**LỰA CHỌN KHUNG NHÌN**

1. **CỤ THỂ HÓA KHUNG NHÌN ĐƠN GIẢN**

**Kiểu khung nhìn cụ thể hóa đơn giản nhất chứa dữ liệu cùng loại với kết quả của một câu truy vấn xác định. Đọc từ khung nhìn cụ thể hóa cho kết quả giống với kết quả được tính toán từ các bảng cơ sở. Chúng tôi sẽ minh họa với một ví dụ cụ thể và giải thích sự tiết kiệm truy vấn I/O.**

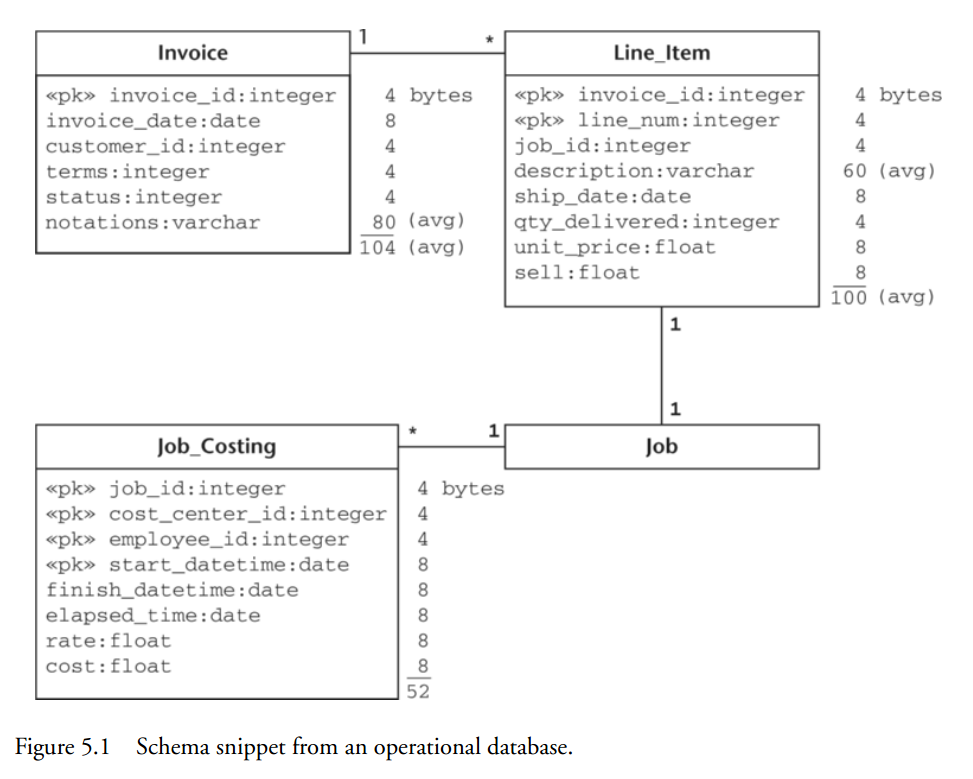
**Hình 5.6 là một phần của một lược đồ từ một CSDL được chuẩn hóa. Chúng tôi chỉ tham khảo các bảng nằm trong phạm vi của chương này. Các bảng thích hợp trong CSDL này chứa các dữ liệu có ích cho chu kỳ tính toán, chi phí công việc tại một nhà máy sản xuất.**

**Mỗi hóa đơn (invoice) có một ngày hóa đơn (invoice date) và một khách hàng liên quan (customer). Các điều khoản (terms) của hóa đơn cho biết trong bao nhiêu ngày khách hàng phải trả hóa đơn. Tình trạng (status) cho biết giai đoạn hiện tại của chu kì thanh toán cho hóa đơn. Các ghi chú (notations) có thể chứa bất kì ý kiến chung nào, chẳng hạn như ghi chú cuộc thoại với khách hàng về hóa đơn.**

**Một hóa đơn có thể có nhiều mặt hàng, mỗi loại một dòng riêng. Ví dụ, bạn có thể đã mua các đồ dùng màu xanh, đã được vận chuyển ngày 23 tháng 10 năm 2006 (ship\_date), số lượng (qty\_delivered) là 100 với đơn giá (unit\_price) là 2.50$ mỗi cái, tổng giá trị (sell) là 250$. Bạn cũng có thể mua các đồ dùng màu đỏ, nằm trên một dòng khác trên cùng một hóa đơn. Dữ liệu này được lưu trữ trong bảng Line\_Item.**

**Mỗi chi tiết hóa đơn có liên quan với một công việc được sản xuất tại nhà máy. Bảng Job có chứa các chi tiết kỹ thuật của công việc. Tuy nhiên các chi tiết này không được thể hiện ở đây vì các chi tiết này không nằm trong nội dung thảo luận của chúng tôi.**

**Bảng Job\_Costing chứa dữ liệu theo dõi thời gian và chi phí sản xuất cho từng công việc. Khi công việc đi qua một máy, các công nhân tại máy đó quét mã ID của công việc trên thẻ CV. Công nhân cũng có thể quét mã chi phí trung tâm đại diện cho hoạt động đã được thực hiện, và mã này chứa mã của nhân viên. Thời gian bắt đầu và thời gian kết thúc cho hoạt động được tự động ghi lại bởi hệ thống máy tính. Dữ liệu được lưu trong bảng Job\_Costing, cùng với mức lương giờ cho chi phí trung tâm và chi phí tính từ thời gian trôi qua và tỉ lệ tính theo giờ. Những thông tin này có ích trong việc so sánh chi phí thực tế và giá trị bán ra của mỗi công việc, dẫn tới quá trình cải tiến và sinh nhiều lợi nhuận hơn.**

****

**Với các tính toán tiếp theo, chúng tôi giả sử các số liệu thống kê trong CSDL như sau:**

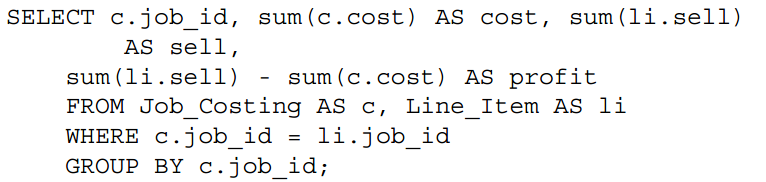
* **Average jobs per day = 10,000.**
* **Average invoices per day = 5,000.**
* **Average Job\_Costing rows per job = 50.**
* **Day on record = 200.**
* **Customers = 5,000.**
* **The rows in all tables are clustered in primary key order.**
* **Anything 200MB (i.e., 50,000 blocks) or less can be sorted in memory.**

**Chúng tôi tiếp tục sử dụng các công thức I/O. Các tính toán của chúng tôi trong chương này đều dựa trên ổ cứng IBM U320 146GB.**

* **I/O time (4KB block in a dedicated disk) = 2.0 ms.**
* **I/O time (64KB buffer in a dedicated disk) = 2.2 ms.**
* **I/O time (4KB block in a shared disk) = 5.6 ms.**
* **I/O time (64KB block in a shared disk) = 5.8 ms.**

**Quét bảng cần giả định sử dụng các bộ đệm nạp trước. Mặc dù thời gian I/O truyền 64KB dài hơn truyền I/O 4KB, truyền 64KB chỉ thực hiện 1 lần so với 16 lần, vì thế hiệu quả tổng thể tốt hơn 16 lần.**

**Bộ phận quản lí của công ty muốn theo dõi lợi nhuận của công việc. Dịch vụ thông tin đã cung cấp một ứng dụng phân tích lợi nhuận tuyệt vời. Thông thường ứng dụng được chạy bởi 10 nhà quản lý mỗi ngày. Ứng dụng đòi hỏi thông tin về chi phí, giá bán và lợi nhuận cho từng công việc. Các thông tin này có thể lấy từ CSDL hiện có bằng câu truy vấn sau đây:**

****

**Hãy tính thời gian cho câu truy vấn này, sau đó chúng tôi sẽ cải thiện hiệu suất bằng cách sử dụng khung nhìn cụ thể hóa. Câu truy vấn liên quan đến 2 bảng Line\_Item và bảng Job\_Costing. Chúng tôi cần tính số lượng hàng, hệ số block và số block cho mỗi bảng. Có 10,00 công việc mỗi ngày, với 200 ngày trong bản ghi, cho tổng cộng 2,000,000 dòng trong bảng Line\_Item. Vì chúng tôi sẽ thực hiện quét bảng, chúng tôi sẽ giả định bộ đệm 64KB cho các hoạt động đọc và ghi I/O. Các phép tính cho bảng Line\_Item như sau:**

