

M. Tamer Özsu · Patrick Valduriez

# Principios de Repartido

Base de datos

Sistemas



 Springer

## Principios de los sistemas de bases de datos distribuidas

M. Tamer Ozsu • Patrick Valduriez

Principios de la distribución

Sistemas de bases de datos

Cuarta edición

123

M. Tamer Ozsu  
Cheriton Escuela de Ciencias de la  
Computación Universidad  
de Waterloo Waterloo, ON, Canadá

Patrick Valduriez  
Inria y LIRMM  
Universidad de Montpellier  
Montpellier, Francia

---

Las dos primeras ediciones de este libro fueron publicadas por: Pearson Education, Inc.

---

ISBN 978-3-030-26252-5 ISBN 978-3-030-26253-2 (libro electrónico) <https://doi.org/>

[10.1007/978-3-030-26253-2](https://doi.org/10.1007/978-3-030-26253-2)

3.<sup>a</sup> edición: © Springer Science+Business Media, LLC 2011 © Springer  
Nature Switzerland AG 2020 Esta obra está  
sujeta a derechos de autor. La editorial se reserva todos los derechos, ya sea sobre la totalidad o parte del material, en  
particular los derechos de traducción, reimpresión, reutilización de ilustraciones, recitación, difusión, reproducción en  
microfilmes o en cualquier otro formato físico, así como la transmisión o el almacenamiento y recuperación de información,  
la adaptación electrónica, el uso de software o cualquier otra metodología similar o diferente, conocida actualmente o  
desarrollada en el futuro.  
El uso de nombres descriptivos generales, nombres registrados, marcas comerciales, marcas de servicio, etc. en esta  
publicación no implica, incluso en ausencia de una declaración específica, que dichos nombres estén exentos de las  
leyes y reglamentos de protección pertinentes y, por lo tanto, sean de uso general.  
La editorial, los autores y los editores pueden asumir con seguridad que los consejos y la información de este libro se  
consideran veraces y precisos en la fecha de publicación. Ni la editorial, ni los autores ni los editores ofrecen garantía  
alguna, expresa o implícita, con respecto al material aquí contenido ni por cualquier error u omisión que pudiera haberse  
cometido. La editorial se mantiene neutral respecto a reclamaciones jurisdiccionales en los mapas publicados y  
afiliaciones institucionales.

Este pie de imprenta de Springer lo publica la empresa registrada Springer Nature Switzerland AG.  
La dirección registrada de la empresa es: Gewerbestrasse 11, 6330 Cham, Suiza

A nuestras  
familias y a  
nuestros padres MTÖ. y PV

# Prefacio

La primera edición de este libro se publicó en 1991, cuando la tecnología era nueva y no existían muchos productos. En el prefacio de la primera edición, citamos a Michael Stonebraker, quien afirmó en 1988 que, en los siguientes 10 años, los SGBD centralizados serían una "curiosidad antigua" y que la mayoría de las organizaciones avanzarían hacia sistemas SGBD distribuidos. Esta predicción ha resultado ser acertada, y una gran proporción de los sistemas en uso hoy en día son distribuidos o paralelos, comúnmente conocidos como sistemas de escalabilidad horizontal. Cuando estábamos preparando la primera edición, los cursos de bases de datos de grado y posgrado no eran tan comunes como ahora; por lo tanto, la versión inicial del libro contenía extensas discusiones sobre soluciones centralizadas antes de presentar sus contrapartes distribuidas/paralelas.

Sin duda, los tiempos también han cambiado en ese aspecto, y ahora es difícil encontrar un estudiante de posgrado que no tenga al menos un conocimiento básico de tecnología de bases de datos. Por lo tanto, un libro de texto de posgrado sobre tecnología de bases de datos distribuidas/paralelas debe tener un enfoque diferente. Ese fue nuestro objetivo en esta edición, manteniendo los numerosos temas nuevos que introdujimos en la tercera edición. Las principales revisiones de esta cuarta edición son las siguientes:

1. Con el paso de los años, las motivaciones y el entorno de esta tecnología han cambiado ligeramente (web, nube, etc.). Por ello, el capítulo introductorio necesitaba una actualización profunda. Revisamos la introducción para ofrecer una perspectiva más contemporánea de la tecnología.
2. Hemos añadido un nuevo capítulo sobre el procesamiento de big data para abarcar los sistemas de almacenamiento distribuido, el procesamiento de flujos de datos, las plataformas MapReduce y Spark, el análisis de grafos y los lagos de datos. Con la proliferación de estos sistemas, es fundamental abordarlos sistemáticamente.
3. De igual forma, abordamos la creciente influencia de los sistemas NoSQL dedicándoles un nuevo capítulo. Este capítulo abarca los cuatro tipos de NoSQL (almacenes clave-valor, almacenes de documentos, sistemas de columnas anchas y sistemas de gestión de bases de datos de grafos), así como los sistemas NewSQL y los polialmacenes.
4. Hemos combinado los capítulos de integración de bases de datos y procesamiento de consultas de múltiples bases de datos de la tercera edición en un capítulo uniforme sobre integración de bases de datos.

5. Realizamos una revisión a fondo del análisis de la gestión de datos web, que anteriormente se centraba principalmente en XML, para reorientarlo hacia la tecnología RDF, más predominante actualmente. En este capítulo, analizamos los enfoques de integración de datos web, incluyendo la importante cuestión de la calidad de los datos.
6. Hemos revisado y actualizado el capítulo de gestión de datos peer to peer y Incluyó una larga discusión sobre blockchain.
7. Como parte de la depuración de los capítulos anteriores, condensamos los capítulos de procesamiento de consultas y gestión de transacciones eliminando las técnicas centralizadas fundamentales y los centramos en técnicas distribuidas/paralelas. En el proceso, incluimos algunos temas que han cobrado importancia desde entonces, como el procesamiento dinámico de consultas (eddies) y el algoritmo de consenso de Paxos y su uso en protocolos de confirmación.
8. Actualizamos el capítulo sobre sistemas de gestión de bases de datos paralelos aclarando los objetivos, en particular, la escalabilidad vertical frente a la horizontal, y analizando las arquitecturas paralelas que incluyen UMA o NUMA. También añadimos una nueva sección sobre algoritmos de ordenación paralela y variantes de algoritmos de unión paralela para aprovechar las memorias principales de gran tamaño y los procesadores multinúcleo que predominan hoy en día.
9. Actualizamos el capítulo sobre diseño de distribución incluyendo una extensa discusión sobre enfoques modernos que combinan fragmentación y asignación. Al reorganizar el material, este capítulo ahora es fundamental para la partición de datos, tanto para la gestión de datos distribuidos como para la gestión paralela, que se abordará en el resto del libro.

Si bien la tecnología de objetos sigue desempeñando un papel importante en los sistemas de información, su importancia en la gestión de datos distribuidos/paralelos ha disminuido. Por lo tanto, eliminamos el capítulo sobre bases de datos de objetos de esta edición.

Como es evidente, todo el libro y cada capítulo han sido revisados y actualizados para un tratamiento más contemporáneo. El material que eliminamos en el proceso no se ha perdido; se incluye como apéndices en línea y aparece en la página web del libro: <https://cs.uwaterloo.ca/ddbs>. Optamos por ofrecerlos en línea en lugar de la versión impresa para mantener el tamaño del libro a un precio razonable (lo que también mantiene el precio razonable). El sitio web también incluye diapositivas de presentación que pueden usarse para enseñar con el libro, así como soluciones a la mayoría de los ejercicios (disponibles solo para instructores que han adoptado el libro para la enseñanza).

Al igual que en ediciones anteriores, muchos colegas contribuyeron con esta edición del libro a quienes nos gustaría agradecer (sin un orden específico). Dan Olteanu proporcionó una interesante discusión sobre dos optimizaciones que pueden reducir significativamente el tiempo de mantenimiento de las vistas materializadas en el capítulo 3. Phil Bernstein proporcionó información clave para nuevos artículos sobre la gestión de transacciones multiversión que resultaron en actualizaciones de esa discusión en el capítulo 5. Khuzaima Daudjee también ayudó al proporcionar una lista de publicaciones más contemporáneas sobre procesamiento distribuido de transacciones que incluimos en la sección de notas bibliográficas de ese capítulo. Ricardo Jiménez-Peris contribuyó con texto sobre sistemas de transacciones de alto rendimiento que se incluye en el mismo capítulo. También contribuyó con una sección sobre LeanXcale en el capítulo sobre NoSQL, NewSQL y polystores. Dennis Shasha revisó la nueva sección de blockchain en el capítulo sobre P2P. Michael Carey leyó los artículos sobre big data, NoSQL, NewSQL y

Los capítulos sobre polystores y DBMS paralelos aportaron comentarios sumamente detallados que los mejoraron considerablemente. Los estudiantes de Tamer, Anil Pacaci, Khaled Ammar y el posdoctorado Xiaofei Zhang, realizaron revisiones exhaustivas del capítulo sobre big data, y este capítulo incluye textos de sus publicaciones. El capítulo sobre NoSQL, NewSQL y polystores incluye textos de publicaciones de Boyan Kolev y Carlyna Bondiombouy, estudiante de Patrick. Jim Webber revisó la sección sobre Neo4j en dicho capítulo. La caracterización de los sistemas de analítica gráfica en ese capítulo se basa parcialmente en la tesis de maestría de Minyang Han, donde también propone el enfoque GiraphUC, que se analiza en dicho capítulo. Semih Salihoglu y Lukasz Golab también revisaron y aportaron comentarios muy útiles sobre partes de este capítulo.

Alon Halevy proporcionó comentarios sobre la discusión de WebTables en el Capítulo 12. La discusión sobre la calidad de los datos en la integración de datos web es aportada por Ihab Ilyas y Xu Chu. Stratos Idreos fue muy útil al aclarar cómo se puede utilizar el craqueo de bases de datos como un enfoque de partición y proporcionó texto que se incluye en el Capítulo 2. Renan Souza y Fabian Stöter revisaron el libro completo.

La tercera edición del libro introdujo varios temas nuevos que se mantuvieron en esta edición, y varios colegas tuvieron una gran influencia en la redacción de esos capítulos. Queremos agradecer, una vez más, su ayuda, ya que su impacto también se refleja en la presente edición. Renée Miller, Erhard Rahm y Alon Halevy fueron fundamentales en la elaboración del análisis sobre la integración de bases de datos, que fue revisado exhaustivamente por Avigdor Gal. Matthias Jarke, Xiang Li, Gottfried Vossen, Erhard Rahm y Andreas Thor aportaron ejercicios para este capítulo.

Hubert Naacke contribuyó a la sección sobre modelado de costos heterogéneos y Fabio Porto a la sección sobre procesamiento adaptativo de consultas. La replicación de datos (Cap. 6) no se habría podido escribir sin la ayuda de Gustavo Alonso y Bettina Kemme.

Esther Pacitti también contribuyó al capítulo sobre replicación de datos, revisándolo y aportando material de referencia. También contribuyó a la sección sobre replicación en clústeres de bases de datos en el capítulo sobre sistemas de gestión de bases de datos paralelos. La gestión de datos entre pares (P2P) se debe en gran medida a las conversaciones con Beng Chin Ooi. La sección de este capítulo sobre el procesamiento de consultas en sistemas P2P utiliza material del trabajo de doctorado de Reza Akbarinia y Wenceslao Palma, mientras que la sección sobre replicación utiliza material del trabajo de doctorado de Vidal Martins.

Agradecemos a nuestra editora en Springer, Susan Lagerstrom-Fife, por impulsar este proyecto dentro de Springer y por animarnos a terminarlo a tiempo. No cumplimos con casi ninguna de sus fechas límite, pero esperamos que el resultado final sea satisfactorio.

Finalmente, nos interesaría mucho conocer sus comentarios y sugerencias sobre el material. Agradecemos cualquier comentario, pero nos gustaría especialmente recibir comentarios sobre los siguientes aspectos:

1. Cualquier error que haya podido persistir a pesar de nuestros mejores esfuerzos (aunque esperamos no hay muchos);

2. Cualquier tema que ya no debería incluirse y cualquier tema que debería agregarse o ampliado;
3. Cualquier ejercicio que hayas diseñado y que te gustaría incluir en Biblia.

Waterloo, Canadá

Montpellier, Francia

Junio de 2019

Sr. Tamer Özsü ([tamer.ozsu@uwaterloo.ca](mailto:tamer.ozsu@uwaterloo.ca))  
Patrick Valduriez ([patrick.valduriez@inria.fr](mailto:patrick.valduriez@inria.fr))

# Contenido

1 Introducción .....	1
1.1 ¿Qué es un sistema de base de datos distribuida? .....	1
1.2 Historia de los DBMS distribuidos .....	3
1.3 Alternativas de entrega de datos .....	5
1.4 Promesas de los DBMS distribuidos .....	7
1.4.1     Gestión transparente de recursos distribuidos y Datos replicados .....	7
1.4.2     Confiabilidad a través de transacciones distribuidas.....	10
1.4.3     Rendimiento mejorado 11 .....	
1.4.4     Escalabilidad .....	13
1.5 Problemas de diseño .....	13
1.5.1     Diseño de bases de datos distribuidas .....	13
1.5.2     Control de datos distribuidos .....	14
1.5.3     Procesamiento de consultas distribuidas .....	14
1.5.4     Control de concurrencia distribuida .....	14
1.5.5     Confiabilidad de los SGBD distribuidos .....	15
1.5.6     Replicación.....	15
1.5.7     DBMS paralelos .....	16
1.5.8     Integración de bases de datos .....	16
1.5.9     Enfoques de distribución alternativos .....	16
1.5.10    Procesamiento de Big Data y NoSQL.....	16
1.6 Arquitecturas de DBMS distribuidos .....	17
1.6.1     Modelos arquitectónicos para sistemas de gestión de bases de datos distribuidos .....	17
1.6.2     Sistemas cliente/servidor .....	20
1.6.3     Sistemas peer-to-peer .....	22
1.6.4     Sistemas multibase de datos .....	25
1.6.5     Computación en la nube .....	27
1.7 Notas bibliográficas .....	31

<b>2 Diseño de bases de datos distribuidas y paralelas .....</b>	<b>33</b>
<b>2.1 Fragmentación de datos .....</b>	<b>35</b>
<b>2.1.1 Fragmentación horizontal .....</b>	<b>37</b>
<b>2.1.2 Fragmentación vertical .....</b>	<b>52</b>
<b>2.1.3 Fragmentación híbrida.....</b>	<b>65</b>
<b>2.2 Asignación.....</b>	<b>66</b>
<b>2.2.1 Información auxiliar .....</b>	<b>68</b>
<b>2.2.2 Modelo de asignación.....</b>	<b>69</b>
<b>2.2.3 Métodos de solución 72 .....</b>	
<b>2.3 Enfoques combinados .....</b>	<b>72</b>
<b>2.3.1 Técnicas de partición independientes de la carga de trabajo .....</b>	<b>73</b>
<b>2.3.2 Técnicas de particionamiento teniendo en cuenta la carga de trabajo .....</b>	<b>74</b>
<b>2.4 Enfoques adaptativos .....</b>	<b>78</b>
<b>2.4.1 Detección de cambios en la carga de trabajo.....</b>	<b>79</b>
<b>2.4.2 Detección de elementos afectados .....</b>	<b>79</b>
<b>2.4.3 Reconfiguración incremental .....</b>	<b>80</b>
<b>2.5 Directorio de datos .....</b>	<b>82</b>
<b>2.6 Conclusión .....</b>	<b>83</b>
<b>2.7 Notas bibliográficas .....</b>	<b>84</b>
<b>3 Control de datos distribuidos .....</b>	<b>91</b>
<b>3.1 Gestión de vistas .....</b>	<b>92</b>
<b>3.1.1 Vistas en DBMS centralizados.....</b>	<b>92</b>
<b>3.1.2 Vistas en sistemas DBMS distribuidos.....</b>	<b>95</b>
<b>3.1.3 Mantenimiento de vistas materializadas.....</b>	<b>96</b>
<b>3.2 Control de acceso .....</b>	<b>102</b>
<b>3.2.1 Control de acceso discrecional .....</b>	<b>103</b>
<b>3.2.2 Control de acceso obligatorio .....</b>	<b>106</b>
<b>3.2.3 Control de acceso distribuido .....</b>	<b>108</b>
<b>3.3 Control de integridad semántica .....</b>	<b>110</b>
<b>3.3.1 Control centralizado de integridad semántica .....</b>	<b>111</b>
<b>3.3.2 Control de integridad semántica distribuida .....</b>	<b>116</b>
<b>3.4 Conclusión .....</b>	<b>123</b>
<b>3.5 Notas bibliográficas .....</b>	<b>123</b>
<b>4 Procesamiento de consultas distribuidas .....</b>	<b>129</b>
<b>4.1 Descripción general.....</b>	<b>130</b>
<b>4.1.1 Problema de procesamiento de consultas .....</b>	<b>130</b>
<b>4.1.2 Optimización de consultas .....</b>	<b>133</b>
<b>4.1.3 Capas de procesamiento de consultas .....</b>	<b>136</b>
<b>4.2 Localización de datos.....</b>	<b>140</b>
<b>4.2.1 Reducción por fragmentación horizontal primaria .....</b>	<b>141</b>
<b>4.2.2 Reducción con unión .....</b>	<b>142</b>
<b>4.2.3 Reducción por fragmentación vertical .....</b>	<b>143</b>
<b>4.2.4 Reducción por fragmentación derivada.....</b>	<b>145</b>
<b>4.2.5 Reducción por fragmentación híbrida.....</b>	<b>148</b>

4.3 Ordenamiento de uniones en consultas distribuidas.....	149
4.3.1 Unir árboles .....	149
4.3.2 Ordenamiento de unión .....	151
4.3.3 Algoritmos basados en semiuniones .....	153
4.3.4 Unión versus Semiunión .....	156
4.4 Modelo de costos distribuidos .....	157
4.4.1 Funciones de costos.....	157
4.4.2 Estadísticas de la base de datos .....	159
4.5 Optimización de consultas distribuidas .....	161
4.5.1 Enfoque dinámico .....	161
4.5.2 Enfoque estático .....	165
4.5.3 Enfoque híbrido .....	169
4.6 Procesamiento de consultas adaptativo .....	173
4.6.1 Proceso de procesamiento de consultas adaptativo .....	174
4.6.2 Enfoque de Eddy .....	176
4.7 Conclusión .....	177
4.8 Notas bibliográficas .....	178
5 Procesamiento de transacciones distribuidas .....	183
5.1 Antecedentes y terminología .....	184
5.2 Control de concurrencia distribuida .....	188
5.2.1 Algoritmos basados en bloqueo .....	189
5.2.2 Algoritmos basados en marcas de tiempo .....	197
5.2.3 Control de concurrencia multiversión .....	203
5.2.4 Algoritmos optimistas .....	205
5.3 Control de concurrencia distribuida mediante aislamiento de instantáneas .....	206
5.4 Confiabilidad de los SGBD distribuidos .....	209
5.4.1 Protocolo de compromiso de dos fases .....	211
5.4.2 Variaciones de 2PC.....	217
5.4.3 Cómo afrontar las fallas del sitio .....	220
5.4.4 Particionado de red .....	227
5.4.5 Protocolo de Consenso de Paxos .....	231
5.4.6 Consideraciones arquitectónicas .....	234
5.5 Enfoques modernos para escalar la gestión de transacciones .....	236
5.5.1 Llave inglesa .....	237
5.5.2 LeanXcale .....	237
5.6 Conclusión .....	239
5.7 Notas bibliográficas .....	241
6 Replicación de datos .....	247
6.1 Consistencia de las bases de datos replicadas .....	249
6.1.1 Coherencia mutua .....	249
6.1.2 Consistencia mutua versus consistencia de transacción .....	251
6.2 Estrategias de gestión de actualizaciones.....	252
6.2.1 Propagación de actualizaciones ansiosas .....	253
6.2.2 Propagación de actualizaciones diferidas .....	254

6.2.3 Técnicas centralizadas .....	254
6.2.4 Técnicas distribuidas .....	255
6.3 Protocolos de replicación .....	255
6.3.1 Protocolos centralizados ansiosos .....	256
6.3.2 Protocolos distribuidos Eager .....	262
6.3.3 Protocolos centralizados perezosos .....	262
6.3.4 Protocolos distribuidos perezosos .....	268
6.4 Comunicación grupal .....	269
6.5 Replicación y fallos .....	272
6.5.1 Fallos y replicación diferida .....	273
6.5.2 Fallos y replicación ansiosa .....	273
6.6 Conclusión .....	276
6.7 Notas bibliográficas .....	277
<b>7 Integración de bases de datos: sistemas multibase de datos.....</b>	<b>281</b>
7.1 Integración de bases de datos .....	282
7.1.1 Metodología de diseño de abajo hacia arriba .....	283
7.1.2 Coincidencia de esquemas .....	287
7.1.3 Integración de esquemas .....	296
7.1.4 Mapeo de esquemas .....	298
7.1.5 Limpieza de datos .....	306
7.2 Procesamiento de consultas en múltiples bases de datos .....	307
7.2.1 Problemas en el procesamiento de consultas en múltiples bases de datos .....	308
7.2.2 Arquitectura de procesamiento de consultas multibase de datos .....	309
7.2.3 Reescritura de consultas mediante vistas.....	311
7.2.4 Optimización y ejecución de consultas .....	317
7.2.5 Traducción y ejecución de consultas.....	329
7.3 Conclusión .....	332
7.4 Notas bibliográficas .....	334
<b>8 Sistemas de bases de datos paralelas .....</b>	<b>349</b>
8.1 Objetivos.....	350
8.2 Arquitecturas paralelas .....	352
8.2.1 Arquitectura general .....	353
8.2.2 Memoria compartida .....	355
8.2.3 Disco compartido .....	357
8.2.4 Nada compartido.....	358
8.3 Ubicación de datos .....	359
8.4 Procesamiento de consultas paralelas .....	362
8.4.1 Algoritmos paralelos para el procesamiento de datos .....	362
8.4.2 Optimización de consultas paralelas .....	369
8.5 Equilibrio de carga .....	374
8.5.1 Problemas de ejecución paralela .....	374
8.5.2 Equilibrio de carga intraoperador .....	376
8.5.3 Equilibrio de carga entre operadores .....	378
8.5.4 Equilibrio de carga entre consultas .....	378

8.6 Tolerancia a fallos .....	383
8.7 Clústeres de bases de datos .....	384
8.7.1     Arquitectura de clúster de bases de datos .....	385
8.7.2     Replicación.....	386
8.7.3     Equilibrio de carga .....	386
8.7.4     Procesamiento de consultas .....	387
8.8 Conclusión .....	390
8.9 Notas bibliográficas .....	390
 9 Gestión de datos punto a punto .....	395
9.1     Infraestructura .....	398
9.1.1 Redes P2P no estructuradas .....	399
9.1.2 Redes P2P estructuradas .....	402
9.1.3 Redes P2P superpeer .....	406
9.1.4 Comparación de redes P2P .....	408
9.2 Mapeo de esquemas en sistemas P2P .....	408
9.2.1 Mapeo de esquemas por pares.....	408
9.2.2 Mapeo basado en técnicas de aprendizaje automático .....	409
9.2.3 Mapeo de acuerdos comunes .....	410
9.2.4 Mapeo de esquemas mediante técnicas IR .....	411
9.3 Consultas a través de sistemas P2P .....	411
9.3.1     Consultas Top-k .....	412
9.3.2     Consultas de unión .....	424
9.3.3     Consultas de rango .....	425
9.4 Consistencia de la réplica .....	428
9.4.1 Soporte básico en DHTs .....	429
9.4.2 Moneda de datos en DHT.....	431
9.4.3 Conciliación de réplicas .....	432
9.5 Cadena de bloques .....	436
9.5.1 Definición de cadena de bloques .....	437
9.5.2 Infraestructura de cadena de bloques .....	438
9.5.3 Blockchain 2.0.....	442
9.5.4 Problemas.....	443
9.6 Conclusión .....	444
9.7 Notas bibliográficas .....	445
 10 Procesamiento de Big Data .....	449
10.1 Sistemas de almacenamiento distribuido .....	451
10.1.1 Sistema de archivos de Google .....	453
10.1.2 Combinación de almacenamiento de objetos y almacenamiento de archivos .....	454
10.2 Marcos de procesamiento de Big Data .....	455
10.2.1 Procesamiento de datos de MapReduce .....	456
10.2.2 Procesamiento de datos con Spark .....	466
10.3 Gestión de datos de transmisión .....	470
10.3.1 Modelos de flujo, lenguajes y operadores .....	472
10.3.2 Procesamiento de consultas sobre flujos de datos.....	476
10.3.3 Tolerancia a fallos del DSS .....	483

10.4 Plataformas de análisis de grafos .....	486
10.4.1 Particionamiento de grafos.....	489
10.4.2 MapReduce y análisis de grafos .....	494
10.4.3 Sistemas de análisis de grafos de propósito especial .....	495
Bloque centrado en vértices síncrono.....	498
Centrado en vértices asíncrono .....	501
10.4.6 Centrado en vértices: Recopilación-Aplicación-Dispersión .....	503
10.4.7 Procesamiento síncrono de bloques centrado en particiones.....	504
10.4.8 Centrado en particiones asíncrono .....	506
10.4.9 Centrado en particiones: Recopilación-Aplicación-Dispersión .....	506
10.4.10 Procesamiento síncrono de bloques centrado en el borde .....	507
10.4.11 Procesamiento asíncrono centrado en el borde .....	507
10.4.12 Recopilación, aplicación y dispersión centrados en el borde .....	507
10.5 Lagos de datos .....	508
10.5.1 Lago de datos frente a almacén de datos .....	508
10.5.2 Arquitectura .....	510
Desafíos .....	511
10.6 Conclusión .....	512
10.7 Notas bibliográficas .....	512
11 NoSQL, NewSQL y Polystores .....	519
11.1 Motivaciones para NoSQL .....	520
11.2 Almacenes de clave-valor .....	521
11.2.1 DynamoDB .....	522
11.2.2 Otros almacenes de clave-valor .....	524
11.3 Almacenes de documentos .....	525
11.3.1 MongoDB .....	525
11.3.2 Otros almacenes de documentos .....	528
11.4 Almacenes de columnas anchas .....	529
11.4.1 Bigtable .....	529
11.5 SGBD de gráficos .....	531
11.5.1 Neo4j.....	531
11.5.2 Otras bases de datos de grafos .....	532
11.6 Almacenes de datos híbridos .....	535
11.6.1 Almacenes NoSQL multimodelo .....	536
11.6.2 SGBD NewSQL .....	537
11.7 Polystores.....	537
11.7.1 Polystores débilmente acoplados .....	540
11.7.2 Polystores fuertemente acoplados .....	544
11.7.3 Sistemas híbridos .....	549
11.7.4 Observaciones finales .....	553
11.8 Conclusión .....	555
11.9 Notas bibliográficas .....	555

<b>12 Gestión de datos web .....</b>	559	<b>12.1 Gestión de gráficos</b>
<b>web.....</b>	560	<b>12.2 Búsqueda web .....</b>
<b>562 12.2.1 Rastreo web .....</b>	563	<b>12.2.2</b>
<b>Indexación .....</b>	566	<b>12.2.3 Clasificación y análisis de</b>
<b>enlaces .....</b>	567	<b>12.2.4 Evaluación de la búsqueda de palabras</b>
<b>clave .....</b>	568	<b>12.3 Consultas web .....</b>
<b>569 12.3.1 Enfoque de datos semiestructurados .....</b>	570	<b>12.3.2 Enfoque</b>
<b>del lenguaje de consulta web .....</b>	574	<b>12.4 Sistemas de preguntas y</b>
<b>respuestas .....</b>	580	<b>12.5 Búsqueda y consulta en la web</b>
<b>oculta .....</b>	584	<b>12.5.1 Rastreo de la web oculta .....</b>
<b>12.5.2 Metabúsqueda .....</b>	586	<b>12.6 Integración de datos</b>
<b>web .....</b>	588	<b>12.6.1 Tablas web/tablas de fusión .....</b>
<b>589 12.6.2 Web semántica y datos abiertos enlazados .....</b>	590	<b>12.6.3</b>
<b>Problemas de calidad de datos en la integración de datos web ..</b>	608	<b>12.7 Notas</b>
<b>bibliográficas .....</b>	615	
<b>Una descripción general de los DBMS relacionales .....</b>	619	
<b>B Procesamiento centralizado de consultas.....</b>	621	
<b>Fundamentos del procesamiento de transacciones C .....</b>	623	
<b>D Revisión de redes de computadoras.....</b>	625	
<b>Referencias.....</b>	627	
<b>Índice .....</b>	663	

# Capítulo 1

## Introducción



El entorno informático actual está en gran medida distribuido: las computadoras están conectadas a Internet para formar un sistema distribuido mundial. Las organizaciones cuentan con centros de datos geográficamente distribuidos e interconectados, cada uno con cientos o miles de computadoras conectadas a redes de alta velocidad, conformando una combinación de sistemas distribuidos y paralelos (Fig. 1.1). En este entorno, la cantidad de datos capturados ha aumentado drásticamente. No todos estos datos se almacenan en sistemas de bases de datos (de hecho, solo una pequeña parte lo hace), pero existe el deseo de proporcionar algún tipo de capacidad de gestión de datos sobre estos datos ampliamente distribuidos. Este es el alcance de los sistemas de bases de datos distribuidos y paralelos, que han pasado de ser una pequeña parte del entorno informático mundial hace unas décadas a ser la norma general.

En este capítulo proporcionamos una descripción general de esta tecnología, antes de examinar los detalles en los capítulos siguientes.

### 1.1 ¿Qué es un sistema de base de datos distribuida?

Definimos una base de datos distribuida como un conjunto de múltiples bases de datos lógicamente interrelacionadas, ubicadas en los nodos de un sistema distribuido. Un sistema de gestión de bases de datos distribuidas (SGBD) se define entonces como el sistema de software que permite la gestión de la base de datos distribuida y hace que la distribución sea transparente para los usuarios. En ocasiones, el término «sistema de base de datos distribuida» (SGBD) se utiliza para referirse conjuntamente a la base de datos distribuida y al SGBD distribuido.

Las dos características importantes son que los datos están lógicamente interrelacionados y que residen en un sistema distribuido.

La existencia de un sistema distribuido es una característica importante. En este contexto, definimos un sistema de computación distribuida como un conjunto de elementos de procesamiento (EP) autónomos interconectados. Las capacidades de estos elementos de procesamiento pueden variar, ser heterogéneos y las interconexiones pueden ser...

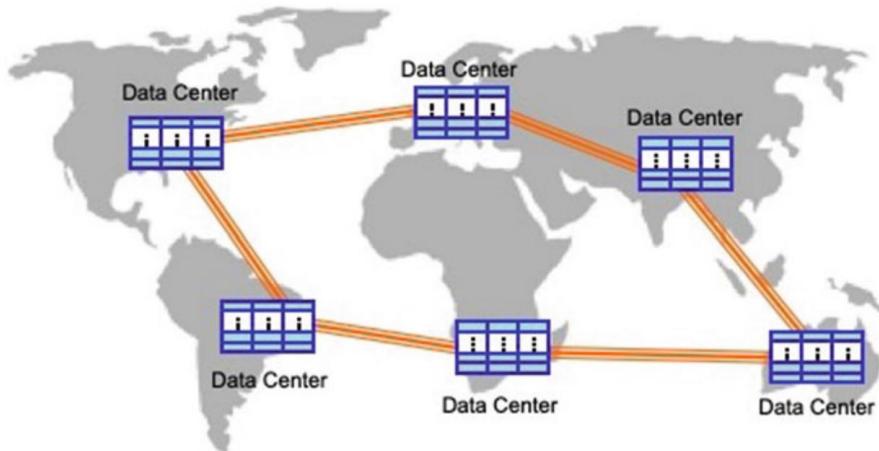


Figura 1.1 Centros de datos distribuidos geográficamente

Diferente, pero lo importante es que los PE no tienen acceso al estado de los demás, el cual solo pueden conocer mediante el intercambio de mensajes que implican un coste de comunicación. Por lo tanto, cuando los datos se distribuyen, su gestión y acceso de forma lógicamente integrada requiere un cuidado especial por parte del software SGBD distribuido.

Un SGBD distribuido no es una colección de archivos que se puedan almacenar individualmente en cada PE de un sistema distribuido (generalmente llamado "sitio" de un SGBD distribuido); los datos en un SGBD distribuido están interrelacionados. No intentaremos ser muy específicos con el significado de "interrelacionado", ya que los requisitos difieren según el tipo de datos. Por ejemplo, en el caso de los datos relacionales, diferentes relaciones o sus particiones podrían almacenarse en diferentes sitios (más sobre esto en el Capítulo 2), lo que requiere operaciones de unión para responder a consultas que generalmente se expresan en SQL. Generalmente, se puede definir un esquema de estos datos distribuidos. En el otro extremo, los datos en sistemas NoSQL (discutidos con más detalle en el Capítulo 11) pueden tener una definición de interrelación mucho más laxa; por ejemplo, pueden ser vértices de un grafo que podrían almacenarse en diferentes sitios.

La conclusión de este debate es que un SGBD distribuido está integrado lógicamente, pero distribuido físicamente. Esto significa que un SGBD distribuido ofrece a los usuarios la visión de una base de datos unificada, mientras que los datos subyacentes están distribuidos físicamente.

