فهرست

بهینه سازی.

۱۸,۱ مقدمه.

بهینه سازی برای سیستم‌های رابطه‌ای یک چالش و یک فرصت است. چالش، از این رو که اگر سیستم بخواهد به کارایی قابل قبولی برسد، بهینه سازی لازم است؛ و فرصت از این رو که عبارات رابطه‌ای در چنان سطح معنایی بالایی قرار دارند که بهینه سازی از همان مرحله اول امکان پذیر است و این دقیقاً یکی از نقاط قوت سیستم‌های رابطه‌ای است. در مقایسه با، سیستم غیر رابطه‌ای که درخواست‌های کاربر در یک سطح پایینی از معنا ارائه می‌شود، هر گونه «بهینه سازی» توسط خود کاربر صورت می‌گیرد («بهینه سازی» داخل گیومه است، زیرا خود واژه بهینه سازی به معنای بهینه سازی خودکار است). به عبارت دیگر در چنین سیستم‌هایی این کاربر است که تصمیم می‌گیرد که چه عملیات سطح پایینی نیاز است و به چه ترتیبی این عملیات‌ها باید اجرا شوند؛ و اگر کاربر تصمیم بدی بگیرد، سیستم نمی‌تواند برای بهبود موضوع، هیچ کاری انجام دهد. همچنین نکته آنکه کاربر در این سیستم‌ها باید تجربه برنامه نویسی داشته باشد این موضوع، سیستم را برای بسیاری که این تجربه را ندارند، خارج از دسترس می‌کند.

مزیت بهینه سازی خودکار، تنها این نیست که کاربران نگران چگونگی مطرح کردن پرسش‌هایشان نباشند (یعنی چطور عبارت بندی درخواست‌های خود را بیان کنند که بهترین کارایی را داشته باشند).حقیقت این است که این محتمل است که بهینه ساز بهتر از کاربر انسانی عمل کند. چندین دلیل برای این ادعا وجود دارد، از جمله انها عبارتند از:

۱. اطلاعاتی که بهینه ساز خوب (سیستم) از وضع پایگاه داده دارد یک کاربر در اختیار ندارد. به طور مشخص، آن یکسری اطلاعات آماری معین (کاردینالتی اطلاعات و شبیه آن) می‌داند که عبارتند:
   - تعداد مقادیر مجزا از هر نوع.
   - تعداد تاپل های هر رابطه.
   - تعداد مقادیر مجزای هر صفت در هر رابطه.
   - تعداد دفعاتی که هر مقدار در هر صفت تکرار می‌شود.

و اطلاعات دیگر، تمام این اطلاعات در کاتالوگ سیستم نگهداری می‌شود. در نتیجه بهینه ساز قادر خواهد بود برآورد دقیقی از هر استراتژی که می‌توان برای پیاده سازی یک درخواست خاص (اجرای پرسش) بکار ببرد، داشته باشد و بنابراین به احتمال فراوان کاراترین استراتژی را برای اجرای پرسش انتخاب می‌نماید.

۲. به علاوه، اگر در طول زمان آمارهای پایگاه داده تغییر کند، آنگاه ممکن است یک استراتژی دیگری برای اجرای پرسش انتخاب شود. به عبارت دیگر، ممکن است دوباره بهینه سازی لازم باشد. در سیستم‌های رابطه‌ای بهینه سازی مجدد آسان است. این کار مستلزم پردازش مجدد اصل درخواست رابطه‌ای توسط بهینه ساز است. در مقایسه با سیستم‌های غیر رابطه‌ای که بهینه سازی مجدد مستلزم بازنویسی برنامه است و به احتمال زیاد به خوبی کار نخواهد کرد.

۳. همچنین، بهینه ساز یک برنامه است و بنابراین طبق تعریف، نسبت به نمونه کاربر انسانی حوصله بیشتری دارد. بهینه ساز به راحتی قادر خواهد بود که برای یک درخواست (پرسش) صدها استراتژی اجرای مختلف را در نظر بگیرد در حالی که یک کاربر انسانی نهایتاً سه یا چهار استراتژی مختلف را در نظر می‌گیرد.

٤. در آخر، بهینه ساز قادر است تا حدودی، مهارت‌ها و سرویس‌های بهترین برنامه نویسان انسانی را در خود جای دهد. در نتیجه بهینه ساز، این مهارت‌ها و سرویس‌ها را در اختیار همه قرار می‌دهد- یعنی در یک روش موثر و سودمند مجموعه کمی از منابع موجود را در اختیار طیف وسیعی از کاربران قرار می‌دهد.

تمام موارد فوق باید گواهی بر این ادعا باشد که گفتیم، قابلیت بهینه سازی (یعنی در حقیقت قابلیت بهینه سازی درخواست‌ها) در واقع یک قدرت سیستم‌های رابطه‌ای است.

بنابراین، انتخاب یک استراتژی کارا برای ارزیابی یک عبارت رابطه‌ای هدف کلی بهینه ساز است. در این فصل اصول اساسی و تکنیک‌هایی که در فرایند بهینه سازی لازم است را به طور خلاصه توصیف می‌کنیم. در ادامه مقدمه، یک مثال در بخش ۱۸,۲ خواهیم دید، در بخش ۱۸,۳ یک دید کلی از چگونگی عملکرد بهینه ساز خواهیم گفت و سپس در بخش ۱۸,٤ به یکی از جنبه‌های بسیار مهم این فرایند یعنی **تبدیل عبارت**[1] (بازنویسی پرسش) می‌پردازد. در بخش ۱۸,۵ در مورد آمارهای پایگاهی بحث خواهد شد و سپس در بخش ۱۸,٦، یک رهیافت ویژه بهینه سازی که **تجزیه پرسش**[2] نام دارد را با جزئیات بررسی می‌کنیم. بخش ۱۸,۷ چگونگی پیاده سازی عملگرهای رابطه‌ای (یا عطفی و غیره) را با در نظر داشتن آمارهای پایگاهی بحث خواهیم کرد.

آخرین توضیحات مقدماتی: معمولاً به این موضوع تحت عنوان بهینه سازی پرسش مراجعه می‌شود، این واژه کمی گمراه کننده است، هر چند، نظر به اینکه عبارتی که باید بهینه شود- «پرسش» - ممکن است که در مواردی غیر از بررسی‌های محاوره‌ای پایگاه داده بکار رود (به ویژه، ممکن است بخشی از عملیات به هنگام

---

[1] Expression transformation
[2] Query decomposition

سازی باشد تا یک پرسش). بهینه سازی یک ادعای بزرگی است، زیرا تضمینی وجود ندارد که استراتژی انتخاب شده از هر نظر بهینه باشد، شاید واقعاً این چنین باشد ولی آنچه همیشه می‌توان مطمئن بود این است که استراتژی «بهینه شده»، بهبودی از نسخه بهینه نشده اصلی است. هر چند در بعضی موارد، به طور صحیح می‌توان این ادعا را کرد که استراتژی انتخاب شده، از بسیاری جهات بهینه است.

۱۸,۲ مثال.

با مثال ساده‌ای شروع می‌کنیم که یکسری ایده امکان پذیر ارائه می‌دهد که اجرای پرسش را به طور چشمگیری بهبود می‌بخشد. پرسش این است: «اسامی عرضه کنندگانی را که قطعه p_2 را عرضه می‌کنند بدست آورید.» یک پاسخ جبر رابطه مربوط به این پرسش بدین صورت است:

( ) SP JOIN S WHERE P# = P# ('S_2') {SNAME}

فرض کنید پایگاه داده شامل ۱۰۰ عرضه کننده (suppliers) و ۱۰,۰۰۰ محموله است که تنها ۵۰ محموله مربوط به قطعه p_2 است. همچنین برای سادگی فرض کنید که حیطه‌های عمل (رابطه‌های) S و SP در دو فایل جداگانه هستند و برای هر تاپل یک رکورد ذخیره شده و این رکوردها به طور مستقیم بر روی دیسک (یعنی در حافظه اصلی نیستند) ذخیره شده‌اند. بنابراین اگر سیستم در این شرایط واقعاً می‌خواست عبارت داده شده را ارزیابی کند–بدون هیچ گونه بهینه سازی– لازم است دنبال‌های از رویدادهای زیر را انجام دهد:

۱. پیوند[1] رابطه‌های S و SP روی S# . این مرحله مستلزم خواندن ۱۰,۰۰۰ محموله: خواندن هر ۱۰۰ عرضه کننده به تعداد ۱۰,۰۰۰ بار (یک‌بار برای هر ۱۰,۰۰۰ محموله) : ایجاد یک نتیجه بینابینی با ۱۰,۰۰۰ تاپل پیوندی: و نوشتن این ۱۰,۰۰۰ تاپل پیوندی بر روی

---

[1] join

دیسک است (فرض کنید که برای این نتیجه بینابینی جایی در حافظه اصلی وجود ندارد).

۲. نتیجه حاصل از مرحله یک را تنها برای تاپل های مربوط به قطعه p_2 گزینش[1] کنید: این مرحله مستلزم دوباره بار کردن ۱۰,۰۰۰ تاپل پیوند شده به حافظه اصلی است و ایجاد نتیجه‌ای (رابطه‌ای) که تنها ۵۰ تاپل دارد که فرض می‌شود که به اندازه کافی کوچک هست که در حافظه اصلی نگهداری شود.

۳. نتیجه حاصل از مرحله دو را بر روی صفت SNAME پرتو[2] کنید (رابطه نتیجه تنها شامل صفت SNAME باشد). این مرحله نتیجه نهایی مورد نظر را تولید می‌کند (حداکثر ۵۰ تاپل، که می‌تواند در حافظه اصلی بماند).

اکنون روال دیگری را شرح می‌دهیم که نتیجه نهایی آن معادل نتیجه نهایی رویه قبلی است، اما بسیار کاراتر است:

۱. رابطه SP را تنها برای تاپل های مربوط به قطعه p2 گزینش کن: این مرحله مستلزم خواندن ۱۰,۰۰۰ تاپل است اما نتیجه حاصل شامل ۵۰ تاپل است و می‌توان آن‌را در حافظه اصلی نگهداری کرد.