**Average rows per prefetch buffer**

**= floor ((65,536 bytes/block) / (100 bytes/row)) = 655.**

**Number of buffers**

**= ceiling (2,000,000 rows / (655 rows/buffer)) = 3,054.**

**Bảng Job\_Costing chứa trung bình 50 dòng cho mỗi công việc, 10,000 công việc mỗi ngày, với 200 ngày trong bản ghi, cho tổng cộng 100,000,000 dòng. Các phép tính của bảng Job\_Costing như sau:**

**Average rows per prefetch buffer**

**= floor ((65,536 bytes/block) / (52 bytes/row)) = 1,260.**

**Number of buffers = ceiling (100,000,000 rows / (1,260 rows/buffer)) = 79,366.**

**Bởi vì chúng tôi kết hai bảng hoàn toàn nên kiểu merge-join là hiệu quả. Dữ liệu trong bảng Job\_Costing được sắp xếp bởi job\_id. Bảng Line\_Item cần phải được sắp xếp bởi job\_id để thực hiện được phép kết merge-join. Bảng Line\_Item đủ nhỏ để sắp xếp trong bộ nhớ, vì thế chúng tôi chỉ cần một lần quét tuần tự để tải bảng vào bộ nhớ.**

**Join cost (shared disk) = scan time (Line\_Item and Job\_Costing tables using 64KB prefetch buffers)**

**= (3,054 + 79,366 buffers) x 5.8 ms**

**= 82,420 x 5.8 ms**

**= 478 second 8 minutes.**

**Đây là một khoảng thời gian khá dài cho một câu truy vấn, đặc biệt khi bạn phải làm chúng nhiều lần và 10 nhà quản lý quyết định chạy chương trình phân tích lợi nhuận tuyệt vời của bạn để lấy thông tin cho cuộc họp lớn của họ sẽ bắt đầu trong 5 phút nữa. Cuối cùng cũng đã có người gõ cánh cửa của bạn. Chúng tôi có thể làm gì? Chúng tôi có thể tính toán các kết quả cần thiết trong giờ nghỉ, và lưu trữ dữ liệu đã được xử lý trong khung nhìn cụ thể hóa. Sau đó khi các nhà quản lý đến, họ có thể có kết quả ngay lập tức. Hình 5.2 cho thấy lược đồ của khung nhìn cụ thể hóa.**

**Đầu tiên, hãy tính thời gian tạo khung nhìn, sau đó chúng tôi sẽ tính thời gian để lấy kết quả từ bảng Profit\_by\_Job. Khung nhìn được tạo bằng cách chạy các câu truy vấn ban đầu, sau đó viết kết quả vào bảng Profit\_by\_Job. Số block (buffer) truy cập tham gia là như nhau, nhưng thời gian I/O bây giờ ngắn hơn bởi vì môi trường đã được dành sẵn trong giờ nghỉ.**

**Join cost (dedicated disk) = 82,420 x 2.2 ms 181 seconds 3 minutes.**

**Chúng tôi cần tính hệ số blocking để tính thời gian ghi kết quả vào bảng Profit\_by\_Job. Chúng tôi có 10,000 công việc mỗi ngày, với 200 ngày trong bản ghi, vì thế chúng tôi có 2,000,000 dòng trong bảng Profit\_by\_Job. Các phép tính của bảng Profit\_by\_Job:**

**Average rows per prefetch buffer**

**= floor ((65,536 bytes/block) / (28 bytes/row)) = 2,340.**

**Number of buffers = ceiling (200,000,000 rows / (2,340 rows/buffer)) = 855.**

**Write cost (dedicated disk) = 855 x 2.2 ms 2 seconds.**

**Profit\_by\_Job creation cost = join cost + write cost = 181 + 2 = 183 seconds.**

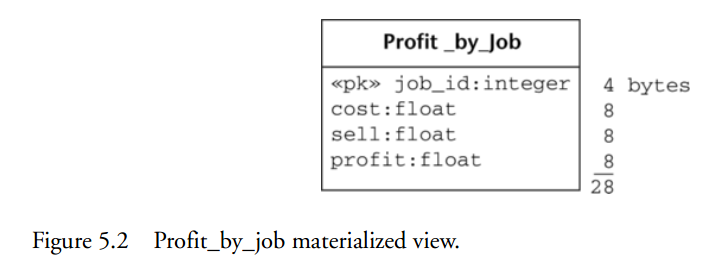
**Câu truy vấn lợi nhuận theo công việc sẽ đơn giản hơn khi dựa trên khung nhìn mới này:**

****

**Chúng tôi đã tính được có 855 buffer cho bảng Profit\_by\_Job. Câu truy vấn được chạy trên đĩa chia sẻ (5 phút trước khi họp).**

**Query I/O time (table scan, shared disk) = 855 × 5.8 ms ≈ 5 seconds.**

**Những người quản lý có bản báo cáo của họ trong tay, hơn thế nữa còn có nhiều thời gian để nói chuyện trước cuộc họp.**

****

**Cùng tóm tắt lại:**

**Disk I/O time before Profit\_by\_Job**

**= query frequency × I/O time per query**

**= (10 queries/day) × (478 sec/query)**

**= 4,780 seconds (or about 1 hour, 20 min)**

**Disk I/O time with Profit\_by\_Job**

**= creation cost + query frequency × I/O time per query**

**= 183 sec creation cost + (10 queries/day) × (5 sec/query)**

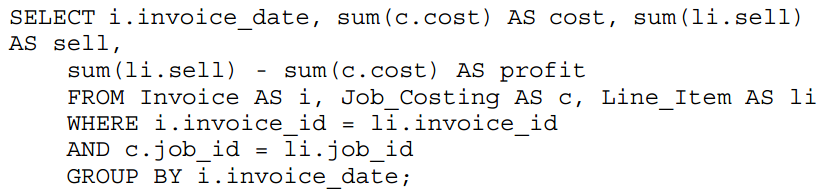
**= 233 seconds.**

* **Sử dụng khung nhìn cụ thể hóa có thể mang lại cải thiện đáng kể cho cả đĩa I/O và tốc độ truy vấn. Tuy một số lợi ích có được từ việc tận dụng ưu điểm của đĩa chuyên dùng để tạo ra khung nhìn, nhưng phần lớn đến từ 2 yếu tố đầu tiên. Thứ nhất, khung nhìn cụ thể hóa có thể nhỏ hơn so với các bảng cơ sở, dẫn tới đạt hiệu quả lớn trong hiệu suất truy vấn đĩa I/O. Thứ 2 là các câu truy vấn thường xuyên làng tăng hiệu quả này.**

1. **KHAI THÁC PHỔ BIẾN**

**Có một số câu truy vấn thường xuyên xem lợi nhuận từ các quan điểm khác nhau. Một sự phân tích hợp lý để theo dõi lợi nhuận bởi khách hàng. Ví dụ, nếu cúng tôi biết ai là khách hàng lợi nhuận cao nhất, chúng tôi có thể thu hút thêm nhiều công việc bằng cách giảm giá số lượng lớn nếu họ đồng ý mang lại số công việc ít nhất trong một khung thời gian nhất định. Chúng tôi kiểm tra khả năng của hai khung nhìn để trả lời nhanh chóng các câu truy vấn này. Sau đó chúng tôi sẽ xem xét một khung nhìn thay thế khác tận dụng lợi thế của sự tương đồng của ba câu truy vấn lợi nhuận của chúng tôi.**

**Xu hướng lợi nhuận theo ngày được xem tóm tắt mỗi ngày một lần bởi người quản lý cấp trên và cũng có thể quản lý tính toán. Chúng tôi xác định rằng câu truy vấn thường được chạy mỗi ngày 5 lần. Lợi nhuận tính bằng cách gom nhóm ngày của hóa đơn được trình bày dưới đây:**

****

**Câu truy vấn này đọc từ các bảng Invoice, Job\_Costing và Line\_Item. Chúng tôi đã tính số block cho bảng Job\_Costing và bảng Line\_Item. Chúng tôi vẫn phải tính số block cho bảng Invoice. Trung bình có 5,000 hóa đơn mỗi ngày, với 200 ngày trong bản ghi, cho 1,000,000 dòng trong bảng Invoice.**

