rồi đến quốc gia. Lớp trên cùng cây phân cấp là mức "Tất cả" đại diện cho toàn bộ cây phân cấp, mức này có 1 giá trị duy nhất cũng là "Tất cả" dùng để lấy tiêu chí đã được tổng hợp đến mức cao nhất đại diện cho toàn bộ cây phân cấp (trong ví dụ này là lấy tổng doanh số bán hàng của tất cả các quốc gia).



*Hình 1.2 Các giá trị của cây phân cấp **Thị trường***

//TODO: cân nhắc viết về cây phân cấp không cân bằng (cây phân cấp có nhánh bị thiếu mức so với nhánh khác).

1.3.2 Số liệu tổng hợp

Việc tổng hợp số liệu xảy ra khi người dùng thay đổi mức chi tiết của dữ liệu lấy ra từ cube, bằng cách duyệt qua cây phân cấp của cắt lớp. Ví dụ hình 1.1, nếu cắt lớp ***Thị trường*** sử dụng ở mức tỉnh thay vì mức thành phố thì doanh số của tất cả các thành phố trong cùng một tỉnh sẽ được tổng hợp bằng phép cộng. Tương tự, dữ liệu ở mức Tất cả được tổng hợp bằng giá trị dữ liệu của tất cả các quốc gia.

Nhằm đảm bảo tổng hợp chính xác, người ta đề suất ra một vài luật tổng hợp. Các luật tổng hợp chính bao gồm:

- **Tính tách biệt (Disjointness of instance)**: giao của các tập lớp cắt có mức cha khác nhau phải là tập rỗng. Ví dụ trong hình 1.2 một thành phố không được phép thuộc 2 tỉnh khác nhau.
- **Tính hoàn thiện (Completeness)**: tất cả lớp cắt đều phải xuất hiện trong cây phân lớp và ứng với mỗi lớp cắt đều phải tồn tại một lớp cắt cha ở tầng trên. Ví dụ hình 1.2 mỗi cửa hàng đều được gán cho một thành phố.
- **Sử dụng đúng phép toán tổng hợp (Correct use of aggregation function)**: mỗi tiêu chí có tính chất khác nhau, chính các tính chất này quyết định phép toán tổng hợp được phép sử dụng cho tiêu chí đó.

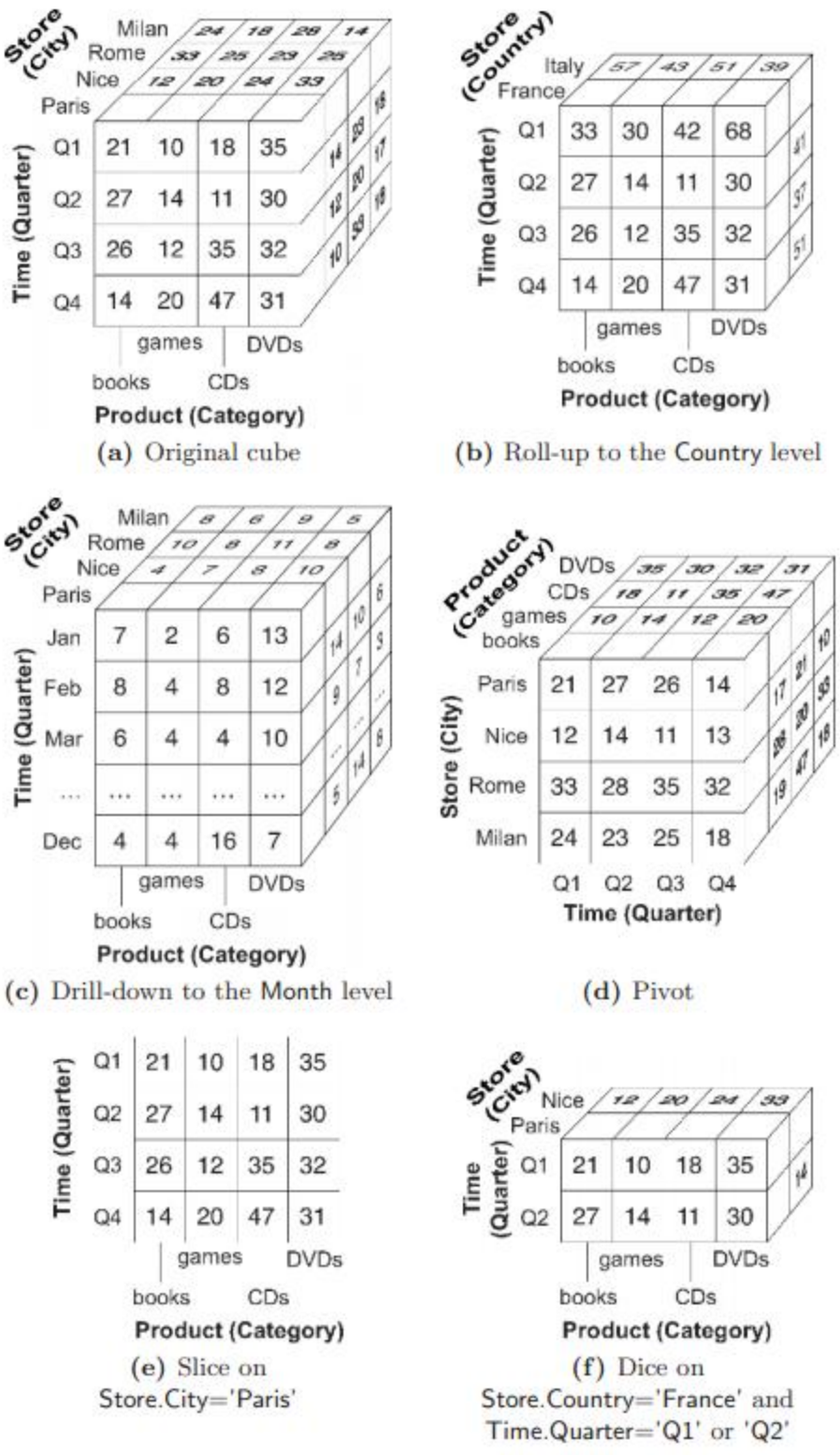
Một số phép toán tổng hợp chính như sau:

- **Các tiêu chí cộng dồn (Additive measure)** thông dụng nhất, là các tiêu chí có thể thực hiện phép tính cộng mà ý nghĩa vẫn chính xác. Ví dụ tiêu chí ***Doanh thu*** ở hình 1.1 là tiêu chí cộng dồn: hệ thống cộng giá trị doanh thu các lớp cắt con sẽ ra doanh thu lớp cắt cha.
- **Các tiêu chí bán cộng dồn (Semiadditive measure)** là tiêu chí cộng dồn nhưng ý nghĩa của nó sẽ bị sai đi nếu dùng với một số cắt lớp nào đó. Ví dụ tiêu chí ***Số lượng hàng*** cho biết trong kho còn bao nhiêu hàng và có thể cộng dồn để biết một thành phố, một tỉnh còn bao nhiêu hàng. Nhưng tiêu chí này nếu kết hợp với dimension ***Thời gian*** sẽ mất ý nghĩa vì nó chỉ mang tính thời điểm.
- **Các tiêu chí không cộng dồn (Nonadditive measure – value-per-unit)** là các tiêu chí không cho phép thực hiện các phép cộng, trừ. Ví dụ cho tiêu chí này chính là các tiêu chí tỷ lệ, trung bình.

Khi xác định tiêu chí cần thiết phải chỉ rõ các phép toán tổng hợp dùng cho các cắt lớp cho data warehouse, đặc biệt quan trọng trong trường hợp tiêu chí bán cộng dồn và không cộng dồn. Ví dụ tiêu chí ***Số lượng hàng*** ở trên là bán cộng dồn, tuy không thể dùng để cộng trừ với cắt lớp ***Thời gian*** nhưng vẫn có thể thực hiện phép tính trung bình, trung vị, tìm max, min.

1.3.3 Tính năng của OLAP

Như đã nói ở trên, tính chất cơ bản của mô hình dữ liệu đa chiều là cho phép người dùng quan sát dữ liệu trên nhiều phương diện khác nhau, ở các mức độ chi tiết khác nhau. OLAP cung cấp một số tính năng cho phép thực hiện điều đó, cụ thể:



Hình 1.3 Các tính năng của OLAP

- Tính năng **nhìn xa (roll-up)** biến tiêu chí từ mức chi tiết sang mức tổng hợp để hiển thị cho người dùng, được thực hiện khi đi từ mức thấp lên mức cao trong cây phân cấp hoặc giảm số cắt lớp xuống. Hình 1.3b là ví dụ cho tính năng nhìn xa này khi cắt lớp ***Thị trường*** chuyển từ mức thành phố lên mức quốc gia, giá trị các lớp cắt thành phố của một quốc gia được cộng dồn vào thành giá trị kết quả.
- Tính năng **đào sâu (drill-down)** thực hiện ngược lại với nhìn xa, tức là đi từ mức tổng hợp cao đến mức chi tiết hơn. Ví dụ như trong hình 1.3c, cắt lớp ***Thời gian*** đi từ mức quý xuống mức các tháng trong quý.
- Tính năng **đảo chiều** (**pivot** hoặc **rotate**) biến hàng thành cột, cột thành hàng giúp cung cấp cho người dùng một cách thể hiện dữ liệu khác. Tính năng này được thể hiện ở hình 1.3d.
- Tính năng **cắt lát mỏng** (**slice**) thực hiện cắt lấy dữ liệu một lớp cắt cụ thể trong một cắt lớp. Ví dụ như hình 1.3e chỉ duy nhất dữ liệu của thành phố Paris được hiển thị.
- Tính năng **cắt khối** (**dice**) thực hiện lựa chọn giá trị cho ít nhất hai lớp cắt. Ví dụ như hình 1.3f là cube thể hiện dữ liệu cho thành phố Paris trong quý 1 và quý 2.

Ngoài 5 tính năng cơ bản trên, các bộ công cụ OLAP trên thị trường cũng cung cấp thêm một loạt các tính năng hỗ trợ khác như các phép toán số học, thống kê, các phép toán kinh tế…

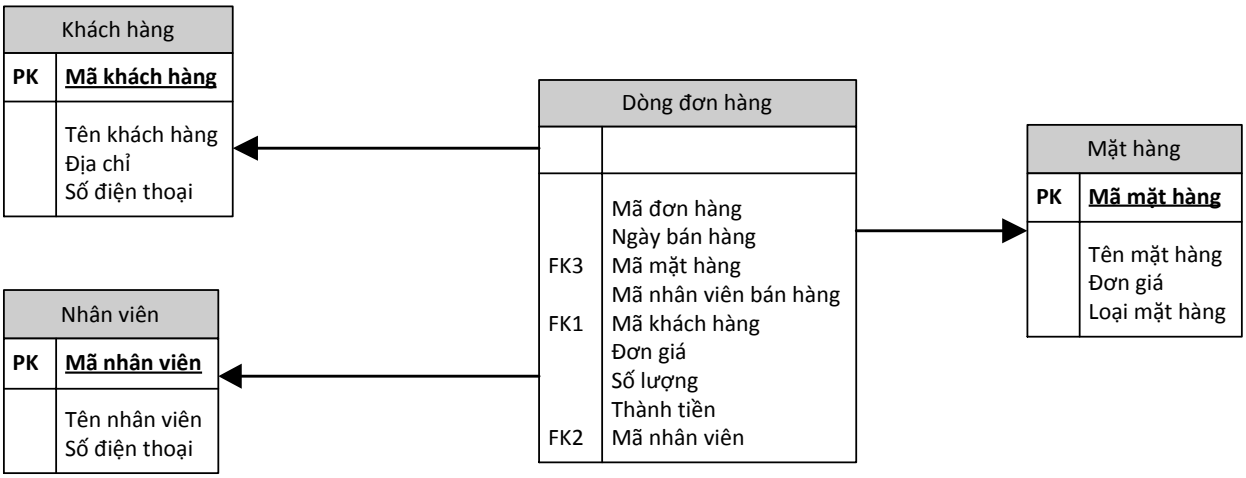
1.3.4 Mô hình thiết kế data warehouse

Căn cứ vào cách thức lưu trữ dữ liệu, người ta thường tiếp cận mô hình dữ liệu đa chiều theo 3 hướng sau:

- **OLAP kiểu quan hệ** (**Relational OLAP** – **ROLAP**) lưu trữ dữ liệu trong cơ sở dữ liệu quan hệ, dùng câu lệnh SQL để thực hiện các tính năng của OLAP.
- **OLAP đa chiều** (**Multidimensional OLAP – MOLAP**) lưu trữ dữ liệu dưới dạng file có cấu trúc đặc thù (ví dụ như cấu trúc dạng mảng – array) và thực hiện các tính năng OLAP trên cấu trúc này. Mặc dù bị hạn chế về lượng dữ liệu lưu trữ và xử lý được so với ROLAP, MOLAP thường cho hiệu năng tốt hơn trong các phép truy vấn hoặc tổng hợp số liệu (vì dữ liệu được thiết kế tối ưu cho truy vấn OLAP trong khi ROLAP phải thông qua database).
- **OLAP lai** (**Hybrid OLAP – HOLAP**) kết hợp 2 công nghệ ROLAP và MOLAP nói trên, tận dụng khả năng lưu trữ của OLAP và khả năng xử lý của MOLAP. Ví dụ HOLAP sẽ lưu dữ liệu chi tiết trên cơ sở dữ liệu quan hệ còn dữ liệu tổng hợp hơn để truy vấn cho người dùng được lưu trên không gian MOLAP.