Como se señaló anteriormente, generalmente consideramos dos tipos de DBMS distribuidos: distribuidos geográficamente (comúnmente denominados geodistribuidos) y de ubicación única (o sitio único).

En el primero, los sitios están interconectados por redes de área amplia que se caracterizan por largas latencias de mensajes y mayores tasas de error.

Estos últimos consisten en sistemas donde los PE se ubican muy cerca, lo que permite intercambios mucho más rápidos, lo que resulta en latencias de mensajes más cortas (incluso insignificantes con las nuevas tecnologías) y tasas de error muy bajas. Los DBMS distribuidos en una sola ubicación se caracterizan típicamente por clústeres de computadoras en un centro de datos y son comúnmente...

Conocidos como SGBD paralelos (y los PE se denominan "nodos" para distinguirlos de los "sitios"). Como se mencionó anteriormente, ahora es bastante común encontrar SGBD distribuidos con múltiples clústeres de un solo sitio interconectados por redes de área extensa (WAN), lo que da lugar a sistemas híbridos multisitio. Durante la mayor parte de este libro, nos centraremos en los problemas de la gestión de datos entre los sitios de un SGBD geodistribuido; nos centraremos en los problemas de los sistemas de un solo sitio en los capítulos 8, 10 y 11, donde analizaremos los SGBD paralelos, los sistemas de big data y los sistemas NoSQL/NewSQL.

## 1.2 Historia de los DBMS distribuidos

Antes de la llegada de los sistemas de bases de datos en la década de 1960, el modo de computación predominante era aquel en el que cada aplicación definía y gestionaba sus propios datos (Fig. 1.2). En este modo, cada aplicación definía los datos que utilizaba, su estructura y métodos de acceso, y gestionaba el archivo en el sistema de almacenamiento. El resultado final era una redundancia significativa e incontrolada en los datos y una alta sobrecarga para los programadores al gestionarlos dentro de sus aplicaciones.

Los sistemas de bases de datos permiten definir y administrar los datos de forma centralizada (Fig. 1.3). Esta nueva orientación genera independencia de los datos, lo que permite que los programas de aplicación sean inmunes a los cambios en la organización lógica o física de los datos y viceversa. En consecuencia, los programadores se liberan de la tarea de gestionar y mantener los datos que necesitan, y se puede eliminar (o reducir) la redundancia de los datos.

Una de las motivaciones originales detrás del uso de sistemas de bases de datos fue el deseo de integrar los datos operativos de una empresa y proporcionar un acceso integrado y, por lo tanto, controlado a dichos datos. Usamos con cuidado el término "integrado" en lugar de...

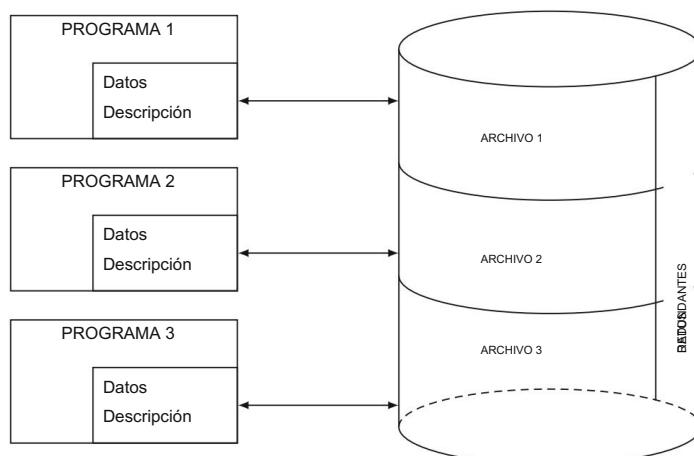


Fig. 1.2 Procesamiento tradicional de archivos

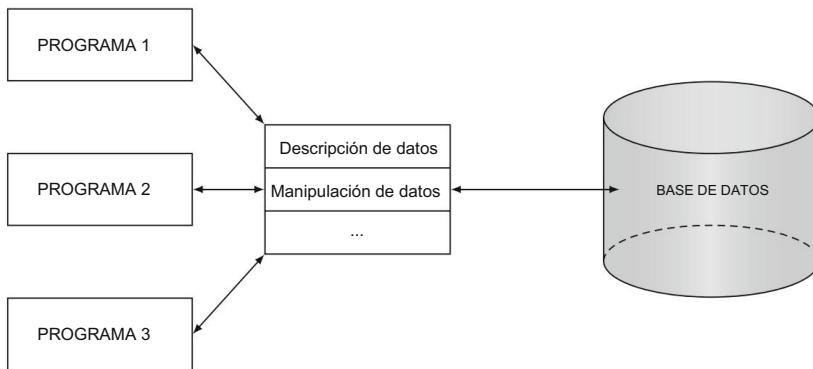


Fig. 1.3 Procesamiento de la base de datos

En lugar de "centralizados", ya que, como se mencionó anteriormente, los datos pueden ubicarse físicamente en diferentes máquinas que podrían estar distribuidas geográficamente. Esto es lo que ofrece la tecnología de bases de datos distribuidas. Como se mencionó anteriormente, esta distribución física puede estar concentrada en una o varias ubicaciones geográficas.

Por lo tanto, cada una de las ubicaciones de la Fig. 1.5 podría ser un centro de datos conectado a otros centros de datos mediante una red de comunicación. Estos son los tipos de entornos distribuidos comunes en la actualidad que estudiamos en este libro.

A lo largo de los años, las arquitecturas de los sistemas de bases de datos distribuidas han experimentado cambios significativos. Los sistemas originales de bases de datos distribuidas, como INGRES Distribuido y SDD-1, se diseñaron como sistemas distribuidos geográficamente con conexiones de red muy lentas; por lo tanto, buscaban optimizar las operaciones para reducir la comunicación en red. Fueron pioneros de los sistemas punto a punto (P2P), en el sentido de que cada sitio tenía una funcionalidad similar en cuanto a la gestión de datos. Con el desarrollo de las computadoras personales y las estaciones de trabajo, el modelo de distribución predominante cambió a cliente/servidor, donde las operaciones de datos se trasladaban a un servidor back-end, mientras que las aplicaciones de usuario se ejecutaban en las estaciones de trabajo front-end. Estos sistemas se volvieron dominantes, especialmente para la distribución en una ubicación específica donde la velocidad de la red era mayor, lo que permitía una comunicación frecuente entre los clientes y el/los servidor(es). En la década del 2000, resurgieron los sistemas P2P, donde no se distingue entre máquinas cliente y servidores. Estos sistemas P2P modernos presentan diferencias importantes con respecto a los sistemas anteriores que analizaremos más adelante en este capítulo. Todas estas arquitecturas todavía se pueden encontrar hoy en día y las analizamos en capítulos posteriores.

El surgimiento de la World Wide Web (generalmente llamada la web) como una importante plataforma de colaboración e intercambio tuvo un profundo impacto en la investigación sobre la gestión distribuida de datos. Se abrió un acceso significativamente mayor a los datos, pero no se trataba de los datos bien estructurados y definidos que suelen manejar los SGBD; en cambio, son datos no estructurados o semiestructurados (es decir, tienen cierta estructura, pero no al nivel de un esquema de base de datos), de procedencia incierta (por lo que podrían estar "sucios" o ser poco fiables) y conflictivos. Además, muchos de los datos se almacenan en sistemas que no son fáciles de gestionar.

Accesible (lo que se conoce como la red oscura). Por consiguiente, las iniciativas de gestión distribuida de datos se centran en acceder a estos datos de forma significativa.

Este desarrollo agregó un impulso particular a una línea de investigación que existía desde el comienzo de los esfuerzos de bases de datos distribuidas, a saber, la integración de bases de datos.

Originalmente, estos esfuerzos se centraban en encontrar maneras de acceder a los datos en bases de datos independientes (de ahí los términos "base de datos federada" y "multibase de datos"). Sin embargo, con la aparición de los datos web, estos esfuerzos se orientaron hacia la integración virtual de diferentes tipos de datos (y el término "integración de datos" se popularizó). El término de moda es "lago de datos", que implica que todos los datos se capturan en un único almacén lógico, del cual se extraen los datos relevantes para cada aplicación. Analizamos el primero en el capítulo [7](#) y el segundo en los capítulos [10](#) y [12](#).

Un avance significativo en los últimos diez años ha sido la aparición de la computación en la nube. La computación en la nube se refiere a un modelo informático en el que varios proveedores de servicios ofrecen recursos informáticos compartidos y geodistribuidos, de modo que los usuarios pueden alquilar algunos de estos recursos según sus necesidades. Los clientes pueden alquilar la infraestructura informática básica para desarrollar su propio software, pero luego deciden el sistema operativo que desean utilizar y crean máquinas virtuales (VM) para crear el entorno en el que desean trabajar: el enfoque de Infraestructura como Servicio (IaaS). Un entorno de nube más sofisticado implica alquilar, además de la infraestructura básica, la plataforma informática completa, lo que da lugar a la Plataforma como Servicio (PaaS), en la que los clientes pueden desarrollar su propio software. La versión más sofisticada consiste en que los proveedores de servicios ofrecen software específico que los clientes pueden alquilar; esto se denomina Software como Servicio (SaaS). Existe una tendencia a ofrecer servicios de gestión de bases de datos distribuidas en la nube como parte de la oferta de SaaS, y este ha sido uno de los desarrollos más recientes.

Además de los capítulos específicos donde discutimos estas arquitecturas en profundidad, Proporcionamos una descripción general de todos ellos en la Sección [1.6.1.2](#).

## 1.3 Alternativas de entrega de datos

En bases de datos distribuidas, la entrega de datos se realiza entre sitios, ya sea desde servidores a clientes en respuesta a consultas o entre múltiples servidores. Caracterizamos las alternativas de entrega de datos según tres dimensiones ortogonales: modos de entrega, frecuencia y métodos de comunicación. La combinación de alternativas en cada una de estas dimensiones proporciona un amplio espacio de diseño.

Los modos de entrega alternativos son solo extracción, solo inserción e híbrido. En el modo de entrega de datos solo extracción , la transferencia de datos se inicia mediante una extracción (es decir, una solicitud) de un sitio a un proveedor de datos; este puede ser un cliente que solicita datos de un servidor o un servidor que solicita datos de otro servidor. En adelante, usaremos los términos "receptor" y "proveedor" para referirnos a la máquina que recibe los datos y a la máquina que los envía, respectivamente. Cuando el proveedor recibe la solicitud, los datos se localizan y se transfieren. La principal característica de la extracción...

La entrega basada en datos implica que los receptores solo conocen los nuevos datos o actualizaciones del proveedor cuando los consultan explícitamente. Además, en el modo de extracción, los proveedores deben interrumpirse continuamente para atender las solicitudes. Además, los datos que los receptores pueden obtener de un proveedor se limitan a cuándo y qué datos los clientes saben solicitar.

Los DBMS convencionales ofrecen principalmente entrega de datos basada en extracción.

En el modo de entrega de datos basado en push, la transferencia de datos de los proveedores se inicia mediante un push sin una solicitud específica. La principal dificultad del enfoque basado en push reside en decidir qué datos serían de interés común y cuándo enviarlos a los receptores potencialmente interesados; las alternativas son periódicas, irregulares o condicionales. Por lo tanto, la utilidad del push depende en gran medida de la precisión del proveedor para predecir las necesidades de los receptores. En el modo basado en push, los proveedores difunden información a un conjunto ilimitado de receptores (difusión aleatoria) que pueden escuchar un medio o a un conjunto selectivo de receptores (multidifusión) que pertenecen a ciertas categorías de destinatarios.

El modo híbrido de entrega de datos combina los mecanismos de extracción e inserción. El enfoque de consulta persistente (véase la sección 10.3) presenta una posible manera de combinar los modos de extracción e inserción: la transferencia de datos de los proveedores a los receptores se inicia primero mediante una extracción (al plantear la consulta), y la posterior transferencia de datos actualizados se inicia mediante una inserción del proveedor.

Existen tres medidas de frecuencia típicas que permiten clasificar la regularidad de la entrega de datos: periódica, condicional y ad hoc (o irregular).

En la entrega periódica, los proveedores envían datos a intervalos regulares. Estos intervalos pueden definirse por defecto del sistema o por los receptores en sus perfiles. Tanto la extracción como la inserción pueden realizarse de forma periódica. La entrega periódica se realiza según un programa regular y predefinido. Una solicitud semanal del precio de las acciones de una empresa es un ejemplo de extracción periódica. Un ejemplo de inserción periódica es cuando una aplicación puede enviar la lista de precios de acciones regularmente, por ejemplo, todas las mañanas. La inserción periódica es especialmente útil en situaciones en las que los receptores podrían no estar disponibles en todo momento o no poder reaccionar a lo que se ha enviado, como en el entorno móvil, donde los clientes pueden desconectarse.

En la entrega condicional, los proveedores envían datos siempre que se cumplan ciertas condiciones especificadas por los receptores en sus perfiles. Dichas condiciones pueden ser tan simples como un lapso de tiempo determinado o tan complejas como reglas de evento-condición-acción.

La entrega condicional se utiliza principalmente en sistemas de entrega híbridos o solo push.

Mediante el envío condicional, los datos se envían según una condición predefinida, en lugar de un programa repetitivo específico. Una aplicación que envía los precios de las acciones solo cuando cambian es un ejemplo de envío condicional. Una aplicación que envía un estado de cuenta solo cuando el saldo total es un 5 % inferior al umbral de saldo predefinido es un ejemplo de envío condicional híbrido. El envío condicional asume que los cambios son críticos para los receptores, quienes están siempre atentos y necesitan responder a lo que se envía. El envío condicional híbrido asume, además, que la falta de información de actualización no es crucial para los receptores.

La entrega ad hoc es irregular y se realiza mayoritariamente en un sistema basado puramente en extracción. Los datos se extraen de los proveedores de manera ad hoc en respuesta a las solicitudes. En

## 1.4 Promesas de los DBMS distribuidos

7

Por el contrario, la extracción periódica surge cuando un solicitante utiliza un sondeo para obtener datos de los proveedores según un período regular (programa).

El tercer componente del espacio de diseño de alternativas de entrega de información es el método de comunicación. Estos métodos determinan las diversas maneras en que proveedores y receptores se comunican para entregar información a los clientes. Las alternativas son unidifusión y uno a muchos. En unidifusión, la comunicación entre un proveedor y un receptor es uno a uno: el proveedor envía datos a un receptor utilizando un modo de entrega específico con cierta frecuencia. En uno a muchos, como su nombre indica, el proveedor envía datos a varios receptores. Cabe destacar que no nos referimos a un protocolo específico; la comunicación uno a muchos puede utilizar un protocolo de multidifusión o de difusión.

Cabe señalar que esta caracterización es objeto de considerable debate. No está claro que todos los puntos del espacio de diseño sean significativos. Además, la especificación de alternativas como condicionales y periódicas (que podrían tener sentido) es difícil. Sin embargo, sirve como una caracterización de primer orden de la complejidad de los sistemas emergentes de gestión de datos distribuidos. En este libro, nos centramos principalmente en sistemas de entrega de datos ad hoc basados únicamente en extracción, y en la sección 10.3 analizamos los modos basados en inserción e híbridos en sistemas de transmisión.

## 1.4 Promesas de los DBMS distribuidos

Se pueden citar numerosas ventajas de los SGBD distribuidos; estas se resumen en cuatro fundamentos que también pueden considerarse promesas de la tecnología de SGBD distribuidos: gestión transparente de datos distribuidos y replicados, acceso fiable a los datos mediante transacciones distribuidas, mejor rendimiento y mayor facilidad de expansión del sistema. En esta sección, analizamos estas promesas y, en el proceso, introducimos muchos de los conceptos que estudiaremos en capítulos posteriores.

### 1.4.1 Gestión transparente de recursos distribuidos y replicados

#### Datos

La transparencia se refiere a la separación de la semántica de alto nivel de un sistema de los problemas de implementación de bajo nivel. En otras palabras, un sistema transparente oculta los detalles de la implementación a los usuarios. La ventaja de un SGBD totalmente transparente es el alto nivel de soporte que ofrece para el desarrollo de aplicaciones complejas. La transparencia en los SGBD distribuidos puede considerarse una extensión del concepto de independencia de datos en los SGBD centralizados (más información a continuación).

Comencemos nuestra discusión con un ejemplo. Consideraremos una empresa de ingeniería con oficinas en Boston, Waterloo, París y San Francisco. Ejecutan proyectos en cada una de estas sedes y desean mantener una base de datos de sus empleados, los proyectos,

```

EMP(ENO, ENAME, TITLE)
PROJ(PNO, PNAME, BUDGET, LOC)
ASG(ENO, PNO, RESP, DUR)
PAY(TITLE, SAL)

```

Fig. 1.4 Ejemplo de base de datos de ingeniería

y otros datos relacionados. Suponiendo que la base de datos es relacional, podemos almacenar esta información en varias relaciones (Fig. 1.4): EMP almacena la información de los empleados con su número, nombre y cargo; PROJ contiene la información del proyecto, mientras que LOC registra la ubicación del proyecto. La información salarial se almacena en PAY (suponiendo que todos con el mismo cargo reciben el mismo salario) y la asignación de personas a proyectos se registra en ASG, donde DUR indica la duración de la asignación y la responsabilidad de la persona en ese proyecto se mantiene en RESP. Si todos estos datos se almacenaran en un SGB centralizado y quisiéramos conocer los nombres y empleados que trabajaron en un proyecto durante más de 12 meses, lo especificaríamos mediante la siguiente consulta SQL:

```

SELECCIONAR ENAME, AMT
DE EMP NATURAL JOIN ASG, EMP NATURAL JOIN PAY DONDE ASG.DUR > 12

```

Sin embargo, dada la naturaleza distribuida del negocio de esta empresa, es preferible, en estas circunstancias, localizar los datos de forma que los datos de los empleados de la oficina de Waterloo se almacenen en Waterloo, los de la oficina de Boston en Boston, y así sucesivamente. Lo mismo aplica a la información sobre proyectos y salarios. Por lo tanto, estamos realizando un proceso en el que particionamos cada una de las relaciones y almacenamos cada partición en un sitio diferente. Esto se conoce como particionamiento o fragmentación de datos y se explica con más detalle más adelante en el capítulo 2.

Además, puede ser preferible duplicar algunos de estos datos en otros sitios por razones de rendimiento y fiabilidad. El resultado es una base de datos distribuida, fragmentada y replicada (Fig. 1.5). El acceso totalmente transparente permite a los usuarios realizar la consulta como se especificó anteriormente, sin tener en cuenta la fragmentación, la ubicación ni la replicación de los datos, dejar que el sistema se encargue de resolver estos problemas. Para que un sistema gestione adecuadamente este tipo de consulta en una base de datos distribuida, fragmentada y replicada, debe ser capaz de gestionar diversos tipos de transparencia, como se explica a continuación.

**Independencia de datos.** Este concepto, heredado de los SGBD centralizados, se refiere a la inmunidad de las aplicaciones de usuario a los cambios en la definición y organización de los datos, y viceversa.

Se suelen citar dos tipos de independencia de datos: la independencia lógica de datos y la independencia física de datos. La independencia lógica de datos se refiere a la inmunidad de las aplicaciones de usuario a los cambios en la estructura lógica (es decir, el esquema) de la base de datos.

---

<sup>1</sup>Los atributos clave principales están subrayados.

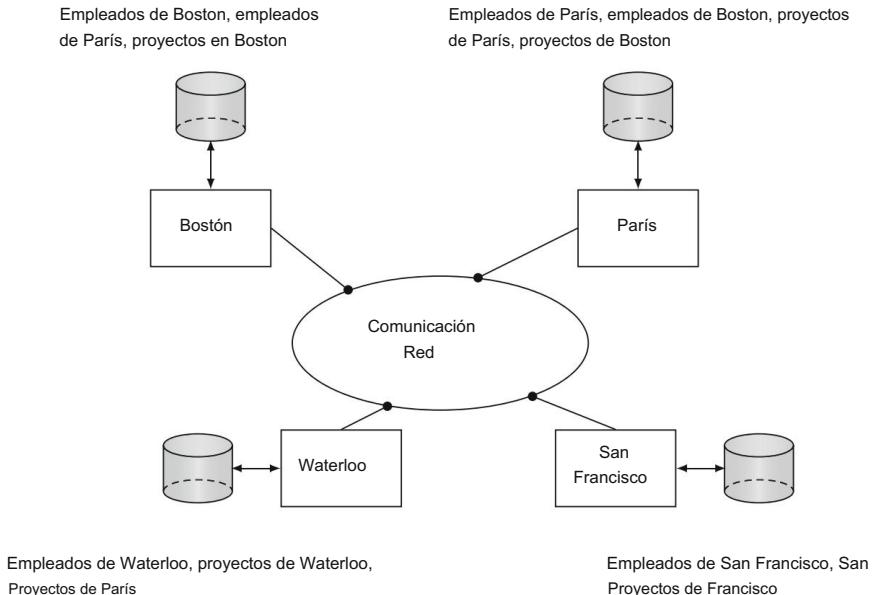


Fig. 1.5 Base de datos distribuida

La independencia física de los datos, por otro lado, implica ocultar los detalles de la estructura de almacenamiento a las aplicaciones de usuario. Al escribir una aplicación de usuario, esta no debería preocuparse por los detalles de la organización física de los datos. Por lo tanto, no debería ser necesario modificarla cuando se produzcan cambios en la organización de los datos debido a consideraciones de rendimiento.

**Transparencia de red.** Preferiblemente, los usuarios deberían estar protegidos de los detalles operativos de la red de comunicación que conecta los sitios; posiblemente incluso ocultando la existencia de la red. De esta manera, no habría diferencia entre las aplicaciones de bases de datos que se ejecutan en una base de datos centralizada y las que se ejecutan en una base de datos distribuida. Este tipo de transparencia se conoce como transparencia de red o transparencia de distribución.

A veces se identifican dos tipos de transparencia de distribución: transparencia de ubicación y transparencia de nombres. La transparencia de ubicación se refiere a que el comando utilizado para realizar una tarea es independiente tanto de la ubicación de los datos como del sistema en el que se realiza la operación. La transparencia de nombres implica que se asigna un nombre único a cada objeto de la base de datos. En ausencia de transparencia de nombres, los usuarios deben incluir el nombre de la ubicación (o un identificador) como parte del nombre del objeto.

**Transparencia de fragmentación.** Como se mencionó anteriormente, suele ser conveniente dividir cada relación de base de datos en fragmentos más pequeños y tratar cada fragmento como un objeto de base de datos independiente (es decir, otra relación). Esto se suele hacer por razones de rendimiento, disponibilidad y fiabilidad (más información).

Se encuentra en el Cap. 2. Sería preferible que los usuarios ignoraran la fragmentación de datos al especificar consultas y que el sistema se encargara de mapear una consulta de usuario especificada en relaciones completas, según lo especificado en el esquema, a un conjunto de consultas ejecutadas en subrelaciones. En otras palabras, el problema radica en encontrar una estrategia de procesamiento de consultas basada en los fragmentos en lugar de las relaciones, aunque las consultas se especifiquen en estas últimas.

Transparencia de la replicación. Por razones de rendimiento, fiabilidad y disponibilidad, suele ser deseable poder distribuir los datos de forma replicada entre las máquinas de una red. Suponiendo que los datos se replican, la cuestión de la transparencia radica en si los usuarios deben conocer la existencia de copias o si el sistema debe gestionarlas y el usuario debe actuar como si existiera una única copia de los datos (cabe destacar que no nos referimos a la ubicación de las copias, sino solo a su existencia). Desde la perspectiva del usuario, es preferible no involucrarse en la gestión de copias ni tener que especificar que una determinada acción puede o debe realizarse en múltiples copias. El tema de la replicación de datos dentro de una base de datos distribuida se presenta en el capítulo 2 y se analiza en detalle en el capítulo 6.

#### 1.4.2 Confiabilidad a través de transacciones distribuidas

Los SGBD distribuidos están diseñados para mejorar la fiabilidad, ya que cuentan con componentes replicados y, por lo tanto, eliminan los puntos únicos de fallo. El fallo de un solo sitio, o el fallo de un enlace de comunicación que impide el acceso a uno o más sitios, no es suficiente para provocar la caída de todo el sistema. En el caso de una base de datos distribuida, esto significa que algunos datos pueden ser inaccesibles, pero con el debido cuidado, se puede permitir a los usuarios acceder a otras partes de la base de datos distribuida.

El "cuidado adecuado" se materializa principalmente en forma de apoyo a las transacciones distribuidas. Un DBMS que brinda soporte total para transacciones garantiza que la ejecución concurrente de transacciones de usuario no violará la consistencia de la base de datos, es decir, cada usuario piensa que su consulta es la única que se ejecuta en la base de datos (llamada transparencia de concurrencia) incluso ante fallas del sistema (llamada transparencia de fallas) siempre que cada transacción sea correcta, es decir, obedezca las reglas de integridad especificadas en la base de datos.

Proporcionar soporte transaccional requiere la implementación de protocolos de control de concurrencia distribuido y confiabilidad distribuida, en particular, los protocolos de confirmación en dos fases (2PC) y de recuperación distribuida, que son significativamente más complejos que sus contrapartes centralizadas. Estos se describen en el capítulo 5.

El soporte de réplicas requiere la implementación de protocolos de control de réplicas que imponen una semántica específica para acceder a ellas. Estos se describen en el capítulo 6.

### 1.4.3 Rendimiento mejorado

La mejora del rendimiento de los SGBD distribuidos se basa generalmente en dos puntos. En primer lugar, un SGBD distribuido fragmenta la base de datos, lo que permite almacenar los datos cerca de sus puntos de uso (también conocido como localización de datos).

Esto tiene dos ventajas potenciales:

1. Dado que cada sitio maneja solo una parte de la base de datos, existe contención por la CPU y Los servicios de E/S no son tan severos como los de las bases de datos centralizadas.
2. La localidad reduce los retrasos en el acceso remoto que suelen estar relacionados con las áreas extensas. redes.

Este punto se relaciona con la sobrecarga de la computación distribuida si los datos residen en sitios remotos y se debe acceder a ellos mediante comunicación remota. El argumento es que, en estas circunstancias, es mejor distribuir la funcionalidad de gestión de datos donde se encuentran, en lugar de mover grandes cantidades de datos. Este tema a veces genera controversia. Algunos argumentan que, con el uso generalizado de redes de alta velocidad y alta capacidad, la distribución de datos y las funciones de gestión de datos ya no tienen sentido, y que podría ser mucho más sencillo almacenar los datos en un sitio central utilizando una máquina muy grande y acceder a ellos a través de redes de alta velocidad. Esto se conoce comúnmente como arquitectura de escalado vertical . Es un argumento atractivo, pero omite un aspecto importante de las bases de datos distribuidas. En primer lugar, en la mayoría de las aplicaciones actuales, los datos se distribuyen; lo que podría ser objeto de debate es cómo y dónde los procesamos. En segundo lugar, y más importante, este argumento no distingue entre ancho de banda (la capacidad de los enlaces informáticos) y latencia (el tiempo que tardan en transmitirse los datos). La latencia es inherente a los entornos distribuidos y existen límites físicos a la velocidad con la que podemos enviar datos a través de redes informáticas. El acceso remoto a los datos puede generar latencias inaceptables para muchas aplicaciones.

El segundo punto es que el paralelismo inherente de los sistemas distribuidos puede ser explotado para el paralelismo interconsulta e intraconsulta. El paralelismo interconsulta permite la ejecución paralela de múltiples consultas generadas por transacciones concurrentes, con el fin de aumentar el rendimiento transaccional. La definición de paralelismo intraconsulta es diferente en los DBMS distribuidos y paralelos. En los primeros, el paralelismo intraconsulta se logra dividiendo una sola consulta en varias subconsultas, cada una de las cuales se ejecuta en un sitio diferente, accediendo a una parte distinta de la base de datos distribuida. En los DBMS paralelos, se logra mediante paralelismo interoperador e intraoperador . El paralelismo interoperador se obtiene ejecutando en paralelo diferentes operadores del trie de consulta en diferentes procesadores, mientras que con el paralelismo intraoperador, el mismo operador es ejecutado por muchos procesadores, cada uno trabajando en un subconjunto de los datos. Tenga en cuenta que estas dos formas de paralelismo también existen en el procesamiento de consultas distribuidas.

El paralelismo intraoperador se basa en la descomposición de un operador en un conjunto de suboperadores independientes, denominados instancias de operador. Esta descomposición se realiza mediante la partición de relaciones. Cada instancia de operador procesará...

Una partición de relación. La descomposición de operadores suele aprovechar la partición inicial de los datos (p. ej., los datos se partitionan según el atributo de unión).

Para ilustrar el paralelismo intraoperador, consideremos una consulta de selección y unión simple. El operador de selección se puede descomponer directamente en varios operadores de selección, cada uno en una partición diferente, sin necesidad de redistribución (Fig. 1.6). Tenga en cuenta que si la relación se partitiona según el atributo de selección, se pueden usar las propiedades de partición para eliminar algunas instancias de selección. Por ejemplo, en una selección de coincidencia exacta, solo se ejecutará una instancia de selección si la relación se partitionó mediante hash (o rango) según el atributo de selección. Descomponer el operador de unión es más complejo. Para obtener uniones independientes, cada partición de una relación R puede unirse a la otra relación S completa. Dicha unión será muy inefficiente (a menos que S sea muy pequeño) porque implicará una difusión de S en cada procesador participante. Una forma más eficiente es usar propiedades de partición. Por ejemplo, si R y S se partitionan mediante hash según el atributo de unión y la unión es una unión equi, podemos partitionar la unión en uniones independientes. Este es el caso ideal que no siempre se puede usar, porque depende de la partición inicial de R y S. En los demás casos, uno o dos operandos pueden ser reparticionados. Finalmente, podemos observar que la función de partición (hash, rango, round robin—discutida en la Secc. 2.3.1) es independiente del algoritmo local (p. ej., bucle anidado, hash, ordenación-combinación) usado para procesar el operador de unión (es decir, en cada procesador). Por ejemplo, una unión hash que usa una partición hash necesita dos funciones hash. La primera,  $h_1$ , se usa para partitionar las dos relaciones base en el atributo de unión. La segunda,  $h_2$ , que puede ser diferente para cada procesador, se usa para procesar la unión en cada procesador.

Se pueden aprovechar dos formas de paralelismo entre operadores. Con el paralelismo de pipeline, varios operadores con un enlace productor-consumidor se ejecutan en paralelo. Por ejemplo, los dos operadores de selección de la Fig. 1.7 se ejecutarán en paralelo con el operador de unión. La ventaja de esta ejecución es que no es necesario materializar completamente el resultado intermedio, lo que ahorra memoria y accesos a disco. Independiente

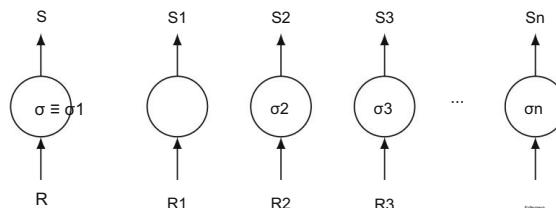


Fig. 1.6 Paralelismo intraoperador.  $\sigma_i$  es la instancia  $i$  del operador;  $n$  es el grado de paralelismo.

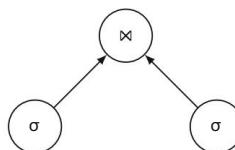


Fig. 1.7 Paralelismo entre operadores

El paralelismo se logra cuando no existe dependencia entre los operadores que se ejecutan en paralelo. Por ejemplo, los dos operadores de selección de la figura 1.7 pueden ejecutarse en paralelo. Esta forma de paralelismo es muy atractiva porque no hay interferencia entre los procesadores.

#### 1.4.4 Escalabilidad

En un entorno distribuido, es mucho más fácil adaptarse al aumento del tamaño de las bases de datos y a mayores cargas de trabajo. La expansión del sistema suele gestionarse añadiendo potencia de procesamiento y almacenamiento a la red. Obviamente, puede que no sea posible obtener un aumento lineal de la potencia, ya que esta también depende de la sobrecarga de la distribución. Sin embargo, aún es posible lograr mejoras significativas. Por ello, los SGBD distribuidos han despertado gran interés en las arquitecturas de escalabilidad horizontal en el contexto de la computación en clúster y en la nube. La escalabilidad horizontal (también denominada escalabilidad horizontal) consiste en añadir más servidores, denominados "servidores de escalabilidad horizontal", de forma flexible, para escalar casi infinitamente. Al facilitar la incorporación de nuevos servidores de bases de datos, un SGBD distribuido puede proporcionar escalabilidad horizontal.

### 1.5 Problemas de diseño

En la sección anterior, analizamos las promesas de la tecnología de SGBD distribuidos, destacando los desafíos que deben superarse para hacerlas realidad. En esta sección, profundizamos en este debate presentando los problemas de diseño que surgen al crear un SGBD distribuido. Estos problemas ocuparán gran parte del resto del libro.