**Phép tính của bảng Invoice như sau:**

**Average rows per prefetch buffer**

**= floor ((65,536 bytes/block) / (104 bytes/row)) = 630.**

**Number of buffers = ceiling (1,000,000 rows / (630 rows/buffer)) = 1,588.**

**Có hai phép kết có thể thực hiện được, dẫn tới hai kế hoạch truy vấn thay thế sau đây.**

**PLAN A:**

* **B1: thực hiện merge-join giữa Line\_Item và Invoice (không yêu cần sắp xếp)**
* **B2: merger-join kết quả bước 1 với Job\_Costing (yêu cần sắp xếp KQ bước 1 the job\_id)**

**PLAN B**

* **B1: merger-join Line\_Item và Job\_Costing (yêu cầu sắp xếp theo job\_id)**
* **B2: merge-join kết quả bước 1 với Invoice (yêu cần sắp xếp KQ bước 1 the invoice\_id)**

**Cả hai kế hoạch trên yêu cầu quét tất cả ba bảng cơ sở. Kế hoạch A là tốt hơn bởi vì nó yêu cầu ít phép sắp xếp hơn kế hoạch B. Sauk hi merge-join bảng Line\_Item với Inovoice, chỉ cần giữ cột invoice\_date, job\_id và sell. Kết quả là 2,000,000 dòng, mỗi dòng 20 byte cho khoảng 10,000 block. Kết quả đủ nhỏ để sắp xếp trong bộ nhớ, vì thế chúng tôi chỉ cần đến thời gian quét các bảng một lần.**

**Query I/O time (shared disk)**

**= scan Line\_Item + scan Invoice + scan Job\_Costing**

**= (3,054 blocks + 1,588 blocks + 79,366 buffers scanned) × 5.8 ms**

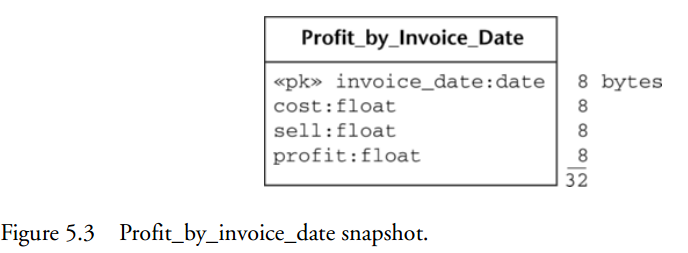
**= 84,008 × 5.8 ms**

**= 487 seconds.**

**Kết quả tương tụ cũng có được từ khung nhìn cụ thể hóa trong hình 5.3.**

**Chúng tôi có thể phát sinh khung nhìn cụ thể hóa từ đầu trong giờ nghỉ bằng cách chạy câu truy vấn tương tự, và sau đó lưu trữ kết quả.**

**Query I/O time (dedicated disk) = 84,008 × 2.2 ms ≈ 185 seconds.**

****

**Chúng tôi có 200 ngày trong bản ghi, vì thế bảng Profit\_by\_Invioce\_Date có 200 dòng. Các phép tính của bảng Profit\_by\_Invioce\_Date:**

**Average rows per block (buffer)**

**= floor((65,536 bytes/buffer)/(32 bytes/row)) = 2,048.**

**Number of blocks (buffers) = ceiling(200 rows/(2,048 rows/block))**

**= 1.**

**Write cost (dedicated disk) = 1 × 2.2 ms = 2.2 ms.**

**Profit\_by\_Invoice\_Date creation cost = query cost + write cost**

**= 185 sec + .0022 sec**

**≈ 185 seconds.**

**Câu truy vấn được biến đổi như sau:**

****

**Thời gian tính toán I/O cho câu truy vấn trên đĩa chia sẻ là:**

**Query I/O time (shared disk) = 1 × 5.8 ms = 5.8 ms.**

**Disk I/O time before Profit\_by\_Invoice\_Date**

**= query frequency × I/O time per query**

**= (5 queries/day) × (487 sec/query)**

**= 2,435 seconds (just over 40 min)**

**Disk I/O time with Profit\_by\_Invoice\_Date**

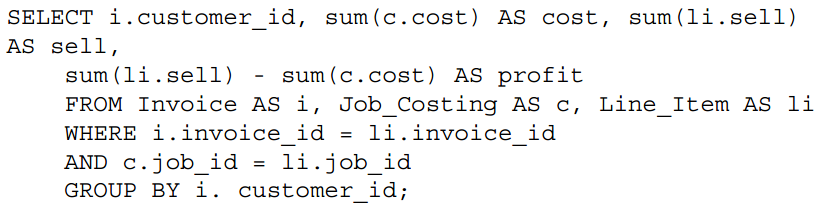
**= creation cost + query frequency × I/O time per query**

**= 185 sec creation cost + (5 queries/day)**

**× (5.8 ms/query) ≈ 185 seconds.**

**Có một cách tiếp cận khác để duy trì khung nhìn cụ thể hóa có thể cải thiện lợi ích trong nhiều trường hợp. Thay vì tính khung nhìn từ đầu mỗi đêm, chúng có thể được duy trì thêm. Ví dụ, trong trường hợp của bảng Profit\_by\_Invoice\_Date, chúng tôi có thể giữ dữ liệu hiện có, và thêm dòng mới bất cứ khi nào một chu trình hóa đơn mới hoàn thành.**

**Câu truy vấn tính lợi nhuận theo khách hàng tương tự như lợi nhuận tính theo ngày hóa đơn. Câu truy vấn này chủ yếu được chạy bởi bộ phận bán hàng. Chúng tôi xác định rằng câu truy vấn này chạy 3 lần mỗi ngày. Đây là câu truy vấn dựa trên bảng cơ sở:**

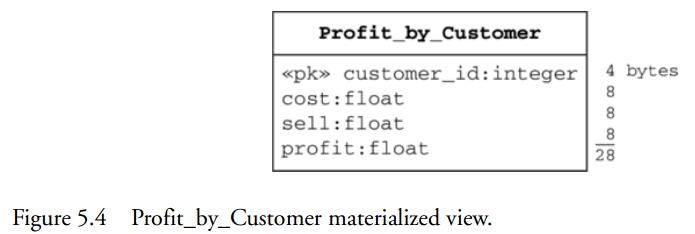
****

**Chú ý rằng các bảng và các phép kết tương tự như khi thực hiện truy vấn theo ngày hóa đơn. Bởi vì cùng một bảng được đọc, và cùng một phép kết được thực hiện, vì thế thời gian truy vấn I/O cũng không thay đổi.**

**Query I/O time (shared disk)**

**= scan Line\_Item + scan Invoice + scan Job\_Costing**

**= 84,008 × 5.8 ms = 487 seconds.**

****

**Hình 5.4 là hình của khung nhìn tương ứng. Chi phí I/O để tạo bảng Profit\_by\_Customer tương đương với chi phí truy vấn trên bảng cơ sở cộng với chi phí viết kết quả vào đĩa.**

**Query I/O time (dedicated disk) = 84,008 × 2.2 ms ≈ 185 seconds.**

**Chúng tôi có 5,000 khách hàng, vì thế bảng Profit\_by\_Customer có 5,000 dòng. Các phép tính của bảng Profit\_by\_Customer:**

**Average rows per block (buffer)**

**= floor((65,536 bytes/block)/(28 bytes/row)) = 2,340.**

**Number of blocks (buffers)**

**= ceiling(5,000 rows/(2,340 rows/block)) = 3.**

**Write cost (dedicated disk) = 3 × 2.2 ms = 6.6 ms.**

**Profit\_by\_Customer creation cost = query cost + write cost**

**≈ 185 sec + .0066 sec ≈ 185 seconds.**

**Câu truy vấn được biến đổi thành:**

****

**Phép tính thời gian truy vấn I/O cho đĩa chia sẻ là:**

**Query I/O time (sequential scan of entire table, shared disk)**

**= 3 × 5.8 ms = 17.4 ms.**

**Disk I/O time before Profit\_by\_Customer**

**= query frequency × I/O time per query**

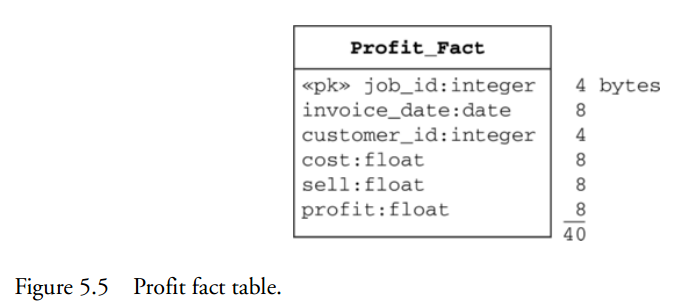
**= (3 queries/day) × (487 sec/query) = 1,461 seconds.**