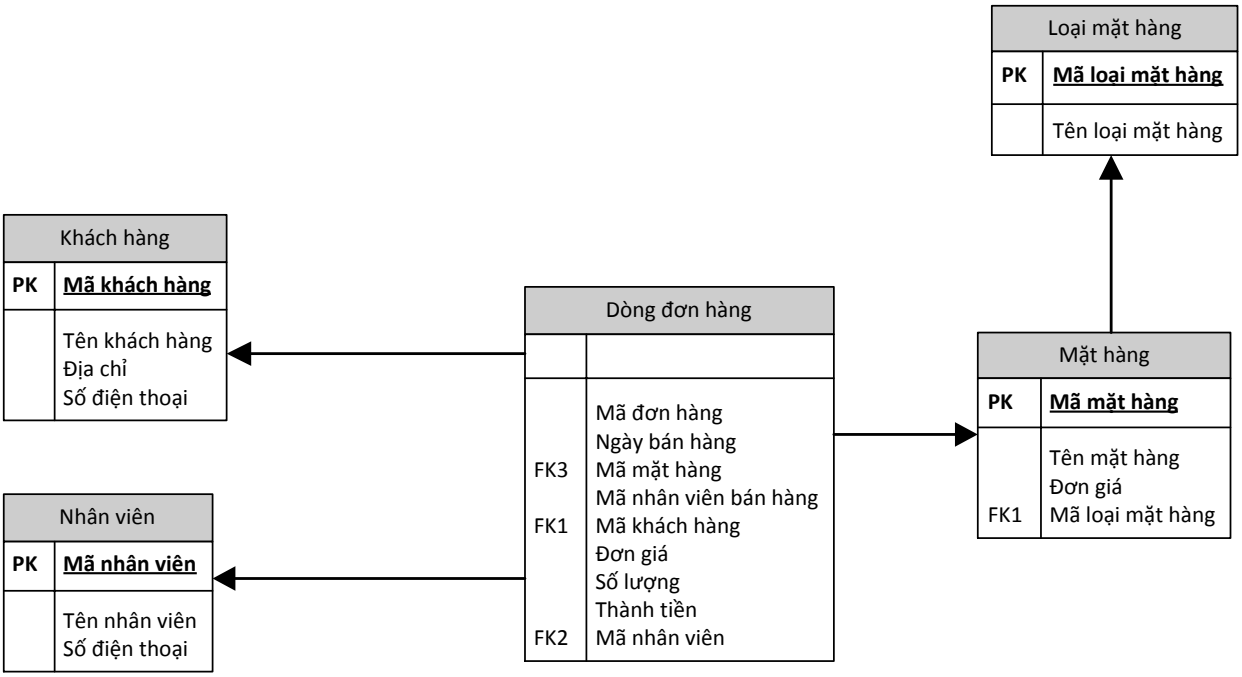
Trong hệ thống ROLAP, dữ liệu đa chiều được lưu trữ dưới dạng bảng quan hệ, tổ chức theo cấu trúc đặc biệt theo lược đồ hình sao, lược đồ hình bông tuyết, lược đồ ánh sao và lược đồ chòm sao như sau:

- **Lược đồ hình sao** (**star schema**) bao gồm duy nhất một bảng fact và nhiều bảng dimension (mỗi bảng cho một dimension). Các thực thể trong lược đồ hình sao không được chuẩn hoá như database nghiệp vụ (các thực thể có cấu trúc phân cấp được nhập chung vào làm một). Ví dụ thực thể *Mặt hàng* trong database nghiệp vụ sẽ được tách thành 2 thực thể *Mặt hàng* và *Loại mặt hàng*.



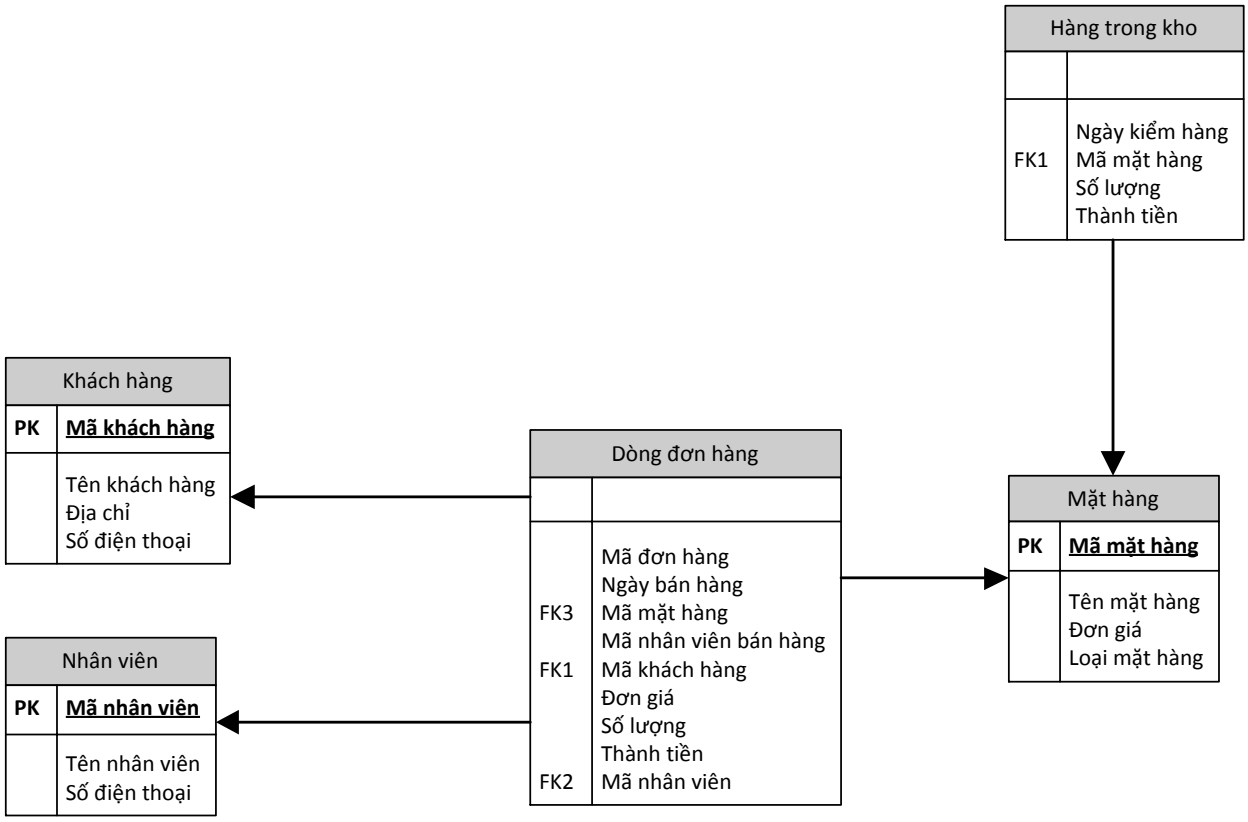
Hình 1.4: Lược đồ hình sao

- **Lược đồ hình bông tuyết** (**snowflake schema**) giảm bớt dư thừa dữ liệu trong lược đồ hình sao bằng cách chuẩn hoá các bảng dimension. Do đó, một thực thể dimension có phân cấp sẽ được thể hiện thành nhiều bảng dữ liệu khác nhau, mỗi bảng một cấp. Hình 1.5 là lược đồ hình bông tuyết, trong đó dimension *Mặt hàng* được thể hiện qua 2 bảng dữ liệu *Mặt hàng* và *Loại mặt hàng*.



Hình 1.5: Lược đồ hình bông tuyết

- **Lược đồ ánh sao** (**starflake schema**) là sự kết hợp giữa lược đồ hình sao và lược đồ hình bông tuyết khi một số dimension được chuẩn hoá trong khi một số khác thì không.
- **Lược đồ chòm sao** (**constellation schema**) là lược đồ thông dụng nhất trong thiết kế data warehouse, là lược đồ trong đó các bảng fact dùng chung dimension với nhau. Ví dụ hình 1.6 bên dưới hai bảng fact *Dòng đơn hàng* và *Hàng trong kho* sử dụng chung dimension *Mặt hàng*.



Hình 1.6: Lược đồ chòm sao

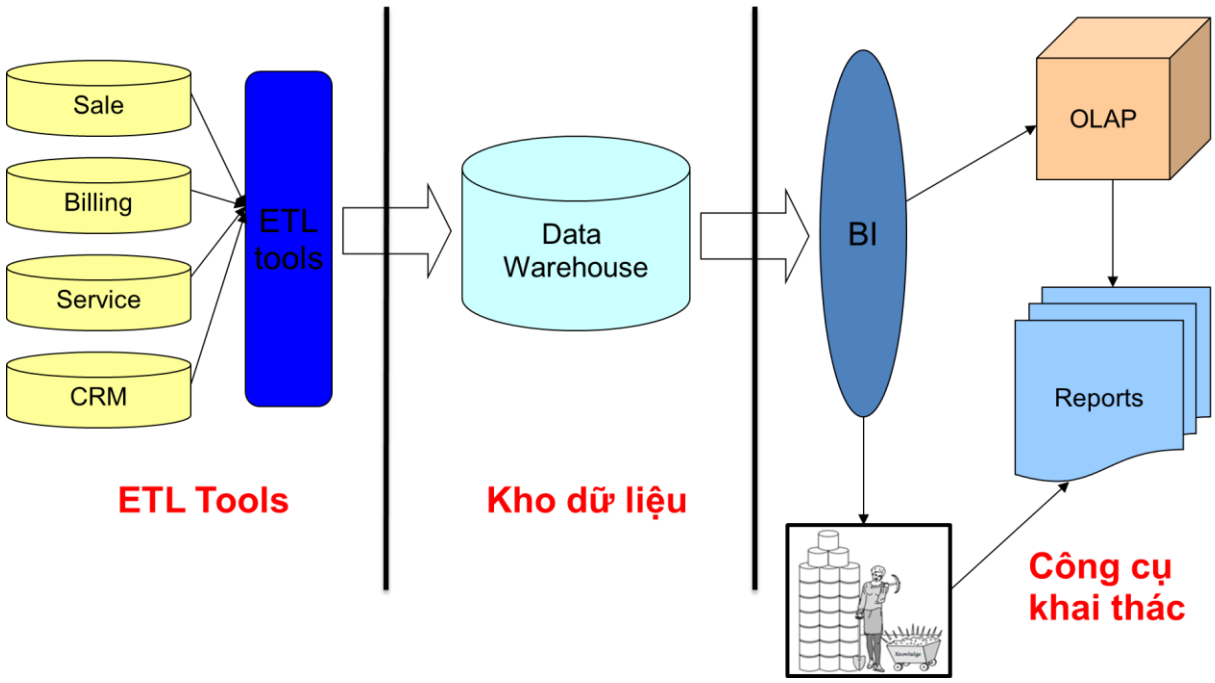
1.4 Ralph Kimball vs Bill Inmon

//TODO: so sánh lý thuyết của Kilball và Inmon

1.5 Kiến trúc hệ thống data warehouse

Một hệ thống data warehouse bao gồm 3 thành phần chính sau:

- Một bộ công cụ để **thu thập** dữ liệu từ hệ thống nghiệp vụ, **chuẩn hoá** chúng về định dạng dữ liệu đa chiều, **nạp** vào data warehouse (**Extract-Transformation-Loading** – **ETL**).
- Một database dùng làm **data warehouse** để lưu trữ dữ liệu
- Một loạt các công cụ khai thác dữ liệu từ data warehouse như hệ thống OLAP, hệ thống báo cáo tĩnh, hệ thống data mining…



Hình 1.7: Kiến trúc hệ thống Data Warehouse

1.5.1 Tầng ETL

Tầng **ETL** (**Extract – Transform – Load**) là tầng thấp nhất, ẩn đi với người dùng cuối, bao gồm 3 bước:

- Bước **thu thập** (**extract**) gom góp dữ liệu từ nhiều khác nhau về. Các nguồn này có thể là database hệ thống nghiệp vụ (MS SQL, mySQL, Oracle, DB2…), cũng có thể là file ở các định dạng khác nhau (CSV, fix-length, excel, XML…), có thể là dữ liệu nội bộ doanh nghiệp hoặc từ bên ngoài. Một hệ thống ETL tốt phải đảm bảo tương thích với các nguồn dữ liệu thông dụng này.
- Bước **chuẩn hoá** (**transform**) biến đổi dữ liệu từ định dạng nguồn sang định dạng của data warehouse (định dạng dữ liệu đa chiều đã nói ở bước trên), bao gồm các bước nhỏ:
    - Bước **dọn dẹp** (**cleaning**) xoá các bản ghi bị sai, lỗi và chuyển hoá dữ liệu về định dạng chuẩn chung.
    - Bước **tập hợp** (**integration**) cắt gọt dữ liệu có chung ý nghĩa từ nhiều nguồn khác nhau về một khung duy nhất.
    - Bước **tổng hợp** (**aggregation**) tổng hợp dữ liệu dựa vào độ chi tiết của data warehouse.
- Bước **nạp** dữ liệu (**load**) ghi dữ liệu đã được chuẩn hoá vào data warehouse. Bước này bao gồm cả quá trình cập nhật thay đổi từ hệ thống nghiệp vụ vào data warehouse, đảm bảo số liệu báo cáo luôn được cập nhật. Tuỳ thuộc vào chính sách

công ty, việc cập nhật này có thể phải thực hiện theo thời gian thực, cập nhật theo giờ, theo ngày hoặc thậm chí theo tháng.