۲. پیوند رابطه حاصل از مرحله اول با رابطه S بر روی #S: این مرحله مستلزم خواننده ۱۰۰ عرضه کننده (تنها یک‌بار، نه اینکه برای هر محموله p_2 یک‌بار خوانده شود) و تولید یک رابطه نتیجه که دوباره تنها شامل ۵۰ تاپل است (و در حافظه اصلی می‌ماند).

---

[1] restrict
[2] project

۳. نتیجه حاصل از مرحله دو را بر روی صفت SNAME پرتو[1] کنید (مثل مرحله ۳ رویه قبل). نتیجه نهایی مورد نظر (حداکثر ۵۰ تاپل) که در حافظه اصلی است.

در اولین رویه لازم است که جمعاً ۱,۰۳۰,۰۰۰ ورودی خروجی تاپل انجام می‌شود، در حالی که برای رویه دوم تنها ۱۰,۰۰۰ ورودی خروجی تاپل لازم است. بنابراین، روشن است که اگر «تعداد ورودی خروجی‌های تاپل» را به عنوان معیار کارایی در نظر بگیریم، آنگاه رویه دوم کمی بیشتر از ۱۰۰ برابر بهتر از رویه اول است. همچنین روشن است که علاقه‌مند هستیم که در پیاده سازی از رویه دوم بیشتر از رویه اول استفاده کنیم! توجه : در عمل ملاک تعداد ورودی خروجی‌ها، صفحه است نه تاپل، بنابراین برای سادگی فرض می‌کنیم که هر تاپل در یک صفحه جای دارد (یعنی هر تاپل صفحه خود را پر کرده است).

همانطور که دیدیم با انجام یکسری تغییرات ساده در اجرای الگوریتم– انجام یک گزینش و سپس یک پیوند بجای یک پیوند و سپس یک گزینش در کارایی بهبود چشمگیری حاصل شد. همچنین اگر محموله‌ها بر روی P# شاخص گذاری یا درهم سازی شده باشند، این بهبود چشمگیرتر خواهد شد. – تعداد محموله‌های خوانده شده در مرحله اول از ۱۰,۰۰۰ به تنها ۵۰ تاپل کاهش یافت و بنابراین رویه جدید در حدود ۷,۰۰۰ بار بهتر از رویه اصلی شد. به طور مشابه اگر عرضه کنندگان نیز بر روی S# شاخص گذاری و یا درهم سازی شده باشند، تعداد تاپل های عرضه کننده که در مرحله ۲ خوانده شده از ۱۰۰ به ۵۰ کاهش می‌یابد. بنابراین تا اینجا رویه ۱۰,۰۰۰ بار بهتر از رویه اصلی شده است. یعنی اگر پرسش بهینه نشده اصلی در ۳ ساعت اجرای شود، نسخه نهایی در کسری از ثانیه اجرا خواهد شد و البته امکان بهبود بیشتری نیز وجود دارد.

---

[1] project

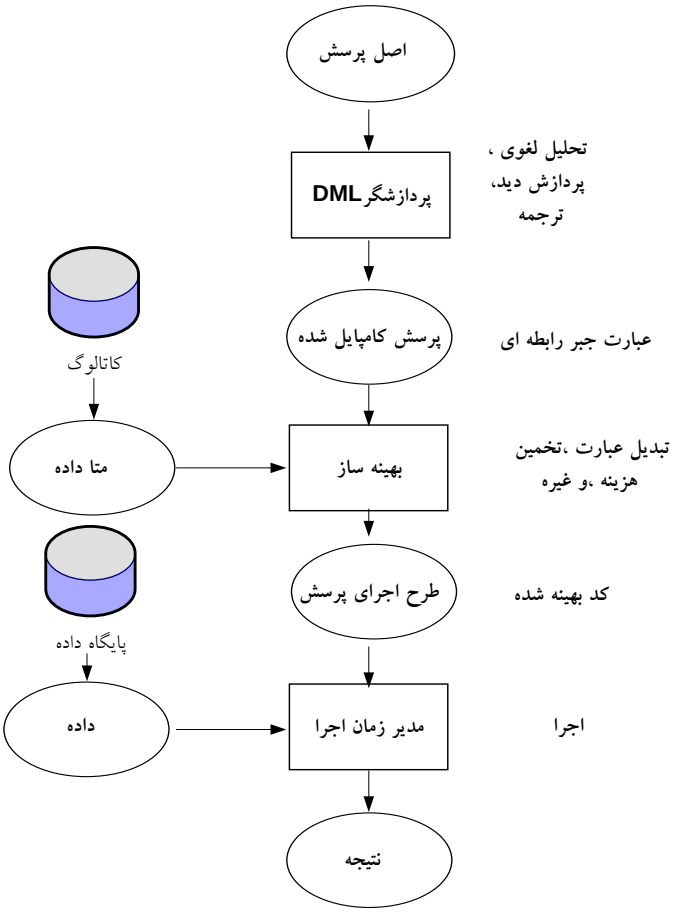
مثال مذکور اگرچه ساده است اما برای نشان دادن اینکه نیاز به بهینه سازی و انواع بهبودها برای اجرای پرسش لازم است، کافی است. در بخش بعدی، بر یک رهیافت سیستماتیک برای انجام کار بهینه سازی مروری خواهیم داشت: به ویژه نشان خواهیم داد که مسئله کلی می‌توانند کم و بیش به یک سری زیر مسئله مستقل تقسیم شود.

۱۸,۳ مراحل کلی پردازش پرسش.

می‌توان برای پردازش پرسش چهار مرحله را مشخص نمود که عبارتند از شکل ۱۸,۱ را ببینید.

۱. تبدیل پرسش به یک فرم بینابینی : معمولاً جبر رابطه‌ای.
۲. تبدیل به یک فرم کانونی (کانونی یا متعارف[1]).
۳. انتخاب رویه‌های کاندید سطح پایین.
٤. تولید طرح‌های اجرای پرسش و انتخاب کم هزینه‌ترین طرح اجرا.

[1] Canonical form



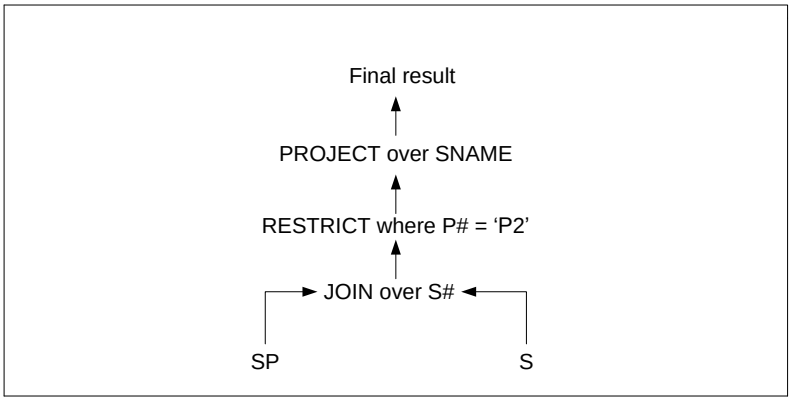
شکل ۱۸,۱: مراحل کلی پردازش پرسش.

مرحله ۱: تبدیل پرسش به یک فرم درونی.

اولین مرحله شامل تبدیل اصل پرسش به یک فرم درونی (نمایشی درونی) است که برای دست‌کاری ماشین، مناسب‌تر است در نتیجه، ملاحظات محض خارجی (مثل، تزئیناتی که مربوط به نحو زبانی است که پرسش توسط آن نوشته شده است) حذف خواهد شد و راه برای مراحل بعدی در کل فرایند هموار خواهد شد. نکته: پردازش دید نیز در همین مرحله صورت می‌پذیرد یعنی فرآیند جایگزین کردن مراجع

(مقادیر جداول پایه) به دیدها، به وسیله بکار بستن عبارات تعریف دید، در این مرحله انجام می‌گیرد.

حال این پرسش مطرح است: فرم درونی باید به چه شکل باشد؟ هر فرمی که انتخاب شود باید آنقدر غنی باشد که هر پرسش ممکن که توسط یک زبان پرسش خارجی (مثلاً SQL) نوشته شده است را نمایش دهد. همچنین باید حتی لامکان بی‌طرف باشد، بی‌طرف بدین مفهوم که در انتخابات بعدی پیش داوری نداشته باشد. فرم درونی که معمولاً انتخاب می‌شود نوعی **درخت نحو انتزاعی**[1] یا **درخت پرسش**[2] است. برای مثال شکل ۱۸,۲ یک نمایش درخت پرسش ممکن را برای مثال بخش ۱۸,۲ («نام عرضه کنندگانی که قطعه p_2 را عرضه می‌کنند بدست آورید») نمایش می‌دهد.



شکل ۱۸,۲: اسامی عرضه کنندگانی را که قطعه p2 را عرضه می‌کنند را بدست آورید (درخت پرسش).

به هر حال بهتر آن است که نمایش داخلی، یکی از شکل‌هایی باشد که با آن آشنایی داریم، یعنی جبر رابطه‌ای با حساب رابطه‌ای. مثال نشان داده شده در شکل ۱۸,۲ می‌تواند به یکی از فرم‌های حساب رابطه‌ای با جبر رابطه باشد، برای تمرکز بیشتر

---

[1] Abstract syntax tree
[2] Query tree

فرض را بر استفاده از جبر رابطه‌ای برمی نهیم. بنابراین از این پس نمایش درونی شکل ۱۸,۲ دقیقاً برابر است با آنچه پیش‌تر نشان داده شد:

```
SP JOIN S WHERE P# = P# ('S2') {SNAME}
```

مرحله ۲: تبدیل به یک فرم کانونی (متعارف).