**Disk I/O time with Profit\_by\_Customer**

**= creation cost + query frequency × I/O time per query**

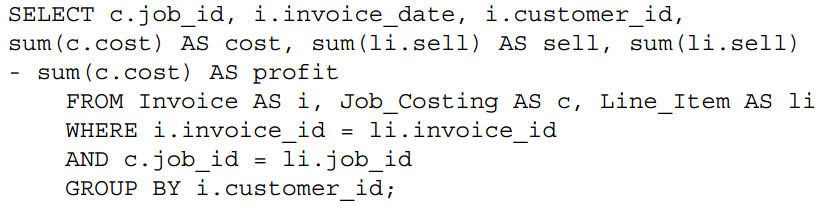
**= 185 sec creation cost + (3 queries/day)**

**× (0.017 sec/query) ≈ 185 seconds.**

****

**Nếu chúng bạn nhìn lại hình 5.2, 5.3 và 5.4, bạn sẽ thấy rằng chúng đều có cost, sell và profit. Lý do là chúng là yếu tố để kết hợp các bảng này. Hình 5.5 là lược đồ bảng kết hợp. job\_id là khóa chính, bởi vì chúng tôi có phụ thuộc hàm: job\_id → invoice\_date, customer\_id, cost, sell, profit .**

**Câu truy vấn phát sinh dữ liệu từ bảng Profit\_Fact là:**

****

**Chi phí tạo bảng Profit\_Fact bằng với chi phí truy vấn từ bảng cơ sở và cho phí viết kết quả lên đĩa. Câu truy vấn không sử dụng bảng cơ sở dẫn tới phép kết giống nhau mà chúng tôi tính được cho bảng Profit\_by\_Customer.**

**Query I/O time (dedicated disk) = 84,008 × 2.2 ms ≈ 185 seconds.**

**Số dòng bằng với số công việc, bởi vì job\_id là khóa chính. Vì thế, có 2,000,000 dòng trogn bảng Profit\_Fact. Các phép tính của bảng Profit\_Fact:**

**Average rows per block (buffer)**

**= floor((65,536 bytes/buffer)/(40 bytes/row)) = 1,638.**

**Number of buffers = ceiling(2,000,000 rows/(1,638 rows/buffer))**

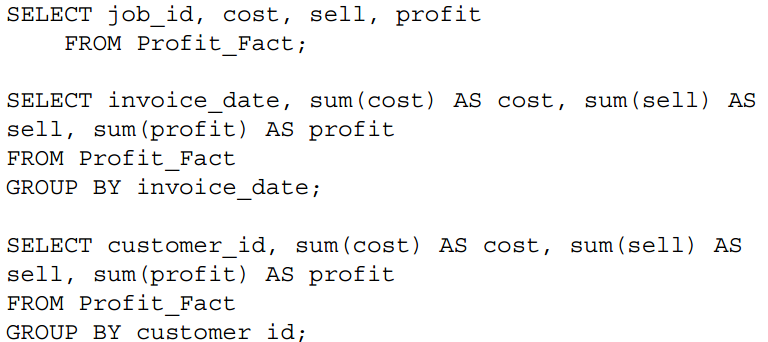
**= 1,222.**

**Write cost (dedicated disk) = 1,222 × 2.2 ms ≈ 3 seconds.**

**Profit\_Fact creation cost = query cost + write cost**

**≈ 185 sec + 3 sec ≈ 188 seconds.**

**Sử dụng khung nhìn cụ thể hóa mô tả trong hình 5.5, 3 câu truy vấn của chúng tôi là:**

****

**Tất cả các câu truy vấn đều quét qua bảng Profit\_Fact\_Table.**

**Query I/O time (sequential scan, shared disk)**

**= 1,222 × 5.8 ms ≈ 7 seconds.**

**Tóm lại, chúng tôi sẽ tính tổng thời gian cho ba câu truy vấn: lợi nhuận theo công việc, theo ngày, theo khách hàng. Chúng tôi sẽ tính số trường hợp khác nhau cho ba câu truy vấn: sử dụng bảng cơ sở, khung nhìn dành riêng cho mỗi câu và sử dụng bảng Profit\_Fact\_Table.**

**Disk I/O times using base tables**

**= (4,780 sec + 2,435 sec + 1,461 sec)/day = 8,676 seconds.**

**Disk I/O times using Profit\_by\_Job, Profit\_by\_Invoice\_Date,**

**and Profit\_by\_Customer = (233 sec + 185 sec + 185 sec)/day**

**= 603 sec/day.**

**Disk I/O time using Profit\_Fact table**

**= creation cost + query frequency × I/O time per query**

**= 188 sec creation cost + (10 profit by job queries/day)**

**× (7 sec/query)**

**+ (5 profit by date queries/day) × (7 sec/query)**

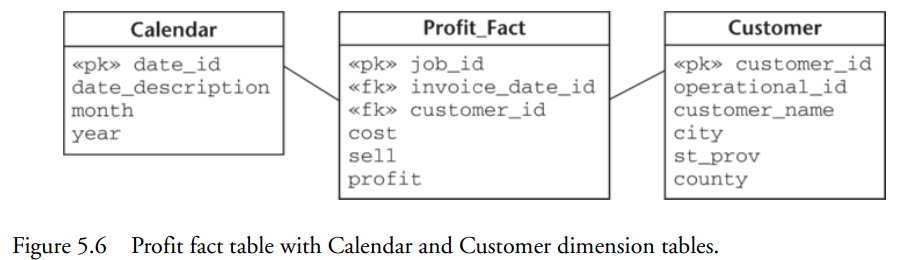
**+ (3 profit by customer queries/day) × (7 sec/query)**

**≈ 314 seconds per day.**

**Sử dụng khung nhìn chung tiết kiệm nhiều thời gian hơn so với sử dụng ba khung nhìn cho mỗi câu truy vấn. Nếu như cách này không được chấp nhận trong một số trường hợp, phương án thay thế là tạo bảng Profit\_Fact, sau đó tạo các khung nhìn khác từ bảng Profit\_Fact hơn là tạo từ các bảng cơ sở. Bởi vì bảng Profit\_Fact nhỏ hơn các bảng cơ sở, thời gian tạo các khung nhìn sẽ được giảm đáng kể.**

1. **KHAI THÁC THEO NHÓM VÀ TỔNG QUÁT**

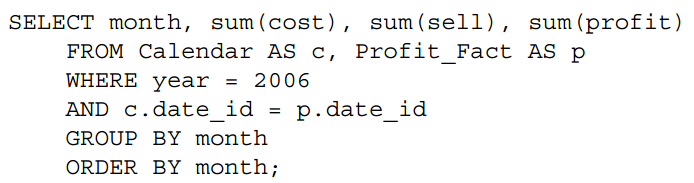
**Tình hình của công ty phụ thuộc vào các xu hướng ngắn hạn và dài hạn. Theo dõi mỗi ngày là cần thiết, nhưng chúng ta cũng cần các báo cáo tháng và năm. Tương tự như vậy, theo dõi lợi nhuận theo khách hàng cũng rất cần thiết. Chúng ta có thể mở rộng bảng trong hình 5.5 để hỗ trợ cho các câu truy vấn khác. Hình 5.6 cho thấy việc bổ sung hai bảng để tạo điều kiện thuận lợi cho các câu truy vấn khác. Bảng Calendar chứa cột tháng và năm có thể được gom nhóm để tính xu hướng từ bảng Profit\_Fact. Cột state\_province và country của bảng Customer có thể được gom nhóm để tính chiến lược về địa lý.**