1.5.2 Tầng data warehouse

Tầng data warehouse đứng ở trung tâm một hệ thống data warehouse làm nhiệm vụ lưu trữ dữ liệu bao quanh tất cả các hoạt động nghiệp vụ, các phòng ban của doanh nghiệp. Data warehouse thường bao gồm một hoặc nhiều **data mart**, với data mart chính là data warehouse thu nhỏ tập trung vào một nghiệp vụ nhất định nào đó của doanh nghiệp (ví dụ data mart về sale, data mart về kho bãi, data mart về nhân sự…)

Ngoài nhiệm vụ lưu trữ dữ liệu, tầng data warehouse còn có một thành phần khác rất quan trọng gọi là **siêu dữ liệu** (**metadata**). Siêu dữ liệu lại được chia làm 2 nhóm là nhóm siêu dữ liệu kỹ thuật và siêu dữ liệu nghiệp vụ. **Siêu dữ liệu nghiệp vụ** (**business metadata**) mô tả ý nghĩa dữ liệu, các luật và ràng buộc tác động lên dữ liệu. **Siêu dữ liệu kỹ thuật** (**technical metadata**) mô tả cách thức tổ chức, lưu trữ và điều khiển dữ liệu trong hệ thống máy tính.

Trong phạm vi data warehouse, siêu dữ liệu kỹ thuật được sử dụng để mô tả thông tin về data warehouse, về dữ liệu nguồn và các tiến trình ETL. Cụ thể:

- Siêu dữ liệu mô tả cấu trúc data warehouse và các data mart ở mức logic (mô tả bảng fact, bảng dimension, cây phân cấp, nguồn gốc dữ liệu) và mức vật lý (cấu trúc bảng, index, partition). Ngoài ra nó còn chứa thông tin bảo mật dữ liệu (xác thực, phân quyền người dùng) và các thông tin giám sát (thống kê hiệu năng sử dụng, báo cáo lỗi…)
- Siêu dữ liệu mô tả dữ liệu nguồn, cũng ở mức logic (cách thức và tham số kết nối lấy dữ liệu, tần suất cập nhật dữ liệu, ý nghĩa dữ liệu) và vật lý (cấu trúc dữ liệu)
- Siêu dữ liệu mô tả các tiến trình ETL, bao gồm cả gốc gác dữ liệu (truy được dữ liệu trên data warehouse về đến gốc gác của nó trong hệ thống nghiệp vụ), các luật thu thập, làm sạch, chuyển hoá dữ liệu.

1.5.3 Tầng khai thác dữ liệu

Tầng khai thác dữ liệu chứa các công cụ cho người dùng cuối khai thác, sử dụng các dữ liệu trong data warehouse. Một số công cụ chính:

- **Báo cáo OLAP** (**OLAP tool**) là báo cáo động cho phép người dùng sử dụng các tính năng của OLAP (đã nói ở phần 1.3.3) để tạo báo cáo. Các truy vấn đột xuất này được gọi là **truy vấn tuỳ biến** (**ad hoc query**) vì hệ thống không hề được chuẩn bị trước cho thao tác của người dùng. Báo cáo OLAP được sử dụng khi

người dùng muốn các thông tin cắt lớp, chuyên sâu hoặc toàn cảnh trước khi ra quyết định.

- **Báo cáo tĩnh** (**reporting tool**) là các báo cáo có cấu trúc, format, sử dụng truy vấn được định nghĩa trước đó, đôi khi bao gồm cả biểu đồ. Báo cáo tĩnh được sử dụng khi người dùng muốn xem các thông tin đánh giá, điều hành.
- Bộ công cụ **khai phá dữ liệu** (**data mining**) cho phép người dùng phân tích dữ liệu để tìm ra các thông tin quý giá còn bị ẩn dấu, ví dụ như các xu hướng, các mẫu chung.

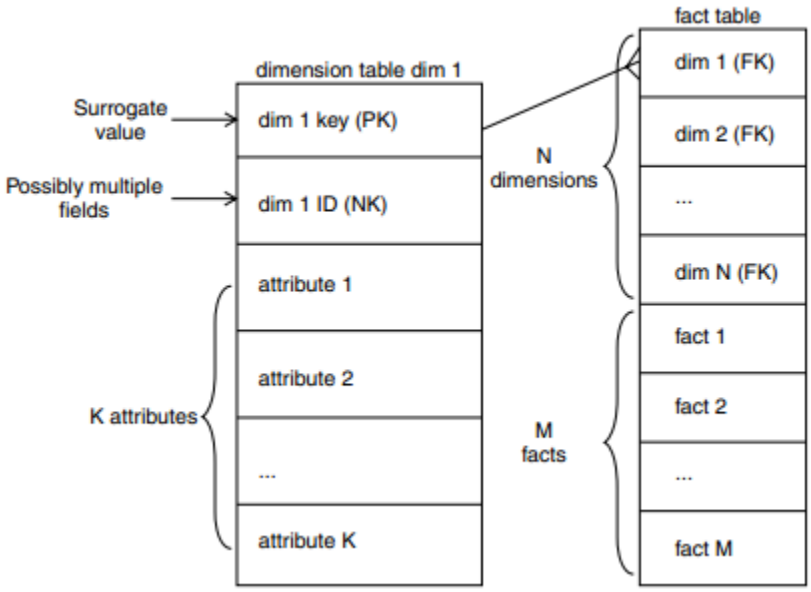
Phần 2: Xây dựng Data Warehouse

Phần 1 của tài liệu đã mô tả data warehouse, mục đích, ý nghĩa và kiến trúc data warehouse. Phần 2 này sẽ giới thiệu các khái niệm trong thiết kế data warehouse.

2.1. Xây dựng bảng dimension

Bảng dimension cung cấp các thông tin, ngữ cảnh cho bảng fact và do đó cũng là cung cấp cho tất cả số liệu thể hiện trong data warehouse. Dù có quy mô nhỏ hơn bảng fact rất nhiều lần, các bảng dimension lại là trái tim và khối óc của data warehouse vì muốn truy cập số liệu data warehouse đều phải thông qua chúng. Có người nói rằng, tốt xấu trong thiết kế của một data warehouse chính là tốt xấu trong thiết kế các bảng dimension của nó.

2.1.1 Cấu trúc bảng dimension



Hình 2.1: Cấu trúc cơ bản một bảng dimension

Về cơ bản các bảng dimension đều có cấu trúc vật lý như hình 2.1. Khoá chính của bảng dimension là trường dữ liệu (thường là kiểu số) lưu những giá trị duy nhất, không có ý nghĩa, gọi là **khoá thay thế** (**surrogate key**). Khoá thay thế này được nội tại hệ thống data warehouse sinh ra bằng các luồng ETL xử lý dữ liệu. Giá trị khoá này chỉ được tạo ra duy nhất trong nội tại data warehouse, các thao tác thay đổi bên ngoài đều bị cấm.

Trước đây việc sinh giá trị khoá thay thế thường được phó mặc cho database dùng làm data warehouse, cụ thể là cho tính năng database trigger. Càng ngày người ta càng nhận ra rằng việc dùng data trigger làm chậm cả tiến trình ETL và hạn chế dùng nó. Việc

dùng database tạo khoá chính cũng không được khuyến khích nữa vì khi đó data warehouse phải phụ thuộc một tiến trình khác từ bên ngoài, dễ gây mất đồng bộ (trong trường hợp database dùng làm data warehouse trong giai đoạn xây dựng hệ thống và giai đoạn triển khai khác nhau). Do đó hướng tiếp cận an toàn nhất vẫn là dùng chính luồng ETL để sinh giá trị cho khoá thay thế này.

Một thành phần khác của bảng dimension là khoá chính của dữ liệu trong hệ thống nghiệp vụ, được gọi là **khoá tự nhiên** (**natural key**), giá trị của khoá tự nhiên thường không phải vô nghĩa. Ví dụ bảng dimension *Nhân viên* sẽ có trường EMP_ID lưu mã nhân viên lấy từ hệ thống nghiệp vụ. Mặc dù trường EMP_ID cũng có thể dùng làm khoá chính cho bảng dimension *Nhân viên* này nhưng khi thiết kế vẫn phải cung cấp cho bảng dimension một khoá thay thế (trong trường hợp phải nhập dữ liệu từ 2 hệ thống khác nhau dẫn đến khoá tự nhiên có thể trùng nhau, hoặc trường hợp giá trị dimension bị thay đổi, sẽ trình bày sau).

Có ý kiến cho rằng dimension đã có khoá tự nhiên có thể dùng làm khoá chính rồi, không nhất thiết phải dùng khoá thay thế kiểu số vô nghĩa nữa mà có thể dùng một giá trị có nghĩa nào đó như thời gian thay đổi, khi đó tập thuộc tính {khoá tự nhiên, thời gian thay đổi} chính là khoá chính của bảng dimension. Hướng tiếp cận này có thể có lợi trong một vài trường hợp nhưng lại bế tắc trong các tình huống sau:

- **Sai định nghĩa**: Khoá thay thế, theo định nghĩa, tự bản thân nó không có ý nghĩa gì cả. Nếu như cố tình gán ý nghĩa cho khoá thay thế thì người thiết kế ETL phải thêm luồng xử lý để quản lý ý nghĩa các giá trị này, khiến cho việc xây dựng luồng ETL phức tạp hơn do đó cũng chạy lâu hơn.
- **Giảm hiệu năng**: Thêm ý nghĩa cho khoá thay thế khiến các câu truy vấn từ người dùng cuối phải thêm điều kiện xác thực, khiến câu lệnh phức tạp hơn, truy vấn tốn tài nguyên và thời gian hơn so với phép so sánh 2 giá trị kiểu số đơn thuần.

Thành phần cuối cùng của bảng dimension, bên cạnh khoá chính và khoá tự nhiên, là một loạt các **thuộc tính mô tả** (**desciptive attribute**). Các thuộc tính mô tả có thể ở nhiều kiểu dữ liệu khác nhau và số lượng có thể rất lớn (đặc biệt với dimension như khách hàng, nhân viên, sản phẩm…). Nhìn chung là không nên quá sợ hãi khi thiết kế ra bảng dimension có hơn 100 thuộc tính mô tả, làm vậy không sai đâu. Chỉ cần lưu ý làm sạch dữ liệu cẩn thận cho các thuộc tính này là được.

Một lưu ý nhỏ là nên chú ý đến các thuộc tính mô tả có kiểu dữ liệu là kiểu số, vì tuỳ vào ý nghĩa, nó có thể lại là một thuộc tính đo đếm được và phải đặt vào bảng fact. Thông tin mô tả chỉ được dùng để mô tả, không phải dùng để cộng dồn. Cũng không nên

quá lo lắng vì trong 99% trường hợp giá trị này được phân ra là fact hay thuộc tính dimension ngay. 1% còn lại phân vào đâu cũng được, và việc phân thuộc tính vào đâu không làm thay đổi ý nghĩa của thuộc tính, mà chỉ thay đổi cách xử lý thuộc tính đó.