در این مرحله، بهینه ساز تعدادی عمل بهینه سازی را که «تضمین می‌شود خوب باشند» بدون در نظر گرفتن مقادیر واقعی داده‌ها ذخیره شده در پایگاه داده و مسیرهای فیزیکی دست‌یابی به آن‌ها، انجام می‌دهد. نکته این است که زبان‌های رابطه‌ای اجازه می‌دهند که همه (بجز ساده‌ترین) پرسش‌ها به روش‌های گوناگونی بیان شوند که حداقل ظاهراً مجزا هستند. برای مثال اگر در SQL یک پرسش به سادگی «نام عرضه کنندگانی که قطعه p_2 را عرضه می‌کنند بدست آورید» باشد می‌تواند تحت لفظی به روش‌های گوناگون نمایش داده شود (بدون در نظر گرفتن شکل‌های بدیهی مثل جایگزینی A = B با B = A یا p AND q با q AND p). البته نباید کارایی پرسش به روش خاصی که کاربر برای نوشتن آن استفاده کرده بستگی داشته باشد. بنابراین مرحله بعدی در پردازش پرسش تبدیل فرم درونی (بینابینی) به فرم کانونی معادل آن با هدف حذف این تمایزات ظاهری و مهم‌تر از همه، پیدا کردن نمایشی که از اصل پرسش به نحوی کاراتر باشد، است.

یک نکته درباره «فرم کانونی» : مفهوم فرم کانونی، مفهومی محوری برای بسیاری از شاخه‌های ریاضی و علوم مرتبط است. آن‌را می‌توان بدین صورت تعریف کرد: یک مجموعه از اشیاء (مثلاً پرسش‌ها) و یک مفهوم هم ارزی بین این اشیاء (مثلاً مفهومی که دو پرسش q_1 و q_2 معادلند اگر و تنها اگر ضمانت شود که نتیجه اجرای آن‌ها یکسان خواهند بود)، زیر مجموعه C از مجموعه Q را یک مجموعه از فرم کانونی هم ارز گوییم اگر و تنها اگر هرشی q از مجموعه Q درست معادل یک شئ c در مجموعه C باشد. به شئ c فرم کانونی شئ q گفته می‌شود. تمام خصوصیات «جالبی» که به q اعمال می‌شود نیز به فرم کانونی c اعمال می‌شود، بنابراین بررسی مجموعه

کوچک C بجای مجموعه بزرگ Q، برای اثبات نتایج «جالب» گوناگون، کافی خواهد بود.

برای بازگشت به رشته اصلی بحث، برای تبدیل فرم خروجی مرحله اول به فرمی هم ارز اما کاراتر، لازم است بهینه ساز از **قوانین** یا **قواعد تبدیل**[1] مشخصی استفاده نماید. مثلاً عبارت:

(A JOIN B) WHERE restriction on A

می‌تواند به عبارتی هم ارز اما کاراتر تبدیل شود:

(A WHERE restriction on A) JOIN B

قواعد بیشتری در بخش ٤ همین فصل بحث خواهد شد.

انتخاب رویه کاندید سطح پایین.

پس از آن که نمایش داخلی پرسش (معمولاً پرسش به شکل جبر رابطه‌ای) به فرم مناسب‌تر تبدیل شد، بهینه ساز باید تصمیم بگیرد که چطور پرسش تبدیل شده را اجرا کند. در این مرحله مواردی از قبیل وجود شاخص‌ها، دیگر مسیرهای دسترسی فیزیکی توزیع مقادیر داده، خوشه بندی فیزیکی داده‌های ذخیره شده و غیره در نظر گرفته می‌شود. (توجه داشته باشید که این موارد را در مراحل ۱ و ۲ در نظر نگرفتیم).

استراتژی اصلی بدین صورت است که عبارت پرسش را به عنوان یک سری از عملیات سطح پایین[2] که وابستگی متقابلی دارند، در نظر می‌گیریم. مثالی برای وابستگی متقابل می‌تواند بدین صورت باشد: کدی که یک پرتو را انجام می‌دهد معمولاً نیازمند این است که تاپل های ورودی به ترتیب خاصی ذخیره شوند تا حذف تاپل های تکراری میسر گردد، یعنی عملیات بلافاصله قبلی در این مجموعه باید تاپل های خروجی را به همان ترتیب تولید کند.

---

[1] Transformation rules

[2] سطح یک مفهوم نسبی است، منظور از سطح پایین در اینجا عملگرهای جبر رابطه ای از قبیل پیوندف گزینش، گروه بندی و غیره است که بیشترسطح بالا در نظر گرفته می شوند.

حال برای هر عملیات سطح پایین (و همچنین احتمالاً برای هر ترکیب متداول چندین عملیات)، بهینه ساز یک مجموعه رویه‌های پیاده سازی[1] از پیش تعریف شده، در دست خواهد داشت. برای مثال، یک مجموعه از رویه‌های برای پیاده سازی عمل گزینش وجود دارد: یک رویه برای حالتی که گزینش، مقایسه برابری انجام می‌دهد. یکی برای جایی که صفتی از رابطه که گزینش روی آن انجام می‌گیرد، شاخص گذاری شده است، و دیگری اینکه صفت درهم سازی شده است و غیره. مثال‌هایی از این رویه‌ها در بخش 18.7 همین فصل آمده است.

هر روال یک **فرمول هزینه‌ای**[2] (دارای پارامتر) مربوط به خود دارد که هزینه اجرای آن را مشخص می‌کند- معمولاً بر حسب I/O های دیسک است، گر چه در بعضی سیستم‌ها، بهره بری CPU و عوامل دیگری را نیز در نظر می‌گیرند. این فرمول‌های هزینه در مرحله ٤ مورد استفاده قرار می‌گیرند.

بنابراین، با استفاده از اطلاعات موجود در کاتالوگ سیستم، که نشان دهنده وضعیت فعلی سیستم است (شاخص‌های موجود، کاردینالتی های جاری) تعداد تاپل های موجود در رابطه‌ها در همان زمان مشخص، و غیره به همراه اطلاعات وابستگی متقابل مقتضی، بهینه ساز یک یا چند رویه کاندید را برای پیاده سازی هر یک از عملیات سطح پایین در عبارت پرسش، انتخاب می‌کند. این فرایند را گاهی انتخاب مسیر دسترسی[3] می‌نامند. نکته: در واقع منبع [33] از اصطلاح انتخاب مسیر دسترسی برای پوشش مراحل ۳ و ٤ استفاده کرده نه فقط در مرحله ۳. حقیقتاً تفکیک دقیق این دو در عمل دشوار است- مرحله ۳ کم و بیش در مرحله ٤ جریان دارد.

---

[1] Implementation procedures
[2] Cost formula
[3] Access path selection

مرحله ٤: ایجاد طرح اجرا و انتخاب کم هزینه‌ترین طرح.

مرحله آخر در فرایند بهینه سازی شامل، ایجاد یک مجموعه از طرح‌های اجرای پرسش[1] کاندید است که بعد از آن بهترین طرح اجرا (کم هزینه‌ترین) انتخاب می‌شود. هر طرح اجرا پرسش، به وسیله ترکیب مجموعه‌ای از رویه‌های پیاده سازی کاندید ساخته می‌شود. برای هر عملیات سطح پایین در پرسش یکی از این رویه‌ها وجود دارد. توجه داشته باشد که برای هر پرسش معمولاً چندین طرح اجرای پرسش وجود دارد. در حقیقت، در عمل تولید تمام طرح‌های اجرای پرسش به دلیل تعداد زیاد آن‌ها ایده خوبی نیست، چون انتخاب کم هزینه‌ترین آن‌ها گران تمام می‌شود. برخی تکنیک‌های مکاشفه‌ای[2] برای ضمانت نگه‌داشتن مجموعه در داخل کران‌های معقول، اگر اساسی نباشد، بسیار مناسب است. "نگهداری این مجموعه داخل کران‌ها" معمولاً به کاهش فضای حالت می انجامد، زیرا می‌توان به عنوان کاهش بازه("فضا") مواردی دانست که باید توسط بهینه ساز به نسبت‌های قابل مدیریت بررسی("جستجو") شود.

انتخاب کم هزینه‌ترین طرح اجرای پرسش، مستلزم روشی برای تخمین هزینه برای هر طرح اجرا است. اساساً، هزینه یک طرح اجرا برابر با مجموع هزینه‌های رویه‌های منفردی است که آن طرح اجرا را می‌سازند. بنابراین کاری که بهینه ساز باید انجام دهد ارزیابی فرمول هزینه برای هر یک از آن رویه‌های منفرد است. مسئله اینجا است که این فرمول‌های هزینه به اندازه رابطه‌ای که باید پردازش شوند بستگی دارد. از آن جایی که همه پرسش‌ها بجز ساده‌ترین آن‌ها نیازمند تولید نتیجه بینابینی در حین اجرا می‌باشند (حداقل از نظر مفهومی)، بنابراین بهینه ساز برای آن که فرمول هزینه را ارزیابی کند باید بتواند اندازه این نتایج بینابینی را تخمین بزند. متأسفانه، این اندازه‌ها به مقادیر واقعی داده‌ها بستگی دارند و در نتیجه تخمین دقیق هزینه می‌تواند کار مشکلی باشد.

---

[1] Query plans
[2] heuristic

۱۸,٤ تبدیل عبارت.

در این بخش برخی قواعد تبدیل را که ممکن است برای مرحله دوم از فرایند بهینه سازی مفید باشد را شرح می‌دهیم. ارائه مثال‌ها برای تشریح قواعد و علت مفید بودنشان به عنوان تمرین در نظر گرفته شده است.

البته، شما باید درک کنید که برای تبدیل یک عبارت مفروض، به‌کارگیری یک قاعده ممکن است عبارتی را ایجاد نماید که آن عبارت می‌تواند طبق قاعده دیگر، به عبارت دیگری تبدیل شود. برای مثال بعید است که اصل پرسش به نحوی باشد که به دو پرتو پی در پی نیاز داشته باشد. اما چنین عبارتی ممکن است به عنوان نتیجه بینابینی از اعمال دیگر تبدیلات معین بدست آید. (یک مورد مهم و مناسب، توسط پردازش دید ارائه شد، برای مثال این پرسش را در نظر بگیرید «تمام شهرها در دید V را بدست آور» که دید V به صورت یک پرتو از رابطه عرضه کنندگان بر روی صفات S# و CITY است). به عبارت دیگر، بهینه ساز از عبارت اصلی (اصل پرسش) شروع می‌کنند، و مکرراً قواعد تبدیل را اعمال می‌کند تا سرانجام به عبارتی برسد که تشخیص دهد برای پرسش مورد نظر «بهینه» است.