****

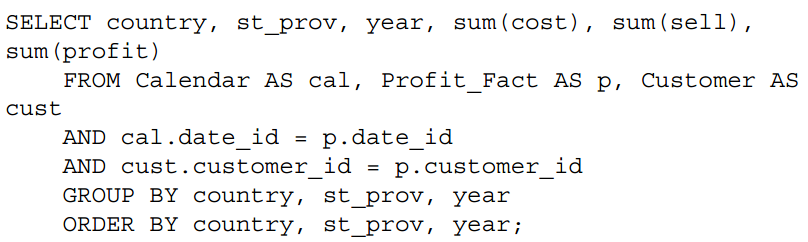
**Nếu bạn đã quen thuộc với kho dữ liệu, bạn sẽ nhận ra hình 5.6 là một lược đồ sao đơn giản với các bảng hai chiều, cụ thể là bảng Calendar và Customer.**

**Chúng tôi đề nghị hai câu truy vấn ví dụ minh họa làm thế nào các bảng này có thể được sử dụng để nhóm dữ liệu trong các kết quả chung. Sau đó chúng tôi sẽ tổng hợp mô hình liệt kê và tổng hợp các mối kết hợp trong mỗi chiều.**

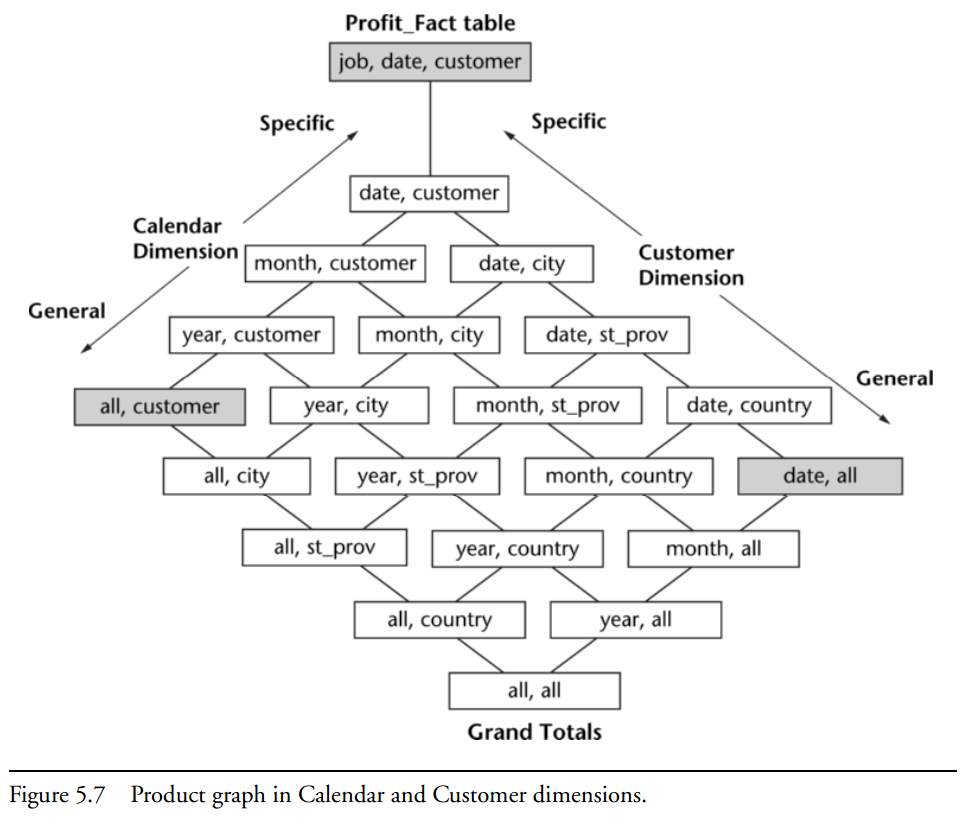
**Đây là câu truy vấn lấy thông tin lợi nhuận theo tháng năm 2006:**

****

**Đây là câu truy vấn lấy thông tin theo năm:**

****

**Có nhiều phép kết của các cấp có thể. Chúng tôi có thể liệt kê các phép kết và minh họa mối quan hệ trong mô hình. Hình 5.7 là mô hình cho lược đồ sao trong hình 5.6. Trên thực tế, mô hình sẽ có ba chiều, bởi vì có chiều Job. Tuy nhiên. Để giữ cho đồ thị đơn giản, chúng tôi đã bỏ bậc gồm job\_id, ngoại trừ bảng Profit\_Fact quan trọng trong dữ liệu nguồn. Chiều Calendar và Customer là trực giao. Mỗi nút là một khung nhìn.**

****

1. **XEM XÉT TÀI NGUYÊN**

**Có một số hạn chế về tài nguyên khi bạn thiết kế khung nhìn. Bao gồm số lượng khung nhìn, dung lượng đĩa yêu cầu, chiều dài của cửa sổ cập nhật có sẵn. Chúng tôi sẽ thảo luận từng yếu tố.**

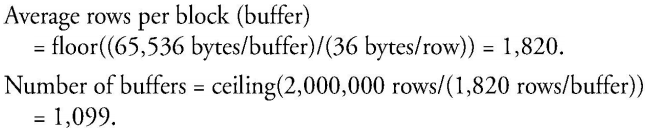
**Số lượng các khung nhìn có thể có trong một cơ sở dữ liệu thông thường rất lớn. Cho một tập hợp các kích thước, số lượng các khung nhìn là sản phẩm của số lượng bậc trong mỗi kích thước. Số lượng các khung nhìn có thể có trong ví dụ hình 5.7 là 4 bậc Calendat X 5 bậc Customer X 2 bậc Job = 40 khung nhìn có thể có. Hãy nhớ rằng, chúng tôi bỏ bậc của bảng Job để mô hình dễ đọc hơn. Chúng tôi chỉ kiểm tra quá trình kinh doanh của phân tích lợi nhuận. Thông thường có nhiều tiến trình kinh doanh cần phải xem xét khi thiết kế CSDL. Rõ ràng là chúng ta không thể cụ thể hóa mọi khung nhìn, vì thế chúng tôi phải chọn ra một tập hợp các chiến lược của các khung nhìn cho sự cụ thể hóa.**

**Nghiên cứu trước đây trong vấn đề chọn khung nhìn cụ thể hóa đã sử dụng hạn chế về số lượng khung nhìn được cụ thể hóa [Harinarayan, Rajaraman, and Ullman 1996]. Ngày nay, nó vẫn còn hữu dụng để ép buộc số lượng khung nhìn cụ thể hóa vì ít nhất hai lý do: thời gian có thể để thiết kế CSDL là hạn chế và cụ thể hóa nhiều khung nhìn đòi hỏi nhiều thời gian thiết kế hơn. Cụ thể hóa nhiều khung nhìn còn đòi hỏi nhiều tiến trình cập nhật. Không những làm tăng số lượng các khung nhìn cụ thể hóa sẽ sử dụng nhiều hơn các tài nguyên máy tính, nó còn yêu cần nhiều CSDL quản lý. Khi các bảng cơ sở phát triển với nhu cầu của công ty, các khung nhìn cụ thể hóa và các tiến trình cập nhật phải được điều chỉnh. Thiết lập một giới hạn về số lượng các khung nhìn, bạn đang sẵn sang để thiết kế và duy trì nó.**

**Tập trung vào các khung nhìn được dùng để trả lời thường xuyên các câu truy vấn là một ý kiến hay để giảm sự phức tạp của quá trình thiết kế để dễ quản lý hơn. Xây dựng một sản phẩm đồ thị hoàn chỉnh là một vấn đề có độ phức tạp tăng theo cấp số nhân của số lượng kích thước (dimension). Tuy nhiên, một mạng đơn giản có thể được xây dựng dựa trên các câu truy vấn thường xuyên. Mạng lưới trong hình 5.8 minh họa một cấu trúc mạng đơn giản. Các bảng cơ sở thích hợp nằm ở trên cùng. Khung nhìn Profit\_Fact có thể đượ tạo từ các bảng cơ sở. Ba khung nhìn khác có thể được tính từ khung nhìn Profit\_Fact.**

**Về cơ bản, mạng lưới trong hình 5.8 là một phiên bản giảng lược của đồ thị trong hình 5.7. Các khung nhìn không được truy vấn thường xuyên không được thêm vào, làm cho đồ thị đơn giản hơn. Chúng tôi đã thêm vào các bảng cơ sở và các lượn đồ khung nhìn vào mạng, một cách ngắn gọn nắm bắt được tình hình. Mỗi nút được gắn với kích thước trong các block, và tần số truy vấn liên quan khi thích hợp. Siêu dữ liệu này rất quan trọng khi tính toán các lợi ích có liên quan của các khung nhìn cụ thể hóa khác nhau.**