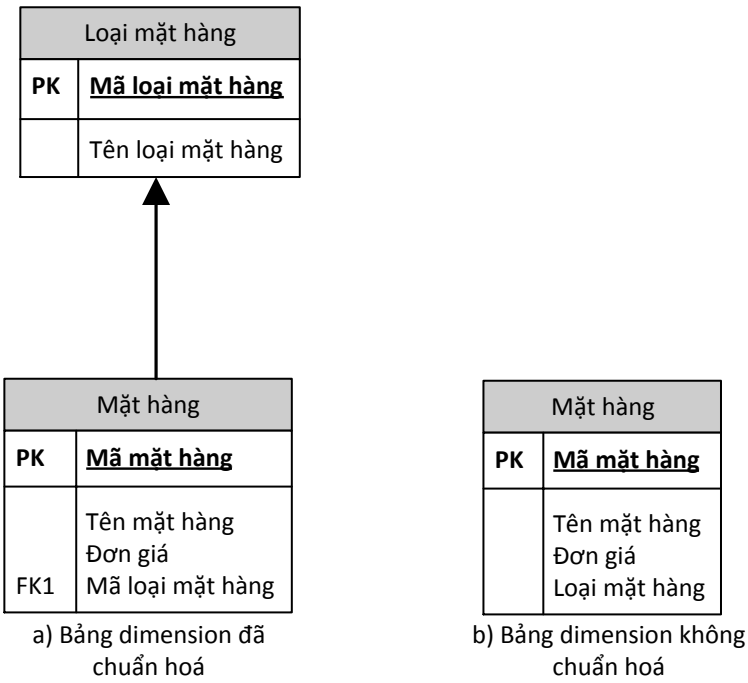
//TODO: ví dụ thuộc tính dimension có thể dùng trong fact

2.1.2 Thiết kế phẳng và thiết kế bông tuyết

Bảng dimension là các bảng dữ liệu được phi chuẩn hoá về dạng bảng phẳng. Trong đó, tất cả dữ liệu phân cấp và các cấu trúc được chuẩn hoá của hệ thống nghiệp vụ được thiết kế lại về dạng phẳng. Dữ liệu trong bảng dimension, vì thế, là dư thừa và tương đương với dạng chuẩn 2 trong thiết kế database nghiệp vụ.

Một bảng dimension có thể bao hàm nhiều hơn một cấu trúc phân cấp. Ví dụ dimension *Cửa hàng* trong hình 1.1 có thể có cấu trúc cây theo phân cấp địa lý theo quản lý hành chính, vừa có cấu trúc cây theo phân cấp của nội bộ doanh nghiệp. Cả hai cây phân cấp này đều có thể nằm trong một cây phân cấp, với điều kiện ràng buộc duy nhất là dù ở cây phân cấp nào, giá trị của thuộc tính phân cấp luôn là duy nhất.

Nếu một bảng dimension ở dạng chuẩn hoá, cấu trúc phân lớp sẽ được thể hiện dưới dạng lược đồ hình bông tuyết (hình 2.2a). Chú ý rằng về mặt nội dung dữ liệu thì hai cách thể hiện này không khác gì nhau, tuy nhiên mỗi cách thể hiện lại có điểm lợi, điểm hại khi người dùng thao tác. Trường hợp bảng phẳng không chuẩn hoá là dư thừa dữ liệu, dễ gây ra sai sót mất đồng bộ giữa các bản ghi (lớp cha giống nhau nhưng thông tin của lớp cha lại khác nhau). Trường hợp của bảng chuẩn hoá là giảm hiệu năng khi query dữ liệu (do phải join nhiều bảng với nhau) và gây khó hiểu cho người dùng không am hiểu kỹ thuật (các nhân viên kinh doanh, phân tích số liệu, cán bộ quản lý cấp cao)

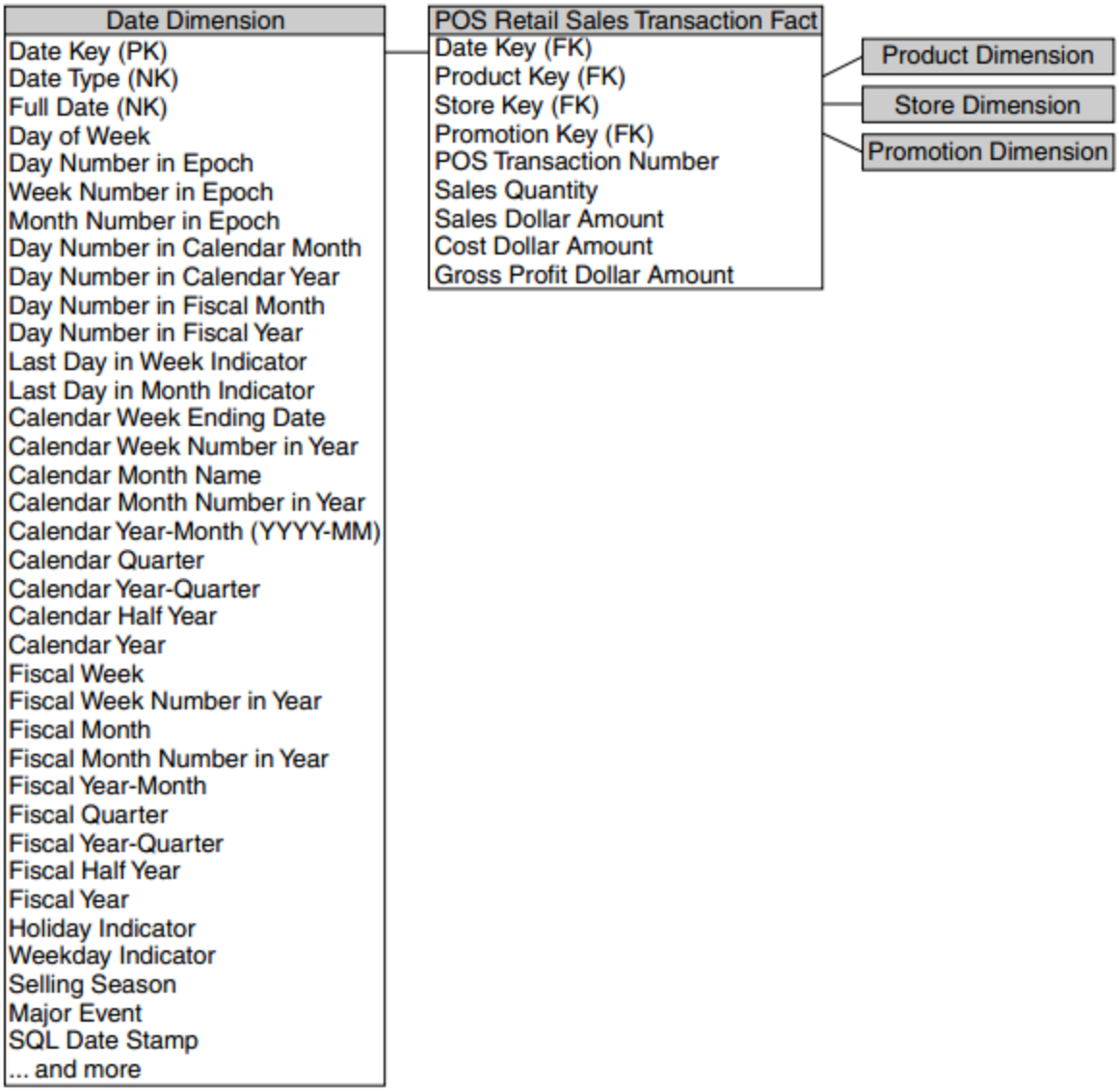


Hình 2.2: Các dạng thể hiện bảng dimension

Mỗi khi thêm bản ghi mới vào bảng dimension, hệ thống phải gán cho bản ghi đó một khoá thay thế, đóng vai trò làm khoá chính cho bảng dimension (hình 2.1). Trong môi trường data warehouse, cần thiết phải có tiến trình ETL quản lý giá trị khoá thay thế này cho mỗi bảng dimension (đơn giản nhất là tìm giá trị khoá cao nhất có sẵn trong bảng, cộng thêm 1 rồi gán cho khoá mới, tuy nhiên cách này bị hạn chế về hiệu năng).

2.1.3 Dimension thời gian

Gần như tất cả bảng fact đều có ít nhất một giá trị dimension là thời gian. Các phép đo đạc đều được thực hiện tại điểm mốc nào đó và được xoay vòng sau mỗi khoảng chu kỳ nhất định.



Hình 2.3: Dimension thời gian

Dimension thời gian được sử dụng nhiều nhất chính là lịch ngày, với đơn vị nhỏ nhất mức bản ghi của bảng dimension chính là một ngày. Hơi bất ngờ tý là dimension này lại có khá nhiều thuộc tính, như hình vẽ 2.3 ở trên. Chỉ một vài thuộc tính là có thể tự sinh ra từ câu lệnh SQL (như thứ, ngày, tháng, quý, năm). Các trường dữ liệu còn lại (như ngày làm việc, ngày nghỉ, năm tài chính…) sẽ khác biệt tuỳ vào chính sách của quốc gia, chính sách công ty (ví dụ Việt Nam có ngày quốc lễ giỗ tổ Hùng Vương tính là ngày nghỉ, có công ty làm 5 ngày/tuần, công ty khác 6 ngày/tuần…). Dimension thời gian là kiểu dimension đặc biệt nhất trong dự án data warehouse, thông dụng nhất và đặc biệt nhất. Thông thường dimension này được tạo một lần vào đầu dự án rồi giữ nguyên gần như không cập nhật trong suốt vòng đời xây dựng, vận hành, nâng cấp dự án đó (trừ khi

chính sách công ty thay đổi, tăng lương giảm giờ làm, trước làm 6 ngày nay giảm xuống còn 5, hoặc nhà nước thêm một ngày quốc lễ mới, ví dụ vậy). Cách tạo bảng dimension thời gian hay nhất là dành ra một buổi chiều ngồi nghịch quyển lịch, viết ra excel rồi ghi dữ liệu excel đó vào data warehouse. Dù có "9 năm làm một Điện Biên – Nên vành hoa đỏ nên thiên sử vàng" thì cũng chưa mất đến 4000 bản ghi, không quá nhiều.

Như trên đã viết, khoá chính của bảng dimension chỉ chứa giá trị có ý nghĩa định danh, tự nó không có giá trị gì cả. Tuy nhiên với dimension thời gian, rất nhiều dự án data warehouse lại gán giá trị có nghĩa cho khoá chính, mà cụ thể là giá trị kiểu YYYYMMDD (ví dụ 20130523 cho ngày 23, tháng 05, năm 2013). Về việc gán giá trị có nghĩa cho khoá thay thế này hay không còn nhiều tranh cãi, và mỗi bên đều đưa ra các trường hợp lợi hơn và bất lợi hơn. Không có trường hợp nào mà một giải pháp bế tắc trong khi giải pháp kia chạy được cả, chỉ có **lợi hơn** hoặc **bất lợi hơn**, nên không đi sâu vào chi tiết nữa. Tuy nhiên đội dự án Viettel BI chọn kiểu YYYYMMDD để nhân viên vận hành dễ thực hiện hơn.

Trong nhiều trường hợp, thời gian trong bảng fact cần được tính toán dưới mức ngày, ở mức giờ hoặc phút. Khi đó theo phản xạ chúng ta nghĩ ngay đến việc tạo một bảng dimension ở mức giờ hoặc phút tương ứng này. Cứ cẩn thận, một năm có 8.760 giờ, 525.600 phút, 31.536.000 giây, lớn thế này mà query không khéo chết cả cái data warehouse. Nếu gặp phải tình hình trên chúng ta có thể tạo bảng dimension 24h, dimension 60 phút, dimension 60 giây rồi kết hợp chúng với nhau. Hoặc có thể ghi luôn giá trị thời gian (kiểu datetime của database) vào bảng fact, coi nó là một giá trị đặc biệt.

2.1.4 Bảng dimension khổng lồ

Khi xây dựng data warehouse đôi khi chúng ta gặp các bảng dimension có số lượng bản ghi khổng lồ, có quy mô tương đương hoặc thậm chí lớn hơn bảng fact. Như ở Viettel, dimension *Thuê bao* lên tới hàng trăm triệu bản ghi.

Chưa hết, các dimension khổng lồ này thường tập trung thuộc tính của nhiều nguồn khác nhau nữa, tạo nên bảng dữ liệu có số trường vô cùng lớn, số cột vô cùng nhiều.

Không may cho chúng ta, các bảng dimension khổng lồ này thường chứa các thông tin khách quan, không phụ thuộc nội tại công ty, doanh nghiệp, do đó tần suất cập nhật, thêm mới dữ liệu là rất lớn. Vấn đề này sẽ được nói đến trong phần sau "Cập nhật giá trị dimension".