پرتوها و گزینش‌ها.

در اینجا برخی از قواعد تبدیل که تنها شامل پرتو و گزینش هستند را بررسی می‌کنیم.

۱. دنباله‌ای از گزینش‌ها بر روی یک رابطه را می‌توان به یک گزینش ("AND شده")(یک گزینش که در قسمت شرط آن چندین شرط با هم AND شده‌اند) تبدیل کرد. برای مثال، عبارت:

$(A \text{ WHERE } p_1) \text{ WHERE } p_2$

معادل عبارت زیر است:

$A \text{ WHERE } p_1 \text{ AND } p_2$

این تبدیل بهتر است زیرا فرمول اصلی مستلزم دو بار گـذر از رابطـه A اسـت در صورتی که نسخه تبدیل شده تنها نیاز به یک گذر دارد.

۲. در دنباله‌ای از پرتوها روی یک رابطه، تمـام آن‌هـا را بجـز آخـری را می‌توان نادیده گرفت. برای مثال، عبارت:

$(A\ \{acl_1\})\ \{acl_2\}$

(که acl_1 و acl_2 لیست‌هایی از اسامی صفات است) این عبارت معادل:

$A\ \{acl_2\}$

البته، در عبارت اصلی acl_2 باید یک زیر مجموعه از acl_1 باشد.

۳. یک گزینش و سپس یک پرتو مـی‌توانـد بـه یـک پرتـو سـپس یـک گزینش تبدیل شود. برای مثال عبارت:

$(A\ \{ac_1\})$ WHERE p

معادل عبارت زیر است:

(A WHERE) p $\{ac_1\}$

عموماً به علت آنکـه گـزینش انـدازه ورودی را بیشـتر از عمـل پرتـو کـاهش می‌دهد و از این رو میزان داده‌ای که به منظور حذف تکراری‌ها نیاز به ذخیره شدن دارد کاهش می‌یابد، انجام گزینش‌ها قبل از پرتوها ایده خوبی است.

توزیع پذیری.

قاعده تبدیلی که در مثال ۱۸,۲ بکار گرفته شد (تبدیل یک پیونـد سـپس یـک گزینش به یک گزینش سپس پیوند)، قاعده توزیع پذیری نامیده می‌شود. بـه طـور کلـی یک عملگر یکانی[1] f بر روی عملگر دوتایی[2] O توزیع پذیر است اگر و تنها اگر بـرای تمام A و B داشته باشیم:

$f(A\ O\ B) \equiv f(A)\ O f(B).$

برای مثال در محاسبات عمومی SQRT (جذر یا ریشـه دوم) بـر روی ضـرب توزیع پذیر است، زیرا برای تمام A و B داریم:

---

[1] Monadic operator
[2] Dyadic operator

SQRT (A * B) ≡ SQRT (A) * SQRT (B)

بنابراین بهینه ساز عبارت محاسباتی، هنگام تبدیل عبارت محاسباتی همیشه می‌توان هر یک از این عبارات را با دیگری جایگزین کند. به عنوان مثال نقض، SQRT (جذر) بر روی جمع توزیع پذیر نیست زیرا در حالت کلی، ریشه دوم A + B با مجموع ریشه‌های دوم A و B برابر نیست.

در جبر رابطه‌ای، گزینش بر روی اجتماع، اشتراک و تفاضل توزیع پذیر است. همچنین گزینش بر روی پیوند توزیع پذیر است اگر و تنها اگر شرط گزینش، در پیچیده‌ترین حالت، شامل دو شرط گزینش ساده باشد که با یکدیگر AND شده‌اند و این گزینش‌ها بر روی هر یک از عمل وندها (رابطه‌هایی که در پیوند شرکت کرده‌اند) توزیع می‌شوند. در مورد مثال ۱۸,۲ این شرط برآورده گشته و می‌توان گزینش را بر روی پیوند توزیع نمود. چون شرط گزینش تنها یک شرط ساده بود که تنها بر روی یکی از عمل وندهای پیوند (یکی از رابطه‌های دخیل در عمل پیوند) انجام می‌شد. بنابراین می‌توانستیم با استفاده از قاعده توزیع پذیری بجای عبارت اصلی یک عبارت کاراتر داشته باشیم که نتیجه اساسی آن این بود که «گزینش زودتر انجام شود». انجام زودتر گزینش ایده خوبی است زیرا باعث می‌گردد تعداد تاپل هایی که باید در عملیات بعدی پردازش پرسش، شرکت کنند، کاهش یابد و احتمالاً تعداد تاپل های خروجی عملیات بعدی نیز کاهش پیدا کند.

در اینجا حالت‌ها دیگری از قاعده توزیع پذیری که مربوط به عملگر پرتو هست را بحث می‌کنیم. پرتو بر روی اجتماع و اشتراک توزیع پذیر است اما بر روی تفاضل توزیع پذیر نیست.

(A UNION B) $\{ac_1\}$ ≡ A $\{ac_l\}$ UNION B $\{acl_1\}$
(A INTERSECT B) ≡ A $\{ac_1\}$ INTERSECT B $\{acl_1\}$
$\{ac_1\}$

البته A و B باید همنوع باشند.

همچنین پرتو تا هنگامی که تمام صفات پیوند را حفظ کند، بر روی پیوند توزیع پذیر خواهد بود. بنابراین:

(A JOIN B) {acl} ≡ (A $\{acl_1\}$) JOIN (B $\{acl_2\}$}).

در اینجا acl_1 برابر است با اجتماع صفات پیوند و آن صفاتی از acl که تنها در A ظاهر می‌شوند و acl_2 برابر است با اجتماع صفات پیوند و آن صفاتی از acl که تنها در B ظاهر می‌شوند.

این قواعد (قوانین) برای «اجرای زود هنگام پرتو» مورد استفاده قرار می‌گیرد و به همان دلایلی که برای گزینش مطرح شد می‌تواند ایده خوبی باشد.

جا بجا پذیری و شرکت پذیری.

دو قاعده کلی مهم‌تر، جا بجا پذیری[1] و شرکت پذیری[2] هستند. ابتدا جا بجا پذیری، عملگر دوتایی O را جا بجا پذیر گوییم اگر و تنها اگر برای هر A و B داشته باشیم:

$$A\ O\ B \equiv B\ O\ A$$

برای مثال در حساب، ضرب و جمع جا بجا پذیرند اما تقسیم و منها جا بجا پذیر نیستند. در جبر رابطه‌ای، اجتماع، اشتراک و پیوند جا بجا پذیرند اما تفاضل و تقسیم جا بجا پذیر نیستند. بنابراین، برای مثال، اگر در یک پرسش یک پیوند بین دو رابطه A و B وجود داشته باشد، بنا به قاعده جا بجا پذیری از لحاظ منطقی فرقی نمی‌کند کدام یک از رابطه‌های A و B در عمل پیوند رابطه «بیرونی» و کدام یک رابطه «درونی» باشند. بنابراین سیستم در تصمیم گیری آزاد است مثلاً رابطه کوچک‌تر در محاسبه عمل پیوند رابطه «بیرونی» باشد (بخش ۱۸,۷ را ببینید).

در مورد شرکت پذیری: عملگر دوتایی O را شرکت پذیر گوییم اگر و تنها اگر برای هر A و B داشته باشیم:

$$A\ O\ (B\ O\ C) \equiv (A\ O\ B)\ O\ C$$

در حساب، ضرب و جمع شرکت پذیر هستند، اما تقسیم و منها شرکت پذیر نیستند. در جبر رابطه‌ای، اجتماع، اشتراک و پیوند همگی شرکت پذیرند اما تفاضل و

---

[1] commutativity
[2] associativity

تقسیم شرکت پذیر نیستند. بنابراین برای مثال، اگر در پرسشی سه رابطه A و B و C با یکدیگر پیوند شده باشند، بر طبق قاعده‌های شرکت پذیری و جا بجا پذیری، از لحاظ منطقی فرقی نمی‌کند که این پیوندها به چه ترتیبی انجام شوند. بنابراین سیستم در تصمیم گیری در مورد اینکه کدام ترتیب کاراتر است، آزاد است.

همانی[1] و جذب[2]

دیگر قاعده مهم قاعده همانی (همانی) است. عملگر دوتایی O را همانی گوییم اگر و تنها اگر برای هر A و B داشته باشیم:

A O A ≡ A

همان‌گونه که انتظار می‌رود، خاصیت همانی نیز می‌تواند در تبدیل عبارت مفید باشد. در جبر رابطه‌ای، اجتماع، اشتراک و پیوند همگی همانی هستند اما تقسیم و تفاضل همانی نیستند.

همچنین اجتماع و اشتراک در قواعد مفید زیر نیز صدق می‌کنند:

A UNION (A INTERSECT B) ≡ A
A INTERSECT (A UNION B) ≡ A

عبارات محاسباتی.

قواعد تبدیل تنها برای عبارات رابطه‌ای نیست. همان‌طور که قبلاً گفتیم بعضی از تبدیلات برای عبارات محاسباتی نیز اعتبار دارند. برای مثال عبارت:

A * B + A * C

می‌تواند بر اساس این حقیقت که «*» بر روی «+» توزیع پذیر است تبدیل به عبارت زیر گردد:

A * (B + C)

---

[1] idempotence
[2] absorption

یک بهینه ساز رابطه‌ای، لازم است اطلاعاتی در مورد این تبدیلات داشته باشد زیرا با چنین عباراتی در متن عملگرهایی چون گروه بندی (summarize) و بسط (Extend)، برخورد خواهد کرد.

ضمناً نکته اینکه این مثال اندکی شکل کلی‌تر توزیع پذیری را نشان می‌دهد. قبلاً توزیع پذیری را بر حسب یک عملگر یکانی که بر روی یک دوتایی توزیع می‌شد، تعریف کردیم اما در این حالت «*»و «+» هر دو عملگر های دوتایی هستند. به طور کلی، عملگر دوتایی را بر روی عملگر O توزیع پذیر می‌گوییم اگر و تنها اگر برای تمام A، B، C داشته باشیم:

A (B O C) ≡ (A) B O (A) C

در یک مثال حسابی می‌توان بجای «*» گذاشت و بجای O از + استفاده کرد.