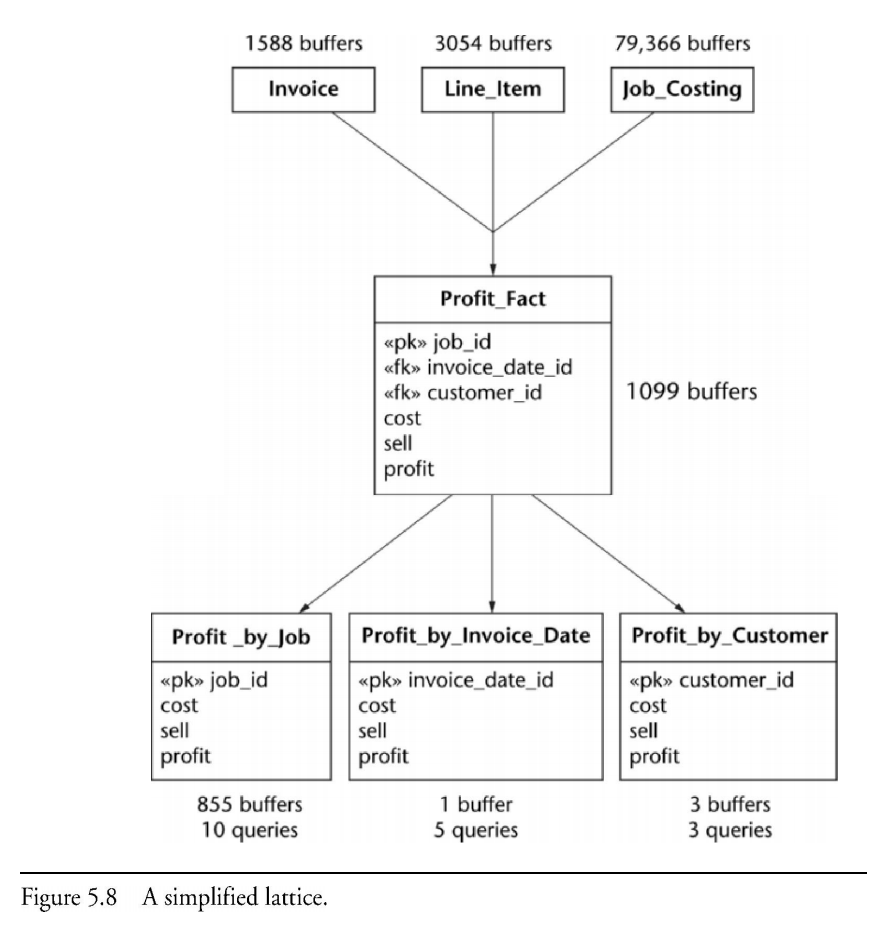
**Hãy chú ý trường invoice\_date\_id 4 byte thay cho trường invoice\_date 8 byte trong hình 5.5. Số lượng các block được tính lại cho phù hợp. Bảng Profit\_Fact được tính như sau:**

****

**Lượng không gian đĩa yêu cầu để lưu trữ các khung nhìn cụ thể hóa là một hạn chế hiển nhiên. Kimball [1998, p544] có đề cập đến quy luật ngón tay cái trong ngữ cảnh của kho dữ liệu, nhiều hệ quản trị CSDL hạn chế tăng không gian cần cho khung nhìn tổng hợp. Điều này có thể khác nhau tùy vào mục đích của CSDL và tài nguyên của tổ chức. Nếu bạn đang hy vọng cải thiện hiệu suất mà không cần kho dữ liệu chính thức, bạn có thể thiết lập một giới hạn không gian đĩa vừa phải hơn. Nếu các khung nhìn mang lại lợi ích lớn nhất được chọn để khai thác đầu tiên, sau đó, cam kết các mức độ vừa phải của không gian đĩa có thể mang lại các cải tiến cho sự trả lời câu truy vấn. Cam kết lượng không gian đĩa lớn hơn sẽ mang lại sự trả lời truy vấn nhanh hơn, nhưng sự trả về thì chậm hơn. Hãy quyết định giới hạn cho số lượng của không gian đĩa.**

**Nếu bạn nhận ra rằng quá trình phát triển đang thúc đẩy giới hạn không gian đĩa, có nhiều lựa chọn để mở rộng tính hữu dụng trong khi bạn vẫn duy trì trong khả năng tổ chức của bạn. Ví dụ, ban quản trị quyết định rằng họ muốn theo dõi các dữ liệu tổng kết trong năm năm. Bạn có thể tiếp tục thanh lọc dữ liệu từ các bảng cơ sở sau khi lấy nửa năm, và cùng lúc đó duy trì dữ liệu của 5 năm trong các khung nhìn cụ thể hóa. Tương tự như vậy, nếu khung nhìn cụ thể hóa chứa dữ liệu trở nên quá lớn, các chi tiết có thể được thanh lọc sau một thời gian thích hợp, trong khi khung nhìn tổng hợp có thể tiếp tục giữa các lịch sử mở rộng. Ví dụ, nếu bảng Profit\_Fact trở nên lớn hơn, các dữ liệu cũ hơn 2 năm có thể được thanh lọc, trong khi bảng Profit\_by\_Invoice\_Date có thể giữ các dữ liệu có giá trị 5 năm và vẫn rất nhỏ. Sự thanh lọc của các dữ liệu chi tiết trong khi vẫn giữ trong dữ liệu tổng hợp đòi hỏi một chiến lược cập nhật phát triển.**

**Thời gian cần cho việc cập nhật là một hạn chế khác để ta xem xét. Thông thường, một cửa sổ cập nhật được dự trữ trong giờ nghỉ, khi các khung nhìn cụ thể hóa được cập nhật trong đĩa chuyên dụng. Hãy xác định lượng thời gian có thể cho việc cập nhật cửa sổ nếu đã áp dụng. Bạn có thể xác định một cửa sổ cập nhật là không cần thiết. Có thể tổ chức của bạn hoạt động 24 giờ 1 ngày, và hệ thống không thể bị làm cho không dùng được cho bất kì cửa sổ cập nhật nào. Có thể người dùng cần dữ liệu tổng hợp tại thời điểm hiện tại, và không thể sử dụng dữ liệu đã được tính tối qua. Cập nhật thường xuyên có thể giải quyết những vấn đề này.**

****

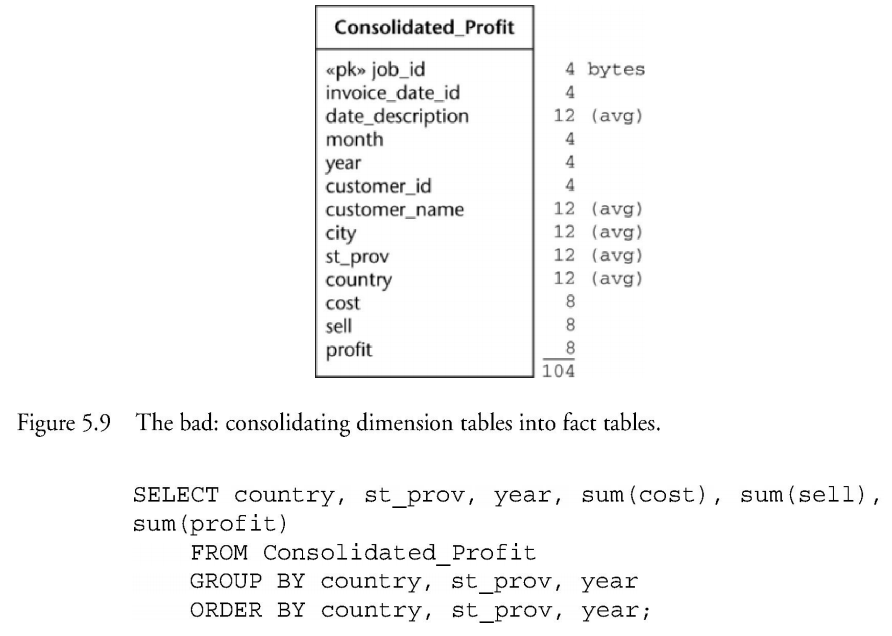
**Thay vì phải tính lại khung nhìn, cập nhật tiếp tục giữ đa số các dòng của khung nhìn cụ thể hóa nguyên vẹn, và thêm các dòng mới nếu thích hợp. Ví dụ, hãy xem bảng Profit\_by\_Invoice\_Date. Hãy xem các hóa đơn được xử lú theo tiến trình hàng ngày. Thông tin được tập hợp cho các công việc đã được vận chuyển từ chu trình hóa đơn trước đó. Mỗi khách hàng có các công việc trong chu trình hiện tại có một hóa đơn giao. Các công việc của khách hàng được liệt kê trong hóa đơn. Sau khi các hóa đơn được xác minh, các hóa đơn được in và gửi cho các khách hàng, và trạng thái của mỗi hóa đơn được cập nhật trong CSDL. Khi chu trình hoàn thành, giá trị bán cho từng công việc đều được biết. Giá của mỗi công việc cũng được biết, bởi vì bảng Job\_Costing đã được cập nhật trong thời gian đó và công việc hoàn thành mỗi công việc trước khi hóa đơn được gửi. chúng tôi có thể tính tổng giá trị mỗi ngày và thêm một dòng vào bảng Profit\_by\_Invoice\_Date. Cập nhật trên khung nhìn cụ thể hóa trong thời gian thực thỉnh thoảng tương tự như phương pháp** *trickle feed***.**