2.1.5 Bảng dimension tí hon

Trong data warehouse cũng tồn tại các bảng dimension tí hon chỉ có một hai cột dữ liệu và một vài bản ghi. Các dimension này thường không có nguồn riêng biệt mà được trích rút từ một thuộc tính nào đó của dữ liệu nguồn. Ví dụ dimension *Giới tính* được trích từ bảng *Thông tin thuê bao* với 3 giá trị "Nam", "Nữ" và "Không cung cấp". Hoặc trong viễn thông có dimension *Loại cuộc gọi* được trích xuất từ chi tiết cước thoại chứa mã code phân biệt cuộc gọi thường, gọi video call hoặc gọi chuyển tiếp.

Mỗi dimension dạng này, dù tí hon, vẫn cần xử lý rất cẩn thận. Thông thường chúng không có luồng ETL riêng biệt để thêm dữ liệu vào, mà ăn luôn trong luồng xử lý nguồn dữ liệu chúng được đúc rút ra.

2.1.6 Dimension lồng nhau

Dimension lồng, hay dimension nằm trong dimension khác, là một kỹ thuật thường gặp khi xây dựng data warehouse. Ví dụ dimension *Thuê bao* thường sẽ có ngày kích hoạt (là một dimension) và huyện kích hoạt (cũng là một dimension khác). Khi gặp tình huống này, nên nhớ rằng đây là một tình huống thường xuyên xảy ra và không có gì sai sót khi thiết kế data warehouse cả.

2.1.7 Một bảng dimension hay tách ra làm hai?

Khi thiết kế data warehouse, quan điểm thường thấy là các bảng dimension độc lập nhau. Quan điểm này không hoàn toàn chính xác, dù đúng trong 90% các trường hợp. Ví dụ như mối quan hệ giữa nhân viên bán hàng và cửa hàng chẳng hạn. Có công ty quy định nhân viên bán hàng phải thuộc một cửa hàng nhất định, chỉ được lấy hàng từ cửa hàng đó thôi. Khi đó mối quan hệ giữa Nhân viên – Cửa hàng là quan hệ 1 – nhiều, là dạng cây thư mục và nhất định phải nằm trong một dimension duy nhất thôi. Nhưng trong công ty khác mối quan hệ lại lỏng hơn, một nhân viên, dù biên chế thuộc một cửa hàng nhưng lại có thể lấy hàng từ nhiều nguồn khác nhau chẳng hạn. Khi đó không nhất thiết và cũng không nên tạo một dimension duy nhất mà nên chia ra thành 2 thực thể dimension khác nhau.

Nhìn chung nên thiết kế các bảng dimension độc lập hoàn toàn với nhau (hoặc lồng nhau, như đã trình bày ở mục 2.1.6), dimension có quan hệ nhiều – nhiều nên tách ra thành một bảng fact.

2.1.8 Cập nhật giá trị dimension

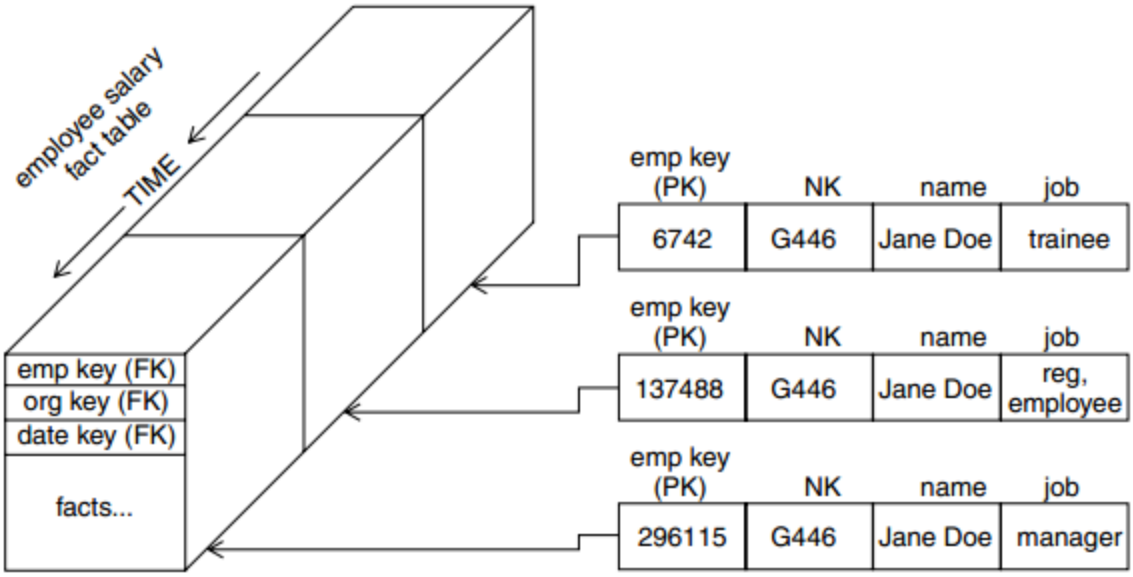
Khi data warehouse nhận thấy có sự thay đổi giá trị trong một bản ghi của dimension, nó phải được lập trình để có hành động tương ứng với sự thay đổi đó. Có 3

phương án hành xử chính được đánh số thứ tự là Kiểu thay đổi 1, Kiểu thay đổi 2 và Kiểu thay đổi 3.

Kiểu thay đổi 1 (**Type 1 Slowly Changing Dimension**) đơn giản ghi đè các dữ liệu bị thay đổi vào bảng dimension. Người thiết kế data warehouse chọn kiểu 1 khi dữ liệu nguồn thay đổi dưới dạng sửa sai hoặc khi phần bị cập nhật này không quan trọng, không làm thay đổi ý nghĩa bảng fact. Kiểu thay đổi 1 này luôn dùng phép UPDATE dữ liệu thay đổi trong data warehouse, chính vì vậy cần lưu ý rằng khi cập nhật dữ liệu bảng dimension, dữ liệu các bảng fact tổng hợp sử dụng các trường dimension thay đổi cũng thay đổi theo.

Kiểu thay đổi 2 (**Type 2 Slowly Changing Dimension**) cho phép theo dõi các thay đổi xảy ra trong bảng dimension và liên kết chính xác bản ghi fact với bản ghi dimension đang có hiệu lực tại thời điểm bản ghi fact. Ý tưởng rất đơn giản: khi data warehouse nhận ra dữ liệu nguồn có cập nhật, thay vì ghi đè, hệ thống cập nhật trạng thái bản ghi cũ và sinh thêm một bản ghi mới vào bảng dimension. Bản ghi mới này được gán cho một khoá thay thế mới toanh (không dính dáng gì đến bản ghi cũ nữa) và từ lúc này hệ thống data warehouse dùng bản ghi mới để liên kết với các bản ghi fact được sinh ra. Các bản ghi fact sinh ra trước đó vẫn liên kết với bản ghi dimension cũ.

Kiểu thay đổi 2 này thể hiện rõ nhất sự thay đổi của dữ liệu theo dòng thời gian vì mỗi sự thay đổi dù nhỏ nhất của thực thể trên dữ liệu nguồn đều được ghi nhận trong data warehouse. Hình 2.4 là một ví dụ cho phương án thay đổi giá trị dimension theo kiểu 2. Nhân viên Jane Doe qua một khoảng thời gian nhận những vị trí công việc khác nhau, mỗi lần thay đổi công việc đều được data warehouse ghi nhận lại với mỗi lần thay đổi công việc là một bản ghi mới trong data warehouse (mỗi bản ghi lại nhận một khoá thay thế mới tương ứng).



Hình 2.4: Ví dụ cho thay đổi dimension kiểu 2

Với bảng dimension quy mô nhỏ khoảng chục trường, vài nghìn bản ghi, việc ghi nhận thay đổi có thể thực hiện bằng cách rà soát từng trường dữ liệu của từng bản ghi. Nhưng nếu số trường, số bản ghi quá lớn thì phương pháp rà soát này mất quá nhiều thời gian, không khả thi nữa. Thay vào đó có thể thay thế bằng phương pháp **kiểm tra checksum** (**cyclic redundancy checksum – CRC**) bản ghi data warehouse và bản ghi nghiệp vụ, nếu checksum không giống nhau tức là có thay đổi. Dữ liệu dùng để checksum có thể dùng phép cộng chuỗi tất cả các trường dữ liệu cần kiểm tra (đổi hết kiểu dữ liệu về dạng chuỗi). Các phép checksum thông dụng nhất hiện nay (CRC32, MD5, SHA1, SHA3, SHA1-Base32, SHA256, SHA384, SHA512…) đều có thể sử dụng để tính checksum cho kết quả chính xác.

Nhưng nếu muốn sinh ra dữ liệu bảng fact, data warehouse lại phải tìm bản ghi có khoá thay thế mới nhất. Điều này không phải lúc nào cũng dễ dàng và thường tiêu tốn tài nguyên cũng như hiệu năng hệ thống. Thay vào đó, bảng dimension có thể thêm một trường is_active để tiện truy vấn. Trong trường hợp này data warehouse không chỉ đơn giản thêm bản ghi mới vào bảng dimension mà trước đó còn phải cập nhật giá trị is_active trong bản ghi cũ về trạng thái không sử dụng. Ngoài trường is_active ra, việc ghi nhận các thông tin bổ sung cho sự kiện cập nhật bản ghi này đôi khi cũng cần thiết. Một số thông tin bổ sung thường được lưu lại bao gồm:

- Ngày bắt đầu có hiệu lực
- Ngày hết hiệu lực (sau khi hết hiệu lực mới được cập nhật thêm vào bảng dimension)

- Lý do cập nhật

Kiểu thay đổi 3 (**Type 3 Slowly Changing Dimension**) được dùng khi giá trị dimension thay đổi nhưng người dùng data warehouse có thể lựa chọn sử dụng giá trị mới hoặc cũ. Nhìn chung kiểu thay đổi 3 này ít được sử dụng (vì có thể thay bằng bảng fact)

Việc lựa chọn áp dụng kiểu thay đổi nào vào thực thể dimension tuỳ thuộc vào yêu cầu nghiệp vụ bảng dimension đó. Nhưng trong nhiều trường hợp thiết kế data warehouse kết hợp 2 kiểu thay đổi trong một dimension, mà thông dụng nhất là kết hợp kiểu 1 và kiểu 2. Khi đó các thuộc tính trong dimension được đánh ưu tiên, nếu thuộc tính thay đổi thuộc độ ưu tiên thấp thì dùng kiểu 1, ghi đè giá trị lên tất cả các bản ghi lịch sử từ trước đến nay. Nếu thuộc tính thay đổi thuộc độ ưu tiên cao hơn thì dùng kiểu 2, vô hiệu hoá bản ghi cũ và sinh bản ghi mới.

2.1.9 Bản ghi dimension về trễ và việc cập nhật dữ liệu dimension sai

Dữ liệu về muộn là vấn đề rất đau đầu trong xây dựng và vận hành hệ thống data warehouse, khắc phục rất mệt mỏi nhưng lại thường xảy ra. Nguyên nhân về muộn thường là do mất đồng bộ giữa data warehouse và hệ thống nghiệp vụ, do lỗi của hệ thống nghiệp vụ không đẩy dữ liệu kịp thời cho data warehouse… Khi phát hiện về muộn nhân viên vận hành data warehouse cần yêu cầu nhân viên vận hành hệ thống nghiệp vụ xử lý vấn đề đảm bảo không xảy ra nữa. Tuy nhiên nguyên nhân mất đồng bộ không được bàn đến ở đây mà sẽ chủ yếu nói về biện pháp khắc phục.

Ví dụ trong viễn thông, theo chính sách mới một loạt thuê bao di động được thay đổi sang gói cước mới nhưng sự thay đổi này chưa kịp về hệ thống data warehouse, phải vài ngày sau mới được ghi nhận. Khi đó rất nhiều bảng fact tổng hợp số liệu vẫn dùng thông tin gói cước cũ, báo cáo sai. Khi đó các bước xử lý nên thực hiện như sau:

- Bước 1: Sao lưu các bản ghi cần thay đổi
- Bước 2: Cập nhật phần dữ liệu bị thay đổi đó trong dữ liệu đã sao lưu ở trên
- Bước 3: Xoá bản ghi cũ trong data warehouse
- Bước 4: Ghi dữ liệu đã cập nhật vào data warehouse

Việc cập nhật bản ghi trong bảng fact data warehouse không được khuyến khích do số lượng bản ghi quá lớn (trong khi data warehouse không tối ưu cho cập nhật).