عبارت منطقی (بولی).

فرض کنید که A و B صفاتی از دو رابطه مجزا هستند. پس عبارت :

A > B AND B > 3

روشن است که معادل عبارت زیر است (و بنابراین می‌توان آن‌را به این عبارت تبدیل نمود) :

A > B AND B > 3 AND A > 3

به خاطر اینکه عملگر مقایسه «<»، تراگذار[1] (رابطه غیر مستقیم) است، دو عبارت معادل هستند. توجه کنید که این تبدیل ارزشمند است، زیرا سیستم را قادر می‌سازد که قبل از انجام پیوند بزرگ‌تر–از که با اعمال "A > B" انجام می‌شود یک گزینش اضافی (بر روی A) انجام دهد. بر اساس نکته‌ای که قبلاً به آن رسیدیم انجام گزینش‌های زودرس به طور کلی ایده خوبی است؛ همچنین اگر سیستم مانند این مثل، بتواند گزینش‌های «زودرس» دیگری را استنتاج کند، ایده خوبی است. نکته : این تکنیک در چندین محصول تجاری پیاده سازی شده است برای مثال DB2 و Integers.

---

[1] transitive

در اینجا مثال دیگر آورده‌ایم، عبارت:

A > B OR (C = D AND E < F).

با توجه به این واقعیت که OR بر روی AND توزیع پذیر است می‌توان تبدیل به عبارت زیر گردد.

(A > B OR C = D) AND (A > B OR E < F).

این مثال یک قاعده کلی دیگری را نشان می‌دهد، هر عبارت منطقی می‌تواند به یک عبارت معادل که **فرم نرمال عطفی**[1] (CNF) نام دارد، تبدیل شود. یک عبارت CNF، عبارتی است به شکل:

C_1 AND C_2 AND … AND C_n

که هر یک از C_1، C_2، ...، C_n عبارات منطقی (عطف نامیده می‌شوند) هستند که فاقد عملگر AND می‌باشند. مزیت CNF در این است که عبارت CNF در صورتی دست است که **هر یک** از عطف‌ها **درست** باشند یا این عبارت در صورتی نادرست است که **یکی** از عطف‌ها **نادرست** باشد. از آن جایی که عملگر AND جابجاپذیر است (A AND B معادل B AND A است). بهینه ساز می‌تواند هر یک از عطف‌ها را به هر ترتیبی که دوست داشت ارزیابی کند. به خصوص می‌تواند عطف‌ها را به ترتیب آسانی به سختی مرتب کند و هر زمان که یکی از عطف‌ها نتیجه ارزیاب آن نادرست شد لازم نیست بقیه ارزیابی شوند و نتیجه کلی نادرست خواهد بود. همچنین در سیستم پردازش موازی، ممکن است بتوان هر یک از عطف‌ها را به طور موازی ارزیابی نمود. باز هم وقتی یکی از آن‌ها نادرست بود کل فرایند متوقف شده و نتیجه نادرست خواهد بود.

پیرو مطالب این گفتار و گفتارهای قبلی، بهینه ساز نه تنها لازم است از قواعد تبدیل و خصوصیات کلی مثل توزیع پذیری برای عملگرهای رابطه‌ای مانند پیوند آگاه باشد بلکه آن باید از وجود قواعد و خصوصیات عملگرهای مقایسه‌ای مثل «<»و

---

[1] Conjunctive normal form

عملگرهای منطقی مثل AND و OR و عملگرهای محاسباتی مثل «+» و غیره نیز آگاه باشد.

تبدیلات معنایی[1]

عبارت زیر را در نظر بگیرید:

(SP JOIN S) {P#}

این پیوند از نوع طبیعی بوده و کلید خارجی رابطه SP با کلید کاندید S تطبیق داده می‌شود (در پیوند طبیعی عملگر مقایسه، تساوی است). بر طبق این پیوند هر تاپل رابطه SP با تاپلی از رابطه S پیوند داده می‌شود و چون شرط پیوند بر روی کلید کاندید و کلید خارجی است و از قبل می‌دانیم که مقدار کلید خارجی در هر تاپل یا NULL است و یا یک مقدار کلید کاندید در رابطه مقابل، پس کاردینالتی نتیجه این پیوند برابر است با کاردینالتی رابطه SP و چون در این عبارت پرتو بر روی #P داریم پس نتیجه همان مقادیر #P در رابطه SP است به عبارت دیگر اصلاً نیاز به انجام پیوند نداریم! می‌توان این عبارت را به صورت ساده شده زیر نوشت:

SP {P#}

به هر حال دقت داشته باشید که این تبدیل تنها به جهت معنایی، معتبر است. به طور کلی، هر یک از عمل وندها (رابطه‌ها) در پیوند، دارای تاپل هایی هستند که همتایی در دیگری ندارد (و از این رو بعضی از تاپل ها سهمی در نتیجه کلی ندارند)، و بنابراین تبدیلاتی طبق آنچه که گفتیم معتبر نیستند. اما در این حالت، هر تاپل SP همتایی در رابطه S دارد، زیرا قانون C2 یا جامعیت ارجاعی می‌گوید که هر محموله باید یک عرضه کننده داشته باشد، در نتیجه این تبدیل معتبر است.

تبدیلی که تنها به خاطر یک محدودیت جامعیتی مشخص معتبر باشد **تبدیل معنایی**[2] نامیده می‌شود و بهینه سازی حاصل از آن **بهینه سازی معنایی**[3] نامیده می‌شود.

---

[1] Semantic transformations
[2] Semantic transformation
[3] Semantic optimization

بهینه سازی معنایی را می‌توان به عنوان فرآیند تبدیل پرسشی خاص بـه پرسشـی دیگـر که از نظر کیفیت متفاوت است، تعریف کرد. پرسشی که علیرغم تبدیل، تضمین می‌کنـد که همان نتیجه پرسش اصلی را ارائه دهد. زیـرا تضـمین مـی‌شـود کـه داده‌هـا در یـک محدودیت جامعیتی مشخص صدق کنند.

درک این نکته مهم است که، در اصل، هیچ‌گونه محدودیت جامعیتی نمی‌تواند در بهینه سازی معنایی بکار رود (این تکنیک محدود به محـدودیت‌هـای ارجـاعی مثـل اینکه در این مثال دیدیم، نیست). برای مثال فرض کنید پایگـاه داده عرضـه کننـدگان و قطعات از این محدودیت پیروی می‌کنند، «تمام قطعات قرمز رنگ در لنـدن انبـار شـده است». حال پرسش زیر را در نظر بگیرید:

عرضه کنندگانی را مشخص کنید که تنها قطعات قرمز رنگ را عرضه می‌کننـد و در همان شهری که حداقل یکی از قطعات را عرضه می‌کنند، مستقر هستند.

بی شک این پرسش پیچیده اسـت! گرچـه بـا توجـه بـه محـدودیت (قاعـده) جامعیت، می‌توان به شکل ساده‌تری تبدیل شود:

عرضه کنندگان لندنی را بدست آور که تنها قطعه قرمز رنگ عرضه می‌کنند.

تا آنجایی که نویسـنده خبـر دارد، تعـداد کمـی از محصـولات (سیسـتم‌هـای مدیریت پایگاه داده) موجود از بهینه سازی معنایی استفاده می‌کننـد. هـر چنـد در عمل این گونه بهینه سازی‌ها، به احتمال زیـاد بهبودهـای بیشـتری نسـبت تکنیک‌های بهینه سازی امروزی، خواهند داشت.

۱۸,۵ آمارهای پایگاهی.

مراحل ۳ و ٤ فرایند کلی بهینه سازی، مراحل «انتخاب مسیر دسـت‌یـابی» کـه در آن‌هـا رویه‌های سطح پایین برای هر عملگر جبر رابطه انتخاب و سـپس طـرح اجـرای بهینـه انتخاب می‌گردد، لازم است که پایگاه داده از آمارهای پایگاهی که در کاتـالوگ سیسـتم ذخیره شده‌اند، استفاده نمایند (برای اطلاعات بیشتر از چگونگی استفاده از ایـن آمارهـا به بخش 18.7 مراجعه کنید). این آمارها خیلی دقیق نیستند چون لازمه چنین دقتی، آن

است که این آمارها پس از هر عمل که پایگاه داده را تغییرمی دهد، اصلاح گردند. معمولاً آمارهای پایگاهی در زمانی که بار کاری سیستم کم باشند، به هنگام می‌شوند. در این بخش برخی از آمارهای اصلی نگهداری شده توسط دو محصول تجاری یعنی DB_2 و Ingers را با کمی توضیح خواهیم گفت. ابتدا برخی آمارهای اصلی که در DB_2 نگهداری می‌شوند عبارتند از:

- برای هر جدول پایه:
  - کاردینالتی.
  - تعداد صفحات اشغال‌شده توسط این جدول.
  - کسری از «فضای جدول» که توسط این جدول اشغال‌شده است.
- برای هر ستون هر جدول پایه.
  - تعداد مقادیر متمایز در این ستون.
  - دومین بیشترین مقدار در این ستون.
  - دومین کوچک‌ترین مقدار در این ستون.
  - ۱۰ مقداری که بیشترین فراوانی را در هستند به همراه تعداد دفعات تکرار آن‌ها، این اطلاعات تنها در مورد ستون‌های شاخص گذاری شده است.
- برای هر شاخص.
  - نشانه‌ای مبنی بر اینکه شاخص یک «شاخص خوشه ساز» است یا خیر (یعنی اینکه: شاخص به خوشه‌ای که حاوی داده‌هایی که از لحاظ منطقی به یکدیگر مرتبط هستند) رکوردهای مرتبط اشاره دارد [•] .

•شاخص می تواند متراکم باشد یعنی هر شاخص تنها به یک رکورد جدول اشاره کند یا غیر متراکم یا خوشه ساز باشد که هر شاخص به یک خوشه رکورد های مرتبط اشاره دارد.