#### 1.5.1 Diseño de bases de datos distribuidas

La pregunta que se está abordando es cómo se colocan los datos en los sitios. El punto de partida es una base de datos global y el resultado final es una distribución de los datos entre los sitios. Esto se conoce como diseño descendente. Existen dos alternativas básicas para la ubicación de los datos: particionados (o no replicados) y replicados. En el esquema particionado, la base de datos se divide en varias particiones disjuntas, cada una ubicada en un sitio diferente. Los diseños replicados pueden ser completamente replicados (también llamados completamente duplicados), donde la base de datos completa se almacena en cada sitio, o parcialmente replicados (o parcialmente duplicados), donde cada partición de la base de datos se almacena en más de un sitio, pero no en todos. Los dos aspectos fundamentales del diseño son la fragmentación (la separación de la base de datos en particiones llamadas fragmentos) y la distribución (la distribución óptima de los fragmentos).

Un problema relacionado es el diseño y la gestión del directorio del sistema. En los SGBD centralizados, el catálogo contiene metainformación (es decir, descripción) sobre los datos.

En un sistema distribuido, tenemos un directorio que contiene información adicional, como la ubicación de los datos. Los problemas relacionados con la gestión de directorios son similares al problema de ubicación de bases de datos analizado en la sección anterior. Un directorio puede ser global para todo el SGBD distribuido o local para cada sitio; puede estar centralizado en un sitio o distribuido en varios; puede haber una sola copia o varias. El diseño de bases de datos distribuidas y la gestión de directorios se tratan en el capítulo 2.

### 1.5.2 Control de datos distribuidos

Un requisito importante de un SGBD es mantener la consistencia de los datos controlando el acceso a ellos. Esto se denomina control de datos e implica la gestión de vistas, el control de acceso y la aplicación de la integridad. La distribución plantea desafíos adicionales, ya que los datos necesarios para verificar las reglas se distribuyen a diferentes sitios, lo que requiere una verificación y aplicación distribuidas de las reglas. Este tema se aborda en el capítulo 3.

### 1.5.3 Procesamiento de consultas distribuidas

El procesamiento de consultas consiste en diseñar algoritmos que analizan las consultas y las convierten en una serie de operaciones de manipulación de datos. El problema radica en cómo determinar la estrategia para ejecutar cada consulta en la red de la manera más rentable; sin embargo, el costo está definido. Los factores a considerar son la distribución de datos, los costos de comunicación y la falta de suficiente información disponible localmente. El objetivo es optimizar dónde se utiliza el paralelismo inherente para mejorar el rendimiento de la ejecución de la transacción, sujeto a las restricciones mencionadas. El problema es de naturaleza NP-hard, y los enfoques suelen ser heurísticos. El procesamiento distribuido de consultas se describe en detalle en el capítulo 4.

### 1.5.4 Control de concurrencia distribuida

El control de concurrencia implica la sincronización de los accesos a la base de datos distribuida, de modo que se mantenga su integridad. El problema del control de concurrencia en un contexto distribuido es algo diferente al de un entorno centralizado. No solo es necesario preocuparse por la integridad de una única base de datos, sino también por la consistencia de múltiples copias de esta. La condición que exige que todos los valores de múltiples copias de cada elemento de datos converjan al mismo valor se denomina consistencia mutua.

Las dos clases generales de soluciones son pesimistas, que sincronizan la ejecución de las solicitudes del usuario antes de que comience la ejecución, y optimistas, que ejecutan las solicitudes y luego verifican si la ejecución ha comprometido la consistencia de la base de datos.

Dos primitivas fundamentales que pueden utilizarse con ambos enfoques son el bloqueo, que se basa en la exclusión mutua de accesos a elementos de datos, y el sellado de tiempo, donde las ejecuciones de transacciones se ordenan según las marcas de tiempo. Existen variaciones de estos esquemas, así como algoritmos híbridos que intentan combinar ambos mecanismos básicos.

En los enfoques basados en bloqueos, los bloqueos mutuos son posibles debido al acceso mutuamente excluyente a los datos por parte de diferentes transacciones. Las conocidas alternativas de prevención, evitación y detección/recuperación también se aplican a los SGBD distribuidos. El control de concurrencia distribuida se trata en el Capítulo 5.

### 1.5.5 Confiabilidad de los DBMS distribuidos

Ya mencionamos que una de las ventajas potenciales de los sistemas distribuidos es la mejora de la confiabilidad y la disponibilidad. Sin embargo, esto no es una característica automática. Es importante proporcionar mecanismos para garantizar la consistencia de la base de datos, así como para detectar fallos y recuperarse de ellos. La implicación para los SGBD distribuidos es que, cuando se produce un fallo y varios sitios dejan de funcionar o de ser accesibles, las bases de datos de los sitios operativos se mantienen consistentes y actualizadas. Además, cuando el sistema informático o la red se recuperan del fallo, los SGBD distribuidos deben ser capaces de recuperar y actualizar las bases de datos de los sitios fallidos. Esto puede ser especialmente difícil en el caso de la partición de la red, donde los sitios se dividen en dos o más grupos sin comunicación entre ellos. Los protocolos de confiabilidad distribuida se tratan en el capítulo 5.

### 1.5.6 Replicación

Si la base de datos distribuida se replica (parcial o totalmente), es necesario implementar protocolos que garanticen la consistencia de las réplicas; es decir, que las copias de un mismo dato tengan el mismo valor. Estos protocolos pueden ser diligentes , ya que fuerzan la aplicación de las actualizaciones a todas las réplicas antes de que se complete la transacción, o pueden ser perezosos , de modo que la transacción actualice una copia (denominada maestra), desde la cual se propagan las actualizaciones a las demás una vez completada la transacción. Los protocolos de replicación se abordan en el capítulo 6.

### 1.5.7 DBMS paralelos

Como se mencionó anteriormente, existe una estrecha relación entre las bases de datos distribuidas y las bases de datos paralelas. Si bien las primeras asumen que cada sitio es un único ordenador lógico, la mayoría de estas instalaciones son, de hecho, clústeres paralelos. Esta es la distinción que destacamos anteriormente entre la distribución en un solo sitio, como en los clústeres de centros de datos, y la distribución geográfica. Los objetivos de los SGBD paralelos difieren ligeramente de los de los SGBD distribuidos, ya que sus principales objetivos son la alta escalabilidad y el rendimiento. Si bien la mayor parte del libro se centra en los problemas que surgen al gestionar datos en bases de datos geodistribuidas, existen interesantes problemas de gestión de datos dentro de una distribución en un solo sitio como sistema paralelo. Abordamos estos problemas en el capítulo 8.

### 1.5.8 Integración de bases de datos

Uno de los avances más importantes ha sido la transición hacia una federación más flexible entre las fuentes de datos, que también pueden ser heterogéneas. Como se explica en la siguiente sección, esto ha dado lugar al desarrollo de sistemas multibase de datos (también llamados sistemas de bases de datos federadas) que requieren una nueva investigación de algunas de las técnicas fundamentales de las bases de datos. La entrada es un conjunto de bases de datos ya distribuidas y el objetivo es facilitar el acceso mediante su integración (física o lógica). Esto implica un diseño ascendente. Estos sistemas constituyen una parte importante del entorno distribuido actual. En el capítulo 7, analizamos los sistemas multibase de datos, o como se denomina más comúnmente ahora la integración de bases de datos, incluyendo los problemas de diseño y los desafíos del procesamiento de consultas .

### 1.5.9 Enfoques de distribución alternativos

El crecimiento de Internet como plataforma de red fundamental ha planteado preguntas importantes sobre los supuestos que subyacen a los sistemas de bases de datos distribuidas.

Dos cuestiones nos preocupan especialmente: la recuperación de la informática entre pares (P2P) y el desarrollo y crecimiento de la World Wide Web.

Ambos buscan mejorar el intercambio de datos, pero adoptan enfoques diferentes y plantean distintos desafíos para su gestión. Analizamos la gestión de datos entre pares en el capítulo 9 y la gestión de datos web en el capítulo 12.

### 1.5.10 Procesamiento de Big Data y NoSQL

La última década ha presenciado la explosión del procesamiento de "big data". La definición exacta de big data es difícil de definir, pero generalmente se acepta que tiene cuatro características conocidas como las "cuatro V": los datos son de gran volumen, son multimodales (variedad) y, por lo general ,

Se transmite a gran velocidad en forma de flujos de datos (velocidad) y puede presentar problemas de calidad debido a la incertidumbre de las fuentes y los conflictos (veracidad). Se han realizado importantes esfuerzos para desarrollar sistemas que gestionen el big data, impulsados por la percepción de la inadecuación de los SGBD relacionales para diversas aplicaciones nuevas. Estos esfuerzos suelen adoptar dos formas: por un lado, se han desarrollado plataformas informáticas de propósito general (casi siempre escalables) para el procesamiento, y por otro, se han desarrollado SGBD especiales que no cuentan con la funcionalidad relacional completa, con capacidades de gestión de datos más flexibles (los denominados sistemas NoSQL). Analizamos las plataformas de big data en el capítulo 10 y los sistemas NoSQL en el capítulo 11.

## 1.6 Arquitecturas de DBMS distribuidos

La arquitectura de un sistema define su estructura. Esto significa que se identifican sus componentes, se especifica la función de cada uno y se definen las interrelaciones e interacciones entre ellos. La especificación de la arquitectura de un sistema requiere la identificación de los distintos módulos, con sus interfaces e interrelaciones, en términos del flujo de datos y control a través del sistema.

En esta sección, desarrollamos cuatro arquitecturas de referencia<sup>2</sup> para un SGBD distribuido: cliente/servidor, punto a punto, multibase de datos y nube. Estas son visiones idealizadas de un SGBD, ya que muchos de los sistemas comerciales pueden diferir de ellas; sin embargo, estas arquitecturas servirán como un marco razonable para abordar los problemas relacionados con los SGBD distribuidos.

Comenzamos con una discusión del espacio de diseño para posicionar mejor las arquitecturas que se presentarán.

### 1.6.1 Modelos arquitectónicos para sistemas DBMS distribuidos

Utilizamos una clasificación (Fig. 1.8) que reconoce tres dimensiones según las cuales se pueden diseñar los SGBD distribuidos: (1) la autonomía de los sistemas locales, (2) su distribución y (3) su heterogeneidad. Estas dimensiones son ortogonales, como explicamos en breve, y en cada una de ellas identificamos diversas alternativas.

En consecuencia, existen 18 arquitecturas posibles en el espacio de diseño; no todas estas alternativas arquitectónicas son significativas, y la mayoría no son relevantes desde la perspectiva de este libro.

Las tres en las que nos centramos se identifican en la figura 1.8.

---

<sup>2</sup>Los desarrolladores de estándares suelen crear una arquitectura de referencia para definir claramente interfaces que necesitan ser estandarizadas.

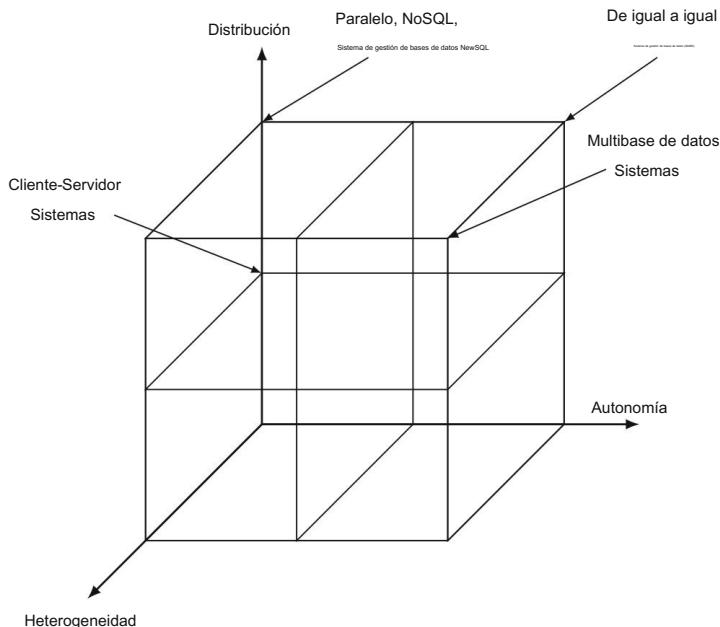


Fig. 1.8 Alternativas de implementación del DBMS

#### 1.6.1.1 Autonomía

La autonomía, en este contexto, se refiere a la distribución del control, no de los datos. Indica el grado en que los SGBD individuales pueden operar de forma independiente. La autonomía depende de diversos factores, como si los sistemas que los componen (es decir, los SGBD individuales) intercambian información, si pueden ejecutar transacciones de forma independiente y si se permite su modificación.

Utilizaremos una clasificación que cubra los aspectos importantes de estas características. Esta clasificación destaca tres alternativas. Una de ellas es la integración estrecha, donde una imagen única de toda la base de datos está disponible para cualquier usuario que desee compartir los datos que pueden residir en múltiples bases de datos. Desde la perspectiva de los usuarios, los datos se integran lógicamente en una sola base de datos. En estos sistemas estrechamente integrados, los administradores de datos se implementan de tal manera que uno de ellos controla el procesamiento de cada solicitud de usuario, incluso si esta es atendida por más de un administrador de datos. Los administradores de datos normalmente no funcionan como DBMS independientes, aunque normalmente tienen la funcionalidad para hacerlo.

A continuación, identificamos sistemas semiautónomos que consisten en SGBD que pueden (y generalmente lo hacen) operar de forma independiente, pero que han decidido participar en una federación para que sus datos locales sean compartibles. Cada uno de estos SGBD determina qué partes de su propia base de datos hará accesibles a los usuarios de otros SGBD. No son completamente...

sistemas autónomos porque necesitan ser modificados para permitirles intercambiar información entre sí.

La última alternativa que consideramos es el aislamiento total, donde los sistemas individuales son SGBD independientes que desconocen la existencia de otros SGBD y cómo comunicarse con ellos. En estos sistemas, el procesamiento de las transacciones de usuario que acceden a múltiples bases de datos es especialmente difícil, ya que no existe control global sobre la ejecución de cada SGBD.

#### 1.6.1.2 Distribución

Mientras que la autonomía se refiere a la distribución (o descentralización) del control, la dimensión de distribución de la taxonomía se ocupa de los datos. Por supuesto, consideramos la distribución física de los datos en múltiples sitios; como se mencionó anteriormente, el usuario los ve como un conjunto lógico. Existen diversas maneras de distribuir los SGBD. Estas alternativas se resumen en dos clases: distribución cliente-servidor y distribución punto a punto (o distribución completa). Junto con la opción no distribuida, la taxonomía identifica tres arquitecturas alternativas.

La distribución cliente/servidor concentra las tareas de gestión de datos en los servidores, mientras que los clientes se centran en proporcionar el entorno de la aplicación, incluida la interfaz de usuario. Las tareas de comunicación se comparten entre los equipos cliente y los servidores. Los SGBD cliente/servidor representan una solución práctica para distribuir la funcionalidad. Existen diversas maneras de estructurarlos, cada una con un nivel de distribución diferente. El análisis detallado se describe en la sección [1.6.2](#).

En los sistemas peer to peer, no hay distinción entre máquinas cliente y servidores.

Cada máquina cuenta con la funcionalidad completa de un SGBD y puede comunicarse con otras máquinas para ejecutar consultas y transacciones. La mayoría de los primeros trabajos sobre sistemas de bases de datos distribuidos partían de la base de datos de arquitectura punto a punto. Por lo tanto, este libro se centra principalmente en los sistemas punto a punto (también conocidos como totalmente distribuidos), aunque muchas de las técnicas también se aplican a los sistemas cliente-servidor.

#### 1.6.1.3 Heterogeneidad

La heterogeneidad puede presentarse de diversas formas en sistemas distribuidos, desde la heterogeneidad del hardware y las diferencias en los protocolos de red hasta las variaciones en los gestores de datos. Desde la perspectiva de este libro, las más importantes se relacionan con los modelos de datos, los lenguajes de consulta y los protocolos de gestión de transacciones. La representación de datos con diferentes herramientas de modelado genera heterogeneidad debido a las capacidades expresivas y limitaciones inherentes de cada modelo de datos. La heterogeneidad en los lenguajes de consulta no solo implica el uso de paradigmas de acceso a datos completamente distintos en distintos modelos de datos (acceso conjunto a conjunto en sistemas relationales frente a acceso registro a registro en algunos sistemas orientados a objetos), sino que también abarca las diferencias entre los lenguajes, incluso cuando los sistemas individuales utilizan el mismo modelo de datos. Aunque SQL es ahora el lenguaje de consulta relacional estándar, hay muchos

Las diferentes implementaciones y el lenguaje de cada proveedor tienen un carácter ligeramente distinto (a veces incluso una semántica distinta, lo que produce resultados distintos). Además, las plataformas de big data y los sistemas NoSQL tienen lenguajes y mecanismos de acceso muy variables.

## 1.6.2 Sistemas cliente/servidor

El modelo cliente/servidor irrumpió en el panorama informático a principios de la década de 1990 y ha tenido un impacto significativo en la tecnología de los SGBD. La idea general es muy simple y elegante: distinguir la funcionalidad que debe proporcionarse en un servidor de la que debe proporcionarse en un cliente. Esto proporciona una arquitectura de dos niveles que facilita la gestión de la complejidad de los SGBD modernos y la complejidad de la distribución.

En los SGBD relacionales cliente-servidor, el servidor realiza la mayor parte de la gestión de datos. Esto significa que todo el procesamiento y la optimización de consultas, la gestión de transacciones y la gestión del almacenamiento se realizan en el servidor. El cliente, además de la aplicación y la interfaz de usuario, cuenta con un módulo cliente del SGBD que se encarga de gestionar los datos almacenados en caché y, en ocasiones, los bloqueos de transacciones que también se hayan almacenado en caché. También es posible realizar la comprobación de consistencia de las consultas del usuario en el lado del cliente, pero esto no es habitual, ya que requiere la replicación del catálogo del sistema en los equipos cliente. Esta arquitectura, representada en la Fig. 1.9, es bastante común en sistemas relacionales donde la comunicación entre los clientes y el/los servidor(es) se realiza a nivel de SQL.

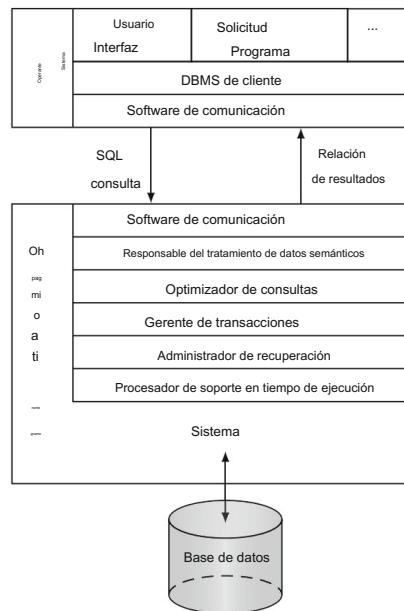


Fig. 1.9 Arquitectura de referencia cliente/servidor

Declaraciones. En otras palabras, el cliente envía consultas SQL al servidor sin intentar comprenderlas ni optimizarlas. El servidor realiza la mayor parte del trabajo y devuelve la relación de resultados al cliente.

Hay varias realizaciones diferentes de la arquitectura cliente/servidor.

El caso más simple es el de un solo servidor al que acceden varios clientes. A esto lo llamamos multiciente/servidor único. Desde la perspectiva de la gestión de datos, esto no difiere mucho de las bases de datos centralizadas, ya que la base de datos se almacena en una sola máquina (el servidor) que también aloja el software para su gestión. Sin embargo, existen diferencias importantes con los sistemas centralizados en la forma en que se ejecutan las transacciones y se gestionan las cachés: dado que los datos se almacenan en la caché del cliente, es necesario implementar protocolos de coherencia de caché.

Una arquitectura cliente/servidor más sofisticada es aquella en la que existen múltiples servidores en el sistema (el denominado enfoque multiciente/multiservidor). En este caso, existen dos estrategias de gestión alternativas: cada cliente gestiona su propia conexión al servidor correspondiente o cada cliente solo conoce su "servidor local", que se comunica con otros servidores según sea necesario. El primer enfoque simplifica el código del servidor, pero impone responsabilidades adicionales a los equipos cliente. Esto da lugar a lo que se ha denominado sistemas de "cliente pesado". El segundo, en cambio, concentra la funcionalidad de gestión de datos en los servidores. De este modo, la transparencia del acceso a los datos se proporciona en la interfaz del servidor, lo que da lugar a "clientes ligeros".

En los sistemas con múltiples servidores, los datos se partitionan y pueden replicarse entre ellos. Esto es transparente para los clientes en el caso de un enfoque de cliente ligero, y los servidores pueden comunicarse entre sí para responder a una consulta del usuario. Este enfoque se implementa en sistemas de gestión de bases de datos (SGBD) paralelos para mejorar el rendimiento mediante el procesamiento en paralelo.

El modelo cliente/servidor puede extenderse de forma natural para proporcionar una distribución de funciones más eficiente en diferentes tipos de servidores: los clientes ejecutan la interfaz de usuario (p. ej., servidores web), los servidores de aplicaciones ejecutan los programas de aplicación y los servidores de bases de datos gestionan las bases de datos. Esto da lugar a la arquitectura de sistema distribuido de tres niveles.

El enfoque de servidor de aplicaciones (de hecho, un enfoque distribuido de n niveles) puede ampliarse mediante la introducción de múltiples servidores de bases de datos y de aplicaciones (Fig. 1.10), como ocurre en las arquitecturas cliente-servidor clásicas. En este caso, cada servidor de aplicaciones suele estar dedicado a una o varias aplicaciones, mientras que los servidores de bases de datos operan con el modelo multiservidor mencionado anteriormente. Además, la interfaz con la aplicación suele realizarse mediante un balanceador de carga que enruta las solicitudes del cliente a los servidores correspondientes.

El enfoque de servidor de bases de datos, como extensión de la arquitectura clásica cliente/servidor, ofrece varias ventajas potenciales. En primer lugar, centrarse exclusivamente en la gestión de datos permite el desarrollo de técnicas específicas para aumentar la fiabilidad y la disponibilidad de los datos, por ejemplo, mediante el paralelismo. En segundo lugar, el rendimiento general de la gestión de bases de datos puede mejorarse significativamente mediante la estrecha integración del sistema de base de datos y un sistema operativo dedicado. Por último, los servidores de bases de datos también pueden aprovechar las ventajas de hardware avanzado, como las GPU y las FPGA, para mejorar tanto el rendimiento como la disponibilidad de los datos.

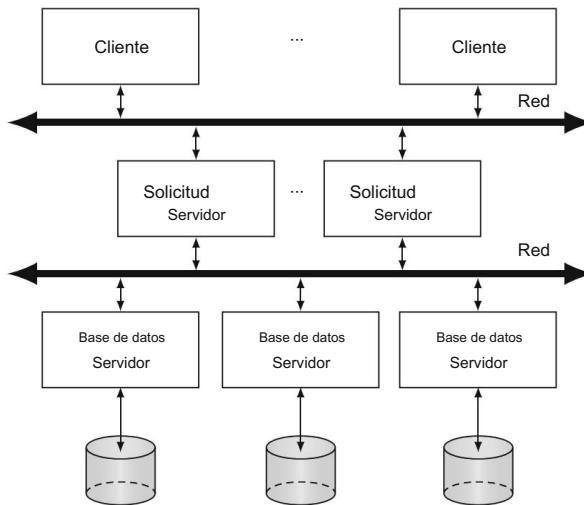


Fig. 1.10 Servidores de bases de datos distribuidos

Si bien estas ventajas son significativas, existe la sobrecarga adicional que supone otra capa de comunicación entre la aplicación y los servidores de datos. El coste de la comunicación puede amortizarse si la interfaz del servidor es lo suficientemente avanzada como para permitir la expresión de consultas complejas que requieren un procesamiento intensivo de datos.

### 1.6.3 Sistemas punto a punto

Los primeros trabajos sobre sistemas de gestión de bases de datos distribuidos se centraron en arquitecturas peer-to-peer, donde no existía diferenciación entre la funcionalidad de cada sitio del sistema. Los sistemas peer-to-peer modernos presentan dos diferencias importantes con respecto a sus predecesores. La primera es la distribución masiva de los sistemas más recientes. Mientras que en sus inicios se centraban en unos pocos sitios (quizás decenas como máximo), los sistemas actuales consideran miles. La segunda es la heterogeneidad inherente de cada aspecto de los sitios y su autonomía. Si bien esto siempre ha sido una preocupación de las bases de datos distribuidas, como se mencionó anteriormente, junto con la distribución masiva, la heterogeneidad y la autonomía de los sitios adquieren una mayor relevancia, lo que impide considerar algunos enfoques. En este libro, nos centramos inicialmente en el significado clásico de peer-to-peer (la misma funcionalidad en cada sitio), ya que los principios y las técnicas fundamentales de estos sistemas son muy similares a los de los sistemas cliente-servidor, y analizamos los problemas de las bases de datos peer-to-peer modernas en un capítulo aparte (Cap. 9).

En estos sistemas, el diseño de la base de datos sigue un diseño descendente, como se explicó anteriormente. Por lo tanto, la entrada es una base de datos (centralizada) con su propia definición de esquema (esquema conceptual global o GCS). Esta base de datos está particionada y asignada a las ubicaciones del SGBD distribuido. De esta manera, en cada ubicación existe una base de datos local con su propio esquema (denominado esquema conceptual local o LCS). El usuario formula consultas según el GCS, independientemente de su ubicación. El SGBD distribuido traduce las consultas globales en un grupo de consultas locales, que son ejecutadas por componentes del SGBD distribuido en diferentes ubicaciones que se comunican entre sí. Desde la perspectiva de las consultas, los sistemas punto a punto y los SGBD cliente/servidor ofrecen la misma vista de los datos. Es decir, ofrecen al usuario la apariencia de una base de datos lógicamente única, mientras que a nivel físico los datos están distribuidos.

Los componentes detallados de un SGBD distribuido se muestran en la Fig. 1.11. Un componente gestiona la interacción con los usuarios y otro el almacenamiento.

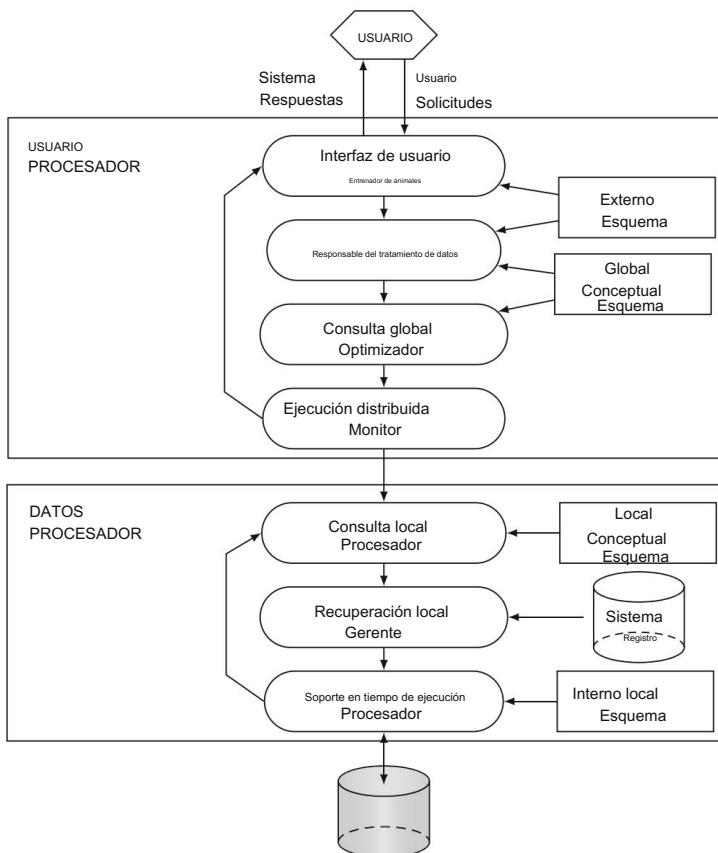


Fig. 1.11 Componentes de un DBMS distribuido

El primer componente principal, al que llamamos procesador de usuario, consta de cuatro elementos:

1. El controlador de la interfaz de usuario es responsable de interpretar los comandos del usuario a medida que... entrar y formatear los datos del resultado a medida que se envían al usuario.
2. El responsable del tratamiento de datos utiliza las restricciones de integridad y las autorizaciones definidas en el esquema conceptual global para comprobar si la consulta del usuario puede procesarse. Este componente, que se analiza en detalle en el capítulo 3, también es responsable de la autorización y otras funciones.
3. El optimizador y descomponedor de consultas globales determina una estrategia de ejecución para minimizar una función de coste y traduce las consultas globales a locales utilizando los esquemas conceptuales globales y locales, así como el directorio global. El optimizador de consultas globales es responsable, entre otras cosas, de generar la mejor estrategia para ejecutar operaciones de unión distribuida. Estos aspectos se abordan en el capítulo 4.
4. El monitor de ejecución distribuida coordina la ejecución distribuida de la solicitud del usuario. También se denomina gestor de transacciones distribuidas. Al ejecutar consultas de forma distribuida, los monitores de ejecución de los distintos sitios pueden comunicarse entre sí, y normalmente lo hacen. La funcionalidad del gestor de transacciones distribuidas se describe en el capítulo 5.

El segundo componente principal de un SGBD distribuido es el procesador de datos y consta de los tres elementos siguientes. Estos son aspectos que abordan los SGBD centralizados, por lo que no nos centraremos en ellos en este libro.

1. El optimizador de consultas local, que en realidad actúa como selector de ruta de acceso, es responsable de elegir la mejor ruta de acceso3 para acceder a cualquier elemento de datos.
2. El administrador de recuperación local es responsable de asegurarse de que la base de datos local permanece consistente incluso cuando ocurren fallas.
3. El procesador de soporte en tiempo de ejecución accede físicamente a la base de datos según los comandos físicos de la programación generada por el optimizador de consultas. Este procesador es la interfaz con el sistema operativo y contiene el administrador de búferes (o caché) de la base de datos , responsable del mantenimiento de los búferes de la memoria principal y de la gestión de los accesos a los datos.

Es importante tener en cuenta que nuestro uso de los términos "procesador de usuario" y "procesador de datos" no implica una división funcional similar a los sistemas cliente/servidor.

Estas divisiones son meramente organizativas y no se sugiere que deban ubicarse en máquinas diferentes. En sistemas peer-to-peer, se espera encontrar tanto los módulos de procesador de usuario como los de procesador de datos en cada máquina.

Sin embargo, puede haber "sitios de solo consulta" que solo tengan el procesador de usuario.

---

El término " ruta de acceso" se refiere a las estructuras de datos y los algoritmos utilizados para acceder a ellos. Una ruta de acceso típica, por ejemplo, es un índice en uno o más atributos de una relación.

### 1.6.4 Sistemas multibase de datos

Los sistemas multibase de datos (MDBS) representan el caso en el que los SGBD individuales son completamente autónomos y no tienen el concepto de cooperación; es posible que ni siquiera sepan de la existencia de los demás ni cómo comunicarse entre sí. Nos centramos, naturalmente, en los MDBS distribuidos, es decir, en los MDBS donde los SGBD participantes se ubican en diferentes ubicaciones. Muchos de los problemas que analizamos son comunes tanto a los MDBS de un solo nodo como a los distribuidos; en esos casos, simplemente utilizaremos el término MDBS sin calificarlo como de un solo nodo o distribuido. En la mayor parte de la literatura actual, se utiliza el término integración de bases de datos . Analizamos estos sistemas con más detalle en el capítulo 7. Sin embargo, observamos que existe una considerable variabilidad en el uso del término "multibase de datos" en la literatura. En este libro, lo utilizamos consistentemente como se definió anteriormente, lo que puede diferir de su uso en parte de la literatura existente.

Las diferencias en el nivel de autonomía entre los MDBS y los DBMS distribuidos también se reflejan en sus modelos arquitectónicos. La diferencia fundamental radica en la definición del esquema conceptual global. En el caso de los DBMS distribuidos con integración lógica, el esquema conceptual global define la visión conceptual de toda la base de datos, mientras que en el caso de los MDBS, representa únicamente el conjunto de algunas de las bases de datos locales que cada DBMS local desea compartir.

Los SGBD individuales pueden optar por poner algunos de sus datos a disposición de otros. Por lo tanto, la definición de una base de datos global difiere en los SGBD que en los SGBD distribuidos. En estos últimos, la base de datos global equivale a la unión de bases de datos locales, mientras que en los primeros es solo un subconjunto (posiblemente propio) de la misma unión. En un SGBD, el GSC (también denominado esquema mediado) se define integrando (posiblemente partes de) esquemas conceptuales locales.

El modelo arquitectónico basado en componentes de un MDBS distribuido difiere significativamente del de un DBMS distribuido, ya que cada sitio es un DBMS completo que gestiona una base de datos diferente. El MDBS proporciona una capa de software que se ejecuta sobre estos DBMS individuales y ofrece a los usuarios la posibilidad de acceder a diversas bases de datos (Fig. 1.12). Cabe destacar que en un MDBS distribuido, la capa del MDBS puede ejecutarse en varios sitios o puede haber un sitio central donde se ofrecen dichos servicios.

Tenga en cuenta también que, en lo que respecta a los DBMS individuales, la capa MDBS es simplemente otra aplicación que envía solicitudes y recibe respuestas.

Una arquitectura de implementación popular para MDBS es el enfoque mediador/encapsulador (Fig. 1.13). Un mediador es un módulo de software que aprovecha el conocimiento codificado sobre ciertos conjuntos o subconjuntos de datos para crear información para una capa superior de aplicaciones [Wiederhold 1992]. Por lo tanto, cada mediador realiza una función específica con interfaces claramente definidas. Al utilizar esta arquitectura para implementar un MDBS, cada módulo de la capa MDBS de la Fig. 1.12 se realiza como mediador.