Ví dụ tiếp theo là Trung tâm kinh doanh đưa vào sử dụng một gói cước mới, nhưng thông tin về gói cước đó lại chưa kịp cập nhật vào data warehouse. Đây là ví dụ tiêu biểu nhất

2.1.10 Tổng kết

Mục này nói về khái niệm và các kỹ thuật xây dựng bảng dimension trong data warehouse. Mặc dù bảng dimension có quy mô số bản ghi nhỏ hơn bảng fact rất nhiều, nhưng nó lại vô cùng quan trọng, cung cấp ý nghĩa cho số liệu trong bảng fact.

Các kỹ thuật trong mục này đều được sử dụng trong thiết kế thực tế. Đặc biệt là ba kiểu thay đổi dữ liệu dimension hiện đã trở thành kiến thức cơ bản mà bất kể người thiết kế data warehouse nào cũng phải biết.

Mục tiếp theo sẽ bàn về bảng fact và các kỹ thuật dùng để xây dựng bảng fact.

2.2 Xây dựng bảng Fact

Bảng fact lưu các tiêu chí, chỉ tiêu về hoạt động sản xuất kinh doanh của doanh nghiệp. Mối quan hệ giữa tiêu chí và bảng fact cũng đơn giản: tiêu chí chính là bản ghi fact. Một **tiêu chí** (**measurement**) được định nghĩa là một lượng quan sát được theo một đơn vị đo lường thống nhất.

Mô hình dữ liệu đa chiều được xây dựng xung quanh các tiêu chí này. Bảng fact chứa tiêu chí, còn bảng dimension chứa ngữ cảnh của các tiêu chí đó. Mối quan hệ tưởng chừng đơn giản này cung cấp cho người dùng cuối góc nhìn rất trực quan, đầy đủ và dễ sử dụng về dữ liệu trong data warehouse.

Mục 2.1 đã nói về kỹ thuật thiết kế bảng dimension, việc của mục này sẽ là thiết kế bảng fact. Mặc dù số liệu (fact) mới là thứ người dùng muốn, là linh hồn của data warehouse, nhưng linh hồn là cái gì nếu không có da thịt bao bọc bên ngoài? Các kiến thức về dimension trong mục trước giúp người dùng dễ dàng nắm bắt nội dung mục này hơn rất nhiều.

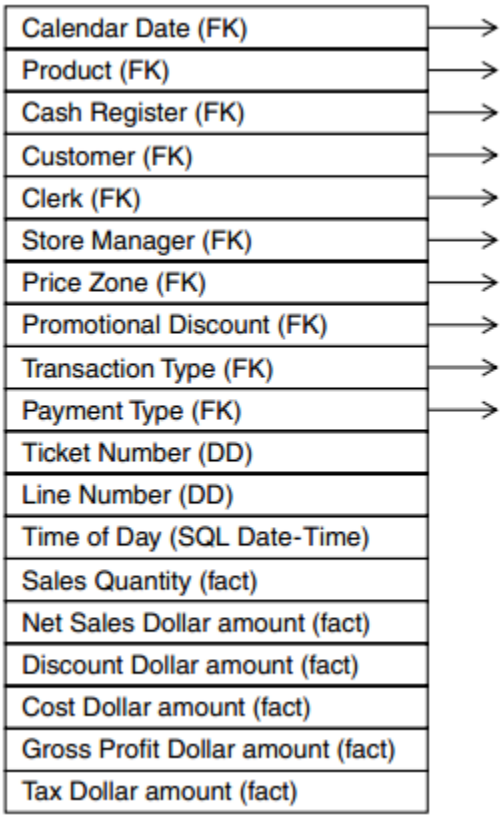
2.2.1 Cấu trúc bảng fact

Mỗi bảng fact được xác định bằng độ chi tiết của bảng dữ liệu. Độ chi tiết này lại được định nghĩa qua sự kiện đo lường trong thực tế. Người thiết kế luôn phải chỉ rõ mức chi tiết của bảng fact, tức là cách thức các tiêu chí trong bảng được đo trong thế giới thực như thế nào. Hình 2.5 là ví dụ một bảng fact ở mức chi tiết nhất, với mỗi bản ghi thể hiện một dòng đơn hàng (dòng hoá đơn).

Bảng fact thông thường không có khoá chính riêng, mà chứa một tập các khoá ngoại để kết nối đến bảng dimension (để cung cấp ngữ cảnh cho cho thông tin trong bảng fact). Hầu hết bảng fact còn có thêm các trường lưu số liệu, chính là các tiêu chí đo lường sản xuất kinh doanh. Trong hình 2.5 còn có một vài dimension đặc biệt gọi là **dimension thoái hoá** (**degenerate dimension**), chính là các dimension có tồn tại trong bảng fact

nhưng không nhất thiết phải tạo ra bảng dimension cho nó. Dimension thoái hoá được viết tắt là DD.

Trong thực tế bảng fact nào cũng có ít nhất 3 dimension, thường là nhiều hơn. Độ chi tiết số liệu càng cao, data warehouse càng cần nhiều bảng dimension. Đáng buồn thay càng ngày nhu cầu sử dụng dữ liệu data warehouse càng khó lường, độ chi tiết dữ liệu càng lớn, số lượng dimension càng lúc càng tăng.



Hình 2.5: Bảng fact giao dịch bán hàng ở mức chi tiết nhất

Không dùng khoá thay thế như dimension, bảng fact dùng các khoá phụ làm khoá chính cho nó. Trong hình 2.5 trên, khoá chính được sử dụng là tập hợp {Ticket Number, Line Number}, hai trường này định nghĩa ra bản ghi duy nhất trong toàn bộ quá trình thanh toán của hệ thống nghiệp vụ, và do đó cũng định nghĩa bản ghi duy nhất trong data warehouse. Cùng là bảng fact về giao dịch bán hàng nhưng ở mức tổng hợp cao hơn, khoá chính sẽ sử dụng những trường dữ liệu khác.

Trong ví dụ trên, khoá chính cũng có thể tập hợp {Cash Register (cửa thanh toán), Time of day, Line Number}. Trong trường hợp này, nếu không chú ý kỹ có thể xảy ra tình huống một sự kiện khác có thông tin y hệt như trên xảy ra, khi đó không có cách nào

biết bản ghi đó là bản ghi trùng, hay là một sự kiện khác cả. Để chặt chẽ hơn, trong quá trình nạp dữ liệu vào data warehouse, các bản ghi fact được gán thêm trường dữ liệu sequence.

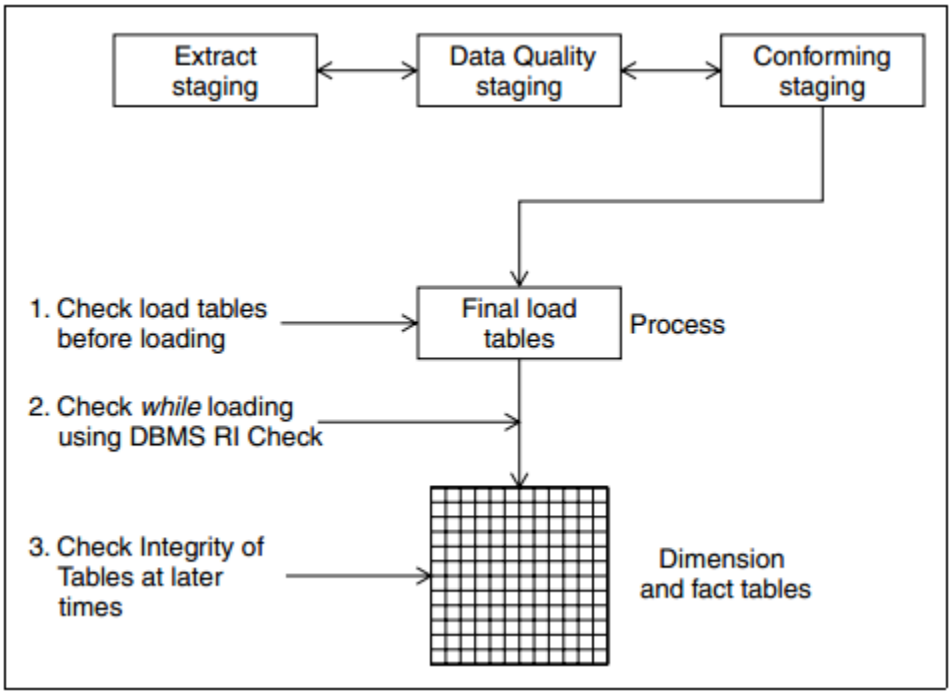
2.2.2 Toàn vẹn thực thể

Toàn vẹn thực thể trong mô hình dữ liệu đa chiều nghĩa là mỗi bản ghi trong bảng fact đều phải được gán giá trị dimension đúng đắn, nói cách khác từng bản ghi fact phải tìm thấy các giá trị dimension tương ứng với nó.

Thông thường có 2 loại vi phạm toàn vẹn chính:

- Ghi dữ liệu vào bảng fact nhưng không tìm thấy giá trị dimension tương ứng
- Xoá bản ghi dimension trong khi giá trị bị xoá đã được sử dụng từ trước đó rồi

Bảng dữ liệu trong data warehouse thường không đặt ràng buộc khoá chặt chẽ như hệ thống nghiệp vụ, vì thế việc vi phạm một trong hai lỗi trên (hoặc cả 2) cực kỳ dễ xảy ra, và mỗi lần xảy ra đều gây ảnh hưởng nghiêm trọng đến dữ liệu. Lỗi vi phạm toàn vẹn thực thể trong data warehouse không phải lỗi nhỏ, nó là lỗi nguy hiểm. Kết quả truy vấn trên dữ liệu fact vi phạm toàn vẹn thực thể sẽ bị thiếu hoặc sai, gây ảnh hưởng đến độ chính xác của quá trình ra quyết định.



Hình 2.6: Thời điểm kiểm tra toàn vẹn thực thể

Hình 2.6 mô tả 3 thời điểm hợp lý kiểm tra toàn vẹn thực thể trong các luồng ETL khi ghi dữ liệu fact vào data warehouse. Chúng là:

- Kiểm tra kỹ dữ liệu fact trước khi ghi vào data warehouse và trước khi xoá dữ liệu khỏi bảng dimension
- Đặt ràng buộc toàn vẹn vào data warehouse đảm bảo không ghi dữ liệu sai hoặc xoá dữ liệu đúng trong data warehouse
- Phát hiện và sửa các lỗi toàn vẹn thực thể sau khi ghi vào data warehouse bằng cách định kỳ rà soát bảng fact, tìm ra giá trị khoá ngoại lỗi.

Trên thực tế thì thời điểm kiểm tra toàn vẹn tốt nhất là giai đoạn trước khi ghi vào data warehouse. Lúc này chỉ cần thêm bước tìm kiếm giá trị dimension nào chưa có trong bảng dimension để ghi tương ứng, sau đó sẽ cập nhật thông tin chi tiết cho dimension sau (công việc này sẽ được mô tả bên dưới). Nếu bước xử lý này được thực hiện cẩn thận, bảng fact sẽ tuân theo ràng buộc ràng buộc toàn vẹn. Tương tự, khi muốn xoá bản ghi trong bảng dimension, trước hết phải kiểm bằng cách join các bản ghi muốn xoá với bảng fact, sau khi chắc chắn không trả về kết quả nào mới thực hiện xoá.

Giai đoạn thứ hai thường không được sử dụng do làm giảm tốc độ ghi dữ liệu vào data warehouse xuống đáng kể. Một số data warehouse cho tốc độ ghi rất tốt ngay cả khi có ràng buộc toàn vẹn (như hệ thống Red Brick của IBM), tuy nhiên đây chỉ là các trường hợp cá biệt, phần lớn data warehouse vẫn xử lý chậm khi có ràng buộc toàn vẹn.

Kiểm tra toàn vẹn sau khi ghi dữ liệu vào data warehouse trên thực tế cũng ít thực hiện do khối lượng cần kiểm tra quá lớn. Ví dụ nếu có kiểm tra thật thì câu lệnh sẽ ở dạng:

```
select f.product_key
from fact_table f
where f.product_key not in (select p.product_key from product_dimension p)
```

Nhìn chung là không thể thực hiện được câu lệnh này trong hệ thống thật. Việc kiểm tra, nếu có, sẽ bị giới hạn trong một khoảng thời gian nhất định (hôm nay, tuần này, tháng này), việc vi phạm toàn vẹn vẫn có thể xảy ra trong phần dữ liệu trước đó.

Dù sao, cách thức trung dung nhất, vừa đảm bảo toàn vẹn vừa đảm bảo hiệu năng chính là kiểm tra dữ liệu bảng fact trước khi ghi dữ liệu vào và không xoá dữ liệu trong bảng dimension. Dữ liệu trong dimension chỉ nên bị xoá khi có sai sót cần sửa, còn nói chung thì cứ để nguyên. Một bản ghi (hoặc nhiều bản ghi) không được sử dụng chưa chắc sau này đã không được sử dụng, quy mô bảng dimension dù sao cũng không quá lớn để ảnh hưởng nghiêm trọng đến hiệu năng hệ thống.