- اگر شاخص خوشه ساز است، کسری از جدول شاخص گذاری شده که هنوز در دنباله خوشه سازی وجود دارد.
- تعداد صفحات برگ در این شاخص.
- تعداد سطح در این شاخص.

نکته: این آمارها بلافاصله به هنگام نمی‌شوند چون سربار این روش زیاد است. این آمارها به طور انتخابی به وسیله یک از ابزار سیستم بنام RUNSTATE به هنگام می‌شوند که بنا به درخواست مدیر پایگاه داده (DBA) اجرا می‌شود چنین ابزاری در سیستم مدیریت پایگاه داده Ingres وجود دارد که OPTIMIZDB نامیده می‌شود.

در اینجا برخی از آمارهای پایگاهی Ingres عنوان شده است. توجه: در Ingres، شاخص به عنوان نوع خاصی از جدول ذخیره شده است، بنابراین آمارهایی که برای ستون‌ها و جداول پایه نگهداری می‌شود برای شاخص‌ها نیز نگهداری می‌شود:

- برای هر جدول پایه:
  - کاردینالتی.
  - تعداد صفحات اصلی برای این جدول.
  - تعداد صفحات سرریز برای این جدول.
- برای هر ستون از جدول پایه:
  - تعداد مقادیر متمایز در این ستون.
  - بیشترین مقدار، کم‌ترین مقدار و میانگین مقادیر برای این ستون.
  - مقادیر واقعی (مقادیر مجزا) و تعداد دفعات تکرار آن‌ها در این ستون.

۱۸,٦ استراتژی تقسیم و حل.

همان‌گونه که در انتهای بخش ۱۸,٤ متذکر شدیم، عبارات رابطه‌ای به صورت بازگشتی بر حسب زیر عبارات بیان می‌شوند، و این واقعیت اجازه می‌دهد که بهینه ساز

استراتژی‌های تقسیم و حل گوناگونی را اتخاذ کند. توجه داشته باشید که این استراتژی‌ها احتمالاً در محیط‌های پردازش موازی (خصوصاً یک سیستم توزیع شده) که چندین بخش جداگانه یک پرسش می‌تواند بر روی چندین پردازشگر مختلف اجرا شود، جذابیت ویژه‌ای دارد. در این بخش یکی از این استراتژی‌ها را که **تجزیه پرسش**[1] نام دارد و اولین بار توسط نمونه اولیه Ingres بکار گرفته شد را بررسی می‌کنیم.

ایده اصلی تجزیه پرسش به این صورت است که یک پرسش که در آن چندین حیطه عمل (رابطه) دخالت دارد به پرسش‌هایی کوچک‌تر که (معمولاً) در هر کدام یک یا دو حیطه عمل (رابطه) دخالت دارد، شکسته می‌شود. این کار با انجام **جداسازی**[2] و **جایگزینی تاپل**[3] برای بدست آوردن تجزیه مطلوب صورت می‌پذیرد.

- جداسازی، فرایند حذف یک قسمت (جزء) از پرسش است که با بقیه پرسش تنها یک حیطه (رابطه) مشترک دارد.
- جایگزینی تاپل، فرایند جایگزین کردن یکی از حیطه‌ها (رابطه‌ها) با یک تاپل در هر بار است.

تا زمانی که انتخاب داشته باشیم، جداسازی نسبت به جایگزینی تاپل ارجحیت دارد. تا آن جا که ممکن است پرسش، از طریق تکنیک جداسازی، به مجموعه‌ای از پرسش‌های کوچک‌تر تجزیه می‌شود تا سرانجام دیگر نتوان با استفاده از این تکنیک پرسش را تجزیه نمود. پس از این باید جایگزینی تاپل را بکار برد.

مثال را از منبع [34] ارائه می‌دهیم. پرسش عبارت است «اسامی عرضه کنندگان لندنی که برخی قطعات قرمز رنگ را که وزنشان کمتر از ۲۵ پوند است را به بیش از ۲۰۰ تهیه می‌کنند» در اینجا این پرسش به زبان QUEL بیان شده است («پرسش Q_0»):

Q_0: RETERIEVE (S.SNAME) WHERE S.CITY = "London"

---

[1] Query decomposition
[2] detachment
[3] Tuple substitution

AND S.S# = SP.S#
AND SP.QTY > 200
AND SP.P# = P.P#
AND P.COLOR = "Red"
AND P.WEIGHT < 25.0

حیطه‌های عمل در اینجا عبارتند از S، Pو SP که محدوده هر کدام همان رابطه همنام می‌باشد.

حال اگر این پرسش را بررسی کنیم، از دو مقایسه آخر پی می‌بریم که آن‌ها تنها قطعاتی را که قرمز رنگ بوده و وزنشان کمتر از ۲۵ پوند هست را گزینش می‌کند. بنابراین می‌توان «پرسشی یک–متغیره[1]» (در واقع یک پرتو از یک گزینش بر روی یک رابطه) که تنها شامل متغیر (حیطه عمل) P است را از پرسش Q_0 جدا کنیم:

D1: RETRIEVE INTO P' (P.P#) WHERE P.COLOR = "Red"
AND P.WEIGHT < 25.0

این پرسش یک–متغیره (تک حیطه) قابل جدا سازی است، زیرا آن با بقیه پرسش تنها یک متغیر (خود P) یا بهتر است بگوییم یک حیطه عمل مشترک دارد. چون که با بقیه پرسش اصلی از طریق صفت P# پیوند دارد (با این مقایسه SP.P# = P.P#)، صفت P# باید در «تاپل شالوده[2]» نسخه جدا شده ظاهر گردد (در واقع صفت P# در رابطه حاصل از اجرا پرسش جدا شده از پرسش اصلی وجود داشته باشد) یعنی: پرسش جدا شده باید شماره قطعات قرمز رنگ که وزنشان کمتر از ۲۵ پوند است را بازیابی کند. این پرسش را تحت عنوان D_1 ذخیره می‌کنیم که نتیجه اجرای آن در رابطه موقت P' ریخته می‌شود. در آخر در نسخه کاهش یافته پرسش Q_0، ارجاع به P را با ارجاع به P'(نتیجه موفقت پرسش D_0) جایگزین می‌کنیم. این نسخه جدید کاهش یافته را پرسش Q_1 می‌نامیم:

Q_1: RETERIEVE (S.SNAME) WHERE S.CITY = "London"
AND S.S# = SP.S#

---

[1] One-variable query
[2] Proto tuple

AND SP.QTY > 200
AND SP.P# = P'.P#

حال، فرایند جداسازی مشابهی را روی پرسش Q_1 انجام می‌دهیم تـا پرسشـی یک متغیره (یک حیطه عمل) را که تنها شامل متغیر SP(تنها بر روی رابطه SP پـرس و جو می‌کند) را جدا کنیم و نسخه اصلاح شده دیگری را که پرسش D_2 می‌نـامیم، ایجـاد کنیم:

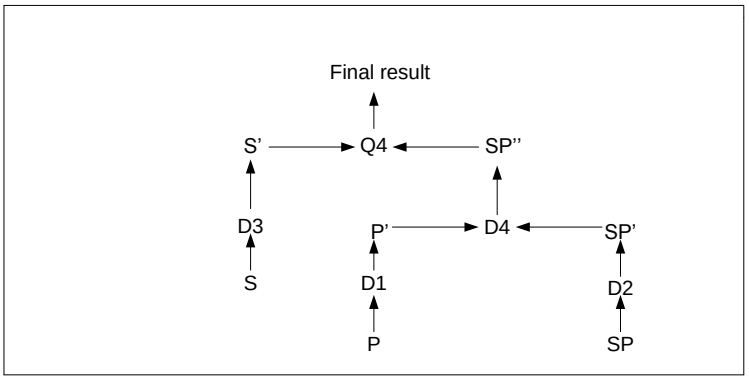
D_2: RETREIVE INTO SP' (SP.S#, SP.P#) WHERE SP.QTY > 200
Q_2: RETERIEVE (S.SNAME) WHERE S.CITY = "London"
AND S.S# = SP'.S#
AND SP'.P# = P'.P#

سپس، پرسش یک متغیر شامل پرسش یک متغیر شامل S را جدا می‌کنیم:

D_3 : RETREIVE INTO S' (S.S#,S.SNAME) WHERE S.CITY = "London"
Q_3 : RETERIEVE (S'.SNAME) WHERE S'.S# = SP'.S#
AND SP'.P# = P'.P#

سرانجام، پرسش دو متغیره شامل 'SP و 'P را جدا می‌کنیم:

D_4 : RETREIVE INTO SP" (SP'.S#) WHERE SP'.P# = P'.P#
Q_4 : RETERIEVE (S'.SNAME) WHERE S'.S# = SP".S#



شکل ۱۸،۳ درخت تجزیه برای پرسش Q_0

بنابراین، پرسش اصلی Q_0 به سه پرسش یک متغیره (تنها پرس و جو از یک رابطه) D_1، D_2، D_3

(هر کدام شامل یک گزینش و یک پرتو) و دو پرسش دو متغیره D_4 و Q_4 (هر یک شامل یک پرتو از یک پیوند بین دو رابطه هستند) تجزیه می‌شود. چگونگی ایجاد نتیجه نهایی در شکل ۳ نشان داده شده است این شکل می‌تواند به این صورت خوانده شود:

- پرسش‌های D_1، D_2 و D_3 به ترتیب رابطه‌های P، SP و S را به عنوان ورودی می‌پذیرند و رابطه‌های موقتی به نام‌های 'P'، 'SP' و 'S' را به عنوان خروجی تولید می‌کنند.
- پرسش D_4، رابطه‌های موقت 'P' و 'SP' را به عنوان ورودی گرفته و رابطه موقت "SP" را تولید می‌کند.
- سرانجام، پرسش Q_4 رابطه‌های 'S' و "SP" را به عنوان ورودی گرفته و نتیجه مورد نظر کلی را تولید می‌کند.