Dado que los mediadores pueden construirse sobre otros mediadores, es posible construir una implementación en capas. El nivel del mediador implementa el GCS. Es este nivel el que gestiona las consultas de los usuarios a través del GCS y realiza la funcionalidad MDBS.

Los mediadores suelen operar utilizando un modelo de datos y un lenguaje de interfaz comunes. Para gestionar las posibles heterogeneidades de los SGBD de origen, se utilizan envoltorios.

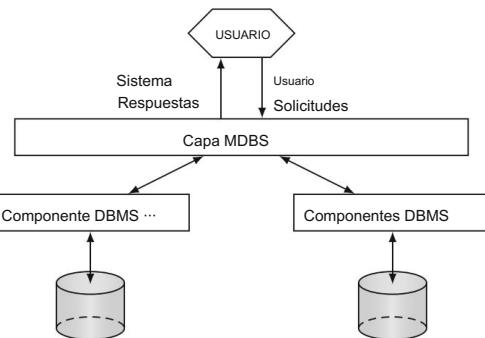


Fig. 1.12 Componentes de un MDBS

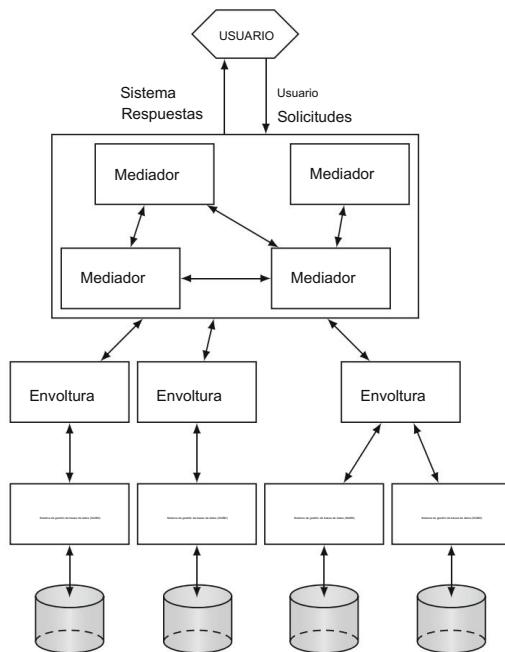


Fig. 1.13 Arquitectura del mediador/envoltorio

Se implementan cuya tarea es proporcionar un mapeo entre un DBMS de origen vista y la vista de los mediadores. Por ejemplo, si el SGBD de origen es un sistema relacional uno, pero las implementaciones del mediador están orientadas a objetos, las asignaciones requeridas Se establecen mediante los envoltorios. El rol y la función exactos de los mediadores difieren de una implementación a otra. En algunos casos, los mediadores no hacen más que... Traducción; estos se llaman mediadores "delgados". En otros casos, los envoltorios toman el control de ejecución de algunas de las funciones de consulta.

El conjunto de mediadores se puede considerar como una capa de middleware que proporciona servicios por encima de los sistemas fuente. El middleware ha sido objeto de un estudio considerable en la última década, y se han desarrollado sistemas de middleware muy sofisticados que proporcionan servicios avanzados para el desarrollo de aplicaciones distribuidas. Los mediadores que analizamos solo representan un subconjunto de la funcionalidad que ofrecen estos sistemas.

### 1.6.5 Computación en la nube

La computación en la nube ha impulsado un cambio significativo en la forma en que usuarios y organizaciones implementan aplicaciones escalables, en particular, las de gestión de datos. Esta visión abarca servicios fiables y bajo demanda, proporcionados a través de Internet (normalmente representados como la nube), con fácil acceso a recursos informáticos, de almacenamiento y de red prácticamente infinitos. Mediante interfaces web muy sencillas y con un pequeño coste incremental, los usuarios pueden externalizar tareas complejas, como el almacenamiento de datos, la gestión de bases de datos, la administración de sistemas o la implementación de aplicaciones, a grandes centros de datos operados por proveedores de la nube. De este modo, la complejidad de la gestión de la infraestructura de software y hardware se traslada de la organización de los usuarios al proveedor de la nube.

La computación en la nube es una evolución natural y una combinación de diferentes modelos de computación propuestos para soportar aplicaciones en la web: arquitecturas orientadas a servicios (SOA) para la comunicación de alto nivel de aplicaciones a través de servicios web, computación de utilidad para empaquetar recursos de computación y almacenamiento como servicios, tecnologías de clúster y virtualización para administrar grandes cantidades de recursos de computación y almacenamiento, y computación autónoma para permitir la autogestión de infraestructura compleja. La nube ofrece varios niveles de funcionalidad como:

- Infraestructura como servicio (IaaS): la entrega de una infraestructura informática (es decir, recursos informáticos, de red y de almacenamiento) como un servicio;
- Plataforma como servicio (PaaS): la entrega de una plataforma informática con herramientas de desarrollo y API como un servicio;
- Software como servicio (SaaS): la entrega de software de aplicación como un servicio;
- o
- Base de datos como servicio (DaaS): la entrega de una base de datos como servicio.

Lo que hace única a la computación en la nube es su capacidad de proporcionar y combinar todo tipo de servicios para satisfacer las necesidades de los usuarios. Desde un punto de vista técnico, el gran reto reside en dar soporte de forma rentable a la enorme escala de la infraestructura que debe gestionar una gran cantidad de usuarios y recursos con una alta calidad de servicio.

Llegar a un acuerdo sobre una definición precisa de computación en la nube es difícil, ya que existen diversas perspectivas (empresarial, de mercado, técnica, de investigación, etc.). Sin embargo, una buena definición práctica es que «la nube proporciona recursos y servicios bajo demanda a través de Internet, generalmente a la escala y con la fiabilidad de un centro de datos» [Grossman y Gu, 2009]. Esta definición refleja a la perfección el objetivo principal (proporcionar...

Recursos y servicios de demanda a través de Internet) y los principales requisitos para su soporte (a la escala y con la confiabilidad de un centro de datos). Dado que se accede a los recursos a través de servicios, todo se entrega como un servicio.

Por tanto, al igual que en el sector de servicios, esto permite a los proveedores de la nube proponer un modelo de precios de pago por uso, mediante el cual los usuarios solo pagan por los recursos que consumen.

Las principales funciones que ofrecen las nubes son: seguridad, gestión de directorios, gestión de recursos (aprovisionamiento, asignación, monitorización) y gestión de datos (almacenamiento, gestión de archivos, gestión de bases de datos, replicación de datos). Además, las nubes ofrecen soporte para la gestión de precios, contabilidad y acuerdos de nivel de servicio.

Las ventajas típicas de la computación en la nube son las siguientes:

- Costo. El costo para el cliente se puede reducir considerablemente, ya que no es necesario poseer ni administrar la infraestructura; la facturación se basa únicamente en el consumo de recursos. En cuanto al proveedor de la nube, el uso de una infraestructura consolidada y la distribución de costos para múltiples clientes reduce el costo de propiedad y operación. • Facilidad de acceso y uso. La nube oculta la complejidad de la infraestructura de TI y permite la transparencia en la ubicación y la distribución. De esta manera, los clientes pueden acceder a los servicios de TI en cualquier momento y desde cualquier lugar con conexión a Internet. • Calidad del servicio. La operación de la infraestructura de TI por parte de un proveedor especializado con amplia experiencia en la gestión de infraestructuras de gran tamaño (incluida su propia infraestructura) aumenta la calidad del servicio y la eficiencia operativa.

- Innovación. El uso de herramientas y aplicaciones de vanguardia que ofrece la nube fomenta las prácticas modernas, aumentando así la capacidad de innovación de los clientes.

- Elasticidad. La capacidad de escalar recursos horizontal y verticalmente de forma dinámica para adaptarse a las condiciones cambiantes es una ventaja fundamental. Esto se logra generalmente mediante la virtualización de servidores, una tecnología que permite que varias aplicaciones se ejecuten en el mismo equipo físico como máquinas virtuales (VM), es decir, como si se ejecutaran en equipos físicos distintos. Los clientes pueden entonces requerir instancias de computación como VM y conectar recursos de almacenamiento según sea necesario.

Sin embargo, también existen desventajas que es necesario comprender bien antes de migrar a la nube. Estas desventajas son similares a las de externalizar aplicaciones y datos a una empresa externa.

- Dependencia del proveedor. Los proveedores de servicios en la nube tienden a retener a sus clientes mediante software propietario, formatos propietarios o altos costos de transferencia de datos salientes, lo que dificulta la migración de servicios en la nube.

Pérdida de control. Los clientes pueden perder el control de la gestión sobre operaciones críticas. como el tiempo de inactividad del sistema, por ejemplo, para realizar una actualización de software.

- Seguridad. Dado que los datos en la nube de un cliente son accesibles desde cualquier lugar de Internet, los ataques de seguridad pueden comprometer los datos de la empresa. La seguridad en la nube se puede mejorar mediante funciones avanzadas, como la nube privada virtual (VPN), pero puede resultar complejo integrarla con la política de seguridad de la empresa.

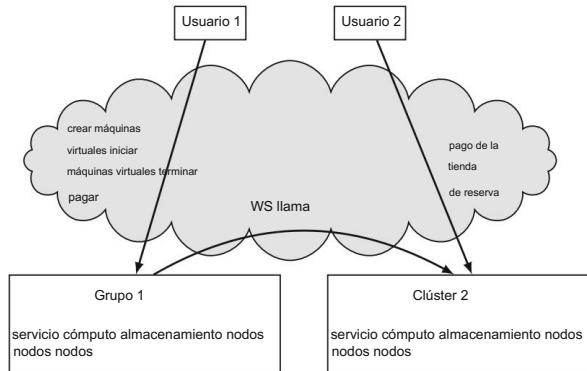


Fig. 1.14 Arquitectura de nube simplificada

- Costos ocultos. Personalización de aplicaciones para que estén listas para la nube mediante SaaS. PaaS puede generar costos de desarrollo significativos.

No existe una arquitectura de nube estándar y probablemente nunca la habrá, ya que los distintos proveedores de nube ofrecen distintos servicios (IaaS, PaaS, SaaS, etc.) de distintas maneras (pública, privada, virtual privada, etc.) según sus modelos de negocio. Por lo tanto, en esta sección, analizamos una arquitectura de nube simplificada, con énfasis en la gestión de bases de datos.

Una nube suele ser multisitio (Fig. 1.14), es decir, está compuesta por varios sitios (o centros de datos) distribuidos geográficamente, cada uno con sus propios recursos y datos. Los principales proveedores de la nube dividen el mundo en varias regiones, cada una con varios sitios. Esto se debe a tres razones principales. En primer lugar, el acceso con baja latencia en la región del usuario, ya que las solicitudes pueden dirigirse al sitio más cercano. En segundo lugar, la replicación de datos entre sitios en diferentes regiones proporciona alta disponibilidad, en particular, resistencia a fallos catastróficos (del sitio). En tercer lugar, algunas normativas nacionales que protegen la privacidad de los datos de los ciudadanos obligan a los proveedores de la nube a ubicar centros de datos en su región (p. ej., Europa). La transparencia multisitio suele ser una opción predeterminada, por lo que la nube parece "centralizada" y el proveedor de la nube puede optimizar la asignación de recursos a los usuarios. Sin embargo, algunos proveedores de la nube (p. ej., Amazon y Microsoft) hacen que sus sitios sean visibles para los usuarios (o desarrolladores de aplicaciones). Esto permite elegir un centro de datos específico para instalar una aplicación con su base de datos, o implementar una aplicación muy grande en varios sitios que se comunican mediante servicios web (WS). Por ejemplo, en la Fig. 1.14, podríamos imaginar que el Cliente 1 primero se conecta a una aplicación en el Centro de Datos 1, que a su vez llamaría a una aplicación en el Centro de Datos 2 usando WS.

La arquitectura de un sitio en la nube (centro de datos) suele ser de tres niveles. El primer nivel consta de clientes web que acceden a los servidores web en la nube, generalmente a través de un enrutador o balanceador de carga en el sitio. El segundo nivel consta de servidores web/de aplicaciones que dan soporte a los clientes y proporcionan lógica de negocio. El tercer nivel consta de servidores de bases de datos. Puede haber otros tipos de servidores, por ejemplo, servidores de caché entre los servidores de aplicaciones y los servidores de bases de datos. Por lo tanto, la arquitectura en la nube proporciona dos

Niveles de distribución: distribución geográfica entre sitios que utilizan una WAN y dentro de un sitio, distribución entre los servidores, generalmente en un clúster de computadoras. Las técnicas utilizadas en el primer nivel son las de un SGBD distribuido geográficamente, mientras que las técnicas utilizadas en el segundo nivel son las de un SGBD paralelo.

La computación en la nube fue diseñada originalmente por gigantes de la web para ejecutar sus aplicaciones a gran escala en centros de datos con miles de servidores. Los sistemas de big data (Cap. 10) y los sistemas NoSQL/NewSQL (Cap. 11) abordan específicamente los requisitos de dichas aplicaciones en la nube mediante técnicas de gestión distribuida de datos. Con la llegada de las soluciones SaaS y PaaS, los proveedores de la nube también necesitan ofrecer pequeñas aplicaciones a un gran número de clientes, denominados inquilinos, cada uno con su propia (pequeña) base de datos a la que acceden sus usuarios. Dedicar un servidor a cada inquilino supone un desperdicio de recursos de hardware. Para reducir el desperdicio de recursos y los costes operativos, los proveedores de la nube suelen compartir recursos entre los inquilinos mediante una arquitectura multiinquilino, en la que un único servidor puede alojar a varios inquilinos. Los distintos modelos multiinquilino ofrecen diferentes equilibrios entre rendimiento, aislamiento (tanto de seguridad como de rendimiento) y complejidad de diseño. Un modelo sencillo utilizado en IaaS es el uso compartido de hardware, que suele lograrse mediante la virtualización de servidores, con una máquina virtual para cada base de datos y sistema operativo del inquilino. Este modelo proporciona un sólido aislamiento de seguridad.

Sin embargo, la utilización de recursos es limitada debido a instancias DBMS redundantes (una por VM) que no cooperan ni realizan una gestión de recursos independiente.

En el contexto de SaaS, PaaS o DaaS, podemos distinguir tres modelos principales de bases de datos multiinquilino con un aumento del uso compartido de recursos y del rendimiento a expensas de un menor aislamiento y una mayor complejidad.

Servidor DBMS compartido. En este modelo, los usuarios comparten un servidor con una instancia de DBMS, pero cada usuario tiene una base de datos diferente. La mayoría de los DBMS admiten varias bases de datos en una sola instancia. Por lo tanto, este modelo se puede implementar fácilmente con un DBMS. Proporciona un aislamiento sólido a nivel de base de datos y es más eficiente que el hardware compartido, ya que la instancia de DBMS tiene control total sobre los recursos de hardware. Sin embargo, administrar cada base de datos por separado puede resultar en una gestión ineficiente de los recursos.

- Base de datos compartida. En este modelo, los usuarios comparten una base de datos, pero cada uno tiene su propio esquema y tablas. La consolidación de la base de datos suele proporcionarse mediante una capa de abstracción adicional en el SGBD. Algunos SGBD (p. ej., Oracle) implementan este modelo utilizando una única base de datos contenedora que aloja varias bases de datos. Ofrece un buen uso de recursos y aislamiento a nivel de esquema.

Sin embargo, con muchos (miles) de inquilinos por servidor, hay una gran cantidad de tablas pequeñas, lo que genera mucha sobrecarga.

- Tablas compartidas. En este modelo, los usuarios comparten una base de datos, un esquema y tablas. Para distinguir las filas de los diferentes inquilinos en una tabla, suele haber una columna adicional llamada "tenant\_id". Si bien se comparte mejor los recursos (p. ej., la memoria caché), el aislamiento es menor, tanto en seguridad como en rendimiento. Por ejemplo, los clientes más grandes tendrán más filas en tablas compartidas, lo que perjudica el rendimiento de los clientes más pequeños.

## 1.7 Notas bibliográficas

No existen muchos libros sobre sistemas de gestión de bases de datos distribuidos. Los dos primeros, de Ceri y Pelagatti [1983] y Bell y Grimson [1992], están agotados. Un libro más reciente, de Rahimi y Haug [2010], abarca algunos de los temas clásicos que también se tratan en este libro. Además, casi todos los libros sobre bases de datos incluyen un capítulo sobre sistemas de gestión de bases de datos distribuidos.

Los sistemas pioneros INGRES distribuido y SDD-1 se analizan en [Stonebraker y Neuhold 1977] y [Wong 1977], respectivamente.

El diseño de bases de datos se analiza de forma introductoria en [Levin y Morgan, 1975] y de forma más exhaustiva en [Ceri et al., 1987]. En [Dowdy y Foster, 1982] se ofrece un resumen de los algoritmos de distribución de archivos . La gestión de directorios no se ha considerado en detalle en la comunidad científica, pero se pueden encontrar técnicas generales en [Chu y Nahouraii, 1975] y [Chu, 1976]. En [Sacco y Yao, 1982] se puede encontrar un resumen de las técnicas de procesamiento de consultas . Los algoritmos de control de concurrencia se revisan en [Bernstein y Goodman, 1981] y [Bernstein et al.

1987]. La gestión de interbloqueos también ha sido objeto de una extensa investigación; un artículo introductorio es [Isloor y Marsland 1980] y un artículo ampliamente citado es [Obermack 1982]. Para la detección de interbloqueos, buenos estudios son [Knapp 1987] y [Elmagarmid 1986]. La fiabilidad es uno de los temas tratados en [Gray 1979], que es uno de los artículos de referencia en el campo. Otros artículos importantes sobre este tema son [Verhofstadt 1978] y [Härder y Reuter 1983]. [Gray 1979] es también el primer artículo que trata los problemas de soporte del sistema operativo para bases de datos distribuidas; el mismo tema se aborda en [Stonebraker 1981]. Desafortunadamente, ambos artículos enfatizan los sistemas de bases de datos centralizados. Un muy buen estudio temprano de sistemas de múltiples bases de datos es por Sheth y Larson [1990]; Wiederhold [1992] propone el enfoque mediador/encapsulador para los sistemas de bases de datos multiusuario (MDBS). La computación en la nube ha sido tema de numerosos libros recientes; quizás [Agrawal et al. 2012] sea un buen punto de partida y [Cusumano 2010] una buena y breve descripción general. La arquitectura que utilizamos en la sección 1.6.5 proviene de [Agrawal et al. 2012]. Diferentes modelos multiusuario en entornos de nube se analizan en [Curino et al. 2011] y [Agrawal et al. 2012].

Se han presentado diversas propuestas de marcos arquitectónicos. Entre las interesantes se incluyen la extensión bastante detallada de Schreiber del marco ANSI/SPARC, que intenta dar cabida a la heterogeneidad de los modelos de datos [Schreiber 1977], y la propuesta de Mohan y Yeh [1978]. Como era de esperar, estas se remontan a los inicios de la introducción de la tecnología de SGBD distribuidos.

La arquitectura detallada del sistema, por componentes, que se muestra en la Fig. 1.11 se deriva de [Rahimi 1987]. Una alternativa a la clasificación que proporcionamos en la Fig. 1.8 se puede encontrar en [Sheth y Larson 1990].

El libro de Agrawal et al. [2012] ofrece una muy buena presentación de los desafíos y conceptos de la gestión de datos en la nube, incluidas las transacciones distribuidas, los sistemas de big data y las bases de datos multiinquilino.

## Capítulo 2

# Diseño de bases de datos distribuidas y paralelas



Un diseño típico de base de datos es un proceso que parte de un conjunto de requisitos y da como resultado la definición de un esquema que define el conjunto de relaciones. El diseño de la distribución parte de este esquema conceptual global (ECG) y consta de dos tareas: partición (fragmentación) y asignación. Algunas técnicas combinan estas dos tareas en un solo algoritmo, mientras que otras las implementan en dos tareas separadas, como se muestra en la figura 2.1. El proceso suele utilizar información auxiliar, que se muestra en la figura, aunque parte de esta información es opcional (de ahí las líneas discontinuas).

Las principales razones y objetivos de la fragmentación en sistemas de gestión de bases de datos (SGBD) distribuidos y paralelos son ligeramente diferentes. En el caso de los primeros, la razón principal es la localización de los datos. En la medida de lo posible, se busca que las consultas accedan a los datos en un único sitio para evitar el costoso acceso remoto a los datos. Una segunda razón importante es que la fragmentación permite la ejecución simultánea de varias consultas (mediante paralelismo entre consultas). La fragmentación de las relaciones también permite la ejecución paralela de una misma consulta al dividirla en un conjunto de subconsultas que operan sobre fragmentos, lo que se conoce como paralelismo intraconsulta. Por lo tanto, en los SGBD distribuidos, la fragmentación puede reducir potencialmente el costoso acceso remoto a los datos y aumentar el paralelismo entre consultas e intraconsultas.

En los DBMS paralelos, la localización de datos no es una gran preocupación ya que el costo de comunicación entre nodos es mucho menor que en los DBMS geodistribuidos.

Lo más preocupante es el equilibrio de carga, ya que queremos que cada nodo del sistema realice aproximadamente la misma cantidad de trabajo. De lo contrario, existe el riesgo de que todo el sistema se colapse, ya que uno o varios nodos realizan la mayor parte del trabajo, mientras que muchos permanecen inactivos. Esto también aumenta la latencia de las consultas y transacciones, ya que deben esperar a que finalicen estos nodos sobrecargados. El paralelismo entre consultas y dentro de ellas es importante, como se explica en el capítulo 8, aunque algunos sistemas modernos de big data (capítulo 10) prestan más atención al paralelismo entre consultas.

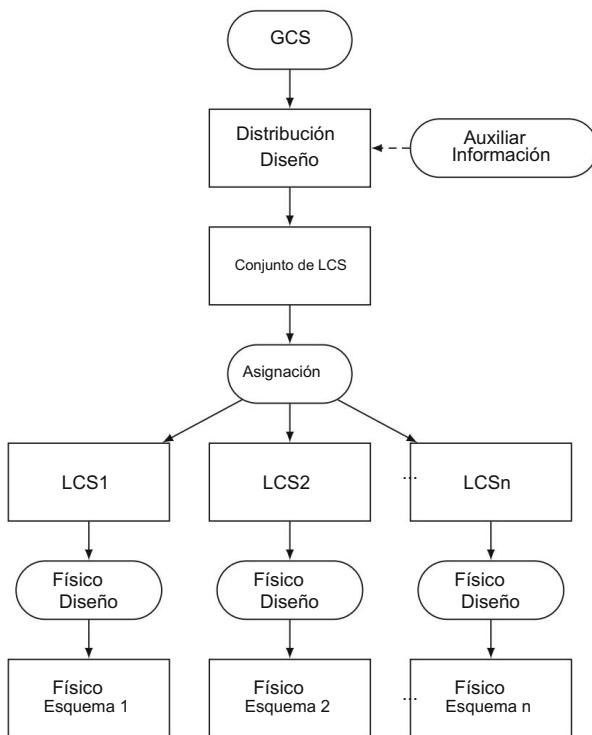


Fig. 2.1 Proceso de diseño de distribución

La fragmentación es importante para el rendimiento del sistema, pero también plantea dificultades en los SGBD distribuidos. No siempre es posible localizar completamente las consultas y transacciones para acceder a los datos solo en un sitio; estas se denominan consultas y transacciones distribuidas. Su procesamiento implica una reducción del rendimiento debido, por ejemplo, a la necesidad de realizar uniones distribuidas y al coste de la asignación de transacciones distribuidas (véase el capítulo 5). Una forma de superar esta reducción en las consultas de solo lectura es replicar los datos en varios sitios (véase el capítulo 6), pero esto agrava aún más la sobrecarga de las transacciones distribuidas. Un segundo problema está relacionado con el control semántico de datos, en concreto con la comprobación de integridad. Como resultado de la fragmentación, los atributos que participan en una restricción (véase el capítulo 3) pueden descomponerse en diferentes fragmentos que se asignan a diferentes sitios. En este caso, la comprobación de integridad implica una ejecución distribuida, lo cual resulta costoso. En el siguiente capítulo, abordaremos el tema del control de datos distribuidos. Por lo tanto, el reto resi-

---

Un pequeño detalle terminológico es el uso de los términos «fragmentación» y «particionamiento»: en los SGBD distribuidos, el término «fragmentación» se usa con mayor frecuencia, mientras que en los SGBD paralelos se prefiere el «particionamiento de datos». No preferimos uno sobre el otro y los usaremos indistintamente en este capítulo y en este libro.

The diagram illustrates the decomposition of a global database into fragments. It consists of four tables:

- PROYECTO** (Project):
 

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
Instrumentación P1		150000 Montreal	
Desarrollo de base de datos P2.	135000 Nueva York		
P3 CAD/CAM 250000 Nueva York			
Mantenimiento P4		310000 Paris	
- PAGAR** (Pay):
 

TÍTULO	SAL
Ing. Elect.	40000
Anal sistémico.	34000
Ingeniería mecánica	27000
Programador	24000
- ASG** (Assignment):
 

ENO PNO	RESP	DUR
E1	Gerente P1	12
E2	Analista P1	24
E3	E2 P2	6
E4	Consultor E3 P3	10
E5	Ingeniero E3 P4	48
E6	Programador E4 P2	18
E7	Gerente 24	
E8	Gerente 48	
E9	Ingeniero 36	
E10	Gerente 40	
- A detailed view of the ASG fragment, showing the mapping between employees (ENO) and their assigned roles (PNO) and duration (DUR). The data is as follows:
 

ENO	PNO	RESP	DUR
E1	Gerente P1		12
E2	Analista P1		24
E3	E2 P2	6	
E4	Consultor E3 P3	10	
E5	Ingeniero E3 P4	48	
E6	Programador E4 P2	18	
E7	Gerente 24		
E8	Gerente 48		
E9	Ingeniero 36		
E10	Gerente 40		

Fig. 2.2 Base de datos de ejemplo

los datos de tal manera que la mayoría de las consultas y transacciones de los usuarios sean locales en un sitio, minimizando consultas y transacciones distribuidas.

Nuestra discusión en este capítulo seguirá la metodología de la Figura 2.1:

Primero analicemos la fragmentación de una base de datos global (Sección 2.1) y luego analicemos cómo para asignar estos fragmentos entre los sitios de una base de datos distribuida (Sección 2.2). En

En esta metodología, la unidad de distribución/asignación es un fragmento. También existen enfoques que combinan los pasos de fragmentación y asignación y los discutimos

Estos se tratan en la Sección 2.3. Finalmente, analizamos técnicas que se adaptan a los cambios en la base de datos y la carga de trabajo del usuario en la Secc. 2.4.

En este capítulo y a lo largo del libro, utilizamos la base de datos de ingeniería Introducido en el capítulo anterior. La Figura 2.2 muestra una instancia de esta base de datos.

## 2.1 Fragmentación de datos

Las tablas relacionales se pueden particionar horizontal o verticalmente. La base de fragmentación horizontal es el operador de selección donde los predicados de selección determinan la fragmentación, mientras que la fragmentación vertical se realiza mediante del operador del proyecto. La fragmentación puede, por supuesto, estar anidada. Si las anidaciones son de diferentes tipos, se obtiene una fragmentación híbrida.

Ejemplo 2.1 La figura 2.3 muestra la relación PROJ de la figura 2.2 dividida horizontalmente en dos fragmentos: PROJ1 contiene información sobre los proyectos cuyos presupuestos son menos de \$200,000, mientras que PROJ2 almacena información sobre proyectos con mayor tamaño. presupuestos.

PROY1

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P1	Instrumentación 150000	Montreal	
	Desarrollo de base de datos P2.	135000 Nueva York	

PROY2

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P3	CAD/CAM	255000 Nueva York	
	Mantenimiento P4	310000 París	

Fig. 2.3 Ejemplo de partición horizontal

PROY1

PRESUPUESTO DE LA PNO
P1 150000
P2 135000
250000 pesos filipinos
P4 310000

PROY2

PNO	Nombre de usuario	LOCALIDAD
P1	Instrumentación P2	Montreal
	Desarrollo de Base de Datos.	Nueva York
P3	CAD/CAM P4	Nueva York
	Mantenimiento	París

Fig. 2.4 Ejemplo de partición vertical

Ejemplo 2.2 La figura 2.4 muestra la relación PROJ de la figura 2.2 dividida verticalmente en dos fragmentos: PROJ1 y PROJ2. PROJ1 contiene solo la información sobre los presupuestos de los proyectos, mientras que PROJ2 contiene los nombres y las ubicaciones de los proyectos. Es importante tener en cuenta que la clave principal de la relación (PNO) está incluida en ambos fragmentos.

La fragmentación horizontal es más frecuente en la mayoría de los sistemas, en particular en DBMS paralelos (donde la literatura prefiere el término fragmentación). La razón para la prevalencia de la fragmentación horizontal es el paralelismo intraconsulta<sup>2</sup> que la mayoría de las plataformas de big data recientes abogan por ello. Sin embargo, la fragmentación vertical ha sido utilizado con éxito en DBMS paralelos de almacenamiento en columnas, como MonetDB y Vertica, para aplicaciones analíticas, que normalmente requieren acceso rápido a unos pocos atributos.

Las técnicas de fragmentación sistemática que analizamos en este capítulo garantizan que la base de datos no sufra cambios semánticos durante la fragmentación, como la pérdida de datos como consecuencia de la fragmentación. Por lo tanto, es necesario poder... para discutir sobre la completitud y reconstrucción. En el caso de la horizontal, la fragmentación y la disyunción de los fragmentos también pueden ser una propiedad deseable (a menos que deseamos explícitamente replicar tuplas individuales (como discutiremos más adelante)).

1. Completitud. Si una instancia de relación R se descompone en fragmentos FR = {R1, R2, ..., Rn}, cada elemento de datos que está en R también se puede encontrar en uno o más de R's. Esta propiedad, que es idéntica a la propiedad de descomposición sin pérdidas de

---

<sup>2</sup>En este capítulo, utilizamos los términos "consulta" y "transacción" indistintamente, ya que ambos se refieren a la carga de trabajo del sistema, que constituye uno de los principales insumos para el diseño de la distribución, como se destaca en el capítulo 1, y como se analizará en detalle en el capítulo 5, las transacciones proporcionan garantías adicionales y, por lo tanto, sus gastos generales son mayores y los incorporaremos a nuestra discusión cuando sea necesario.

## 2.1 Fragmentación de datos

La normalización (Apéndice A) también es importante en la fragmentación, ya que garantiza que los datos de una relación global se asignen a fragmentos sin pérdida alguna. Cabe destacar que, en el caso de la fragmentación horizontal, el elemento suele referirse a una tupla, mientras que en el caso de la fragmentación vertical, se refiere a un atributo.

2. Reconstrucción. Si una relación R se descompone en fragmentos  $FR = \{R_1, R_2, \dots, R_n\}$ , debería ser posible definir un operador relacional tal que

$$R = R_1 \quad R_2 \quad FR$$

El operador será diferente para las distintas formas de fragmentación; sin embargo, es importante que pueda identificarse. La reconstructibilidad de la relación a partir de sus fragmentos garantiza que se conserven las restricciones definidas en los datos en forma de dependencias.

3. Disjunción. Si una relación R se descompone horizontalmente en fragmentos  $FR = \{R_1, R_2, \dots, R_n\}$  y el dato di está en  $R_j$ , no está en ningún otro fragmento  $R_k$  ( $k = j$ ). Este criterio garantiza que los fragmentos horizontales sean disjuntos.

Si la relación R se descompone verticalmente, sus atributos de clave primaria suelen repetirse en todos sus fragmentos (para su reconstrucción). Por lo tanto, en caso de partición vertical, la disyunción se define únicamente en los atributos de clave no primaria de una relación.

### 2.1.1 Fragmentación horizontal

Como explicamos anteriormente, la fragmentación horizontal partitiona una relación a lo largo de sus tuplas. Por lo tanto, cada fragmento contiene un subconjunto de las tuplas de la relación. Existen dos versiones de la fragmentación horizontal: primaria y derivada. La fragmentación horizontal primaria de una relación se realiza mediante predicados definidos en ella. La fragmentación horizontal derivada, por otro lado, es la partición de una relación resultante de la definición de predicados en otra relación.

Más adelante en esta sección, consideraremos un algoritmo para realizar ambas fragmentaciones. Sin embargo, primero investigaremos la información necesaria para realizar la fragmentación horizontal.