1. **CÁC VÍ DỤ: TỐT, XẤU**

**Các khung nhìn và lược đồ là cách tiếp cận tốt để giảm đĩa I/O, kết quả là câu truy vấn chạy nhanh hơn. Hình 5.6 là một ví dụ của một lược đồ sao, minh họa cách tiếp cận thiết kế không gian. Thông thường, có nhiều chiều trong một lược đồ sao, nhưng ví dụ đã đủ để thảo luận. Bảng thực tế là một khung nhìn lấy được từ các bản cơ sở. Khung nhìn chung quy là để lưu trữ các phép tính để giảm lượng đĩa I/O. Các bảng này cung cấp một phương tiện để gom nhóm dữ liệu ở các cấp độ khác nhau của cùng một mức độ. Các bảng này còn thêm tính uyển chuyển, tăng lợi ích của các bảng thực tế trong một gia đình lớn hơn của các câu truy vấn. Lược đồ sao và cách tiếp cận thiết kế không gian có thể cách tốt để cải thiện tốc độ câu truy vấn.**

**Một câu hỏi tự nhiên phát sinh. Nếu tính lại tiết kiệm thời gian, vậy tại sao không thực hiện lại phép kết giữa các bản thực và các bảng ảo? Liệu làm như vậy có tiết kiệm thời gian hơn không? Lược đồ trong hình 5.9 có thể là sự hợp nhất giữa bảng thực và bảng ảo từ hình 5.6. Bây giờ chúng tôi có thể truy vấn trên một bản, và chúng tôi vẫn có khả năng gom nhóm tương tự như khi chúng tôi thực hiện trên những bảng ảo.**

**Cùng viết lại câu truy vấn thứ 2 trong phần 5.3. Câu truy vấn bây giờ đã đơn giản hơn:**

****

**Tất cả những gì chúng ta cần làm là duyệt trên một bảng. Hãy tính xem có bao nhiêu khối được đưa ra. Có 2.000.000 dòng bởi vì có 2.000.000 công việc. Phép tính của bảng Consolidated\_Profit là:**

**Average rows per block (buffer)**

**= floor((65,536 bytes/buffer)/(104 bytes/row)) = 630.**

**Number of buffers = ceiling(2,000,000 rows/(630 rows/buffer)) = 3,175.**

**Hãy tính số dòng của một block được quét nếu lược đồ hình 5.6 được thay thế. Bảng Calendar có 200 dòng, mỗi dòng trung bình 24 bytes. Bảng Profit\_Fact có 2,000,000 dòng, mỗi dòng 36 bytes. Bảng Customer có 5,000 dòng, trung bình 64 bytes một dòng. Bảng Calendar được tính như sau:**

**Average rows per block (buffer)**

**= floor((65,536 bytes/buffer)/(24 bytes/row)) = 2,730.**

**Number of buffers = ceiling(200 rows/(2730 rows/buffer)) = 1.**

**Bảng Profit: 1,099 buffer.**

**Bảng Customer:**

**Average rows per block (buffer)**

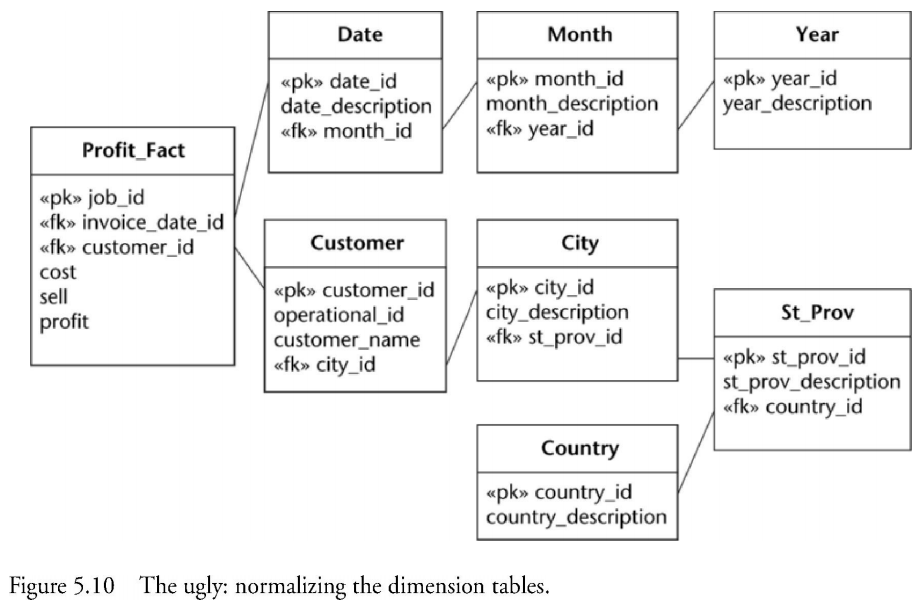
**= floor((65,536 bytes/buffer)/(64 bytes/row)) = 1,024.**

**Number of buffers = ceiling(5,000 rows/(1024 rows/buffer)) = 5.**

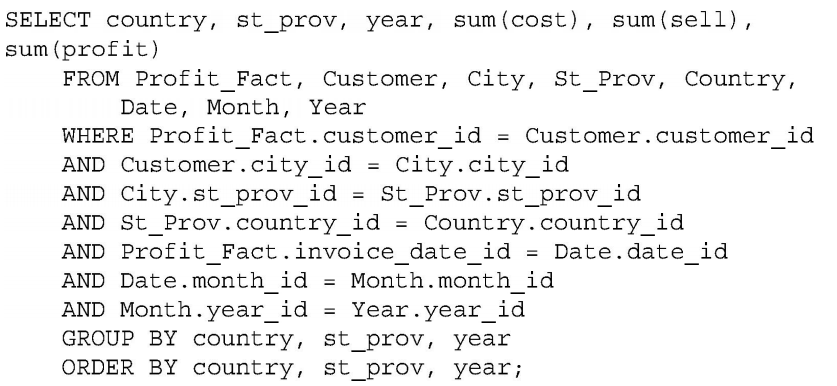
**Tổng số block được quét sử dụng lược đồ sao = 1 + 1,099 + 5 = 1,105 block.**

**Hợp nhất bảng ảo và bảng thực thực chất làm chậm đi câu truy vấn gấp 3 lần. Sự hợp nhất bảng ảo và bảng thực là một ý kiến tồi.**

**Tại sao lại là bảng ảo không bình thường trong hình 5.6? Nếu hợp nhất bảng ảo với bảng thực là một ý kiến tồi, có lẽ nếu chúng tôi bình thường hóa bảng ảo thành một bảng nhỏ hơn thì chúng tôi có thể đạt được nhiều thứ hơn. Hình 5.10 cho thấy bảng ảo được bình thường hóa. Trong các tài liệu gọi nó là lược đồ hoa tuyết.**