Đôi lúc xảy ra trường hợp dữ liệu dimension mới toanh được phát hiện trong bảng fact (dữ liệu dimension về muộn). Lúc này chúng ta sẽ ghi bản ghi mới vào bảng dimension (và do đó tạo ra khoá thay thế mới) với thuộc tính của dimension đơn giản là *Unknown*.

2.2.3 Phân loại bảng fact

Bảng fact chứa tất cả tiêu chí về hoạt động sản xuất kinh doanh của doanh nghiệp. Nhiều vậy nhưng bảng fact chỉ được phân loại về 3 dạng chính:

Bảng fact mức chi tiết (**transactional grain fact table**) thường mô tả một sự kiện nào đó xảy ra thế giới thực được ghi nhận vào data warehouse. Các tiêu chí và dimension trong bảng fact này vì thế không mô tả một quá trình, mà chỉ ghi nhận giá trị thời điểm sự kiện xảy ra.

Bảng fact mức chi tiết là bảng có quy mô lớn nhất (về lượng bản ghi) và chi tiết nhất trong 3 dạng bảng fact chính. Dữ liệu của nó thường chứa thông tin đầy đủ, chi tiết nhất về sự kiện, bao gồm cả thời gian chính xác sự kiện xảy ra. Do quy mô lớn vậy nên việc sử dụng bảng này ở mức người dùng cuối rất ít khi xảy ra mà thường đây là đầu vào để tổng hợp lên các bảng fact có mức tổng hợp cao hơn.

Bảng fact tổng hợp thường kỳ (**periodic snapshot fact table**) đại diện cho một khung thời gian nhất định nào đó và sẽ lặp lại sau mỗi chu kỳ. Một số dạng thường thấy nhất của bảng fact tổng hợp thường kỳ là tổng hợp theo ngày, theo tháng, theo năm.

Có 2 cách xây dựng dữ liệu cho bảng fact tổng hợp thường kỳ. Một cách là cứ đợi đến cuối tháng rồi tính một thể, nhanh và gọn. Cách khác là duy trì và cập nhật bản ghi luỹ kế tháng, cộng dồn kết quả hàng ngày lại cho đến ngày đầu tháng sau thì chốt tháng cũ và sinh ra bản ghi mới.

Bảng fact luỹ kế (**accumulating snapshot fact table**) là bảng lưu lại số liệu có thời gian không định trước, như là doanh số của một sản phẩm từ khi sản phẩm ra đời đến thời điểm hiện tại (và tiếp tục được cập nhật trong tương lai)

2.2.4 Ghi dữ liệu vào bảng fact

Bảng fact, đặc biệt là bảng fact chi tiết, là nơi tập trung số lượng bản ghi lớn nhất data warehouse. Vì thế việc ghi dữ liệu vào bảng fact thường không phải là việc đơn giản và thường rất tai vạ nếu không được xử lý cẩn thận. Mục này sẽ nói đến các vấn đề thường gặp của người vận hành ETL trong quá trình ghi dữ liệu trên.

Xử lý index

Index rất tốt khi query nhưng lại tai hại vô cùng khi ghi dữ liệu vào database. Các bảng dữ liệu có nhiều index làm chậm việc ghi đến mức có cảm giác cả tiến trình đứng lại không hoạt động. Việc cần làm ở đây là: xoá hết index trước tiến trình ghi dữ liệu vào data warehouse, sau khi ghi xong lại tạo lại index từ đầu.

Xử lý partition

Partition cho phép bảng dữ liệu (và cả index nữa) được chia ra thành các bảng dữ liệu nhỏ hơn về mặt vật lý. Phép chia này cho phép câu truy vấn có thể chạy đến đúng phân khu chứa dữ liệu cần thiết mà không cần tìm kiếm trên toàn bộ bảng dữ liệu. Được xử lý đúng đắn, bảng partition giúp làm tăng hiệu năng truy vấn lên đáng kể trên các bảng fact lớn. Việc partition này thường trong suốt với người dùng, và được vận hành kết hợp bởi cả đội DBA và ETL.

Kỹ thuật đánh partition thông dụng nhất trên bảng fact là đánh partition theo trường thời gian. Ưu điểm của trường thời gian là luôn cố định, lại được định nghĩa sẵn nên chúng ta luôn biết được khoá thay thế sắp được sử dụng là gì để sử dụng. Sai lầm thường thấy khi dùng trường thời gian là người thiết kế thêm một trường thời gian vào bản ghi fact (có thể dùng luôn thời gian ghi bản ghi vào warehouse) và dùng trường đó đánh partition. Nhưng nếu trường thời gian đó không xuất hiện trong câu truy vấn của người dùng cuối thì đánh partition như vậy là vô nghĩa. Bài học rút ra: chỉ đánh partition vào trường thời gian được người dùng quan tâm, sử dụng.

Bảng fact đánh partition theo thời gian thường là đánh theo năm, theo quý, theo tháng, đôi khi là tuần hoặc ngày. Thông thường đội thiết kế data warehouse phải làm việc với đội DBA để xác định phương án đánh partition tốt nhất cho từng bảng fact một. Cũng thông thường đội DBA không trực tiếp tác động vào bảng fact mà việc tăng partition phải thực hiện bằng luồng ETL. Trong phần lớn trường hợp, đội DBA sẽ không can thiệp vào việc vận hành các bảng dữ liệu trong data warehouse, khi đó việc quản lý partition này do đội vận hành ETL đảm nhiệm, và việc của đội vận hành là dựng luồng sửa partition cho từng bảng fact một. Với bảng partition thì thỉnh thoảng đội vận hành sẽ gặp lỗi sau:

```
ORA-14400: inserted partition key is beyond highest legal partition key
```

Khi đó chắc chắn luồng tăng ETL đã bị lỗi và việc cần làm là sửa luồng ETL tăng partition hoặc liên hệ đội DBA ngay lập tức.

Để tránh lỗi trên, trước khi ghi dữ liệu vào data warehouse, luồng ETL có thể chủ động kiểm tra bằng cách so sánh giá trị thời gian lớn nhất trên phần dữ liệu sắp được ghi và giá trị partition cao nhất của bảng fact. Cụ thể so sánh

```
select max(date_key) from FACT_TABLE
```

với

```
select high_value from all_tab_partitions
where table_name = 'FACT_TABLE'
and partition_position = (select max(partition_position)
from all_tab_partitions where table_name= 'FACT_TABLE')
```

Nếu giá trị trên script 1 lớn hơn script 2, ta có thể tăng partition trước khi ghi dữ liệu bằng đoạn script:

```
ALTER TABLE FACT_TABLE
ADD PARTITION year_2013 VALUES LESS THAN (20140101)
```

Cả quá trình theo dõi, cập nhật partition nói trên có thể được viết vào thủ tục stored procedure trên data warehouse và được luồng xử lý ETL gọi trước mỗi lần ghi dữ liệu.

Loại bỏ rollback log

Mặc định tất cả cơ sở dữ liệu quan7 hệ đều hỗ trợ xử lý lỗi khi giao dịch thất bại. Hệ thống trả các bản ghi lỗi về trạng thái trước khi commit, bằng cách ghi nhận lại tất cả thao tác thay đổi dữ liệu. Khi có lỗi, database đọc lại bản ghi log này và sửa chữa tất cả thao tác cập nhật chưa được commit. Việc commit một giao dịch, nghĩa là thông báo cho database biết giao dịch đã thành công và các tác động giao dịch gây ra phải được cập nhật vào cơ sở dữ liệu.

Rollback log, hay còn gọi là redo log, do đó có ý nghĩa rất lớn trong hệ thống nghiệp vụ. Nhưng trong data warehouse khi mà tất cả giao dịch đều được luồng ETL quản lý thì redo log lại là tính năng phiền phức, gây cản trở ghi dữ liệu và cần phải bị loại bỏ để tăng hiệu năng xử lý. Tổng kết lại các nguyên nhân data warehouse không cần redo log:

- Dữ liệu được ghi vào data warehouse bằng một tiến trình được giám sát kỹ càng – luồng ETL
- Dữ liệu được ghi hàng loạt vào data warehouse
- Nếu chẳng may bị lỗi, người vận hành có thể dễ dàng khắc phục và cho chạy lại tiến trình

Ghi chú nhỏ là các hãng cung cấp cơ sở dữ liệu có cơ chế quản lý log khác nhau và đôi khi cung cấp tính năng log mà hãng khác không có. Khi xây dựng luồng ETL cần chú ý các điểm khác biệt này để tối ưu đạt hiệu quả cao nhất.

Ghi dữ liệu

Ghi một lượng lớn dữ liệu vào data warehouse không chỉ đơn giản là câu insert, các kỹ thuật chính khi ghi dữ liệu bao gồm:

- **Tách ghi dữ liệu cũ với cập nhật dữ liệu mới**: một số công cụ ETL (và cả database) cung cấp tính năng *cập nhật hoặc ghi mới*. Tính năng này mới nghe qua thì có vẻ rất tiện lợi và làm cho luồng ETL trông đơn giản hơn rất nhiều nhưng xử lý thực tế lại quá chậm chạp (do mỗi khi có bản ghi mới đều phải kiểm tra bảng dữ liệu xem đã tồn tại chưa). Không nên dùng tính năng này, thay vào đó nên tạo ra 2 luồng khác nhau: cập nhật trước, sau đó ghi mới dữ liệu.
- **Dùng tool ghi dữ liệu của database**: mỗi database thường cung cấp một tool ghi dữ liệu vào database, áp dụng nhiều kỹ thuật độc quyền của riêng database nên tốc độ ghi nhanh hơn dùng câu insert rất nhiều.
- **Chia luồng ghi dữ liệu chạy song song**: khi phải ghi số lượng lớn bản ghi vào data warehouse nên chia nhỏ khối dữ liệu ra làm nhiều phần chia cho nhiều luồng ETL chạy song song.
- **Hạn chế sửa đổi**: chạy lệnh UPDATE trong data warehouse thường rất chậm chạp và khó theo dõi, do đó khi thật sự muốn sửa đổi dữ liệu thường phải sử dụng các kỹ thuật khác. Trong đó một kỹ thuật thông dụng và đơn giản là xoá các bản ghi cũ cần cập nhật đi rồi ghi lại chính các bản ghi đó với thông tin đã được sửa chữa.
- **Tạo bảng tổng hợp bên ngoài database**: các phép sắp xếp, phép giao, phép tổng hợp dữ liệu nên được thực hiện bên ngoài data warehouse để đạt hiệu quả cao hơn. Nguyên nhân là do tài nguyên dành cho data warehouse thường là có hạn và rất khó nâng cấp về phần cứng (cả theo chiều dọc – thêm RAM, thêm CPU hoặc chiều ngang – thêm server), trong khi bộ ETL có thể dễ dàng triển khai trên nhiều server khác nhau để share tải.

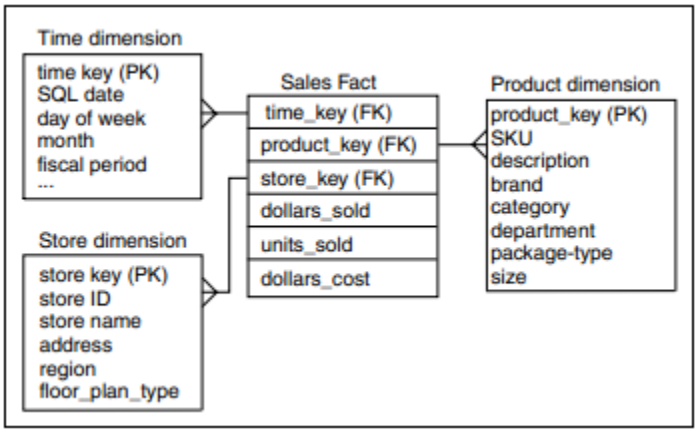
2.2.5 Bảng fact không số liệu

Bảng fact chứa các *sự kiện* có số liệu đo đếm được. Nhưng trong một số trường hợp các sự kiện này lại không có số liệu đo đếm được. Ví dụ bảng fact về việc khách hàng check-in khách sạn, hoặc viễn thông là bảng fact về việc khách hàng đổi gói cước từ Economy sang Tomato. Các bảng fact này chỉ ghi thời gian sự kiện xảy ra và các thuộc tính, không có số liệu nào cả.

Các bảng fact này gọi là **bảng fact không có số liệu** (**factless fact table**). Phép toán dùng chủ yếu trên bảng fact này là phép đếm (count) và đếm không lặp (count distinct).