#### 2.1.1.1 Requisitos de información auxiliar

La información de la base de datos requerida se refiere al esquema conceptual global, principalmente a cómo se conectan las relaciones entre sí, especialmente mediante uniones. Una forma de capturar esta información es modelar explícitamente las relaciones de unión de clave primaria y clave externa en un grafo de uniones. En este grafo, cada relación  $R_i$  se representa como un vértice y existe una arista dirigida  $L_k$  de  $R_i$  a  $R_j$  si existe una unión de igualdad de clave primaria y clave externa de  $R_i$  a  $R_j$ . Cabe destacar que  $L_k$  también representa una relación un-

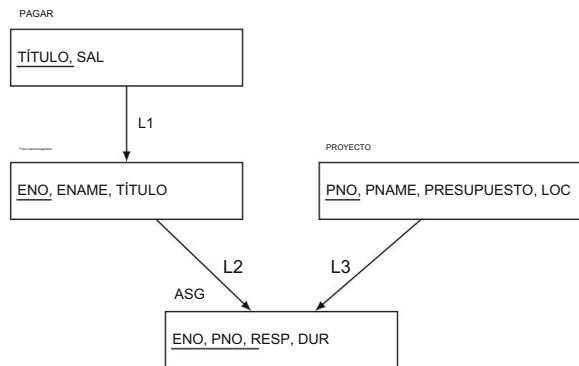


Fig. 2.5 Gráfico de unión que representa relaciones entre relaciones

Ejemplo 2.3. La Figura 2.5 muestra las aristas entre las relaciones de la base de datos de la Figura 2.2. Nótese que la dirección de la arista muestra una relación de uno a muchos. Por ejemplo, para cada puesto hay varios empleados con ese puesto; por lo tanto, existe una arista entre las relaciones PAY y EMP. De igual forma, la relación de muchos a muchos entre las relaciones EMP y PROJ se expresa con dos aristas en la relación ASG.

La relación en la cola de una arista se denomina origen de la arista y la relación en la cabeza, destino. Definamos dos funciones: origen y destino, que proporcionan asignaciones del conjunto de aristas al conjunto de relaciones. Considerando L1 de la Fig. 2.5,  $\text{origen}(L1) = \text{PAGAR}$  y destino ( $L1 = \text{EMP}$ ).

Además, la cardinalidad de cada relación R denotada por  $\text{card}(R)$  es útil en fragmentación horizontal.

Estos enfoques también utilizan la información de la carga de trabajo, es decir, las consultas que se ejecutan en la base de datos. Los predicados utilizados en las consultas de usuario son de especial importancia. En muchos casos, puede que no sea posible analizar la carga de trabajo completa, por lo que el diseñador normalmente se centra en las consultas importantes. Existe una conocida regla general del "80/20" en informática que también se aplica en este caso: el 20% más común de las consultas de usuario representa el 80% del total de accesos a los datos, por lo que centrarse en ese 20% suele ser suficiente para obtener una fragmentación que mejore la mayoría de los accesos a bases de datos distribuidas.

En este punto, nos interesa determinar predicados simples. Dada una relación  $R(A_1, A_2, \dots, A_n)$ , donde  $A_i$  es un atributo definido sobre el dominio  $D_i$ , un predicado simple  $p_j$  definido en  $R$  tiene la forma

$$p_j : A_i \theta \text{Valor}$$

donde  $\theta \in \{=, <, =, \leq, >, \geq\}$  y Valor se elige del dominio de  $A_i$  (Valor  $\in D_i$ ). Usamos  $P_{ri}$  para denotar el conjunto de todos los predicados simples definidos en una relación  $R_i$ . Los miembros de  $P_{ri}$  se denotan por  $p_{ij}$ .

## 2.1 Fragmentación de datos

Ejemplo 2.4 Dada la instancia de relación PROJ de la Fig. 2.2,

PNAME = "Mantenimiento" y PRESUPUESTO  $\leq$  200000

es un predicado simple.

Las consultas de usuario suelen incluir predicados más complejos, que son combinaciones booleanas de predicados simples. Una de estas combinaciones, denominada predicado mintérmino, es la conjunción de predicados simples. Dado que siempre es posible transformar una expresión booleana en la forma normal conjuntiva, el uso de predicados mintérmino en los algoritmos de diseño no implica pérdida de generalidad.

Dado un conjunto  $P_{ri} = \{pi_1, pi_2, \dots, pi_m\}$  de predicados simples para la relación  $R_i$ , El conjunto de predicados minterm  $M_i = \{mi_1, mi_2, \dots, mi_z\}$  se define como

$$M_i = \{mij = pik \quad p \quad ik\}, \quad 1 \leq k \leq m, \quad 1 \leq j \leq z$$

donde  $p \quad y \quad pik$  o  $p \quad = \neg pik$ . Por lo tanto, cada predicado simple puede aparecer en un predicado de mintérmino en su forma natural o en su forma negada.

La negación de un predicado es sencilla para predicados de igualdad de la forma Atributo = Valor. Para predicados de desigualdad, la negación debe tratarse como el complemento. Por ejemplo, la negación del predicado simple Atributo  $\leq$  Valor es Atributo  $>$  Valor. Existen problemas teóricos para encontrar el complemento en conjuntos infinitos, y también el problema práctico de que el complemento puede ser difícil de definir. Por ejemplo, si se definen dos predicados simples de la forma Límite\_inferior  $\leq$  Atributo\_1 y Atributo\_1  $\leq$  Límite\_superior, sus complementos son  $\neg(Límite\_inferior \leq Atributo\_1)$  y  $\neg(Atributo\_1 \leq Límite\_superior)$ . Sin embargo, los dos predicados simples originales pueden escribirse como Límite inferior  $\leq$  Atributo\_1  $\leq$  Límite superior con un complemento  $\neg(Límite\_inferior \leq Atributo\_1 \leq Límite\_superior)$ , que puede no ser fácil de definir. Por lo tanto, nos limitamos a predicados simples.

Ejemplo 2.5. Considere la relación PAY de la figura 2.2. A continuación, se presentan algunos predicados simples que pueden definirse en PAY.

p1 : TÍTULO = "Ing. Elect."

p2 : TÍTULO = "Sist. Anal."

p3 : TÍTULO = "Ingeniería mecánica"

p4 : TÍTULO = "Programador"

p5 : SAL  $\leq$  30000

Los siguientes son algunos de los predicados minterm que se pueden definir basándose en estos predicados simples.

m1 : TÍTULO = "Ing. Elect." SAL ≤ 30000  
 m2 : TÍTULO = "Ing. Elect." SAL > 30000  
 m3 :  $\neg(TÍTULO = \text{"Ing. Elect."})$  SAL ≤ 30000  
 m4 :  $\neg(TÍTULO = \text{"Ing. Elect."})$  SAL > 30000  
 m5 : TÍTULO = "Programador" SAL ≤ 30000  
 m6 : TÍTULO = "Programador" SAL > 30000

Ésta es sólo una muestra representativa, no el conjunto completo de predicados minterm.

Además, algunos de los mintérminos pueden carecer de sentido dada la semántica de la relación PAY, en cuyo caso se eliminan del conjunto. Finalmente, cabe destacar que estas son versiones simplificadas de los mintérminos. La definición de mintérmino requiere que cada predicado esté en un mintérmino en su forma natural o negada. Por lo tanto, m1, por ejemplo, debería escribirse como

m1 : TÍTULO = "Ing. Elect." TÍTULO = "Anal. Sist." TÍTULO = "Ing. Mec."  
 TÍTULO = "Programador" SAL ≤ 30000

Esto claramente no es necesario y utilizamos el formato simplificado.

También necesitamos información cuantitativa sobre la carga de trabajo:

1. Selectividad de mintérminos: número de tuplas de la relación que satisfacen un predicado de mintérmino dado. Por ejemplo, la selectividad de  $m^2$  del Ejemplo 2.5 es 0,25, ya que una de las cuatro tuplas de PAY satisface  $m^2$ . Denotamos la selectividad de un mintérmino mi como sel(mi).
2. Frecuencia de acceso: frecuencia con la que las aplicaciones de usuario acceden a los datos. Si  $Q = \{q_1, q_2, \dots, q_n\}$  es un conjunto de consultas de usuario, acc(qi) indica la frecuencia de acceso de la consulta qi en un período determinado.

Tenga en cuenta que las frecuencias de acceso de minterm se pueden determinar a partir de la frecuencia de consulta. Nos referimos a la frecuencia de acceso de un minterm mi como acc(mi).

### 2.1.1.2 Fragmentación horizontal primaria

La fragmentación horizontal primaria se aplica a las relaciones sin aristas entrantes en el grafo de unión y se ejecuta mediante los predicados definidos en dicha relación. En nuestros ejemplos, las relaciones PAY y PROJ están sujetas a fragmentación horizontal primaria, y EMP y ASG a fragmentación horizontal derivada.

En esta sección, nos centramos en la fragmentación horizontal primaria y dedicamos la siguiente sección a la fragmentación horizontal derivada.

Una fragmentación horizontal primaria se define mediante una operación de selección en las relaciones de origen de un esquema de base de datos. Por lo tanto, dada la relación R, su fragmentación horizontal...

## 2.1 Fragmentación de datos

Los fragmentos se dan por

$$R_i = \sigma F_i(R), 1 \leq i \leq w$$

Donde  $F_i$  es la fórmula de selección utilizada para obtener el fragmento  $R_i$  (también llamado predicado de fragmentación). Nótese que si  $F_i$  está en forma normal conjuntiva, es un predicado mintérmino ( $m_i$ ). El algoritmo requiere que  $F_i$  sea un predicado mintérmino.

Ejemplo 2.6 La descomposición de la relación PROJ en los fragmentos horizontales PROJ1 y PROJ2 en el Ejemplo 2.1 se define de la siguiente manera<sup>3</sup>:

$$PROY1 = \sigma PRESUPUESTO \leq 200000(PROJ)$$

$$PROY2 = \sigma PRESUPUESTO > 200000(PROY)$$

El ejemplo 2.6 demuestra uno de los problemas de la partición horizontal. Si el dominio de los atributos que participan en las fórmulas de selección es continuo e infinito, como en el ejemplo 2.6, resulta bastante difícil definir el conjunto de fórmulas  $F = \{F_1, F_2, \dots, F_n\}$  que fragmentaría la relación correctamente. Una posible solución es definir rangos como se hizo en el ejemplo 2.6. Sin embargo, siempre existe el problema de gestionar los dos extremos. Por ejemplo, si se insertara una nueva tupla con un valor de PRESUPUESTO de, por ejemplo, \$600,000 en PROJ, habría que revisar la fragmentación para decidir si la nueva tupla debe ir a PROJ2 o si es necesario revisar los fragmentos y definir un nuevo fragmento como

$$PROJ2 = \sigma 200000 < PRESUPUESTO \quad PRESUPUESTO \leq 400000(PROJ)$$

$$PROY3 = \sigma PRESUPUESTO > 400000(PROY)$$

Ejemplo 2.7. Considere la relación PROJ de la Fig. 2.2. Podemos definir los siguientes fragmentos horizontales según la ubicación del proyecto. Los fragmentos resultantes se muestran en la Fig. 2.6.

$$PROJ1 = \sigma LOC = "Montreal"(PROJ)$$

$$PROJ2 = \sigma LOC = "Nueva York"(PROJ)$$

$$PROJ3 = \sigma LOC = "París"(PROJ)$$

Ahora podemos definir un fragmento horizontal con más precisión. Un fragmento horizontal  $R_i$  de la relación  $R$  consta de todas las tuplas de  $R$  que satisfacen un predicado mintérmino  $m_i$ .

---

<sup>3</sup>Suponemos que la no negatividad de los valores de PRESUPUESTO es una característica de la relación impuesta por una restricción de integridad. De lo contrario, también debe incluirse en P un predicado simple de la forma  $0 \leq PRESUPUESTO$ . Asumimos que esto es cierto en todos los ejemplos y análisis de este capítulo.

PROY1

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P1	Instrumentación	150000	Montreal

PROY2

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P2	Desarrollo de bases de datos.	135000 Nueva York	
P3 CAD CAM		255000 Nueva York	
P4	Mantenimiento	310000	París

PROY3

PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P4	Mantenimiento	310000	París

Fig. 2.6 Fragmentación horizontal primaria de la relación PROJ

Por lo tanto, dado un conjunto de predicados minterm M, hay tantos fragmentos horizontales de la relación R, ya que hay predicados minterm. Este conjunto de fragmentos horizontales también es comúnmente denominado el conjunto de fragmentos minterm.

Queremos que el conjunto de predicados simples que forman los predicados minterm sea completo y mínimo. Un conjunto de predicados simples P r se dice que es completo si y sólo si existe una probabilidad igual de acceso por parte de cada aplicación a cualquier tupla perteneciente a cualquier fragmento minterm que esté definido según P r.

4

Ejemplo 2.8 Considere la fragmentación de la relación PROJ dada en el Ejemplo 2.7.

Si la única consulta que accede a PROJ quiere acceder a las tuplas según la ubicación, el conjunto está completo ya que cada tupla de cada fragmento PROJ<sub>i</sub> tiene la misma probabilidad de ser accedido. Sin embargo, si hay una segunda consulta que accede solo aquellas tuplas de proyectos donde el presupuesto es menor o igual a \$200,000, entonces P r no es completa. Algunas de las tuplas dentro de cada PROJ<sub>i</sub> tienen una mayor probabilidad de ser accedido debido a esta segunda aplicación. Para completar el conjunto de predicados, es necesario agregar (PRESUPUESTO ≤ 200000, PRESUPUESTO > 200000) a P r:

$$P r = \{LOC = "Montreal", LOC = "Nueva York", LOC = "París",$$

$$PRESUPUESTO \leq 200000, PRESUPUESTO > 200000\}$$

La completitud es deseable porque los fragmentos obtenidos de acuerdo con un conjunto completo de predicados son lógicamente uniformes, ya que todos satisfacen el predicado mintérmino.

También son estadísticamente homogéneos en la forma en que las aplicaciones acceden a ellos. Estos

4Es claro que la definición de completitud de un conjunto de predicados simples es diferente de la regla de completitud de la fragmentación que discutimos anteriormente.

Las características garantizan que la fragmentación resultante genere una carga equilibrada (con respecto a la carga de trabajo dada) en todos los fragmentos.

La minimalidad establece que si un predicado influye en cómo se realiza la fragmentación (es decir, provoca que un fragmento  $f$  se fragmente aún más en, por ejemplo,  $f_1$  y  $f_2$ ), debería haber al menos una aplicación que acceda a  $f_1$  y  $f_2$  de forma diferente. En otras palabras, el predicado simple debería ser relevante para determinar una fragmentación. Si todos los predicados de un conjunto  $P_r$  son relevantes,  $P_r$  es minimal.

Una definición formal de relevancia puede darse de la siguiente manera. Sean  $m_i$  y  $m_j$  dos predicados minterm idénticos en su definición, excepto que  $m_i$  contiene el predicado simple  $p_i$  en su forma natural, mientras que  $m_j$  contiene  $\neg p_i$ . Además, sean  $f_1$  y  $f_2$  dos fragmentos definidos según  $m_i$  y  $m_j$ , respectivamente. Entonces  $p_i$  es relevante si y solo si

$$\frac{\text{tarjeta}}{\text{acc}(m_i)(f_i)} = \frac{\text{acc}(m_j)}{\text{tarjeta}(f_j)}$$

Ejemplo 2.9 El conjunto  $P_r$  definido en el Ejemplo 2.8 es completo y mínimo. Sin embargo, si añadiéramos el predicado PNAME = "Instrumentación" a  $P_r$ , el conjunto resultante no sería mínimo, ya que el nuevo predicado no es relevante con respecto a  $P_r$ ; ninguna aplicación accedería a los fragmentos resultantes de forma diferente.

Ahora presentamos un algoritmo iterativo que generaría un conjunto completo y mínimo de predicados  $P_r$  dado un conjunto de predicados simples  $P_r$ . Este algoritmo, llamado COM\_MIN, se proporciona en el Algoritmo 2.1, donde utilizamos la siguiente notación:

Regla 1: cada fragmento es accedido de forma diferente por al menos una aplicación.

$f_i$  de  $P_r$ : fragmento  $f_i$  definido según un predicado minterm definido sobre los predicados de  $P_r$

COM\_MIN comienza encontrando un predicado relevante que particione la relación de entrada. El bucle "repeat-until" añade predicados iterativamente a este conjunto, garantizando la minimalidad en cada paso. Por lo tanto, al final, el conjunto  $P_r$  es mínimo y completo.

El segundo paso en el proceso de diseño horizontal primario es derivar el conjunto de estos Los predicados minterm que se pueden definir en los predicados del conjunto  $P_r$ . minterm  $r$  determinan los fragmentos que se utilizan como candidatos en el paso de asignación. La determinación de predicados de mintérminos individuales es trivial; la dificultad radica en que el conjunto de predicados de mintérminos puede ser bastante grande (de hecho, exponencial respecto al número de predicados simples). Analizamos maneras de reducir el número de predicados de mintérminos que deben considerarse en la fragmentación.

Esta reducción se puede lograr eliminando algunos de los fragmentos de mintérminos que puedan carecer de sentido. Esta eliminación se realiza identificando aquellos mintérminos que podrían ser contradictorios con un conjunto de implicaciones  $I$ . Por ejemplo, si  $P_r = \{p_1, p_2\}$ , donde

---

Algoritmo 2.1: COM\_MIN Entrada: R:

relación; P r: conjunto de predicados simples Salida: P r :

conjunto de predicados simples Declarar: F:

conjunto de fragmentos de mintérmino que

```

comienzan P r ← ; F ←                                {inicializar}
    encuentre pi ∈ P r tal que pi particione a R según la Regla 1 P r ←
    P r - pi P r ← P r
    - pi F ← F   fi
    repita encuentre                               {fi es el fragmento minterm según pi}
    pj ∈ P
        r tal que pj particione a algún fk de P r según la Regla 1 P r ← P r - pj P r ← P r - pj F
        ← F   f si   pk
        P r que no es
        relevante
        entonces P r ← P r - pk F ← F - fk fin si
    |
    |
hasta que P r esté completo
fin

```

---

$$\begin{aligned} p1 : \text{att} &= \text{valor\_1} \\ p2 : \text{att} &= \text{valor\_2} \end{aligned}$$

y el dominio de att es {valor\_1, valor\_2}, por lo que contiene dos implicaciones:

$$\begin{aligned} i1 : (\text{att} = \text{valor\_1}) &\quad \neg(\text{att} = \text{valor\_2}) \\ i2 : \neg(\text{att} = \text{valor\_1}) &\quad (\text{att} = \text{valor\_2}) \end{aligned}$$

Los siguientes cuatro predicados minterm se definen según P r :

$$\begin{aligned} m1 : (\text{att} = \text{valor\_1}) &\quad (\text{att} = \text{valor\_2}) \\ m2 : (\text{att} = \text{valor\_1}) &\quad \neg(\text{att} = \text{valor\_2}) \\ m3 : \neg(\text{att} = \text{valor\_1}) &\quad (\text{att} = \text{valor\_2}) \\ m4 : \neg(\text{att} = \text{valor\_1}) &\quad \neg(\text{att} = \text{valor\_2}) \end{aligned}$$

En este caso los predicados minterm m1 y m4 son contradictorios con las implicaciones I y, por lo tanto, pueden eliminarse de M.

El algoritmo para la fragmentación horizontal primaria, denominado PHORIZONTAL, se presenta en el Algoritmo 2.2. La entrada es una relación R sujeta a fragmentación horizontal primaria y P r, que es el conjunto de predicados simples determinados según las aplicaciones definidas en la relación R.

Ejemplo 2.10 Consideremos ahora las relaciones PAY y PROJ que están sujetas a fragmentación horizontal primaria como se muestra en la figura 2.5.

## 2.1 Fragmentación de datos

Supongamos que solo hay una consulta que accede a PAY, la cual verifica la información salarial y determina el aumento correspondiente. Supongamos que los registros de los empleados se gestionan en dos ubicaciones: una gestiona los registros de aquellos con salarios menores o iguales a \$30,000 y la otra los de aquellos que ganan más de \$30,000. Por lo tanto, la consulta se emite en dos ubicaciones.

Los predicados simples que se usarían para particionar la relación PAY son

$$p1 : SAL \leq 30000$$

$$p2 : SAL > 30000$$

Por lo tanto, el conjunto inicial de predicados simples es  $P_r = \{p1, p2\}$ . Al aplicar el algoritmo COM\_MIN con  $i = 1$  como valor inicial, se obtiene  $P_r = \{p1\}$ . Esto es completo y mínimo, ya que  $p2$  no particionaría  $f1$  (que es el fragmento del mintérmino formado con respecto a  $p1$ ) según la Regla 1. Podemos formar los siguientes predicados del mintérmino como miembros de  $M$ :

$$m1 : SAL < 30000$$

$$m2 : \neg(SAL \leq 30000) = SAL > 30000$$

Por lo tanto, definimos dos fragmentos  $FPAY = \{PAY1, PAY2\}$  de acuerdo con  $M$  (Fig. 2.7).

**Algoritmo 2.2: FORIZONTAL**

Entrada:  $R$ : relación;  $P_r$ : conjunto de predicados simples

Salida:  $FR$ : conjunto de fragmentos horizontales de  $R$

begin

$P_r \leftarrow COM\_MIN(R, P_r)$  determina  
el conjunto  $M$  de predicados minterm determina el  
conjunto  $I$  de implicaciones entre  $p_i \in P_r$  para cada  $m_i \in M$   
hacer si  $m_i$  es

contradicitorio según  $I$  entonces  
|  $M \leftarrow M - m_i$  fin si

fin de foreach

$FR = \{R_i | R_i = \sigma_{m_i} R\}, \quad m_i \in M$  fin

PAGO1

TÍTULO	SAL
Mec. Ing. 27000 Programador	

PAY2

TÍTULO	SAL
Ing. Elect. Anal sistemico.	40000 34000

Fig. 2.7 Fragmentación horizontal de la relación PAY

Consideremos ahora la relación PROY. Supongamos que hay dos consultas. La primera se emite en tres sitios y encuentra los nombres y presupuestos de los proyectos según su ubicación.

En notación SQL, la consulta es

```
SELECCIONAR PNAME, PRESUPUESTO
DEL PROYECTO
DONDE LOC=Valor
```

Para esta aplicación, los predicados simples que se utilizarían son los siguientes:

```
p1 : LOC = "Montreal"
p2 : LOC = "Nueva York"
p3 : LOC = "París"
```

La segunda consulta se emite en dos sitios y se relaciona con la gestión de los proyectos. Los proyectos con un presupuesto menor o igual a \$200,000 se gestionan en un sitio, mientras que los proyectos con presupuestos mayores se gestionan en un segundo sitio. Por lo tanto, los predicados simples que deben usarse para fragmentar según la segunda aplicación son

```
p4 : PRESUPUESTO ≤ 200000
p5 : PRESUPUESTO > 200000
```

Usando COM\_MIN, obtenemos el conjunto completo y mínimo  $P_r = \{p1, p2, p4\}$ .

En realidad, COM\_MIN agregaría dos de  $p1, p2, p3$  a  $P_r$ ; en este ejemplo hemos seleccionado incluir  $p1, p2$ .

Basado en  $P_r$ , Se pueden definir los siguientes seis predicados minterm que forman  $M$ :

```
m1 : (LOC = "Montreal") (PRESUPUESTO ≤ 200000)
m2 : (LOC = "Montreal") (PRESUPUESTO > 200000)
m3 : (LOC = "Nueva York") (PRESUPUESTO ≤ 200000)
m4 : (LOC = "Nueva York") (PRESUPUESTO > 200000)
m5 : (LOC = "París") (PRESUPUESTO ≤ 200000)
m6 : (LOC = "París") (PRESUPUESTO > 200000)
```

Como se señala en el Ejemplo 2.5, estos no son los únicos predicados minterm que se pueden usar. generado. Es posible, por ejemplo, especificar predicados de la forma

```
p1   p2   p3   p4   p5
```

## 2.1 Fragmentación de datos

Sin embargo, las implicaciones obvias (por ejemplo,  $p1 \wedge p2 \wedge p3 \wedge p5 \wedge p4$ ) eliminan estos predicados minterm y nos quedamos con  $m1$  a  $m6$ .

Al observar la instancia de base de datos en la figura 2.2, uno puede verse tentado a afirmar que Las siguientes implicaciones son válidas:

i8 : LOC = "Montreal"  $\neg(\text{PRESUPUESTO} > 200000)$

i9 : LOC = "París"  $\neg(\text{PRESUPUESTO} \leq 200000)$

i10 :  $\neg(\text{LOC} = \text{"Montreal"}) \quad \text{PRESUPUESTO} \leq 200000$

i11 :  $\neg(\text{LOC} = \text{"París"}) \quad \text{PRESUPUESTO} > 200000$

Sin embargo, recuerde que las implicaciones deben definirse según la semántica de la base de datos, no según los valores actuales. No hay nada en el semántica de la base de datos que sugiere que las implicaciones i8-i11 son válidas. Algunas de las Los fragmentos definidos según  $M = \{m1, \dots, m6\}$  pueden estar vacíos, pero lo son, Sin embargo, fragmentos.

El resultado de la fragmentación horizontal primaria de PROJ es la formación de seis fragmentos FPROJ = {PROJ1, PROJ2, PROJ3, PROJ4, PROJ5, PROJ6} de relación PROJ según los predicados del mintermino M (Fig. 2.8). Dado que los fragmentos PROJ2 y PROJ5 están vacíos, no se representan en la Fig. 2.8.

## 2.1.1.3 Fragmentación horizontal derivada

Una fragmentación horizontal derivada se aplica a las relaciones de destino en el gráfico de unión y se realiza en función de predicados definidos sobre la relación de origen de la Unir la arista del grafo. En nuestros ejemplos, las relaciones EMP y ASG están sujetas a derivadas. fragmentación horizontal. Recordemos que el borde entre la fuente y el objetivo Las relaciones se definen como una unión equitativa que se puede implementar por medio de semiuniones.

PROY1			
PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
Instrumentación P1		150000	Montreal

PROY3			
PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
Desarrollo de base de datos P2.	135000	Nueva York	

PROYECTO4			
PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
P3 CAD/CAM		255000	Nueva York

PROYECTOS8			
PNO	Nombre de usuario	PRESUPUESTO	LOCALIDAD
Mantenimiento P4		310000	París

Fig. 2.8 Fragmentación horizontal de la relación PROJ

Este segundo punto es importante, ya que queremos particionar una relación objetivo según a la fragmentación de su fuente, pero también queremos que el fragmento resultante sea definido únicamente en los atributos de la relación de destino.

En consecuencia, dado un borde L donde la fuente ( $L = S$ ) y el objetivo ( $L = R$ ), la Los fragmentos horizontales derivados de R se definen como

$$R_i = R \text{ Si, } 1 \leq i \leq w$$

donde  $w$  es el número máximo de fragmentos que se definirán en R y Si =  $\sigma F_i(S)$ , donde  $F_i$  es la fórmula según la cual el fragmento horizontal primario Si está definido.

Ejemplo 2.11 Consideré el borde L1 en la figura 2.5, donde source(L1) = PAY y objetivo (L1) = EMP. Entonces, podemos agrupar a los ingenieros en dos grupos según su salario: aquellos que ganan menos o igual a \$30,000, y aquellos que ganan más de 30.000 dólares. Los dos fragmentos EMP1 y EMP2 se definen de la siguiente manera:

$$\text{EMP1} = \text{EMP PAGO1}$$

$$\text{EMP2} = \text{EMP PAY2}$$

dónde

$$\text{PAGO1} = \sigma \text{SAL} \leq 30000(\text{PAGO})$$

$$\text{PAGO2} = \sigma \text{SAL} > 30000(\text{PAGO})$$

El resultado de esta fragmentación se muestra en la figura 2.9.

La fragmentación horizontal derivada se aplica a las relaciones de destino en el gráfico de unión y se realizan en función de predicados definidos sobre la relación de origen de la Unir la arista del grafo. En nuestros ejemplos, las relaciones EMP y ASG están sujetas a derivadas. Fragmentación horizontal. Para realizar una fragmentación horizontal derivada, tres Se necesitan entradas: el conjunto de particiones de la relación de origen (por ejemplo, PAY1 y PAY2) en el Ejemplo 2.11, la relación de destino y el conjunto de predicados de semiunión entre

EMP1		
ENO	ENAME	TÍTULO
E3 A. Lee	Ingeniero Mecánico	
Programador E4 J. Miller		
E7 R. Davis	Ingeniero Mecánico	

EMP2		
ENO	ENAME	TÍTULO
E1 J. Doe	Ing. Elect.	
E2 M.	Smith Syst. Anal.	
E5 B.	Casey Anal Sist.	
E6 L.	Chu Ing. Elect.	
E8 J.	Jones Syst. Anal.	

Fig. 2.9 Fragmentación horizontal derivada de la relación EMP

## 2.1 Fragmentación de datos

El origen y el destino (p. ej., EMP.TITLE = PAY.TITLE en el Ejemplo 2.11). El algoritmo de fragmentación, por lo tanto, es bastante simple, por lo que no lo presentaremos en detalle.

Existe una posible complicación que merece atención. En un esquema de base de datos, es común que existan múltiples aristas en una relación R (p. ej., en la Fig. 2.5, ASG tiene dos aristas entrantes). En este caso, existe más de una posible fragmentación horizontal derivada de R. La elección de la fragmentación candidata se basa en dos criterios:

1. La fragmentación con mejores características de unión; 2.

La fragmentación utilizada en más consultas.

Analicemos primero el segundo criterio. Este es bastante sencillo si consideramos la frecuencia con la que la carga de trabajo accede a los datos.

Si es posible se debería intentar facilitar los accesos de los usuarios "pesados" para que su impacto total en el rendimiento del sistema se minimice.

Sin embargo, aplicar el primer criterio no es tan sencillo. Consideremos, por ejemplo, la fragmentación que analizamos en el Ejemplo 2.1. El efecto (y el objetivo) de esta fragmentación es que la unión de las relaciones EMP y PAY para responder a la consulta se facilita (1) al realizarla en relaciones más pequeñas (es decir, fragmentos) y (2) al realizar uniones en paralelo.

El primer punto es obvio. El segundo aborda el paralelismo intraconsulta de las consultas de unión, es decir, la ejecución de cada consulta de unión en paralelo, lo cual es posible en ciertas circunstancias. Considere, por ejemplo, las aristas entre los fragmentos (es decir, el grafo de unión) de EMP y PAY, derivadas del Ejemplo 2.9. Tenemos PAY1 → EMP1 y PAY2 → EMP2; solo hay una arista que entra o sale de un fragmento, por lo que se trata de un grafo de unión simple. La ventaja de un diseño donde la relación de unión entre fragmentos es simple es que el destino y el origen de una arista pueden asignarse a un sitio y las uniones entre diferentes pares de fragmentos pueden realizarse de forma independiente y en paralelo.

Desafortunadamente, no siempre es posible obtener grafos de unión simples. En ese caso, la siguiente alternativa deseable es contar con un diseño que genere un grafo de unión particionado. Un grafo particionado consta de dos o más subgrafos sin aristas entre ellos. Los fragmentos así obtenidos pueden no distribuirse para la ejecución en paralelo con la misma facilidad que los obtenidos mediante grafos de unión simples, pero la asignación sigue siendo posible.

Ejemplo 2.12. Continuemos con el diseño de distribución de la base de datos que iniciamos en el Ejemplo 2.10. Ya decidimos la fragmentación de la relación EMP según la fragmentación de PAY (Ejemplo 2.11). Consideremos ahora ASG.

Supongamos que existen las dos consultas siguientes:

1. La primera consulta busca los nombres de los ingenieros que trabajan en determinados lugares. Se ejecuta en los tres sitios y accede a la información sobre los ingenieros que trabajan en proyectos locales con mayor probabilidad que en proyectos de otras ubicaciones.
2. En cada sitio administrativo donde se gestionan los registros de los empleados, los usuarios desean acceder a las responsabilidades en los proyectos en los que trabajan estos empleados y saber cuánto tiempo trabajarán en esos proyectos.

La primera consulta da como resultado una fragmentación de ASG según el (no vacío) fragmentos PROJ1, PROJ3, PROJ4 y PROJ6 de PROJ obtenidos en el Ejemplo 2.10:

PROY1 :  $\sigma_{LOC="Montreal"} PRESUPUESTO \leq 200000$ (PROY)

PROJ3 :  $\sigma_{LOC="Nueva York"} PRESUPUESTO \leq 200000$ (PROJ)

PROJ4 :  $\sigma_{LOC="Nueva York"} PRESUPUESTO > 200000$ (PROJ)

PROJ6 :  $\sigma_{LOC="París"} PRESUPUESTO > 200000$ (PROJ)

Por lo tanto, la fragmentación derivada de ASG según {PROJ1, PROJ3, PROJ4, PROJ6} se define de la siguiente manera:

ASG1 = ASG PROJ1

ASG2 = ASG PROJ3

ASG3 = ASG PROJ4

ASG4 = ASG PROJ6

Estas instancias de fragmentos se muestran en la figura 2.10.

La segunda consulta se puede especificar en SQL como

SELECCIONAR RESP, DUR  
DESDE ASG NATURAL ÚNETE A EMP<sub>i</sub>

donde i = 1 o i = 2, dependiendo del sitio donde se emite la consulta. El derivado La fragmentación de ASG según la fragmentación de EMP se define a continuación y representado en la figura 2.11.