****

**Đây là cây truy vấn của chúng tôi, được viết lại từ lược đồ trong 5.10:**

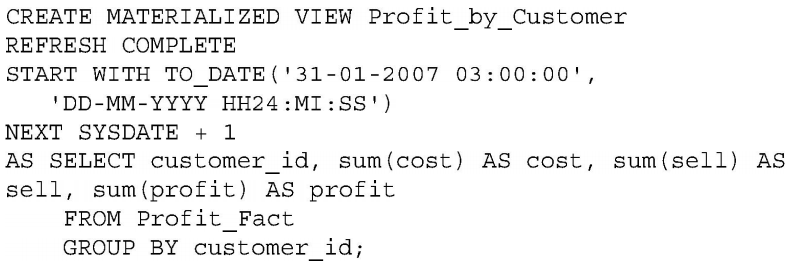
****

**Câu truy vấn bây giờ phức tạp hơn nhiều, có tới 7 phép kết. Kimball và Ross [2002] xác định phép kết phức tạp là một lý do để tránh lược đồ hoa tuyết này. Các câu truy vấn khó viết hơn, và tối ưu hóa câu truy vấn có thể phải tìm một phương án tốt một cách khó khăn. Hơn nữa, lược đồ sao trực quan hơn lược đồ tuyết. Một vài lược đồ có thể tìm thấy các mẫu bình thường hóa của các bảng ảo trong lược đồ hoa tuyết, nhưng thường là các giải pháp rắc rối chứ không phải ngắn gọn.**

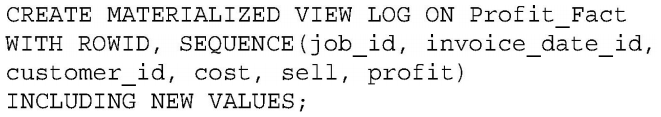
1. **CÁCH SỬ DỤNG CÚ PHÁP VÀ CÁC VÍ DỤ**

**Chúng tôi minh họa một số chiến lược cập nhật có sẵn, sử dụng các ví dụ được viết trên SQL. Các ví dụ SQL trong phần này cũng sử dụng được cho Oracle 10g. Nếu bạn đang sử dụng một phiên bản khác của Oracle, kiểm tra tài liệu của bạn để có thể thay đổi.**

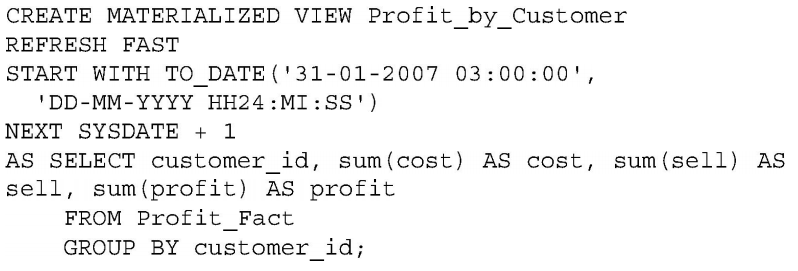
**SQL trong Oracle bao gồm các khái niệm để tạo các khung nhìn và chỉ rõ chiến lược cập nhật. Nếu bạn muốn tạo một khung nhìn để lưu trữ lợi nhuận bởi các dữ liệu khách hàng, hãy dựa trên lược đồ trong hình 5.6. Nếu bạn muốn khung nhìn được nạp lại từ đầu mỗi sáng vào lúc 3 giờ bắt đầu từ ngày 31.7.2007, câu SQL sẽ được viết như sau:**

****

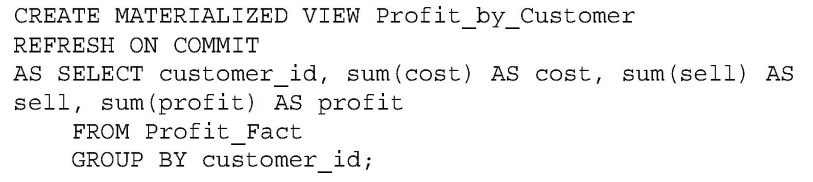
**Dĩ nhiên, nạp lại hoàn toàn sẽ sử dụng nhiều tài nguyên nếu bảng Profit\_Fact có nhiều dòng. Khi bảng nguồn lớn, thường trong trường hợp là một kho dữ liệu, một lựa chọn khác khả thi hơn là sử dụng các lựa chọn nạp lại nhanh. Nạp lại nhanh là một chiến lược cập nhật lớn. Về cơ bản các dữ liệu hiện có trong khung nhìn vẫn còn nguyên vẹn, những thay đổi trên bảng nguồn luôn được theo dõi, và quá trình cập nhật tổng kết các dữ liệu mới và sau đó điều chỉnh hay tạo mới các dòng thích hợp trên khung nhìn. Nếu chúng tôi muốn sử dụng phương pháp nạp lại nhanh, sau đó chúng tôi cần ra lệnh cho Oracle theo dõi sự thay đổi trên bảng dữ liệu nguồn.**

****

**Dưới đây là sự định nghĩa cho lợi nhận bằng cách tính tổng theo khách hàng, tận dụng lợi thế của chiến lược nạp lại nhanh:**

****

**Những chiến lược khác cũng khả thi. Ví dụ như sự định nghĩa dưới đây giữ khung nhìn được cập nhật bất cứ khi nào có sự commit trên bảng Profit\_Fact.**

****

**Thông thường, các kho dữ liệu chứa nhiều bảng lớn, và các tiến trình nạp lại chạy trong lúc các cửa sổ được cập nhật. Sự lựa chọn “nhanh” hỗ trợ chiến lược này. Nếu bạn có một khung nhìn nơi một phần đáng kể của các dòng thay đổi mỗi ngày, thì giải pháp nạp lại hoàn toàn có thể sẽ chạy nhanh hơn, vì các phép tính lớn sẽ hoạt động hiệu quả hơn so với một số lượng lớn những thay đổi gia tăng. Sự lựa chọn “on commit” hỗ trợ việc cập nhật hiện thời (real-time). Nếu bạn cần dữ liệu hiện thời, hãy luôn nhớ rằng việc cập nhật khung nhìn bất cứ khi nào có một sự commit trên bảng nguồn có thể là nguồn tài nguyên lớn. Bạn có thể sẽ không được hỗ trợ nhiều loại khung nhìn như những chiến lược mà bạn đã lập trình sẵn.**

**Oracle có một thiết lập QUERY\_REWRITE\_ENABLE nên được cài trong tập tin thông số cấu hình để làm đầy đủ hơn mục đích của khung nhìn mà bạn tạo. Nếu câu truy vấn viết lại được bật, Oracle sau đó sẽ tự động nhận ra khi có một khung nhìn nhỏ có thể dùng để trả lời câu truy vấn, thay vì đọc nhiều bảng nguồn cơ sở lớn hơn.**

1. **TỔNG KẾT**

**Khung nhìn là kết quả của câu truy vấn được lưu trữ trong đĩa. Khung nhìn lưu trữ có hiệu quả kết quả tính toán, cho phép tái sử dụng kết quả, làm cho câu truy vấn ra kết quả nhanh hơn. Số lượng các khung nhìn có thể có thường rất lớn, yêu cầu người thiết kế phải liên tục lựa chọn các chiến lược của khung nhìn. Sự tìm hiểu tốt sẽ giúp tập trung vào những câu truy vấn thường xuyên và chiếm nhiều dung lượng đĩa I/O. Mỗi câu truy vấn sẽ tự ánh xạ lên một khung nhìn. Các đồ thị kết quả và các cấu trúc mạng có thể chứa các mối quan hệ giữa các khung nhìn, tạo điều kiện thuận lợi cho sự lựa chọn các khung nhìn để sử dụng. Những khung nhìn tốt là khung nhìn tự nhiên của những câu truy vấn thường xuyên, và cũng là hình thức ban đầu chung của những khung nhìn đó. Các lược đồ thường là mô hình tốt để làm theo và tăng khả năng sử dụng của các bảng trong một tập lớn các câu truy vấn. Việc lựa chọn các khung nhìn thông thường theo cách tiếp cận tham lam, chọn khung nhìn thuận lợi nhất ở mỗi bước, cho đến khi đạt đến sự hạn chế tài nguyên. Sự hạn chế tài nguyên có thể bao gồm số lượng khung nhìn, sự sử dụng không gian đĩa và thời gian yêu cầu để duy trì khung nhìn.**