2.2.6 Bảng tổng hợp dữ liệu

Bảng fact có số bản ghi rất lớn, làm chậm việc truy vấn cho người dùng cuối. Vẫn là vấn đề muôn thuở để các chuyên gia DBA, các chuyên gia ETL và người dùng hệ thống BI căng thẳng nhất. Biện pháp tăng tốc độ truy vấn ấn tượng nhất, thường được các chuyên gia data warehouse khuyên dùng là sử dụng **bảng tổng hợp** (**aggregate table**). Bảng tổng hợp là bảng fact nhưng dữ liệu trong đó đã được tổng hợp ở mức cao hơn trong cây phân cấp dimension so với bảng fact gốc. Khi truy vấn, tuỳ mức độ sử dụng cây phân cấp mà người dùng sử dụng bảng fact gốc hay bảng tổng hợp tương ứng (thực chất ra bảng tổng hợp cũng phải trong suốt với người dùng, và việc lựa chọn dùng bảng nào phải là việc của công cụ khai thác). Chưa cần tốn kém chi phí nâng cấp phần cứng, phần mềm, chỉ cần dùng bảng tổng hợp hiệu năng truy vấn đã tăng lên rõ rệt.

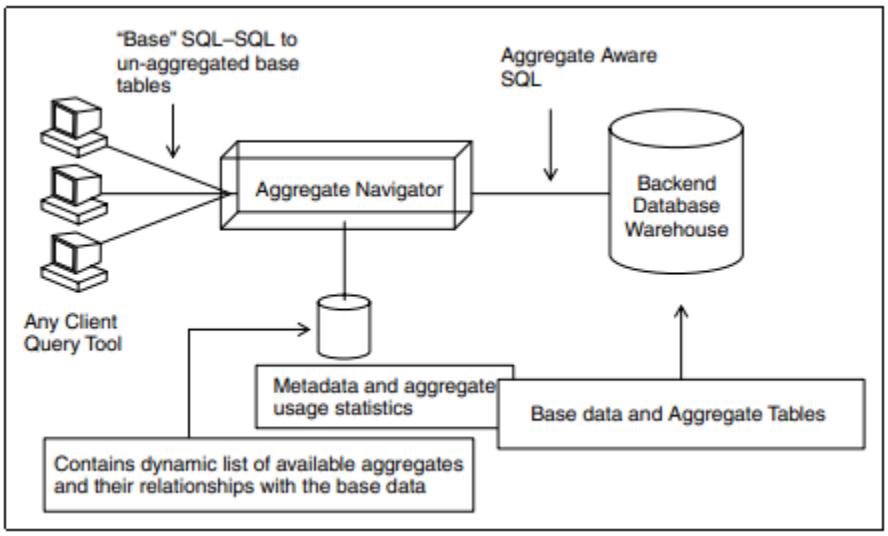


Hình 2.7: ví dụ mô hình dữ liệu đa chiều với dimension ở mức ngày, sản phẩm và kho

Bảng tổng hợp không hề mới lạ, chủ đề này đã được thảo luận rất nhiều trên các diễn đàn data warehouse và được nhiều sách nhắc đến. Một vài điểm chính:

- Trong một thiết kế data warehouse, ứng với mỗi bảng fact lại có một tập các bảng tổng hợp đại diện cho các phép nhóm thông dụng nhất. Phương pháp chuyển qua lại bảng tổng hợp là phương pháp của cơ sở dữ liệu đa chiều, chỉ hỗ trợ cơ sở dữ liệu đa chiều và không có phương pháp tương đương nào trên cho cơ sở dữ liệu quan hệ.
- Module lựa chọn bảng tổng hợp nằm giữa module xử lý truy vấn từ người dùng và data warehouse.

- Module lựa chọn bảng tổng hợp xử lý truy vấn từ người dùng cuối và nếu có thể, chuyển câu lệnh SQL sang lấy dữ liệu từ bảng tổng hợp thay vì bảng fact nguồn
- Module lựa chọn bảng tổng hợp biết khi nào phải dùng bảng tổng hợp, và nếu dùng thì dùng bảng tổng hợp nào vì các thông tin đó đã phải được cấu hình sẵn trong siêu dữ liệu mô tả data warehouse.



Hình 2.8: Cấu trúc module lựa chọn bảng tổng hợp

Tính năng bảng tổng hợp trong data warehouse được coi là tốt phải đáp ứng các tiêu chuẩn sau:

- Cải thiện đáng kể hiệu năng cho phần lớn truy vấn của người dùng (giải pháp tối ưu dành cho tất cả mọi người là không hề có, và nếu có cũng không khả thi trong thực tế)
- Không làm tăng dung lượng dữ liệu lưu trữ lên quá nhiều. Cũng khó nói "quá nhiều" ở đây là bao nhiêu vì tuỳ tình hình cụ thể, nhưng nói chung tổng dung lượng bảng tổng hợp không nên lớn hơn dung lượng bảng chính.
- Hoàn toàn trong suốt với người dùng cuối. Nhân viên kinh doanh hoặc các sếp quản lý cấp cao không cần và cũng không muốn biết quá chi tiết các vấn đề kỹ thuật không liên quan đâu.
- Không làm ảnh hưởng đến việc vận hành hệ thống ETL. Mặc dù mỗi lần tạo bảng tổng hợp đều phải chạy một loạt phép toán sắp xếp, gom nhóm… nhưng việc tạo bảng tổng hợp này nên được dựng tự động bằng tiến trình ETL.
- Ít ảnh hưởng đến việc quản trị viên DBA. Việc tạo bảng tổng hợp nào nên được tự động hoá bằng cách theo dõi thói quen truy vấn của người dùng cuối.

Nếu thiết kế tốt, bảng tổng hợp sẽ thoả mãn cả 5 yêu cầu trên, thiết kế tồi thì chẳng đạt được yêu cầu nào cả. Dưới đây là một số hướng dẫn thiết kế để đạt được các yêu cầu nói trên:

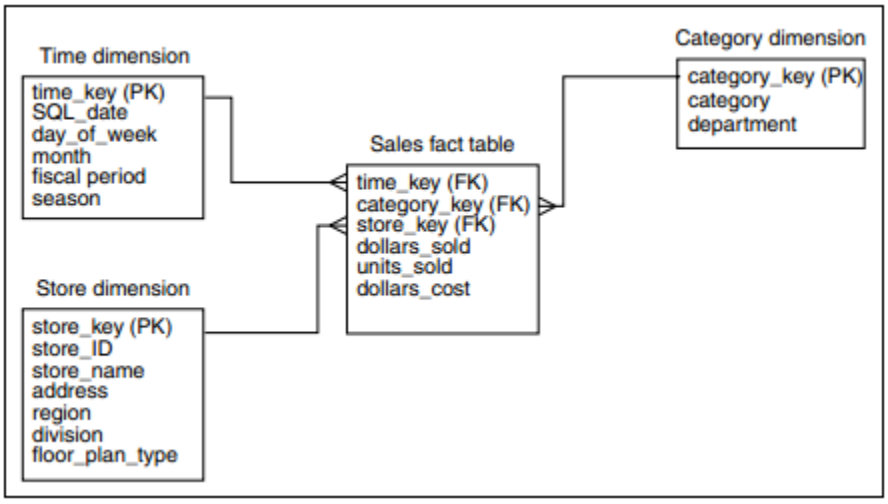
Hướng dẫn thiết kế số 1

Bảng tổng hợp phải có không gian lưu trữ của riêng nó, tách biệt với bảng fact nguồn. Mỗi cấp độ tổng hợp phải có một bảng fact riêng.

Việc chia bảng tổng hợp thành các bảng fact là rất quan trọng và có nhiều hiệu ứng tích cực. Đầu tiên là cấu trúc bảng fact và bảng tổng hợp đơn giản hơn, câu lệnh SQL truy vấn vì thế chỉ cần thay tên bảng, không cần thêm điều kiện phức tạp. Thứ hai, cũng do không phải thêm điều kiện nên người dùng cuối không gặp nguy cơ lấy kết quả sai vì cộng nhầm số liệu các mức tổng hợp khác. Thứ ba, lưu ra bảng khác làm tăng tốc độ truy vấn dữ liệu ở mức database vì làm giảm số bản ghi cần tìm kiếm. Cuối cùng, lưu ra bảng fact riêng khiến việc quản lý bảng tổng hợp dễ dàng hơn, nếu một bảng tổng hợp bị lỗi chỉ cần sửa trong phạm vi bảng đó mà không gây nguy cơ ảnh hưởng các phần dữ liệu khác.

Hướng dẫn thiết kế số 2

Bảng dimension tương ứng với bảng tổng hợp cũng phải ở phân cấp tương ứng với dữ liệu trong bảng tổng hợp. Nói cách khác nếu bảng tổng hợp là phiên bản rút gọn của bảng fact, bỏ đi một vài mức trong cây phân cấp, thì bảng dimension cho bảng tổng hợp cũng phải bỏ các mức phân cấp tương ứng.



Hình 2.9: Mô hình bảng tổng hợp sau khi để lại mức Nhóm sản phẩm

Hình 2.9 là mô hình bảng tổng hợp của ví dụ trong hình 2.7. Mô hình này giải quyết trường hợp người dùng cuối muốn truy vấn dữ liệu bán hàng theo ngày, theo cửa hàng và theo nhóm sản phẩm (chứ không quan tâm đến từng sản phẩm một như hình 2.7 nữa).

Không phải dimension nào cũng xuất hiện ở các mức tổng hợp cao hơn. Một số dimension và số liệu trên bảng fact chỉ có ý nghĩa ở mức bảng fact nguồn và làm sai hẳn ý nghĩa báo cáo khi kết hợp với mức cao hơn của dimension khác.

Các bảng dimension rút gọn này cũng phải tuân theo các quy tắc về bảng dimension như khoá thay thế, khoá tự nhiên, các thuộc tính… Như vậy trước khi xây dựng dimension cho bảng fact gốc, cần xây dựng bảng dimension cho các bảng tổng hợp (và cũng cần thiết thiết phải lưu khoá thay thế của bảng dimension rút gọn trong bảng dimension ở mức chi tiết hơn).

Hướng dẫn thiết kế số 3

Bảng fact và tất cả các bảng tổng hợp của nó phải được liên kết với nhau thành một hệ bảng fact để module lựa chọn biết được phải sử dụng bảng nào để chạy câu truy vấn. Mỗi thành phần trong hệ bảng fact sẽ bao gồm một bảng fact (có thể là bảng gốc hoặc bảng tổng hợp), bao xung quanh bằng các dimension tương ứng với nó. Trong hệ này có một mẫu duy nhất chứa bảng fact gốc, các mẫu khác là mẫu tổng hợp. Hình 2.7 là mẫu gốc, hình 2.9 là một trong số rất nhiều mẫu tổng hợp của mẫu 2.7.

Mối liên hệ giữa bảng fact gốc và các bảng tổng hợp (cũng như các bảng dimension tương ứng) phải được cấu hình để hệ thống truy vấn biết được mà lựa chọn. Thông thường thông tin này được cấu hình trong bộ công cụ OLAP của hệ thống data warehouse.

Hướng dẫn thiết kế số 4

Nói chung thì người dùng cuối không nên biết về sự tồn tại của bảng tổng hợp. Các truy vấn từ người dùng cuối trực tiếp đến data warehouse không thông qua các công cụ khai thác chỉ có điểm dừng duy nhất là bảng fact gốc, không nên đi xa hơn đến bảng tổng hợp.

2.2.7 Tổng kết

Trong mục này chúng ta đã xác định bảng fact là nơi lưu trữ toàn bộ số liệu sản xuất kinh doanh của doanh nghiệp. Các bảng fact này đều có 2 phần chính: số liệu và các khoá ngoại liên kết đến bảng dimension làm ngữ cảnh cho số liệu.

Chúng ta chỉ ra được toàn vẹn thực thể là vô cùng quan trọng trong mô hình dữ liệu đa chiều, và đề xuất ra 3 phương án kiểm tra ràng buộc toàn vẹn.

Rồi chuyển sang các kỹ thuật thường dùng để tăng hiệu năng xử lý bảng fact, bao gồm cả các kỹ thuật tối ưu tiến trình ghi dữ liệu và kỹ thuật tạo bảng tổng hợp để tăng tốc truy vấn vào bảng fact này.

Phần 3: Xây dựng luồng ETL

3.1 Về ETL

3.2 Thu thập dữ liệu

3.3 Làm sạch và chuẩn hoá dữ liệu

Phần 4: Áp dụng thực tế cho viễn thông, xây dựng hệ thống data warehouse cho mobile trả trước

4.1 Thu thập yêu cầu

4.2 Thiết kế data warehouse

4.3 Xây dựng luồng ETL

4.4 Xây dựng OLAP Cube