ASG1 = ASG EMP1

ASG2 = ASG EMP12

ASG1			
ENO	PNO	RESP	DUR
E1	Gerente P1		12
E2	Analista P1		24

ASG3			
ENO	PNO	RESP	DUR
Consultor E3	P3		10
Ingeniero E7	P3		36
Gerente E8	P3		40

ASG2			
ENO	PNO	RESP	DUR
E2	Analista P2	6	
E4	Programador P2	18	
E5	Gerente P2	24	

ASG4			
ENO	PNO	RESP	DUR
Ingeniero E3	P4		48
Gerente E6	P4		48

Fig. 2.10 Fragmentación derivada de ASG con respecto a PROJ

## 2.1 Fragmentación de datos

ASG1				ASG2			
ENO	PNO	RESP	DUR	ENO	PNO	RESP	DUR
Consultor E3 P3		10		E1	Gerente P1	12	
Ingeniero E3 P4		48		E2	Analista P1	24	
E4	Programador P2 18			Analista E2 P2		6	
Ingeniero E7 P3		36		Gerente E5 P2		24	

Fig. 2.11 Fragmentación derivada de ASG con respecto a EMP

Este ejemplo destaca dos observaciones:

1. La fragmentación derivada puede seguir una cadena donde una relación se fragmenta como una resultado del diseño de otro y, a su vez, provoca la fragmentación de otro relación (por ejemplo, la cadena PAY → EMP → ASG).
2. Normalmente, habrá más de una fragmentación candidata para una relación. (p. ej., relación ASG). La elección final del esquema de fragmentación es una decisión problema que puede abordarse durante la asignación.

## 2.1.1.4 Comprobación de la corrección

Ahora comprobamos los algoritmos de fragmentación discutidos hasta ahora con respecto a la tres criterios de corrección que discutimos anteriormente.

Lo completo

La integridad de una fragmentación horizontal primaria se basa en la selección predicados utilizados. Mientras los predicados de selección estén completos, el resultado También se garantiza que la fragmentación sea completa. Dado que la base del algoritmo de fragmentación es un conjunto de predicados completos y mínimos ( $P_r$ ), la completitud es garantizado si  $P_r$  se determina correctamente.

La completitud de una fragmentación horizontal derivada es algo más difícil de definir ya que el predicado que determina la fragmentación involucra dos relaciones.

Sea  $R$  la relación objetivo de una arista cuya fuente es la relación  $S$ , donde  $R$  y  $S$  se fragmentan como  $FR = \{R_1, R_2, \dots, R_w\}$  y  $FS = \{S_1, S_2, \dots, S_w\}$ , respectivamente. Sea  $A$  el atributo de unión entre  $R$  y  $S$ . Entonces, para cada tupla  $t$  de  $R_i$ , debería haber una tupla  $t$  de  $S_j$  tal que  $t[A] = t'[A]$ . Esta es la conocida integridad referencial regla, que garantiza que las tuplas de cualquier fragmento de la relación de destino también estén en La relación de origen. Por ejemplo, no debería haber ninguna tupla ASG que tenga un proyecto Número que no esté incluido en PROJ. De igual forma, no debería haber tuplas EMP. con valores de TÍTULO donde el mismo valor de TÍTULO no aparece en PAGO también.

### Reconstrucción

La reconstrucción de una relación global a partir de sus fragmentos se realiza mediante el operador de unión, tanto en la fragmentación horizontal primaria como en la derivada. Por lo tanto, para una relación R con fragmentación FR = {R1, R2,..., Rw}, R = Ri,  $\cup$  FR.

### Desunión

Es más fácil establecer la disyunción de la fragmentación para la fragmentación horizontal primaria que para la derivada. En el primer caso, la disyunción está garantizada siempre que los predicados del mintérmino que determinan la fragmentación sean mutuamente excluyentes.

Sin embargo, en la fragmentación derivada, existe una semiunión que añade una complejidad considerable. La disyunción puede garantizarse si el grafo de unión es simple.

De lo contrario, es necesario investigar los valores reales de las tuplas. En general, no queremos que una tupla de una relación de destino se una con dos o más tuplas de la relación de origen cuando estas se encuentran en fragmentos diferentes de la relación de origen. Esto puede no ser fácil de establecer, e ilustra por qué los esquemas de fragmentación derivados que generan un grafo de unión simple son siempre recomendables.

Ejemplo 2.13 En la relación de fragmentación PAY (Ejemplo 2.10), los predicados minterm M = {m1, m2} fueron

$$m1 : SAL \leq 30000$$

$$m2 : SAL > 30000$$

Dado que m1 y m2 son mutuamente excluyentes, la fragmentación de PAY es disjunta.

Para la relación EMP, sin embargo, requerimos que

1. Cada ingeniero tiene un título único.
2. Cada título tiene un valor salarial único asociado.

Dado que estas dos reglas se derivan de la semántica de la base de datos, la fragmentación de EMP con respecto a PAY también es disjunta.

## 2.1.2 Fragmentación vertical

Recuerde que una fragmentación vertical de una relación R produce los fragmentos R1, R2, ..., Rr, cada uno de los cuales contiene un subconjunto de los atributos de R, así como la clave principal de los RA. En el caso de la fragmentación horizontal, el objetivo es dividir una relación en un conjunto de relaciones más pequeñas para que muchas de las aplicaciones de usuario se ejecuten en un solo fragmento. La clave principal se incluye en cada fragmento para permitir la reconstrucción, como se explica más adelante. Esto también es beneficioso para el cumplimiento de la integridad, ya que la clave principal...

La clave determina funcionalmente todos los atributos de la relación; tenerla en cada fragmento elimina el cálculo distribuido para imponer la restricción de clave principal.

La partición vertical es inherentemente más compleja que la horizontal, debido principalmente al número total de alternativas posibles. Por ejemplo, en la partición horizontal, si el número total de predicados simples en  $P \leq n$ , existen  $2^n$  predicados mintérmino posibles. Además, sabemos que algunos de estos contradirán las implicaciones existentes, lo que reducirá aún más los fragmentos candidatos a considerar. Sin embargo, en el caso de la partición vertical, si una relación tiene  $m$  atributos de clave no primaria, el número de fragmentos posibles es igual a  $B(m)$ , que es el  $m$ -ésimo número de Bell. Para valores grandes de  $m$ ,  $B(m) \approx m^m$ ; por ejemplo, para  $m = 10$ ,  $B(m) \approx 115\,000$ , para  $m = 15$ ,  $B(m) \approx 109$ , para  $m = 30$ ,  $B(m) = 1023$ .

Estos valores indican que es inútil intentar obtener soluciones óptimas al problema de la fragmentación vertical; es necesario recurrir a la heurística. Existen dos tipos de enfoques heurísticos para la fragmentación vertical de las relaciones globales<sup>5</sup>:

1. Agrupación: comienza asignando cada atributo a un fragmento y, en cada paso, une algunos de los fragmentos hasta que se satisfacen algunos criterios.
2. División: comienza con una relación y decide particiones beneficiosas según el comportamiento de acceso de las aplicaciones a los atributos.

A continuación, analizamos únicamente la técnica de división, ya que se adapta mejor a la metodología de diseño que explicamos anteriormente, ya que la solución óptima probablemente se acerca más a la relación completa que a un conjunto de fragmentos, cada uno compuesto por un solo atributo. Además, la división genera fragmentos no superpuestos, mientras que la agrupación suele generarlos. Preferimos fragmentos no superpuestos por su disyunción. Por supuesto, la no superposición se refiere únicamente a atributos de clave no primaria.

#### 2.1.2.1 Requisitos de información auxiliar

Nuevamente necesitamos información sobre la carga de trabajo. Dado que la partición vertical agrupa en un solo fragmento los atributos a los que se accede habitualmente juntos, se necesita una medida que defina con mayor precisión el concepto de "unión". Esta medida es la afinidad de los atributos, que indica su estrecha relación. No es realista esperar que el diseñador o los usuarios puedan especificar fácilmente estos valores.

Presentamos una forma de obtenerlos a partir de datos más primitivos.

Sea  $Q = \{q_1, q_2, \dots, q_n\}$  el conjunto de consultas de usuario que acceden a la relación  $R(A_1, A_2, \dots, A_n)$ . Luego, para cada consulta  $q_i$  y cada atributo  $A_j$ , asociamos un valor de uso del atributo, denotado como  $use(q_i, A_j)$ :

---

<sup>5</sup>Existe también un tercer enfoque extremo en los SGBD orientados a columnas (como MonetDB y Vertica), donde cada columna se asigna a un fragmento. Dado que en este libro no se abordan los SGBD orientados a columnas, no se profundiza en este enfoque.

$$\text{uso}(q_i, A_j) = \begin{cases} 1 & \text{si el atributo } A_j \text{ es referenciado por la consulta } q_i \\ 0 & \text{en caso contrario} \end{cases}$$

Los vectores  $\text{use}(q_i, \cdot)$  para cada consulta son fáciles de determinar.

Ejemplo 2.14: Consideré la relación PROJ de la Fig. 2.2. Suponga que las siguientes consultas están definidas para ejecutarse en esta relación. En cada caso, también proporcionamos la expresión SQL.

q1: Encontrar el presupuesto de un proyecto, dado su número de identificación.

```
SELECCIONAR PRESUPUESTO
DEL PROYECTO
DONDE PNO=Valor
```

q2: Encuentra los nombres y presupuestos de todos los proyectos.

```
SELECCIONAR PNAME, PRESUPUESTO
DEL PROYECTO
```

q3: Encuentra los nombres de los proyectos ubicados en una ciudad determinada.

```
SELECCIONAR PNAME
DEL PROYECTO
DONDE LOC=Valor
```

q4: Encuentra los presupuestos totales del proyecto para cada ciudad.

```
SELECCIONAR SUMA(PRESUPUESTO)
DEL PROYECTO
DONDE LOC=Valor
```

De acuerdo con estas cuatro consultas, los valores de uso de los atributos se pueden definir en forma de matriz (Fig.

2.12), donde la entrada  $(i, j)$  denota  $\text{uso}(q_i, A_j)$ .

Los valores de uso de atributos no son lo suficientemente generales como para fundamentar la división y fragmentación de atributos, ya que no representan la ponderación de las frecuencias de aplicación. La medida de frecuencia puede incluirse en la definición de la medida de afinidad de atributos  $\text{aff}(A_i, A_j)$ , que mide el vínculo entre dos atributos de una relación según cómo se accede a ellos mediante consultas.

	PNO	PNAME	PRESUPUESTO	LOC
q1	0	1	1	q2 11 1 q3 10 0
q4	0	0	1	1
				0

Fig. 2.12 Ejemplo de matriz de uso de atributos

La medida de afinidad de atributos entre dos atributos  $A_i$  y  $A_j$  de una relación  $R(A_1, A_2, \dots, A_n)$  con respecto al conjunto de consultas  $Q = \{q_1, q_2, \dots, q_k\}$  se define como

$$\text{aff}(A_i, A_j) = k | \begin{array}{l} \text{refl}(q_k) \text{accl}(q_k) \\ \text{uso}(q_k, A_i) = 1 \quad \text{uso}(q_k, A_j) = 1 \quad \text{Si} \end{array}$$

donde  $\text{refl}(q_k)$  es el número de accesos a los atributos  $(A_i, A_j)$  para cada ejecución de la aplicación  $q_k$  en el sitio Si y  $\text{accl}(q_k)$  es la medida de frecuencia de acceso a la aplicación previamente definido y modificado para incluir frecuencias en diferentes sitios.

El resultado de este cálculo es una matriz  $n \times n$ , cada elemento de la cual es uno de las medidas definidas anteriormente. Esta matriz se denomina matriz de afinidad de atributos (AA).

Ejemplo 2.15 Continuemos con el caso que examinamos en el Ejemplo 2.14.

Para simplificar, supongamos que  $\text{refl}(q_k) = 1$  para todos los  $q_k$  y Si. Si la aplicación Las frecuencias son

$$\begin{array}{ll} \text{acc1}(q_1) = 15 & \text{acc1}(q_2) = 5 \\ \text{acc1}(q_3) = 25 & \text{acc1}(q_4) = 3 \\ \text{acc2}(q_1) = 20 & \text{acc2}(q_2) = 0 \\ \text{acc2}(q_3) = 25 & \text{acc3}(q_4) = 0 \\ \text{acc3}(q_1) = 10 & \text{acc3}(q_2) = 0 \\ \text{acc3}(q_3) = 25 & \text{acc2}(q_4) = 0 \end{array}$$

Entonces la medida de afinidad entre los atributos PNO y BUDGET se puede medir como

$$\text{aff}(\text{PNO}, \text{PRESUPUESTO}) = \sum_{k=1}^3 \text{accl}(q_k) = \text{acc1}(q_1) + \text{acc2}(q_1) + \text{acc3}(q_1) = 45$$

ya que la única aplicación que accede a ambos atributos es  $q_1$ . El completo La matriz de afinidad de atributos se muestra en la Fig. 2.13. Nótese que los valores diagonales no son se calculan porque no tienen sentido.

	PNO	PNAME	PRESUPUESTO	LOC
PNO	—	0	45	0
Nombre de usuario 0	—	5	—	75
PRESUPUESTO	45	5	—	3
UBICACIÓN 0	75	3	—	—

Fig. 2.13 Matriz de afinidad de atributos

La matriz de afinidad de atributos se utilizará en el resto de este capítulo para guiar la fragmentación. El proceso primero agrupa los atributos con alta afinidad entre sí y luego divide la relación según corresponda.

### 2.1.2.2 Algoritmo de agrupamiento

La tarea fundamental al diseñar un algoritmo de fragmentación vertical es encontrar la manera de agrupar los atributos de una relación según los valores de afinidad de los atributos en AA. Analizaremos el algoritmo de energía de enlace (BEA), propuesto para este propósito. También se pueden utilizar otros algoritmos de agrupamiento.

BEA toma como entrada la matriz de afinidad de atributos para la relación  $R(A_1, \dots, A_n)$ , permuta sus filas y columnas, y genera una matriz de afinidad agrupada (CA). La permutación se realiza de tal manera que se maximice la siguiente medida de afinidad global (AM):

$$\begin{aligned} AM = & \sum_{i=1}^n \sum_{j=1}^n \\ & [aff(A_i, A_j) \cdot [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1}) \\ & + aff(A_{i-1}, A_j) + aff(A_{i+1}, A_j)]] \end{aligned}$$

dónde

$$aff(A_0, A_j) = aff(A_i, A_0) = aff(A_{n+1}, A_j) = aff(A_i, A_{n+1}) = 0$$

El último conjunto de condiciones se ocupa de los casos en los que un atributo se coloca en CA a la izquierda del atributo más a la izquierda o a la derecha del atributo más a la derecha durante las permutaciones de columnas, y antes de la fila superior y después de la última fila durante las permutaciones de filas. Se denota con  $A_0$  el atributo a la izquierda del atributo más a la izquierda y la fila anterior a la superior, y con  $A_{n+1}$  el atributo a la derecha del atributo más a la derecha o la fila posterior a la última. En estos casos, se establecen en 0 los valores  $aff$  entre el atributo considerado para la colocación y sus vecinos izquierdo o derecho (superior o inferior), ya que no existen en CA.

La función de maximización considera únicamente los vecinos más cercanos, lo que da como resultado la agrupación de valores grandes con valores grandes y valores pequeños con valores pequeños. Además, la matriz de afinidad de atributos (AA) es simétrica, lo que reduce la función objetivo a

$$\begin{aligned} AM = & \sum_{i=1}^n \sum_{j=1}^n \\ & [aff(A_i, A_j) \cdot [aff(A_i, A_{j-1}) + aff(A_i, A_{j+1})]] \end{aligned}$$

---

Algoritmo 2.3: BEA Entrada:

AA: matriz de afinidad de atributos Salida:

CA: matriz de afinidad agrupada begin

```

{inicializar; recuerde que AA es una matriz n × n}
    CA(*, 1) ← AA(*, 1)
    CA(*, 2) ← AA(*, 2) índice
    ← 3 mientras
        índice ≤ n hacer {elegir la “mejor” ubicación para el atributo AAindex } para i de 1 a índice – 1 por 1 hacer
            | calcular cont (Ai–1, Aindex , Ai) calcular cont (Aindex–1, Aindex , Aindex+1)
            | {condición de contorno} loc ← ubicación dada por el valor de cont máximo para j de índice a loc por
            | –1 hacer CA(*,j) ← CA(*, j – 1) fin para CA(*, loc) ← AA(*,
            | índice ← índice + 1 fin mientras
            | | ordenar las filas según el orden
            | | relativo
            | | de las columnas
            |
        fin

```

---

Los detalles de BEA se detallan en el Algoritmo 2.3. La generación de la matriz de afinidad agrupada (CA) se realiza en tres pasos:

1. Inicialización. Coloque y fije una de las columnas de AA arbitrariamente en CA.  
Se eligió la columna 1 en el algoritmo.
2. Iteración. Seleccione cada una de las  $n - i$  columnas restantes (donde  $i$  es el número de columnas ya colocadas en CA) e intente colocarlas en las  $i + 1$  posiciones restantes de la matriz CA. Elija la ubicación que contribuya al máximo a la medida de afinidad global descrita anteriormente. Continúe este paso hasta que no queden más columnas por colocar.
3. Ordenamiento de filas. Una vez determinado el ordenamiento de las columnas, también debe modificarse la ubicación de las filas para que sus posiciones relativas coincidan con las de las columnas.<sup>6</sup> Para que el segundo paso del algoritmo funcione, es necesario definir qué se entiende por la contribución de un atributo a la medida de afinidad. Esta contribución se puede obtener de la siguiente manera. Recordemos que la medida de afinidad global AM se definió previamente como

---

<sup>6</sup>De ahora en adelante, podemos referirnos a los elementos de las matrices AA y CA como  $AA(i, j)$  y  $CA(i, j)$ , respectivamente. La asignación a las medidas de afinidad es  $AA(i, j) = \text{aff}(A_i, A_j)$  y  $CA(i, j) = \text{aff}$  (atributo colocado en la columna  $i$  en CA, atributo colocado en la columna  $j$  en CA). Aunque las matrices AA y CA son idénticas excepto por el orden de los atributos, dado que el algoritmo ordena todas las columnas de CA antes de ordenar las filas, la medida de afinidad de CA se especifica con respecto a las columnas. Tenga en cuenta que la condición de punto final para el cálculo de la medida de afinidad (AM) se puede especificar, utilizando esta notación, como  $CA(0,j) = CA(i, 0) = CA(n + 1,j) = CA(i, n + 1) = 0$ .

$$\text{AM} = \frac{\text{aff}(A_i, A_j) \text{aff}(A_i, A_{j-1}) + \text{aff}(A_i, A_{j+1})}{\text{aff}(A_i, A_j)} \quad i=1, j=1$$

que puede reescribirse como

$$\begin{aligned} \text{AM} &= \frac{[\text{aff}(A_i, A_j) \text{aff}(A_i, A_{j-1}) + \text{aff}(A_i, A_j) \text{aff}(A_i, A_{j+1})]}{\text{aff}(A_i, A_j)} \\ &= \frac{\text{aff}(A_i, A_j) \text{aff}(A_i, A_{j-1})}{\text{aff}(A_i, A_j)} + \frac{\text{aff}(A_i, A_j) \text{aff}(A_i, A_{j+1})}{\text{aff}(A_i, A_j)} \end{aligned}$$

Definamos el vínculo entre dos atributos  $A_x$  y  $A_y$  como

$$\text{bono}(A_x, A_y) = \frac{\text{aff}(A_z, A_x) \text{aff}(A_z, A_y)}{z=1}$$

Entonces AM se puede escribir como

$$\text{AM} = \frac{[\text{enlace}(A_j, A_{j-1}) + \text{enlace}(A_j, A_{j+1})]}{j=1}$$

Consideremos ahora los siguientes  $n$  atributos:

$$\begin{array}{lll} A_1 A_2 \dots A_{i-1} & A_i A_j A_{j+1} \dots A_n \\ \text{SOY} & \text{AM1} \end{array}$$

La medida de afinidad global para estos atributos se puede escribir como

$$\text{AMold} = \text{AM} + \text{AM1}$$

$$\begin{aligned} &+ \text{enlace}(A_{i-1}, A_i) + \text{enlace}(A_i, A_j) + \text{enlace}(A_j, A_i) + \text{enlace}(A_j, A_{j+1}) \\ &= \frac{i}{\text{enlace}(A_l, A_{l-1}) + \text{enlace}(A_l, A_{l+1})} \\ &+ \frac{l=1}{\text{enlace}(A_l, A_{l-1}) + \text{enlace}(A_l, A_{l+1})} \\ &+ 2 \text{enlace}(A_i, A_j) \end{aligned}$$

Ahora considere colocar un nuevo atributo  $A_k$  entre los atributos  $A_i$  y  $A_j$  en la matriz de afinidad agrupada. La nueva medida de afinidad global puede escribirse de forma similar como

## 2.1 Fragmentación de datos

$$\begin{aligned}
 AM_{\text{nuevo}} &= AM + AM_1 + \text{enlace}(A_i, A_k) + \text{enlace}(A_k, A_i) \\
 &\quad + \text{enlace}(A_k, A_j) + \text{enlace}(A_j, A_k) \\
 &= AM + AM_1 + 2\text{enlace}(A_i, A_k) + 2\text{enlace}(A_k, A_j)
 \end{aligned}$$

Por lo tanto, la contribución neta a la medida de afinidad global de colocar el atributo  $A_k$  entre  $A_i$  y  $A_j$  es

$$\begin{aligned}
 \text{cont}(A_i, A_k, A_j) &= AM_{\text{nuevo}} - AM_{\text{viejo}} \\
 &= 2\text{enlace}(A_i, A_k) + 2\text{enlace}(A_k, A_j) - 2\text{enlace}(A_i, A_j)
 \end{aligned}$$

Ejemplo 2.16 Consideremos la matriz AA dada en la Fig. 2.13 y estudiemos la contribución del atributo móvil LOC entre los atributos PNO y PNAME, dada por la fórmula

$$\begin{aligned}
 \text{cont}(PNO, LOC, PNAME) &= 2\text{bonos}(PNO, LOC) + 2\text{bonos}(LOC, PNAME) \\
 &\quad - 2\text{bond}(PNO, PNAME)
 \end{aligned}$$

Calculando cada término, obtenemos

$$\text{enlace}(PNO, LOC) = 45 \quad 0 + 0 \quad 75 + 45 \quad 3 + 0 \quad 78 = 135$$

$$\text{bono}(LOC, PNAME) = 11865$$

$$\text{bono}(PNO, PNAME) = 225$$

Por lo tanto,

$$\text{continuo}(PNO, LOC, PNAME) = 2 \quad 135 + 2 \quad 11865 - 2 \quad 225 = 23550$$

Hasta ahora, tanto el algoritmo como nuestra discusión se han centrado en las columnas de la matriz de afinidad de atributos. Es posible rediseñar el algoritmo para que opere en las filas. Dado que la matriz AA es simétrica, ambos enfoques generarán el mismo resultado.

Tenga en cuenta que el algoritmo 2.3 coloca la segunda columna junto a la primera durante la inicialización. Esto, obviamente, funciona, ya que el enlace entre ambas es independiente de sus posiciones relativas.

El cálculo de cont en los extremos requiere cuidado. Si se considera colocar un atributo  $A_i$  a la izquierda del atributo más a la izquierda, una de las ecuaciones de enlace a calcular es entre un elemento izquierdo inexistente y  $A_k$  [es decir,  $\text{enlace}(A_0, A_k)$ ]. Por lo tanto, debemos referirnos a las condiciones impuestas a la definición.

de la medida de afinidad global AM, donde  $CA(0, k) = 0$ . Se aplican argumentos similares para la colocación a la derecha del atributo más a la derecha.

Ejemplo 2.17 Consideramos la agrupación de los atributos de relación PROJ y utilizamos la matriz de afinidad de atributos AA de la figura 2.13.

De acuerdo con el paso de inicialización, copiamos las columnas 1 y 2 de la matriz AA a la matriz CA (Fig. 2.14a) y comience con la columna 3 (es decir, el atributo PRESUPUESTO). Allí Hay tres lugares alternativos donde se puede colocar la columna 3: a la izquierda de la columna 1, resultando en el ordenamiento (3-1-2), entre las columnas 1 y 2, dando (1-3-2), y a la derecha de 2, resultando en (1-2-3). Nótese que para calcular la contribución del último orden tenemos que calcular  $\text{cont}(\text{PNAME}, \text{BUDGET}, \text{LOC})$  en lugar de  $\text{cont}(\text{PNO}, \text{PNAME}, \text{BUDGET})$ . Sin embargo, tenga en cuenta que el atributo LOC aún no se ha colocado en la matriz CA (Fig. 2.14b), lo que requiere un cálculo especial como descrito anteriormente. Calculemos la contribución a la medida de afinidad global de cada alternativa.

Ordenando (0-3-1):

$$\begin{aligned}\text{cont}(\text{A0}, \text{PRESUPUESTO}, \text{PNO}) &= 2\text{bond}(\text{A0}, \text{PRESUPUESTO}) + 2\text{bond}(\text{PRESUPUESTO}, \text{PNO}) \\ &\quad - 2\text{enlace}(\text{A0}, \text{PNO})\end{aligned}$$

Sabemos que

$$\text{bono}(\text{A0}, \text{PNO}) = \text{bono}(\text{A0}, \text{PRESUPUESTO}) = 0$$

$$\text{bono}(\text{PRESUPUESTO}, \text{PNO}) = 45 \quad 45 + 5 \quad 0 + 53 \quad 45 + 3 \quad 0 = 4410$$

PNO PNAME		PRESUPUESTO PNO PNAME		
PNO	PNAME	PRESUPUESTO	PNO	PNAME
PNO 45	0	PNO 45	45	0
Nombre de usuario 0	80	Nombre de usuario 0	5	80
PRESUPUESTO 45	5	PRESUPUESTO 45	53	5
UBICACIÓN 0	75	UBICACIÓN 0	3	75

(a)

(b)

PRESUPUESTO PNO NOMBRE P LOC				PRESUPUESTO PNO NOMBRE P LOC			
PNO	PRESUPUESTO	PNO	NOMBRE	PNO	PRESUPUESTO	PNO	NOMBRE
PNO 45	45	0	0	PNO 45	45	0	0
Nombre de usuario 0	5	80	75	PRESUPUESTO 45	53	5	3
PRESUPUESTO 45	53	5	3	PNAME 0	5	80	75
UBICACIÓN 0	3	75	78	UBICACIÓN 0	3	75	78

(c)

(d)

Fig. 2.14 Cálculo de la matriz de afinidad agrupada (CA)

## 2.1 Fragmentación de datos

De este modo

$$\text{cont (A0, PRESUPUESTO, PNO)} = 8820$$

Ordenando (1-3-2):

$$\text{cont (PNO, PRESUPUESTO, PNAME)} = 2\text{bono(PNO, PRESUPUESTO)} + 2\text{bono(PRESUPUESTO, PNAME)}$$

$$- 2\text{bono(PNO, PNAME)}$$

$$\text{bono(PNO, PRESUPUESTO)} \quad \quad \quad = \text{bono(PRESUPUESTO, PNO)} = 4410$$

$$\text{bono(PRESUPUESTO, PNAME)} = 890$$

$$\text{bono(PNO, PNAME)} \quad \quad \quad = 225$$

De este modo

$$\text{cont (PNO, PRESUPUESTO, PNAME)} = 10150$$

Ordenando (2-3-4):

$$\text{cont (PNAME, PRESUPUESTO, LOC)} = 2\text{bono(PNAME, PRESUPUESTO)} + 2\text{bono(PRESUPUESTO, LOC)}$$

$$- 2\text{bono(PNAME, LOC)}$$

$$\text{bono(PNAME, PRESUPUESTO)} = 890$$

$$\text{bono(PRESUPUESTO, LOC)} \quad \quad \quad = 0$$

$$\text{vínculo(PNAME, LOC)} \quad \quad \quad = 0$$

De este modo

$$\text{cont (PNAME, PRESUPUESTO, LOC)} = 1780$$

Dado que la contribución del ordenamiento (1-3-2) es la mayor, optamos por ubicar BUDGET a la derecha de PNO (Fig. 2.14b). Cálculos similares para LOC indican que debería ubicarse a la derecha de PNAME (Fig. 2.14c).

Finalmente, las filas se organizan en el mismo orden que las columnas y el resultado es como se muestra en la figura 2.14d.

En la Fig. 2.14d se observa la creación de dos clústeres: uno, en la esquina superior izquierda, contiene los valores de afinidad más bajos, y el otro, en la esquina inferior derecha, contiene los valores de afinidad más altos. Esta agrupación indica cómo deben dividirse los atributos de la relación PROJ. Sin embargo, en general, el límite de esta división puede no ser tan claro. Cuando la matriz CA es grande, se suelen formar más de dos clústeres y hay más de una partición candidata. Por lo tanto, es necesario abordar este problema de forma más sistemática.

### 2.1.2.3 Algoritmo de división

El objetivo de la división es encontrar conjuntos de atributos a los que se accede únicamente, o en su mayoría, mediante distintos conjuntos de consultas. Por ejemplo, si es posible identificar dos atributos A1 y A2 , a los que solo se accede mediante la consulta q1, y los atributos A3 y A4, a los que se accede mediante, por ejemplo, dos consultas q2 y q3, sería bastante sencillo determinar los fragmentos. La tarea consiste en encontrar un método algorítmico para identificar estos grupos.

Consideré la matriz de atributos agrupados de la Fig. 2.15. Si se fija un punto a lo largo de la diagonal, se identifican dos conjuntos de atributos. Un conjunto  $\{A_1, A_2, \dots, A_i\}$  se encuentra en la esquina superior izquierda (denotado TA) y el segundo conjunto  $\{A_{i+1}, \dots, A_n\}$  se encuentra en la esquina inferior derecha (denotado TB) con respecto a este punto.

Ahora particionamos el conjunto de consultas  $Q = \{q_1, q_2, \dots, q_n\}$  que acceden solo a TA, Solo BA, o ambos. Estos conjuntos se definen de la siguiente manera:

$$AQ(q_i) = \{A_j \mid uso(q_i, A_j) = 1\}$$

$$TQ = \{q_i \mid AQ(q_i) = TA\}$$

$$BQ = \{q_i \mid AQ(q_i) = BA\}$$

$$OQ = Q - \{TQ \cup BQ\}$$

La primera de estas ecuaciones define el conjunto de atributos a los que accede la consulta  $q_i$ ;  $TQ$  y  $BQ$  son los conjuntos de consultas que solo acceden a TA o BA, respectivamente, y  $OQ$  es el conjunto de consultas que acceden a ambos.

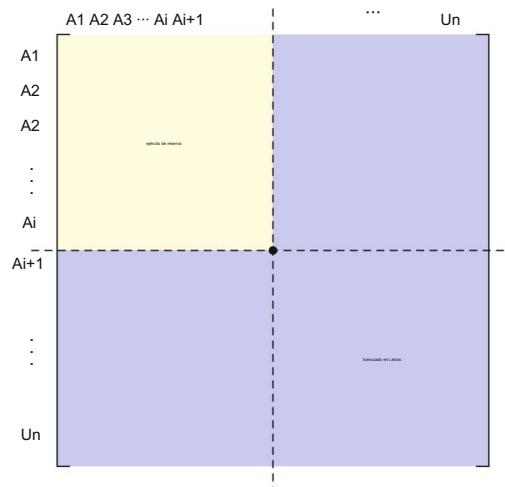


Fig. 2.15 Localización de un punto de división

Aquí se plantea un problema de optimización. Si una relación tiene n atributos, existen n – 1 posibles posiciones donde el punto de división puede ubicarse a lo largo de la diagonal de la matriz de atributos agrupados de dicha relación. La mejor posición para la división es aquella que produce los conjuntos TQ y BQ de manera que se maximice el total de accesos a un solo fragmento, mientras que se minimice el total de accesos a ambos . Por lo tanto, definimos las siguientes ecuaciones de costes:

$$CQ = \sum_{qi} Q_{Sj} \text{refj (qi)} \text{accj (qi)}$$

$$CTQ = \sum_{qi} TQ_{Sj} \text{refj (qi)} \text{accj (qi)}$$

$$CBQ = \sum_{qi} BQ_{Sj} \text{refj (qi)} \text{accj (qi)}$$

$$COQ = \sum_{qi} OQ_{Sj} \text{refj (qi)} \text{accj (qi)}$$

Cada una de las ecuaciones anteriores contabiliza el número total de accesos a atributos por consultas en sus respectivas clases. Con base en estas medidas, el problema de optimización se define como encontrar el punto x ( $1 \leq x \leq n$ ) tal que la expresión

$$z = CTQ - CBQ - COQ$$

Se maximiza. La característica importante de esta expresión es que define dos fragmentos de modo que los valores de CTQ y CBQ sean lo más iguales posible.

Esto permite equilibrar la carga de procesamiento cuando los fragmentos se distribuyen a varios sitios. Es evidente que el algoritmo de partición tiene una complejidad lineal en función del número de atributos de la relación, es decir,  $O(n)$ .

Este procedimiento divide el conjunto de atributos bidireccionalmente. Para conjuntos de atributos más grandes, es muy probable que se requiera una partición de m-direcciones. Diseñar una partición de m-direcciones es posible, pero computacionalmente costoso. A lo largo de la diagonal de la matriz CA, es necesario probar 1, 2, ..., m – 1 puntos de división, y para cada uno de ellos, es necesario verificar qué punto maximiza z. Por lo tanto, la complejidad de dicho algoritmo es  $O(2m)$ . Por supuesto, la definición de z debe modificarse para aquellos casos donde hay múltiples puntos de división. La solución alternativa es aplicar recursivamente el algoritmo de partición binaria a cada uno de los fragmentos obtenidos durante la iteración anterior. Se calcularían TQ, BQ y OQ, así como las medidas de acceso asociadas para cada uno de los fragmentos, y se partitionarían aún más.