مشاهده می‌کنید که پرسش‌های D_1، D_2 و D_3 به طور کامل از یکدیگر مستقل بوده و می‌توانند به هر ترتیبی اجرا شوند (احتمالاً حتی به صورت موازی)، همچنین پس از اینکه پرسش‌های D_1 و D_2 اجرا شدند، پرسش‌های D_3 و D_4 می‌توانند به هر ترتیبی اجرا شوند. اما پرسش‌های D_4 و Q_4 نمی‌توانند بیش از این تجزیه شده و باید با

جایگزین نمودن تاپل (واقعاً به وسیله جستجو جامع، جستجو شاخص، جستجو درهم ساز) جا یاب. برای مثال پرسش Q_4 را در نظر بگیرید. با داده‌های نمونه‌ای‌مان، شماره عرضه کنندگانی که قطعه‌های قرمز رنگ کمتر از ۲۵ پوند را عرضه می‌کنند یعنی رابطه موقت "SP برابر با مجموعه $\{s_1,s_2,s_3\}$ خواهد بود. هر یک از این سه مقدار، به نوبت برای #SP."S مقدار دهی شده و بنابراین نتیجه ارزیابی (اجرای) پرسش Q_4 به گونه خواهد بود که گویی به صورت زیر نوشته شده است:

```
RETREIVE (S'.SNAME) WHERE S'.S# = "S1"
                      OR S'.S# = "S2"
                      OR S'.S# = "S4"
```

در مرجع [34] الگوریتم‌هایی برای شکستن پرسش اصلی به پرسش‌های کوچک‌تر و انتخاب متغیرها برای جایگزین کردن تاپل ها آمده است. این منبع شامل روش‌های هیوریستیک (مکاشفه‌ای) برآورد هزینه جهت انتخاب برای جایگزینی است. Ingers معمولاً (اما نه همیشه) رابطه‌ای را برای جایگزینی انتخاب می‌کند که کم‌ترین کاردینالتی را داشته باشد. اهداف اصلی فرایند بهینه سازی اجتناب از ضرب دکارتی و حداقل کردن تعداد تاپل هایی که در هر مرحله باید بررسی شود.

۱۸,۷ پیاده سازی عملگرهای جبر رابطه‌ای.

اکنون مختصر توضیحی از برخی روش‌های ساده، برای پیاده سازی برخی از عملگرهای جبر رابطه‌ای، به ویژه پیوند را ارائه می‌دهیم. علت اصلی بیان این موضوع رفع ابهاماتی است که ممکن است هنوز در مورد فرایند بهینه سازی وجود داشته باشد. روش‌هایی که مورد بحث قرار می‌گیرد متناظر «رویه‌های سطح- پایین» که در بخش ۱۸,۳ گفته شده خواهد بود.

برای سادگی فرض می‌شود که تاپل ها و رابطه‌ها به طور فیزیکی ذخیره شده‌اند. عملگرهایی که در نظر می‌گیریم پرتو، پیوند و گروه بندی است- "summarize" (گروه بندی) شامل دو حالت زیر است.

۱. عمل وند)PER هیچ صفتی را مشخص نمی‌کند) "PER TABLE_DEE".

۲. عمل وند PER حداقل یک صفت را مشخص می‌کند.

حالت اول آسان است، اساساً، شامل پیمایش کل رابطه‌ای است که گروه بندی در آن صورت می‌پذیرد. به استثنای اینکه اگر صفتی که تجمیع روی آن صورت می‌پذیرد، شاخص گذاری شده باشد، در این صورت ممکن است بدون داشتن دسترسی به خود رابطه، نتیجه مستقیماً از شاخص بدست بیاید. برای مثال عبارت زیر را در نظر بگیرید:

SUMMARIZE SP ADD SUM (QTY) AS TQ

با فرض وجود شاخص بر روی صفت QTY، بدون دسترسی به رابطه SP(محموله) می‌توان از روی پیمایش شاخص عبارت مذکور را ارزیابی نمود. به طور مشابه برای COUNT و AVG نیز می‌توان از شاخص استفاده نمود (البته برای تابع COUNT می‌توان از هر شاخصی استفاده نمود). برای تابع MIN و MAX با یک‌بار دسترسی به شاخص می‌توان نتیجه را بدست آورد برای تابع MIN دسترسی به اولین مدخل[1] شاخص و برای MAX دسترسی به آخرین مدخل شاخص، کافی خواهد بود. البته با فرض اینکه شاخص برای صفت مورد نظر وجود داشته باشد.

در بقیه این بخش "summarize" (گروه بندی) را در حالت دوم بررسی می‌کنیم. در اینجا یک مثال از حالت دوم در نظر بگیرید.

SUMMARIZE SP PER P {P#} ADD SUM {QTY} AS TQTQTY

از نقطه نظر کاربر پرتو، پیوند و گروه بندی (حالت ۲) از یکدیگر کاملاً تفاوت دارند. اما از نقطه نظر پیاده سازی آن‌ها شباهت‌هایی با هم دارند، زیرا در هر مورد لازم است که سیستم تاپل ها را بر اساس مقادیر مشترک صفات مشخص شده‌ای گروه بندی کند. در مورد پرتو این گروه بندی اجازه می‌دهد که مقادیر تکرار شده را حذف کند، در مورد پیوند این گروه بندی اجازه می‌دهد که تاپل هایی که با هم تطبیق

---

[1] ENTRY

می‌شوند (با توجه به شرط پیوند) مشخص گردند، در مورد «گروه بندی» این گروه بندی اجازه می‌دهد که مقادیر تجمیع (بسته به نوع تابع) برای هر گروه مشخص شوند. تکنیک‌های متعددی برای انجام گروه بندی وجود دارد که عبارتند از:

۱. غیر هوشمند[1]
۲. جستجو شاخص.
۳. جستجو درهم ساز (جا یاب).
٤. ادغام.
۵. درهم سازی (جا یاب).
٦. ترکیب ۱ تا ۵

شکل‌های ٤ تا ۸ شبه کد[2] رویه‌هایی را از چگونگی پیاده سازی تکنیک‌ها مذکور تنها برای عملگر پیوند نشان می‌دهد (پرتو و گروه بندی به عنوان تمرین واگذار شده‌اند) نمادهای بکارگرفته شده در این شکل‌ها بدین صورت است: R و S رابطه‌هایی هستند که باید با یکدیگر پیوند داده شوند: C صفت مشترک (احتمالاً مرکب) بین این دو رابطه است. فرض می‌کنیم که هر بار می‌توان به تاپل های R و Sبا هر ترتیبی دست یافت و تاپل هایی که به آن‌ها دسترسی پیدا کردیم به ترتیبی به صورت R[n],…, R[2], R[1] و S[n],…, S[2], S[1] نشان می‌دهیم. عبارت R[i] * S[j] نشان دهنده تاپل حاصل از پیوند تاپل های R[i] و S[j] است. در آخر رابطه‌های R و S را به ترتیب، رابطه‌های بیرونی و درونی می‌نامیم (چون آن‌ها به ترتیب حلقه‌های بیرونی و درونی را کنترل می‌کنند).

غیر هوشمند

تکنیک غیر هوشمند را «حالت ساده» می‌نامند که در آن تمام ترکیب ممکن تاپل ها بررسی می‌شود (یعنی هر تاپل از رابطه R با هر تاپل از رابطه S بررسی می‌شود مانند

---

[1] Brute force
[2] pseudo code

شکل ۱۸,٤) : توجه تکنیک غیر هوشمند در بعضی متون با نام «حلقه‌های تو در تو» آمده است که این نام گمراه کننده است زیرا حلقه‌های تو در تو در تمام الگوریتم‌ها وجود دارد.

می‌خواهیم هزینه مربوط به رهیافت غیر هوشمندانه را بررسی کنیم. لازم به ذکر است که ما توجه خود را تنها معطوف به هزینه ورودی خروجی (I/O) می‌کنیم هرچند که در عمل دیگر هزینه‌ها (مثل CPU) نیز ممکن است مهم باشد.

```
do I := 1 to m ;                    /* outer loop */
   do j := 1 to n  ;                /* inner loop */
      If  R[i].C = S[j].C  then
      add joined tuple R[i] * S[j] to result ;
   end;
end;
```

شکل ۱۸,٤ : غیر هوشمند.

اول از همه، این رهیافت غیر هوشمندانه نیاز به خواندن مجموعاً m+(m*n) تاپل دارد. اما تعداد نوشتن تاپل ها چقدر است؟ یعنی کاردینالتی نتیجه پیوند چقدر خواهد بود؟ (در صورتی که نتیجه پیوند بخواهد بر روی دیسک ذخیره گردد تعداد نوشتن تاپل ها برابر کاردینالتی پیوند خواهد بود).

- در حالت پیوند یک به چند (که صفت مشترک کلید کاندید یکی از دو رابطه است و در دیگری آن صفت، کلید خارجی است) کاردینالتی نتیجه برابر است با کاردینالتی رابطه‌ای که کلید خارجی را دارد می‌باشد. پس اگر R یا S سمت کلید خارجی پیوند باشند پس کاردینالتی یا برابر با m است یا برابر n.
- اکنون حالت کلی‌تری از پیوند یعنی پیوند چند به چند را در نظر می‌گیریم. فرض کنید که dCR تعداد مقادیر متمایز صفت C در رابطه R باشد (دقت داشته باشید که صفت C در هر دو رابطه مشترک است و در واقع صفت پیوند نامیده می‌شود) همچنین به طور مشابه dCS را تعداد مقادیر متمایز صفت C در رابطه S در نظر بگیرید. اگر

فرض کنیم که مقادیر این صفت (C) به طور یکنواخت توزیع شده باشد (یعنی احتمال وقوع هر مقدار از صفت C در رابطه R با مقادیر دیگر یکسان باشد) پس برای هر تاپل از رابطه R به تعداد n/dCS تاپل وجود خواهد داشت که مقادیر صفت مشترک (یعنی C) در هر دو تاپل یکسان خواهد بود. بنابراین تعداد کل تاپل ها در پیوند (یعنی کاردینالتی نتیجه) برابر است با (m * n)/dCR. اگر dCR ≠ dCS باشد، این تخمین متفاوت خواهد بود) یعنی، مقادیری از C وجود دارد که در رابطه R وجود دارند اما در رابطه S وجود ندارند در این حالت باید از تخمین کوچک‌تر استفاده نمود.