Hasta ahora, nuestra discusión ha asumido que el punto de división es único y divide la matriz CA en una partición superior izquierda y una segunda partición formada por el resto de los atributos. Sin embargo, la partición también puede formarse en el centro de la matriz. En este caso, necesitamos modificar ligeramente el algoritmo. La columna más a la izquierda de la matriz CA se desplaza para convertirse en la columna más a la derecha y la fila superior es

---

Algoritmo 2.4: DIVISIÓN

---

Entrada: CA: matriz de afinidad agrupada; R: relación; ref: matriz de uso de atributos; acc: acceso  
 matriz de frecuencias

Salida: F: conjunto de fragmentos

begin

```

{determinar el valor z para la primera columna} {los
subíndices en las ecuaciones de costo indican el punto de división} calcular
CTQn-1 calcular
CBQn-1 calcular
COQn-1 mejor ←
CTQn-1 CBQn-1 - (COQn-1)2 repetir {determinar la
mejor
  {partición} para i de n - 2 a 1 por -1 hacer
  calcular CTQi calcular CBQi calcular
    COQi z ← CTQ
    CBQi - COQ2 si z >
    mejor entonces mejor
    ← z
  i
  {registrar el punto de división dentro del turno}
fin de
llamar a SHIFT(CA)
```

hasta que no sea posible realizar más

SHIFT reconstruir la matriz de acuerdo con la posición de desplazamiento

R1 ← T A(R) K {K es el conjunto de atributos de clave principal de R}

R2 ← BA(R) K

F ← {R1, R2} fin

---

Desplazamiento hacia abajo. Tras la operación de desplazamiento, se verifican las  $n-1$  posiciones diagonales para hallar el valor  $z$  máximo. La idea del desplazamiento es mover el bloque de atributos que debería formar un clúster a la esquina superior izquierda de la matriz, donde se puede identificar fácilmente. Con la adición de la operación de desplazamiento, la complejidad del algoritmo de partición aumenta en un factor de  $n$  y se convierte en  $O(n^2)$ .

Suponiendo que ya se ha implementado un procedimiento de desplazamiento, llamado SHIFT, el algoritmo de división se describe en el Algoritmo 2.4. La entrada del algoritmo es la matriz de afinidad agrupada CA, la relación R a fragmentar y las matrices de uso de atributos y frecuencia de acceso. La salida es un conjunto de fragmentos FR = {R1, R2}, donde  $R_i \subseteq \{A_1, A_2, \dots, A_n\}$  y  $R_1 \cap R_2 = \emptyset$  los atributos clave de la relación R. Tenga en cuenta que, para la partición de  $n$  vías, esta rutina debe invocarse iterativamente o implementarse como un procedimiento recursivo.

Ejemplo 2.18 Cuando se aplica el algoritmo SPLIT a la matriz CA obtenida para la relación PROJ (Ejemplo 2.17), el resultado es la definición de fragmentos FPROJ = {PROJ1,PROJ2}, donde

$$\text{PROJ1} = \{\text{PNO, PRESUPUESTO}\}$$

$$\text{PROJ2} = \{\text{PNO, PNAME, LOC}\}$$

Tenga en cuenta que en este ejercicio realizamos la fragmentación sobre todo el conjunto de atributos, en lugar de solo sobre los que no son clave. Esto se debe a la simplicidad del ejemplo. Por ello, incluimos PNO, que es la clave de PROJ tanto en PROJ2 como en PROJ1.

#### 2.1.2.4 Comprobación de la corrección

Seguimos argumentos similares a los de la partición horizontal para demostrar que el algoritmo SPLIT produce una fragmentación vertical correcta.

##### Lo completo

El algoritmo SPLIT garantiza la completitud, ya que cada atributo de la relación global se asigna a uno de los fragmentos. Siempre que el conjunto de atributos A sobre el que se define la relación R consista en  $A = R_i$ , se garantiza la completitud de la fragmentación vertical.

##### Reconstrucción

Ya hemos mencionado que la reconstrucción de la relación global original es posible gracias a la operación de unión. Por lo tanto, para una relación R con fragmentación vertical  $FR = \{R_1, R_2, \dots, R_r\}$  y atributo(s) clave K,  $R = K R_i, R_i \in FR$ . Por lo tanto, siempre que cada  $R_i$  esté completo, la operación de unión reconstruirá correctamente RA. Otro punto importante es que cada  $R_i$  debe contener el atributo(s) clave de R o los identificadores de tupla (TID) asignados por el sistema.

##### Desunión

Como se señaló anteriormente, los atributos de la clave principal se replican en cada fragmento. Excluyendo estos, el algoritmo SPLIT encuentra grupos de atributos mutuamente excluyentes, lo que genera fragmentos disjuntos con respecto a los atributos.

#### 2.1.3 Fragmentación híbrida

En algunos casos, una simple fragmentación horizontal o vertical de un esquema de base de datos puede no ser suficiente para satisfacer los requisitos de las aplicaciones de usuario. En este caso, una fragmentación vertical puede ir seguida de una horizontal, o viceversa, lo que produce una partición estructurada en tres (Fig. 2.16). Dado que los dos tipos de

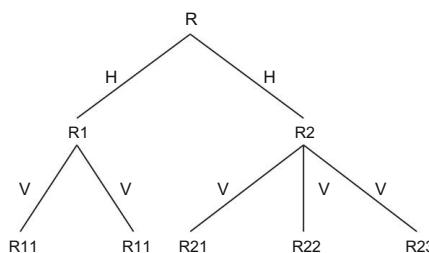


Fig. 2.16 Fragmentación híbrida

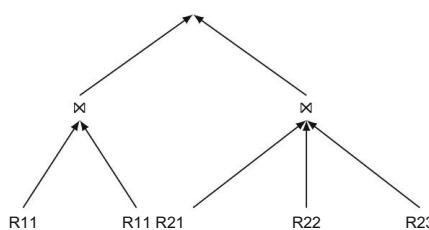


Fig. 2.17 Reconstrucción de la fragmentación híbrida

Las estrategias de partición se aplican una tras otra; esta alternativa se denomina fragmentación híbrida .

También se conoce como fragmentación mixta o fragmentación anidada .

Un buen ejemplo de la necesidad de la fragmentación híbrida es la relación PROJ.

En el Ejemplo 2.10, la dividimos en seis fragmentos horizontales basados en dos aplicaciones. En el Ejemplo 2.18, la dividimos verticalmente en dos.

Lo que tenemos, por tanto, es un conjunto de fragmentos horizontales, cada uno de los cuales está dividido en dos fragmentos verticales.

Las reglas y condiciones de corrección para la fragmentación híbrida se derivan naturalmente de las de las fragmentaciones verticales y horizontales. Por ejemplo, para reconstruir la relación global original en caso de fragmentación híbrida, se comienza por las hojas del árbol de partición y se avanza hacia arriba mediante uniones (Fig. 2.17).

La fragmentación es completa si los fragmentos intermedios y foliares están completos.

De manera similar, la disyunción está garantizada si los fragmentos intermedios y de hojas están disjuntos.

## 2.2 Asignación

Tras la fragmentación, el siguiente problema de decisión es asignar fragmentos a los sitios del SGBD distribuido. Esto puede hacerse colocando cada fragmento en un solo sitio o replicándolo en varios sitios. Las razones para la replicación son la fiabilidad y la eficiencia de las consultas de solo lectura. Si existen varias copias de...

un fragmento, hay una buena posibilidad de que se pueda acceder a alguna copia de los datos en algún lugar, incluso cuando se producen fallos del sistema. Además, las consultas de solo lectura que El acceso a los mismos elementos de datos se puede ejecutar en paralelo ya que existen copias en varios sitios. Por otro lado, la ejecución de consultas de actualización causa problemas ya que El sistema debe garantizar que todas las copias de los datos se actualicen correctamente. Por lo tanto, La decisión sobre la replicación es un equilibrio que depende de la proporción de consultas de solo lectura y consultas de actualización. Esta decisión afecta a casi todos los sistemas distribuidos.

#### Algoritmos DBMS y funciones de control.

Una base de datos no replicada (comúnmente llamada base de datos particionada ) contiene fragmentos que se asignan a sitios de manera que cada fragmento se coloca en un sitio. En caso de replicación, la base de datos existe en su totalidad en cada sitio (completamente base de datos replicada ), o los fragmentos se distribuyen a los sitios de tal manera que Las copias de un fragmento pueden residir en varios sitios (base de datos parcialmente replicada ). En En este último caso, el número de copias de un fragmento puede ser una entrada para la asignación. algoritmo o una variable de decisión cuyo valor está determinado por el algoritmo. La figura 2.18 compara estas tres alternativas de replicación con respecto a varias Funciones de DBMS distribuido. Analizaremos la replicación en detalle en el capítulo 6.

El problema de asignación de archivos se ha estudiado durante mucho tiempo en el contexto de la distribución. sistemas informáticos donde la unidad de asignación es un archivo. Esto se conoce comúnmente como Como el problema de asignación de archivos (FAP) y las formulaciones suelen ser bastante simples, lo que refleja la simplicidad de las API de archivos. Incluso esta versión simple ha demostrado ser... NP-completo, lo que resulta en una búsqueda de heurísticas razonables.

Las formulaciones FAP no son adecuadas para el diseño de bases de datos distribuidas, debido fundamentalmente a las características de los DBMS: los fragmentos no son independientes entre sí. otros, por lo que no se pueden asignar simplemente a archivos individuales; el acceso a los datos en un La base de datos es más compleja que el simple acceso a archivos; y los DBMS refuerzan la integridad y propiedades transaccionales cuyos costos deben ser considerados.

No existen modelos heurísticos generales que tomen como entrada un conjunto de fragmentos y producir una asignación casi óptima sujeta a los tipos de restricciones discutidos aquí. Los modelos desarrollados hasta la fecha hacen una serie de supuestos simplificadores y son aplicable a ciertas formulaciones específicas. Por lo tanto, en lugar de presentar una o

	Replicación completa	Replicación parcial	Particionado
CONSULTA TRATAMIENTO	Fácil	Misma dificultad	
DIRECTORIO GESTIÓN	Fácil o inexistente	Misma dificultad	
CONCURRENCIA CONTROL	Moderado	Difícil	Fácil
FIABILIDAD	Muy alto	Alto	Bajo
REALIDAD	Posible solicitud	Realista	Posible solicitud

Fig. 2.18 Comparación de alternativas de replicación

En más de estos algoritmos de asignación, presentamos un modelo relativamente general y luego analizamos una serie de posibles heurísticas que podrían emplearse para resolverlo.

## 2.2.1 Información auxiliar

Necesitamos datos cuantitativos sobre la base de datos, la carga de trabajo, la red de comunicación, las capacidades de procesamiento y las limitaciones de almacenamiento de cada sitio de la red.

Para realizar la fragmentación horizontal, definimos la selectividad de los mintérminos.

Ahora necesitamos extender esa definición a los fragmentos y definir la selectividad de un fragmento  $F_j$  con respecto a la consulta  $q_i$ . Este es el número de tuplas de  $F_j$  a las que se debe acceder para procesar  $q_i$ . Este valor se denominará como  $sel_i(F_j)$ .

Otro dato necesario sobre los fragmentos de la base de datos es su tamaño.

El tamaño de un fragmento  $F_j$  está dado por

$$\text{tamaño}(F_j) = \text{tarjeta}(F_j) \cdot \text{longitud}(F_j)$$

donde  $\text{length}(F_j)$  es la longitud (en bytes) de una tupla del fragmento  $F_j$ .

La mayor parte de la información relacionada con la carga de trabajo ya se compila durante la fragmentación, pero el modelo de asignación requiere información adicional. Las dos medidas importantes son el número de accesos de lectura que una consulta  $q_i$  realiza a un fragmento  $F_j$  durante su ejecución (denotado como  $RR_{ij}$ ) y su contraparte, los accesos de actualización ( $UR_{ij}$ ). Estos pueden, por ejemplo, contar el número de accesos a bloques requeridos por la consulta.

También necesitamos definir dos matrices  $UM$  y  $RM$ , con elementos  $uij$  y  $rij$ , respectivamente, que se especifican de la siguiente manera:

$$uij = \begin{cases} 1 & \text{si la consulta } q_i \text{ actualiza el fragmento } F_j \\ 0 & \text{en caso contrario} \end{cases}$$

$$rij = \begin{cases} 1 & \text{si la consulta } q_i \text{ recupera del fragmento } F_j \\ 0 & \text{caso contrario} \end{cases}$$

También se define un vector  $O$  de valores  $o(i)$ , donde  $o(i)$  especifica el sitio de origen de la consulta  $q_i$ . Finalmente, para definir la restricción de tiempo de respuesta, se debe especificar el tiempo de respuesta máximo permitido para cada aplicación.

Para cada sitio informático, necesitamos conocer su capacidad de almacenamiento y procesamiento. Obviamente, estos valores pueden calcularse mediante funciones complejas o estimaciones simples. El coste unitario de almacenamiento de datos en el sitio  $S_k$  se denominará como  $USC_k$ .

También es necesario especificar una medida de costo  $LP_{Ck}$  como el costo de procesar una unidad de trabajo en el sitio  $S_k$ . La unidad de trabajo debe ser idéntica a la de  $RR$  y  $UR$ . medidas.

En nuestro modelo, asumimos la existencia de una red simple donde el coste de la comunicación se define en función de un mensaje que contiene una cantidad específica de datos. Por lo tanto,  $g_{ij}$  representa el coste de comunicación por mensaje entre los sitios  $S_i$  y  $S_j$ .

Para calcular el número de mensajes, utilizamos  $msize$  como el tamaño (en bytes) de un mensaje. Existen modelos de red más elaborados que consideran la capacidad del canal, la distancia entre sitios, la sobrecarga del protocolo, etc., pero este modelo simple es suficiente para nuestros propósitos.

## 2.2.2 Modelo de asignación

Analizamos un modelo de asignación que busca minimizar el coste total de procesamiento y almacenamiento, a la vez que se intenta cumplir con ciertas restricciones de tiempo de respuesta. El modelo que utilizamos tiene la siguiente forma:

$$\min(\text{Costo total})$$

sujeto a

restricción de tiempo de respuesta

restricción de

almacenamiento restricción de procesamiento

En el resto de esta sección, ampliamos los componentes de este modelo con base en los requisitos de información analizados en la Sección 2.2.1. La variable de decisión es que se define como  $x_{ij}$ ,

$$x_{ij} = \begin{cases} 1 & \text{si el fragmento } F_i \text{ se almacena en el sitio } S_j \\ 0 & \text{en caso contrario} \end{cases}$$

### 2.2.2.1 Costo total

La función de coste total tiene dos componentes: procesamiento de consultas y almacenamiento. Por lo tanto, se puede expresar como

$$\begin{array}{ccc} \text{Tabla de contenidos} = & QP C_i + & STC_{jk} \\ q_i & Q & S_k \quad S \quad F_j \quad F \end{array}$$

donde  $QP C_i$  es el costo de procesamiento de la consulta  $q_i$ , y  $STC_{jk}$  es el costo de almacenar el fragmento  $F_j$  en el sitio  $S_k$ .

Consideremos primero el costo de almacenamiento. Se calcula simplemente por

$$STC_{jk} = USC_k \quad \text{tamaño}(F_j) \quad x_{jk}$$

y las dos sumas encuentran los costos totales de almacenamiento en todos los sitios para todos los fragmentos.

El costo de procesamiento de consultas es más difícil de especificar. Lo especificamos como compuesto por el costo de procesamiento (CP) y el costo de transmisión (TC). Por lo tanto, el costo de procesamiento de consultas (QP C) para la aplicación  $q_i$  es

$$QP\ C_i = P\ C_i + T\ C_i$$

El componente de procesamiento, PC, consta de tres factores de costo, el costo de acceso (AC), el costo de cumplimiento de integridad (IE) y el costo de control de concurrencia (CC):

$$P\ C_i = A\ C_i + I\ E_i + C\ C_i$$

La especificación detallada de cada uno de estos factores de coste depende de los algoritmos utilizados para realizar estas tareas. Sin embargo, para ilustrar este punto, especificamos CA con cierto detalle:

$$A\ C_i = \frac{(u_{ij} \quad U\ R_{ij} + r_{ij} \quad R\ R_{ij})}{S_k \quad S \quad F_j \quad F} \quad x_{jk} \quad L\ P\ C_k$$

Los dos primeros términos de la fórmula anterior calculan el número de accesos de la consulta de usuario  $q_i$  al fragmento  $F_j$ . Nótese que  $(U\ R_{ij} + R\ R_{ij})$  da el número total de accesos de actualización y recuperación. Suponemos que los costes locales de procesamiento son idénticos. La suma da el número total de accesos para todos los fragmentos referenciados por  $q_i$ . La multiplicación por  $L\ P\ C_k$  da el coste de este acceso en el sitio  $S_k$ .

Nuevamente utilizamos  $x_{jk}$  para seleccionar solo aquellos valores de costo para los sitios donde se almacenan los fragmentos.

La función de costo de acceso supone que procesar una consulta implica descomponerla en un conjunto de subconsultas, cada una de las cuales trabaja en un fragmento almacenado en el sitio, y luego transmitir los resultados al sitio donde se originó la consulta.

La realidad es más compleja; por ejemplo, la función de costo no tiene en cuenta el costo de realizar uniones (si es necesario), que pueden ejecutarse de diversas maneras (ver Cap. 4).

El factor de costo de cumplimiento de la integridad puede especificarse de forma similar al componente de procesamiento, salvo que el costo unitario de procesamiento local probablemente cambiaría para reflejar el costo real del cumplimiento de la integridad. Dado que los métodos de verificación de integridad y control de concurrencia se describen más adelante en el libro, no profundizaremos en estos componentes de costo. El lector debe consultar esta sección después de leer los capítulos 3 y 5 para comprobar que las funciones de costo pueden derivarse.

La función de coste de transmisión puede formularse siguiendo los lineamientos de la función de coste de acceso. Sin embargo, la sobrecarga de transmisión de datos para las solicitudes de actualización y de recuperación puede ser bastante diferente. En las consultas de actualización, es necesario informar a todos los sitios donde existen réplicas, mientras que en las consultas de recuperación, basta con acceder solo a una de las copias. Además, al finalizar una solicitud de actualización, no se transmiten datos al sitio de origen, salvo un mensaje de confirmación, mientras que las consultas de solo recuperación pueden resultar en una transmisión de datos significativa.

El componente de actualización de la función de transmisión es

$$TCUi = \frac{uij \quad xjk \quad go(i),k + uij \quad xjk \quad gk,o(i) \quad Sk \quad S \quad Fj \quad F \quad Sk \quad S \quad Fj \quad F}{Fj \quad F}$$

El primer término se utiliza para enviar el mensaje de actualización desde el sitio de origen o(i) de q(i) a todas las réplicas de fragmentos que deben actualizarse. El segundo término se utiliza para la confirmación.

El coste de recuperación se puede especificar como

$$TCRi = \frac{(rij \quad xjk \quad go(i),k + rij \quad xjk \quad gk,o(i)) \quad rmsize \quad seli(Fj) \quad longitud(Fj) \quad min}{Fj \quad F \quad Sk \quad S}$$

El primer término del TCR representa el coste de transmisión de la solicitud de recuperación a los sitios que tienen copias de los fragmentos a los que se debe acceder. El segundo término representa la transmisión de los resultados desde estos sitios al sitio de origen. La ecuación establece que, entre todos los sitios con copias del mismo fragmento, solo el que ofrezca el coste total de transmisión mínimo debe seleccionarse para la ejecución de la operación.

Ahora la función de costo de transmisión para la consulta q(i) se puede especificar como

$$T Ci = TCUi + TCRi$$

que especifica completamente la función de costo total.

#### 2.2.2.2 Restricciones

Las funciones de restricción pueden especificarse con un detalle similar. Sin embargo, en lugar de describirlas en profundidad, simplemente indicaremos su aspecto. La restricción de tiempo de respuesta debe especificarse como

$$\text{tiempo de ejecución de } q_i \leq \text{tiempo máximo de respuesta de } q_i, \quad q_i \quad Q$$

Preferiblemente, la medida del costo en la función objetivo debe especificarse en términos de tiempo, ya que hace que la especificación de la restricción del tiempo de ejecución sea relativamente sencilla.

La restricción de almacenamiento es

$$\frac{STC_{jk} \leq \text{capacidad de almacenamiento en el sitio } Sk, \quad Sk \quad S}{Fj \quad F}$$

Mientras que la restricción de procesamiento es

$$\frac{\text{carga de procesamiento de } q_i \text{ en el sitio } Sk \leq \text{capacidad de procesamiento de } Sk, \quad Sk \quad S}{qi \quad Q}$$

Con esto finalizamos el desarrollo del modelo de asignación. Si bien no lo hemos desarrollado en su totalidad, la precisión de algunos términos indica cómo se puede formular este problema. Además, hemos indicado las cuestiones importantes que deben abordarse en los modelos de asignación.

### 2.2.3 Métodos de solución

Como se mencionó anteriormente, un problema simple de asignación de archivos es NP-completo. Dado que el modelo desarrollado en la sección anterior es más complejo, es probable que también lo sea. Por lo tanto, es necesario buscar métodos heurísticos que generen soluciones subóptimas. La prueba de "bondad" en este caso radica, obviamente, en la proximidad de los resultados del algoritmo heurístico a la asignación óptima.

Se observó desde el principio una correspondencia entre los problemas de asignación de archivos y de ubicación de instalaciones. De hecho, se ha demostrado el isomorfismo entre el problema simple de asignación de archivos y el problema de ubicación de un almacén de productos básicos. Por lo tanto, las heurísticas desarrolladas para este último se han aplicado al primero.

Algunos ejemplos son la solución del problema de la mochila, las técnicas de ramificación y acotación y los algoritmos de flujo de red.

Se han realizado otros intentos para reducir la complejidad del problema. Una estrategia ha sido asumir que se han determinado todas las particiones candidatas, junto con sus costos y beneficios asociados en términos de procesamiento de consultas. El problema, entonces, se modela como la elección de la partición y la ubicación óptimas para cada relación. Otra simplificación frecuentemente empleada consiste en ignorar la replicación inicialmente y encontrar una solución óptima no replicada. La replicación se gestiona en un segundo paso mediante la aplicación de un algoritmo voraz que parte de la solución no replicada como la solución factible inicial e intenta mejorárla. Sin embargo, para estas heurísticas, no hay suficientes datos para determinar cuán cerca están los resultados del óptimo.

## 2.3 Enfoques combinados

El proceso de diseño representado en la Fig. 2.1 , en el que basamos nuestra discusión, separa los pasos de fragmentación y asignación. La metodología es lineal, donde el resultado de la fragmentación es la entrada para la asignación; lo denominamos enfoque de fragmentar y luego asignar. Esto simplifica la formulación del problema al reducir el espacio de decisión, pero el aislamiento de ambos pasos puede, de hecho, contribuir a la complejidad de los modelos de asignación. Ambos pasos tienen entradas similares, diferenciándose únicamente en que la fragmentación trabaja con relaciones globales, mientras que la asignación considera fragmentos.

Ambos requieren información sobre la carga de trabajo, pero ignoran cómo cada uno utiliza estas entradas. El resultado final es que los algoritmos de fragmentación deciden cómo partitionar una relación basándose parcialmente en cómo las consultas acceden a ella, pero los modelos de asignación...

Ignoran el papel que esta entrada desempeña en la fragmentación. Por lo tanto, los modelos de asignación deben incluir, una vez más, una especificación detallada de la relación entre las relaciones de fragmentos y cómo las aplicaciones de usuario acceden a ellas. Existen enfoques que combinan los pasos de fragmentación y asignación de tal manera que el algoritmo de partición de datos también dicta la asignación, o bien, el algoritmo de asignación dicta cómo se partitionan los datos; estos se denominan enfoques combinados.

Estos enfoques se basan principalmente en la partición horizontal, ya que es el método habitual para lograr un paralelismo significativo. En esta sección, presentamos estos enfoques, clasificados como independientes de la carga de trabajo o conscientes de ella.

### 2.3.1 Técnicas de particionamiento independientes de la carga de trabajo

Esta clase de técnicas ignora la carga de trabajo que se ejecutará en los datos y simplemente se centra en la base de datos, a menudo sin siquiera prestar atención a la definición del esquema. Estos enfoques se utilizan principalmente en DBMS paralelos donde el dinamismo de los datos es mayor que en los DBMS distribuidos, por lo que se prefieren técnicas más simples que se puedan aplicar rápidamente.

La forma más simple de estos algoritmos es la partición round-robin (Fig. 2.19).

Con  $n$  particiones, la  $i$ -ésima tupla en orden de inserción se asigna a la partición  $(i \bmod n)$ . Esta estrategia permite el acceso secuencial a una relación en paralelo.

Sin embargo, el acceso directo a tuplas individuales, basado en un predicado, requiere acceder a la relación completa. Por lo tanto, la partición round-robin es adecuada para consultas de escaneo completo, como en minería de datos.

Una alternativa es la partición hash, que aplica una función hash a algún atributo que genera el número de partición (Fig. 2.20). Esta estrategia permite que las consultas de coincidencia exacta sobre el atributo de selección sean procesadas por un solo nodo y todos

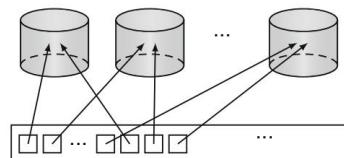


Fig. 2.19 Partición round-robin

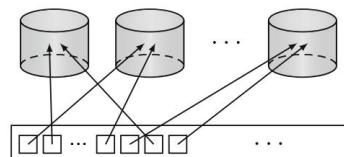


Fig. 2.20 Partición hash

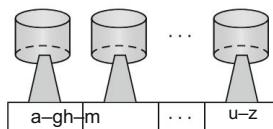


Fig. 2.21 Partición de rango

Otras consultas deben ser procesadas por todos los nodos en paralelo. Sin embargo, si el atributo utilizado para la partición tiene una distribución de datos no uniforme, por ejemplo, con los nombres de las personas, la ubicación resultante puede ser desequilibrada, con algunas particiones mucho más grandes que otras. Esto se denomina sesgo de datos y es un problema importante que puede causar una carga desequilibrada.

Finalmente, existe la partición de rangos (Fig. 2.21), que distribuye tuplas según los intervalos de valores (rangos) de algún atributo y, por lo tanto, puede gestionar distribuciones de datos no uniformes. A diferencia del hash, que se basa en funciones hash, los rangos deben mantenerse en una estructura de índice, por ejemplo, un árbol B. Además de admitir consultas de coincidencia exacta (como en el hash), es ideal para consultas de rangos. Por ejemplo, una consulta con un predicado "A entre A1 y A2" puede ser procesada por el único nodo o los únicos nodos que contienen tuplas cuyo valor A está en el rango [A1, A2].

Estas técnicas son sencillas, se calculan rápidamente y, como se explica en el capítulo 8, se adaptan perfectamente al dinamismo de los datos en sistemas de gestión de bases de datos (SGBD) paralelos. Sin embargo, ofrecen métodos indirectos para gestionar las relaciones semánticas entre las relaciones de la base de datos. Por ejemplo, si consideramos dos relaciones con una relación de unión de clave externa y clave primaria, como R RA=SB S, la partición hash utilizaría la misma función sobre los atributos RA y SB para garantizar que se encuentren en el mismo nodo, localizando así las uniones y paralelizando su ejecución. Se puede utilizar un enfoque similar en la partición por rangos, pero la operación round-robin no consideraría esta relación.

### 2.3.2 Técnicas de particionamiento teniendo en cuenta la carga de trabajo

Esta clase de técnicas considera la carga de trabajo como entrada y realiza particiones para localizar la mayor cantidad posible de carga de trabajo en un sitio. Como se mencionó al principio de este capítulo, su objetivo es minimizar la cantidad de consultas distribuidas.

Un enfoque que se ha propuesto en un sistema llamado Schism utiliza la información de la base de datos y de la carga de trabajo para construir un gráfico  $G = V, E$  donde cada vértice  $v$  en  $V$  representa una tupla en la base de datos y cada borde  $e = (vi, vj)$  en  $E$  representa una consulta que accede a ambas tuplas  $vi$  y  $vj$ .

A cada borde se le asigna un peso que es el recuento de la cantidad de transacciones que acceden a ambas tuplas.

En este modelo, también es fácil considerar las réplicas, representando cada copia con un vértice independiente. El número de vértices de réplica se determina por el número de transacciones que acceden a la tupla; es decir, cada transacción accede a una

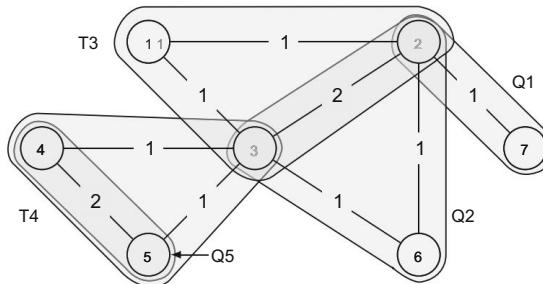


Fig. 2.22 Representación gráfica para la partición en cisma

Copia. Una tupla replicada se representa en el grafo mediante una configuración en forma de estrella que consta de  $n+1$  vértices, donde el vértice "central" representa la tupla lógica y los otros  $n$  vértices representan las copias físicas. El peso de una arista entre el vértice de la copia física y el vértice central es el número de transacciones que actualizan la tupla; el peso de las otras aristas se mantiene como el número de consultas que acceden a la tupla. Esta disposición es lógica, ya que el objetivo es localizar las transacciones tanto como sea posible y esta técnica utiliza la replicación para lograr la localización.

Ejemplo 2.19. Consideremos una base de datos con una relación compuesta por siete tuplas a las que acceden cinco transacciones. En la figura 2.22 se representa el grafo construido: hay siete vértices correspondientes a las tuplas, y las consultas que las acceden conjuntamente se muestran como grupos. Por ejemplo, la consulta Q1 accede a las tuplas 2 y 7, la consulta Q2 a las tuplas 2, 3 y 6, la consulta Q3 a las tuplas 1, 2 y 3, la consulta Q4 a las tuplas 3, 4 y 5, y la consulta Q5 a las tuplas 4 y 5.

Los pesos de los bordes capturan el número de accesos a transacciones.

La replicación se puede incorporar en este grafo, replicando las tuplas a las que acceden múltiples transacciones; esto se muestra en la Fig. 2.23. Nótese que las tuplas 1, 6 y 7 no se replican, ya que solo se accede a ellas una transacción cada una; las tuplas 4 y 5 se replican dos veces, y las tuplas 2 y 3 se replican tres veces. Representamos las aristas de replicación entre el vértice central y cada copia física mediante líneas discontinuas y omitimos los pesos de estas aristas en este ejemplo.

Una vez que la base de datos y la carga de trabajo son capturadas por esta representación gráfica, el siguiente paso es realizar una partición gráfica disjunta por vértices. Dado que estas técnicas se describen en detalle en la sección 10.4.1, no se profundizará en los detalles aquí, sino que simplemente se indica que la partición disjunta por vértices asigna cada vértice del grafo a una partición separada, de modo que las particiones sean mutuamente excluyentes. Estos algoritmos tienen como función objetivo un conjunto de particiones equilibrado (o casi equilibrado), minimizando al mismo tiempo el coste de los cortes de aristas. El coste de un corte de arista considera los pesos de cada arista para minimizar el número de consultas distribuidas.

La ventaja del enfoque Schism es su asignación de grano fino: trata cada tupla como una unidad de asignación y la partición surge a medida que se toma la decisión de asignación para cada tupla. Por lo tanto, la asignación de conjuntos de tuplas a consultas puede

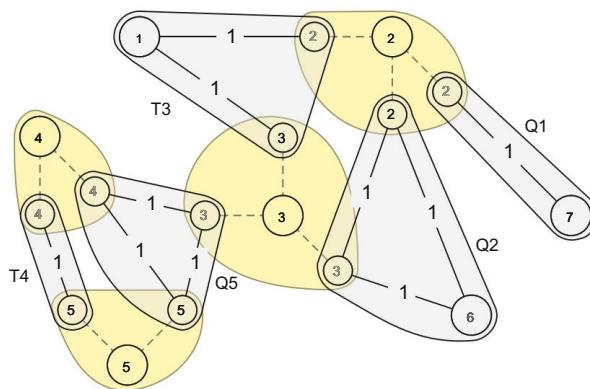


Fig. 2.23 Gráfico de cisma que incorpora replicación

Se pueden controlar y muchos de ellos pueden ejecutarse en un solo sitio. Sin embargo, la desventaja de este enfoque es que el grafo se vuelve muy grande a medida que aumenta el tamaño de la base de datos, en particular al añadir réplicas. Esto dificulta la gestión del grafo y encarece su partición. Otro problema a considerar es que las tablas de mapeo que registran dónde se almacena cada tupla (es decir, el directorio) se vuelven muy grandes y pueden plantear un problema de gestión.

Se ha propuesto un enfoque para superar estos problemas como parte del sistema SWORD, que emplea un modelo de hipergrafo<sup>7</sup> donde cada grupo en la Fig. 2.22 se representa como una hiperarista. Cada hiperarista representa una consulta y el conjunto de vértices abarcados por ella representa las tuplas a las que accede. Cada hiperarista tiene un peso que representa la frecuencia de esa consulta en la carga de trabajo. Por lo tanto, se obtiene un hipergafo ponderado. Este hipergafo se partitiona mediante un algoritmo de partición balanceada de  $k$  vías y corte mínimo, que produce  $k$  particiones balanceadas, cada una de las cuales se asigna a un sitio. Esto minimiza el número de consultas distribuidas, ya que el algoritmo minimiza los cortes en las hiperaristas, y cada uno de estos cortes indica una consulta distribuida.