همان‌طور که در بخش ۱۸,۲ گفته شد، در عمل تعداد صفحات I/O مهم است نه تعداد تاپل های I/O . بنابراین فرض کنید که تاپل های رابطه‌های R و S به ترتیب pR در یک صفحه و pS در یک صفحه ذخیره شده‌اند (به طوری که این دو رابطه به ترتیب m/pR صفحه و n/pS صفحه را پر کرده‌اند) آن گاه به روشنی می‌توان دید که رویه شکل ۱۸,٤ شامل (m/pR) + (m*n)/pS صفحه خواندن خواهد بود. به طور متناوب اگر نقش رابطه‌های R و S را در پیوند عوض کنیم تعداد صفحاتی که خوانده می‌شود برابر خواهد بود با: (n/pS) + (m*n)/pR

برای مثال، فرض کنید m =100، n = 10.000، pR = 1، pS = 10 است. آنگاه تخمین با استفاده از این دو فرمول به ترتیب به ۱۰۰,۱۰۰ و ۱,۰۰۱,۰۰۰ خواندن صفحه، خواهد بود. نتیجه گیری می‌شود که در تکنیک غیر هوشمند باید رابطه کوچک‌تر به عنوان رابطه بیرونی انتخاب شود (منظور از رابطه کوچک‌تر، رابطه‌ای که تعداد صفحات کمتر دارد).

این بحث کوتاه در مورد تکنیک غیر هوشمندانه را با این نکته به پایان می‌بریم که این تکنیک را به عنوان رویه بدترین حالت از آن استفاده می‌کنیم یعنی وقتی که رابطه S بر روی صفت پیوند (مشترک) C نه شاخص بندی شده و نه درهم ساز شده

است. آزمایشات انجام شده توسط Bitton و همکاران نشان می‌دهد که این فرض معتبر است، اما با شاخص گذاری و یا جدول درهم ساز (جا یاب) بر روی صفت پیوند C و سپس پردازش با استفاده از جستجو شاخص و یا جستجو در جدول درهم ساز، کارایی و سایر متریک‌ها بهبود خواهد داشت.

جستجو شاخص.

اکنون حالتی را در نظرمی گیریم که در آن شاخص X بر روی صفت C از رابطه داخلی S (منظور از داخلی رابطه ایست که در عمل پیوند در داخل حلقه قرار می‌گیرد) وجود دارد (شکل ۱۸,۵). مزیت این تکنیک نسبت به تکنیک غیر هوشمندانه در این است که برای هر تاپل در رابطه بیرونی R می‌توان «به طور مستقیم» تاپلهایی از رابطه داخلی S را که با آن منطبق می‌شوند را پیدا نمود. بنابراین روشن است که تعداد کل خواندن تاپل ها برای رابطه‌های R و S برابر با کاردینالتی نتیجه حاصل از پیوند خواهد بود. با فرض بدترین حالت که هر خواندن تاپل از رابطه (درونی) S برابر با خواندن یک صفحه جداگانه است (یعنی هر تاپل یک صفحه را پر نموده)، تعداد صفحاتی که باید خوانده شود برابر است با : (m/pR) + (m*n)/dCS.

```
/* assume index X on S.C */
do i := 1 to m ;                                  /* outer loop */
    /* let there be k index entries X[1], . . . , X[k]  with  */
    /* indexed attribute value = R[i].C                       */
    do j := 1 to n  ;                             /* inner loop */
        /* let tuple of S indexed by X[j] be S[j] */
        If  R[i].C = S[j].C  then
        add joined tuple R[i] * S[j] to result ;
    end;
 end;
```

شکل ۱۸,۵ : جستجو با استفاده از شاخص.

هرچند که اگر رابطه S بر حسب مقادیر صفت پیوند C مرتب شده باشد تعداد خواندن صفحه در مثال مذکور به (m/pR) +m*n/dCS/pS کاهش خواهد یافت. با توجه به مقادیر عددی مثال تکنیک غیر هوشمندانه که در آن، n = 10.000 ،pR = 1،

pS = 10 است و همچنین با فرض اینکه dCS = 100 است، نتیجه ارزیابی دو فرمول فوق به ترتیب برابر با ۱۰,۱۰۰ و ۱,۱۰۰ خواهد شد. تفاوت بین این دو اهمیت نگهداری رابطه‌های ذخیره شده را در یک ترتیب فیزیکی «خوب» نشان می‌دهد.

به هر حال باید سربار ناشی از دسترسی به خود شاخص X را نیز باید در نظر بگیریم. در بدترین حالت هر تاپل از رابطه R نیازمند یک جستجوی شاخص برای منطبق کردن تاپل های رابطه S می‌باشد. که به معنی خواندن یک صفحه از هر سطح شاخص است. برای یک شاخص x سطحی لازم است که به تعداد m*x صفحه خوانده شود. در عمل، معمولاً x برابر یا کمتر از ۳ می‌باشد. (بعلاوه، بالاترین سطح از شاخص در حافظه اصلی مقیم است و برای پردازش خواندن صفحه کاهش می‌یابد).

جستجو درهم ساز (جا یاب).

جستجو در هم ساز شبیه جستجو شاخص است، با این تفاوت که «مسیر دسترسی سریع[1]» به صفت پیوند S.C در رابطه داخلی S بجای شاخص از درهم سازی استفاده شده است (شکل ۱۸,٦). تخمین هزینه این حالت به عنوان تمرین واگذار شده است.

```
/* assume hash table H  on S.C */
do i := 1 to m ;                                  /* outer loop */
     k := hash ( R[i].C) ;
      /* let there be h tuple  S[1], . . . , Sh]  stored at H[k]  */
          do j := 1 to n  ;                              /* inner loop */
                If  R[i].C = S[j].C  then
                add joined tuple R[i] * S[j] to result ;
      end;
 end;
```

شکل ۱۸,٦ جستجو درهم ساز.

[1] Fast access path

ادغام.

تکنیک ادغام فرض می‌کند که هر دو رابطه R و S به ترتیب مقادیر صفت پیوند C به طور فیزیکی ذخیره شده‌اند. اگر چنین باشد دو رابطه می‌توانند به همان ترتیب فیزیکی پیمایش شوند، و هر دو پیمایش می‌تواند هم‌زمان صورت پذیرد و نتیجه کلی پیوند می‌تواند با یک‌بار گذر از داده‌ها به وجود آید (حداقل این ادعا در صورتی درست است که پیوند از نوع یک به چند باشد در حالت چند به چند ممکن است کاملاً درست نباشد). بر اساس فرضیاتی که داریم این تکنیک بهینه است زیرا هر صفحه تنها یک‌بار دست‌یابی می‌شود (شکل ۱۸,۷)، به عبارت دیگر، تعداد صفحاتی که خوانده می‌شود تنها برابر (m/pR) + (n/pS) خواهد بود. نتیجه می‌شود:

- خوشه سازی فیزیکی از داده‌های مرتبط یک عامل مهم در کارایی است. یعنی بسیار مناسب خواهد بود اگر داده‌ها به صورتی خوشه سازی شوند که (تاپل ها بر حسب مقادیری از برخی صفات به طور فیزیکی در کنار هم ذخیره شوند) با پیوندهایی که برای سازمان مهم است تطبیق داده شوند.
- در صورت نبود چنین خوشه سازی، بهتر است که یک یا هر دو رابطه‌ها در زمان اجرا مرتب شوند و سپس عمل پیوند دو رابطه انجام شود. این تکنیک را تکنیک **مرتب سازی/ ادغام**[1] می‌نامند.

---

[1] Sort/merge

```
/* assume R and S are both stored on attribute C ;  */
/* following code assumes join is many-to-many ;    */
/* simpler many-to-one case is left as an exercise   */

r := 1 ;
s := 1;
do while r ≤ m and s ≤  n ;                          /* outer loop */
    v := R[r].C;
     do  j := s  by I while s[j].c < v;
     end ;
     s := j ;
     do j := s by 1 while S[j].C = v ;       /* main inner loop */
        do  I := r by 1 while R[i].C = v ;
             add  joined tuple R[i] * s[j] to result;
        end;
     end;
     s := j ;
     do I := r by 1 while R[i].C = v ;
     end ;
     r := I ;
end;
```

شکل ۱۸,۷ ادغام (حالت چند به چند).

درهم سازی.

مانند تکنیک ادغام، تکنیک درهم سازی نیز نیازمند تنها یک گذر از هر دو رابطه پیوند خواهد بود (شکل ۱۸,۸) گذر اول، یک جدول درهم سازی (جا یابی) برای رابطه S بر اساس مقادیر صفت پیوند S.C می‌سازد، مدخل‌های این جدول شامل مقدار صفت پیوند (احتمالاً شامل مقادیر دیگر هم هست) و اشاره گری به تاپل متناظر با آن صفت (تاپلی که حاوی مقدار صفت پیوند است) بر روی دیسک است. دومین گذر رابطه R را پیمایش نموده و همین تابع درهم ساز را بر روی صفت پیوند R.C اعمال می‌کند (یعنی مدخل‌هایی در این جدول درهم ساز درج می‌کند). زمانی که یک تاپل R در جدول درهم ساز با یک یا چند تاپل S تداخل (تصادم) پیدا کرد، الگوریتم بررسی می‌کند که آیا مقادیر R.C و S.C برابر هستند یا نه. در صورت برابر بودن تاپل (های) پیوندی مناسب را تولید می‌کند. مزیت مهم این تکنیک نسبت به تکنیک ادغام این است

که لازم نیست که رابطه‌های R و S به ترتیب خاصی مرتب باشند و نیاز به مرتب سازی نخواهد داشت.

همانند تکنیک جستجو درهم ساز، تخمین هزینه‌ها در این رهیافت به عنوان تمـرین در نظر گرفته شده است.

```
/*build hash table H on S.C
do j := 1 to n ;
    k := hash (S[j].C);
    add S[j] to hash table entry H[k] ;
end ;
/*  now do hash lookup on R */
```

شکل ۱۸،۸ درهم ساز (جا یاب).