Por supuesto, este cambio en el modelo no es suficiente para abordar los problemas mencionados anteriormente. Para reducir el tamaño del grafo y la sobrecarga de mantener la tabla de mapeo asociada, SWORD comprime este hipergafo de la siguiente manera. El conjunto de vértices  $V$  del hipergafo original  $G$  se mapea a un conjunto de vértices virtuales  $V'$  mediante un hash u otra función que opera sobre las claves primarias de las tuplas. Una vez determinado el conjunto de vértices virtuales, las aristas del hipergafo original se mapean a las hiperaristas del grafo comprimido ( $E'$ ), de modo que si los vértices abarcados por una hiperarista  $e \in E$  se mapean a diferentes vértices virtuales del grafo comprimido, entonces habrá una hiperarista  $e' \in E'$ .

<sup>7</sup>Un hipergafo permite que cada arista (denominada hiperarista) conecte más de dos vértices, como ocurre en los grafos regulares. Los detalles del modelo de hipergafo quedan fuera de nuestro alcance.

## 2.3 Enfoques combinados

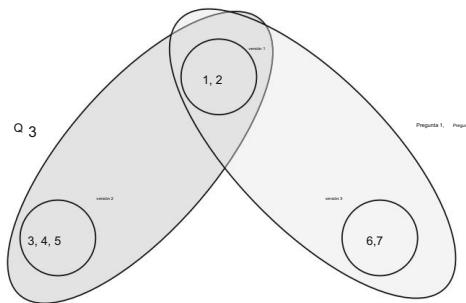


Fig. 2.24 Hipergrafo comprimido de espada

Por supuesto, para que esta compresión tenga sentido,  $|V'| < |V|$ , un aspecto crítico es determinar el nivel de compresión deseado. Una compresión excesiva reducirá el número de vértices virtuales, pero aumentará el número de hiperaristas y, por lo tanto, la posibilidad de consultas distribuidas. El hipergrafo comprimido resultante,  $G = (V', E)$ , será más pequeño que el hipergrafo original, por lo que será más fácil de gestionar y particionar. Las tablas de mapeo también serán más pequeñas, ya que solo considerarán el mapeo de conjuntos de vértices virtuales.

Ejemplo 2.20. Retomemos el caso del Ejemplo 2.19 y consideremos que estamos comprimiendo el hipergrafo en tres vértices virtuales:  $v_6, v_7$ . Entonces, habría dos hiperaristas:  $v_3 = \{1, 2\}$ ,  $v_2 = \{3, 4, 5\}$ ,  $v_3 = \{6, 7\}$  con frecuencia 2 (correspondiente a Q1 y Q2 en el hipergrafo original) y  $e = \{vv_2\}$  con frecuencia 1 (correspondiente a Q3). Las hiperaristas que representan las consultas Q4 y Q5 serían locales (es decir, no abarcarián vértices virtuales), por lo que no se requieren hiperaristas en el hipergrafo comprimido. Esto se muestra en la Fig. 2.24.

La realización de la partición de corte mínimo balanceada de  $k$  vías en el hipergráfico comprimido se puede realizar mucho más rápido y la tabla de mapeo resultante será más pequeña debido al tamaño reducido del gráfico.

SWORD incorpora la replicación en el hipergrafo comprimido. Primero, determina cuántas réplicas se requieren para cada vértice virtual. Esto se logra utilizando las estadísticas del patrón de acceso a nivel de tupla para cada tupla  $t_j$  en cada vértice virtual  $v_i$ , es decir, su frecuencia de lectura  $f_{ij}^l$  y su frecuencia de escritura  $f_{ij}^w$ . Con estos datos, calcula las frecuencias promedio de lectura y escritura (ARF y AWF, respectivamente) del vértice virtual de la siguiente manera: vértice  $v_i$

$$\text{ARF}(v_i) = \log \frac{\sum_j f_{ij}^l}{S(v_i)} \quad \text{AWF}(v_i) = \log \frac{\sum_j f_{ij}^w}{S(v_i)}$$

$S(v_i)$  es el tamaño de cada vértice virtual (en términos del número de vértices reales asignados a él) y se toma su logaritmo para compensar la desviación en los tamaños de los vértices virtuales (por lo tanto, estos son promedios con compensación de tamaño). A partir de estos, SWORD define un

factor de replicación,  $R = \frac{AWF(v_i)}{\delta}$  y un umbral  $\delta$  especificado por el usuario ( $0 < \delta < 1$ ) se definen como ARWF ( $v_i$ ). El número de réplicas (#\_rep) para el vértice virtual  $v_i$  se da entonces como

$$\#_{\text{rep}}(v_i) = \begin{cases} 1 & \text{si } R \geq \delta \\ \text{ARF}(v_i) & \text{de lo contrario} \end{cases}$$

Una vez determinado el número de réplicas para cada vértice virtual, estas se añaden al hipergrafo comprimido y se asignan a las hiperaristas de forma que se minimice el corte mínimo en el algoritmo de partición. Ignoramos los detalles de esta asignación.

## 2.4 Enfoques adaptativos

El trabajo descrito en este capítulo generalmente asume un entorno estático donde el diseño se realiza solo una vez y este diseño puede persistir. La realidad, por supuesto, es bastante diferente. Se producen cambios tanto físicos (p. ej., características de la red, almacenamiento disponible en varios sitios) como lógicos (p. ej., carga de trabajo) que requieren el rediseño de la base de datos. En un entorno dinámico, el proceso se convierte en uno de diseño-rediseño-materialización del rediseño. Cuando las cosas cambian, el enfoque más simple es rebajar el diseño de la distribución desde cero. Para sistemas grandes o altamente dinámicos, esto no es del todo realista ya que la sobrecarga del rediseño probablemente sea muy alta. Un enfoque preferido es realizar un rediseño incremental, centrándose solo en las partes de la base de datos que probablemente se verán afectadas por los cambios. El rediseño incremental puede realizarse cada vez que se detecta un cambio o periódicamente, donde los cambios se agrupan y se evalúan a intervalos regulares.

La mayor parte del trabajo en esta área se ha centrado en los cambios en la carga de trabajo (consultas y transacciones) a lo largo del tiempo, y en estos aspectos nos centraremos en esta sección. Si bien algunos trabajos se han centrado en el enfoque de fragmentación y posterior asignación, la mayoría sigue el enfoque combinado. En el primer caso, una alternativa propuesta consiste en una fase de división donde los fragmentos se subdividen según los requisitos modificados de la aplicación hasta que una subdivisión adicional ya no sea rentable según una función de costes. En este punto, comienza la fase de fusión, donde los fragmentos a los que acceden conjuntamente varias aplicaciones se fusionan en un solo fragmento. Nos centraremos más en los enfoques combinados dinámicos que realizan un rediseño incremental a medida que cambia la carga de trabajo.

El objetivo de los enfoques adaptativos es el mismo que el de las estrategias de particionamiento basadas en la carga de trabajo, descritas en la sección 2.3.2: minimizar el número de consultas distribuidas y garantizar que los datos de cada consulta sean locales. En este contexto, existen tres cuestiones interrelacionadas que deben abordarse en el diseño de distribuciones adaptativas:

1. ¿Cómo detectar cambios en la carga de trabajo que requieren cambios en el diseño de la distribución?
2. ¿Cómo determinar qué elementos de datos se verán afectados en el diseño?
3. ¿Cómo realizar los cambios de manera eficiente?

En el resto analizamos cada uno de estos temas.

### 2.4.1 Detección de cambios en la carga de trabajo

Este es un tema complejo en el que no se ha trabajado mucho. La mayoría de las técnicas adaptativas propuestas asumen que se detecta el cambio en la carga de trabajo y se centran simplemente en el problema de la migración. Para detectar cambios en la carga de trabajo, es necesario monitorizar las consultas entrantes. Una forma de hacerlo es examinar periódicamente los registros del sistema, pero esto puede suponer una sobrecarga elevada, especialmente en sistemas altamente dinámicos. Una alternativa es monitorizar continuamente la carga de trabajo dentro del SGBD. En el sistema SWORD que analizamos anteriormente, el sistema monitoriza el aumento porcentual del número de transacciones distribuidas y considera que el sistema ha cambiado lo suficiente como para requerir una reconfiguración si este aumento supera un umbral definido. Otro ejemplo es el sistema E-Store, que monitoriza tanto las métricas a nivel de sistema como el acceso a nivel de tupla. Comienza recopilando métricas a nivel de sistema en cada nodo informático mediante las funciones del sistema operativo. Actualmente, E-Store se centra principalmente en detectar desequilibrios en la carga de trabajo entre los nodos informáticos y, por lo tanto, solo recopila datos de utilización de la CPU. Si el desequilibrio en la utilización de la CPU supera un umbral, se activa una monitorización más detallada a nivel de tupla para detectar los elementos afectados (véase la siguiente sección). Aunque el desequilibrio en la utilización de la CPU puede ser un buen indicador de posibles problemas de rendimiento, es demasiado simple para capturar cambios más significativos en la carga de trabajo. Por supuesto, es posible realizar una monitorización más sofisticada; por ejemplo, se puede crear un perfil que observe la frecuencia de cada consulta en un período determinado, el porcentaje de consultas que cumplen (o superan) las latencias acordadas (como se refleja en un acuerdo de nivel de servicio, por ejemplo), entre otros. Después, se puede decidir si los cambios en el perfil requieren un rediseño, que puede realizarse de forma continua (es decir, cada vez que el monitor registra información) o periódica. El reto aquí es hacer esto de forma eficiente sin afectar al rendimiento del sistema. Esta es un área de investigación abierta que no se ha estudiado adecuadamente.

### 2.4.2 Detección de elementos afectados

Una vez detectado un cambio en la carga de trabajo, el siguiente paso es determinar qué elementos de datos se ven afectados y deben migrarse para abordar este cambio. La forma de hacerlo depende en gran medida del método de detección. Por ejemplo, si el sistema monitorea la frecuencia de las consultas y detecta cambios, estas identificarán los elementos de datos. Es posible generalizar desde consultas individuales a plantillas de consulta para capturar consultas similares que también podrían verse afectadas por los cambios. Esto se realiza en el sistema Apollo, donde cada constante se reemplaza por un comodín. Por ejemplo, la consulta

SELECCIONAR PNAME DE PROYECTOS DONDE PRESUPUESTO > 200000 Y LOC = "Londres"

se generalizaría a

SELECCIONAR NOMBRE DEL PROYECTO DONDE PRESUPUESTO>? Y LOC = "?"

Si bien esto reduce la granularidad para determinar el conjunto exacto de elementos de datos que se ven afectados, puede permitir la detección de elementos de datos adicionales que podrían verse afectados por consultas similares y reducir la frecuencia de los cambios necesarios.

El sistema E-Store inicia la monitorización a nivel de tupla al detectar un desequilibrio en la carga del sistema. Durante un breve periodo, recopila datos de acceso a las tuplas en cada nodo informático (es decir, cada partición) y determina las tuplas "calientes", que son las k tuplas con mayor acceso en un periodo determinado. Para ello, utiliza un histograma para cada tupla, que se inicializa al habilitar la monitorización a nivel de tupla y se actualiza a medida que se produce el acceso dentro de la ventana de monitorización. Al final de este periodo, se compila la lista de las k tuplas principales. El software de monitorización recopila estas listas y genera una lista global de las k tuplas principales: estos son los elementos de datos que deben migrarse. Un efecto secundario es la determinación de las tuplas frías; son de especial importancia las tuplas que anteriormente eran calientes y que desde entonces se han vuelto frías. La determinación de la ventana de tiempo para la monitorización a nivel de tupla y el valor de k son parámetros establecidos por el administrador de la base de datos.

### 2.4.3 Reconfiguración incremental

Como se mencionó anteriormente, el enfoque ingenuo para rediseñar consiste en rehacer toda la partición y distribución de datos. Si bien esto puede ser útil en entornos donde los cambios en la carga de trabajo ocurren con poca frecuencia, en la mayoría de los casos, la sobrecarga del rediseño es demasiado alta para hacerlo desde cero. El enfoque preferido es aplicar los cambios de forma incremental mediante la migración de datos; en otras palabras, solo examinamos la carga de trabajo modificada y los elementos de datos afectados, y los movemos.<sup>8</sup> Por lo tanto, en esta sección, nos centraremos en los enfoques incrementales.

Siguiendo la sección anterior, un enfoque obvio es utilizar un algoritmo de partición incremental de grafos que reacciona a los cambios en la representación gráfica que analizamos. Esto se ha seguido en el sistema SWORD mencionado anteriormente y en AdaptCache; ambos representan el uso como hipergrafos y realizan partición incremental en estos grafos. La partición incremental de grafos inicia la migración de datos para la reconfiguración.

El sistema de tienda electrónica que hemos estado analizando adopta un enfoque más sofisticado. Una vez identificado el conjunto de tuplas activas, se elabora un plan de migración que identifica dónde deben trasladarse las tuplas activas y qué reasignación de tuplas inactivas es necesaria. Esto puede plantearse como un problema de optimización que crea una carga equilibrada entre los nodos informáticos (el equilibrio se define como la carga promedio entre los nodos).

---

La investigación en esta área se ha centrado exclusivamente en la partición horizontal, que también será nuestro foco aquí, lo que significa que nuestras unidades de migración son tuplas individuales.

± un valor umbral), pero resolver este problema de optimización en tiempo real para la reconfiguración en línea no es fácil, por lo que se utilizan enfoques de ubicación aproximada (p. ej., greedy, first-fit) para generar el plan de reconfiguración. Básicamente, primero se determinan los nodos de cómputo apropiados donde debe ubicarse cada tupla activa y luego se abordan las tuplas inactivas, si es necesario debido al desequilibrio restante, moviéndolas en bloques. Por lo tanto, el plan de reconfiguración generado aborda la migración de las tuplas activas individualmente, pero la migración de las tuplas inactivas como bloques. Como parte del plan, se determina un nodo coordinador para gestionar la migración, y este plan constituye una entrada para el sistema de reconfiguración Squall.

Squall realiza la reconfiguración y la migración de datos en tres pasos. En el primer paso, el coordinador identificado en el plan de reconfiguración inicializa el sistema para la migración. Este paso incluye la coordinación para obtener acceso exclusivo a todas estas particiones mediante una transacción, como se explicará en el capítulo 5. A continuación, el coordinador solicita a cada sitio que identifique las tuplas que se transferirán de la partición local y las que se transferirán. Este análisis se realiza en los metadatos, por lo que puede realizarse rápidamente. Tras esto, cada sitio notifica al coordinador y finaliza la transacción de inicialización. En el segundo paso, el coordinador indica a cada sitio que realice la migración de datos. Esto es crucial, ya que hay consultas que acceden a los datos mientras se transfieren. Si una consulta se ejecuta en un nodo informático determinado donde se supone que deben estar los datos según el plan de reconfiguración, pero las tuplas requeridas no están disponibles localmente, Squall extrae las tuplas faltantes para procesar la consulta. Esto se realiza además de la migración normal de los datos según el plan de reconfiguración. En otras palabras, para ejecutar las consultas de manera oportuna, Squall realiza un movimiento a pedido además de su migración normal.

Una vez completado este paso, cada nodo informa al coordinador, que luego inicia el paso de terminación final e informa a cada nodo que la reconfiguración se ha completado.

Estos tres pasos son necesarios para que Squall pueda realizar la migración mientras ejecuta consultas de usuario al mismo tiempo en lugar de detener toda la ejecución de consultas, realizar la migración y luego reiniciar la ejecución de la consulta.

Otro enfoque es el cracking de bases de datos, una técnica de indexación adaptativa enfocada en cargas de trabajo dinámicas y difíciles de predecir, y en escenarios donde hay poco o ningún tiempo de inactividad para el análisis de la carga de trabajo y la creación de índices. El cracking de bases de datos funciona reorganizando continuamente los datos para que se ajusten a la carga de trabajo de las consultas. Cada consulta sirve como guía sobre cómo almacenar los datos. El cracking logra esto mediante la creación y el refinamiento de índices de forma parcial e incremental como parte del procesamiento de las consultas. Al reaccionar a cada consulta con acciones ligeras, el cracking de bases de datos se adapta instantáneamente a una carga de trabajo cambiante. A medida que llegan más consultas, los índices se refinan y el rendimiento mejora, alcanzando finalmente el rendimiento óptimo, es decir, el que obtendríamos con un sistema ajustado manualmente.

La idea principal del enfoque original de descifrado de bases de datos es que el sistema de datos reorganiza una columna de datos a la vez y solo cuando se realiza una consulta.

En otras palabras, la reorganización aprovecha el hecho de que los datos ya se han leído y decide cómo refinarlos de la mejor manera. En efecto, el enfoque de descifrado original sobrecarga el operador de selección de un sistema de base de datos y utiliza los predicados de cada consulta para determinar cómo reorganizar la columna relevante. La primera vez que un atributo...

Una consulta requiere A y crea una copia de la columna base A, denominada columna de cracker de A. Cada operador de selección sobre A activa la reorganización física de la columna de cracker según el rango solicitado en la consulta. Las entradas con una clave menor que el límite inferior se colocan antes de este, mientras que las entradas con una clave mayor que el límite superior se colocan después de este en la columna correspondiente. La información de partición de cada columna de cracker se guarda en un árbol AVL, el índice de cracker. Las consultas futuras sobre la columna A buscan en el índice de cracker la partición donde se encuentra el rango solicitado. Si la clave solicitada ya existe en el índice (es decir, si consultas anteriores han realizado crackers en exactamente esos rangos), el operador de selección puede devolver el resultado inmediatamente. De lo contrario, el operador de selección refina la columna sobre la marcha; es decir, solo se reorganizan las particiones o partes de la columna donde se encuentran los predicados (como máximo dos particiones en los límites del rango). Progresivamente, la columna se ordena más con más partes, pero más pequeñas.

El concepto principal del cracking de bases de datos y sus técnicas básicas se puede extender a la partición de datos en un entorno distribuido; es decir, al almacenamiento de datos en un conjunto de nodos mediante consultas entrantes. Cada vez que un nodo necesita una parte específica de los datos para una consulta local, pero los datos no existen en él, esta información puede utilizarse como indicio de que los datos podrían transferirse a él. Sin embargo, a diferencia de los métodos de cracking de bases de datos en memoria, donde el sistema reacciona inmediatamente a cada consulta, en un entorno distribuido debemos considerar que transferir los datos es más costoso. Al mismo tiempo, por la misma razón, el beneficio que puedan tener las consultas futuras será mayor. De hecho, ya se ha estudiado la misma desventaja en variaciones del enfoque original de cracking de bases de datos para optimizar los datos en disco. El efecto neto es doble: (1) en lugar de reaccionar ante cada consulta, deberíamos esperar a tener más evidencia de la carga de trabajo antes de embarcarnos en costosas reorganizaciones de datos, y (2) deberíamos aplicar reorganizaciones más complejas para aprovechar el hecho de que leer y escribir datos fuera de memoria es más costoso. Esperamos que futuros enfoques exploren y desarrollemos estos métodos de indexación adaptativa para aprovechar la eficacia de la partición en escenarios donde la carga de trabajo no es fácil de predecir y no hay tiempo suficiente para ordenar/particionar completamente todos los datos antes de que llegue la primera consulta.

## 2.5 Directorio de datos

El último problema de diseño de distribución que analizamos está relacionado con el directorio de datos. El sistema debe almacenar y mantener el esquema de base de datos distribuida. Esta información es necesaria durante la optimización de consultas distribuidas, como se explicará más adelante. La información del esquema se almacena en un catálogo/diccionario de datos/directorio (simplemente un directorio). Un directorio es una metabase de datos que almacena información diversa, como definiciones de esquemas y mapeos, estadísticas de uso, información de control de acceso, etc.

En el caso de un SGBD distribuido, la definición del esquema se realiza a nivel global (es decir, el esquema conceptual global [GCS]) y en los sitios locales (es decir, los esquemas conceptuales locales [LCS]). El GCS define la base de datos general, mientras que cada LCS describe los datos en ese sitio específico. Por consiguiente, existen dos tipos de directorios: un directorio/diccionario global (GD/D)<sup>9</sup>, que describe el esquema de la base de datos tal como lo visualizan los usuarios finales, y un directorio/diccionario local (LD/D), que describe las asignaciones locales y el esquema en cada sitio. De este modo, los componentes locales de gestión de bases de datos se integran mediante funciones globales del SGBD.

Como se mencionó anteriormente, el directorio es en sí mismo una base de datos que contiene metadatos sobre los datos almacenados en ella. Por lo tanto, las técnicas que analizamos en este capítulo, en relación con el diseño de bases de datos distribuidas, también se aplican a la gestión de directorios, pero de una forma mucho más sencilla. En resumen, un directorio puede ser global para toda la base de datos o local para cada sitio. En otras palabras, puede haber un único directorio con información sobre todos los datos de la base de datos (el GD/D), o varios directorios, cada uno con la información almacenada en un sitio (el LD/D). En este último caso, podríamos crear jerarquías de directorios para facilitar las búsquedas o implementar una estrategia de búsqueda distribuida que implique una comunicación considerable entre los sitios que albergan los directorios.

Un segundo problema es la replicación. Puede haber una sola copia del directorio o varias copias. Múltiples copias proporcionarían mayor fiabilidad, ya que la probabilidad de acceder a una copia del directorio sería mayor. Además, los retrasos en el acceso al directorio serían menores, debido a la menor contención y la relativa proximidad de las copias. Por otro lado, mantener el directorio actualizado sería considerablemente más difícil, ya que sería necesario actualizar varias copias. Por lo tanto, la elección debe depender del entorno en el que opera el sistema y debe realizarse sopesando factores como los requisitos de tiempo de respuesta, el tamaño del directorio, la capacidad de las máquinas en los sitios, los requisitos de fiabilidad y la volatilidad del directorio (es decir, la cantidad de cambios que experimenta la base de datos, lo que provocaría un cambio en el directorio).

## 2.6 Conclusión

En este capítulo, presentamos las técnicas que se pueden utilizar para el diseño de bases de datos distribuidas, con especial énfasis en los problemas de partición y asignación. Hemos analizado en detalle los algoritmos que se pueden utilizar para fragmentar un esquema relacional de diversas maneras. Estos algoritmos se han desarrollado de forma bastante independiente y no existe una metodología de diseño subyacente que combine las técnicas de partición horizontal y vertical. Si se parte de una relación global, existen algoritmos para descomponerla horizontalmente, así como algoritmos para descomponerla verticalmente en un conjunto de relaciones fragmentadas. Sin embargo, no existen algoritmos que fragmen-

---

En el resto, nos referiremos a esto simplemente como el directorio global.

Una relación global se divide en un conjunto de relaciones fragmentarias, algunas de las cuales se descomponen horizontalmente y otras verticalmente. Se suele señalar que la mayoría de las fragmentaciones reales serían mixtas, es decir, implicarían la partición horizontal y vertical de una relación, pero falta investigación metodológica para lograrlo. Para seguir esta metodología de diseño, se requiere una metodología de diseño de distribución que abarque los algoritmos de fragmentación horizontal y vertical y los utilice como parte de una estrategia más general. Dicha metodología debería considerar una relación global junto con un conjunto de criterios de diseño y generar un conjunto de fragmentos, algunos de los cuales se obtienen mediante fragmentación horizontal y otros mediante fragmentación vertical.

También analizamos técnicas que no separan los pasos de fragmentación y asignación; la forma en que se partitionan los datos determina cómo se asignan, o viceversa. Estas técnicas suelen tener dos características. La primera es que se centran exclusivamente en la partición horizontal. La segunda es que son más precisas y la unidad de asignación es una tupla; los fragmentos en cada sitio surgen como la unión de tuplas de la misma relación asignada a ese sitio.

Finalmente, analizamos las técnicas adaptativas que consideran los cambios en la carga de trabajo. Estas técnicas, de nuevo, suelen implicar particionamiento horizontal, pero monitorizan los cambios en la carga de trabajo (tanto en el conjunto de consultas como en los patrones de acceso) y ajustan el particionamiento de datos en consecuencia. La forma más sencilla de lograr esto es ejecutar el algoritmo de particionamiento por lotes, pero esto obviamente no es deseable. Por lo tanto, los mejores algoritmos de este tipo ajustan la distribución de datos de forma incremental.

## 2.7 Notas bibliográficas

El diseño de bases de datos distribuidas se ha estudiado sistemáticamente desde los inicios de esta tecnología. Un artículo temprano que caracteriza el espacio de diseño es [Levin y Morgan, 1975]. Davenport [1981], Ceri et al. [1983] y Ceri et al. [1987] ofrecen excelentes descripciones generales de la metodología de diseño. Ceri y Pernici [1985] analizan una metodología particular, denominada DATAID-D, similar a la que presentamos en la figura 2.1. Otros intentos de desarrollar una metodología se deben a Fisher et al. [1980], Dawson [1980], Hevner y Schneider [1980] y Mohan [1979].

En este capítulo se han abordado la mayoría de los resultados conocidos sobre fragmentación. El trabajo sobre fragmentación en bases de datos distribuidas se centró inicialmente en la fragmentación horizontal. La discusión sobre este tema se basa principalmente en [Ceri et al. 1982b] y [Ceri et al. 1983]. La partición de datos en SGBD paralelos se trata en [DeWitt y Gray 1992]. El tema de la fragmentación vertical para el diseño de distribuciones se ha abordado en varios artículos (p. ej., Navathe et al. [1984] y Sacca y Wiederhold [1985]). El trabajo original sobre fragmentación vertical se remonta a la tesis doctoral de Hoffer [Hoffer 1975, Hoffer y Severance 1975] y a Niamir [1978] y Hammer y Niamir [1979]. McCormick et al. [1972] presentan el vínculo

Algoritmo de energía que ha sido adoptado para la fragmentación vertical por Hoffer y Severance [1975] y Navathe et al. [1984].

La investigación del problema de asignación de archivos en redes de área extensa se remonta al trabajo de Chu [Chu 1969, 1973]. La mayor parte del trabajo inicial sobre este tema se ha abordado en el excelente estudio de Dowdy y Foster [1982]. Grapa y Belford [1977] y Kollias y Hatzopoulos [1981] presentan algunos resultados teóricos . El trabajo sobre asignación distribuida de datos se remonta a mediados de la década de 1970, con los trabajos de Eswaran [1974] y otros. En su trabajo anterior, Levin y Morgan [1975] se centraron en la asignación de datos, pero posteriormente consideraron conjuntamente la asignación de programas y datos [Morgan y Levin 1977]. El problema de la asignación distribuida de datos también se ha estudiado en muchos entornos especializados. Se ha trabajado para determinar la ubicación de las computadoras y los datos en un diseño de red de área extensa [Gavish y Pirkul 1986]. Las capacidades de canal se han examinado junto con la ubicación de datos [Mahmoud y Riordon, 1976] y la asignación de datos en sistemas de supercomputadoras [Irani y Khabbaz, 1982] , así como en un clúster de procesadores [Sacca y Wiederhold, 1985]. Un trabajo interesante es el de Apers [1981], donde las relaciones se ubican óptimamente en los nodos de una red virtual, y luego se encuentra la mejor coincidencia entre los nodos de la red virtual y la red física. El isomorfismo del problema de asignación de datos con el problema de ubicación de un solo almacén de productos básicos se debe a Ramamoorthy y Wah [1983]. Para otros enfoques de solución, las fuentes son las siguientes: solución del problema de la mochila [Ceri et al. 1982a], técnicas de ramificación y acotación [Fisher y Hochbaum, 1980] y algoritmos de flujo de red [Chang y Liu , 1982].

El enfoque Schism para el particionamiento combinado (Sección 2.3.2) se debe a Curino et al. [2010] y SWORD a Quamar et al. [2013]. Otros trabajos similares son [Zilio 1998], [Rao et al. 2002] y [Agrawal et al. 2004], que se centran principalmente en el particionamiento para sistemas de gestión de bases de datos (SGBD) paralelos.

Wilson y Navathe [1986] analizan una técnica adaptativa temprana . El rediseño limitado, en particular el problema de la materialización, se estudia en [Rivera-Vega et al. 1990, Varadarajan et al. 1989]. Se han estudiado problemas de rediseño completo y materialización en [Karlapalem et al. 1996, Karlapalem y Navathe 1994, Kazerouni y Karlapalem 1997]. Kazerouni y Karlapalem [1997] describen la metodología de rediseño gradual a la que nos referimos en la sección 2.4 . AdaptCache se describe en [Asad y Kemme 2016].

El impacto de los cambios en la carga de trabajo en los sistemas de gestión de bases de datos distribuidos/paralelos y la conveniencia de localizar los datos para cada transacción han sido estudiados por Pavlo et al. [2012] y Lin et al. [2016]. Diversos trabajos abordan el particionamiento adaptativo ante estos cambios. Nuestro análisis se centró en E-Store [Taft et al. 2014] como ejemplo. E-Store implementa los sistemas E-Monitor y E-Planner, respectivamente, para supervisar y detectar cambios en la carga de trabajo, y para detectar los elementos afectados con el fin de crear un plan de migración. Para la migración real, utiliza una versión optimizada de Squall [Elmore et al. 2015]. Existen otros trabajos en la misma línea; por ejemplo, P-Store [Taft et al. 2018] predice las demandas de carga (en lugar de que E-Store reaccione a ellas).

La determinación de los cambios en la carga de trabajo basada en la inspección de registros se debe a Levan-doski y otros [2013].

Holze y Ritter [2008] describen un trabajo centrado en la detección de cambios en la carga de trabajo para la computación autónoma. El sistema Apollo, mencionado en la discusión sobre cómo detectar elementos de datos afectados, y que abstrae las consultas en plantillas de consulta para realizar computación predictiva, se describe en Glasbergen et al. [2018].

El concepto de cracking de bases de datos se ha estudiado en el contexto de almacenes de columnas en memoria principal [Idreos et al. 2007b, Schuhknecht et al. 2013]. Los algoritmos de cracking se han adaptado para funcionar en muchos problemas básicos de la arquitectura de bases de datos, como: actualizaciones para absorber los cambios de datos de forma incremental y adaptativa [Idreos et al. 2007a], consultas multiatributo para reorganizar relaciones completas en lugar de solo columnas [Idreos et al. 2009], para usar también el operador de unión como un disparador para la adaptación [Idreos 2010], control de concurrencia para lidiar con el problema de que el cracking convierte efectivamente las lecturas en escrituras [Graefe et al. 2014, 2012], y lógica similar a la fusión de particiones para proporcionar algoritmos de cracking que puedan equilibrar la convergencia de índices frente a los costes de inicialización [Idreos et al. 2011]. Además, se han desarrollado puntos de referencia personalizados para poner a prueba características críticas como la velocidad con la que se adapta un algoritmo [Graefe et al.

2010]. El descifrado estocástico de bases de datos [Halim et al., 2012] muestra cómo lograr robustez en diversas cargas de trabajo, y Graefe y Kuno [2010b] muestran cómo la indexación adaptativa puede aplicarse a columnas clave. Finalmente, trabajos recientes sobre indexación adaptativa paralela estudian implementaciones eficientes en el uso de la CPU y proponen algoritmos de descifrado para utilizar núcleos múltiples [Pirk et al., 2014, Alvarez et al., 2014] o incluso tiempo de inactividad de la CPU [Petraki et al., 2015].

El concepto de cracking de bases de datos también se ha extendido a decisiones más amplias sobre el diseño del almacenamiento, es decir, la reorganización de los datos base (columnas/filas) según las solicitudes de consulta entrantes [Alagiannis et al., 2014], o incluso sobre qué datos deben cargarse [Idreos et al., 2011, Alagiannis et al., 2012]. El cracking también se ha estudiado en el contexto de Hadoop [Richter et al., 2013] para la indexación local en cada nodo, así como para mejorar la indexación tradicional basada en disco, que fuerza la lectura de datos con la granularidad de las páginas y donde la reescritura de los datos reorganizados debe considerarse una sobrecarga importante [Graefe y Kuno, 2010a].

## Ceremonias

Problema 2.1 (\*) Dada la relación EMP de la Fig. 2.2, sean p1: TÍTULO < "Programador" y p2: TÍTULO > "Programador" dos predicados simples. Suponga que las cadenas de caracteres tienen un orden alfabético.

(a) Realice una fragmentación horizontal de la relación EMP con respecto a {p1, p2}. (b) Explique por qué la fragmentación resultante (EMP1, EMP2) no cumple las reglas de corrección de la fragmentación. (c) Modifique los predicados p1 y p2 para que particionen EMP siguiendo las reglas de corrección de la fragmentación. Para ello, modifique los predicados y componga

todos los predicados minterm y deducir las implicaciones correspondientes, y luego realizar una fragmentación horizontal de EMP basada en estos predicados minterm. Por último, demuestre que el resultado tiene propiedades de completitud, reconstrucción y disyunción.

**Problema 2.2 (\*)** Considere la relación ASG de la Fig. 2.2. Supongamos que hay dos aplicaciones que acceden a ASG. La primera se emite en cinco ubicaciones e intenta determinar la duración de la asignación de los empleados según su número. Supongamos que gerentes, consultores, ingenieros y programadores se encuentran en cuatro ubicaciones diferentes. La segunda aplicación se emite en dos ubicaciones; los empleados con una duración de asignación inferior a 20 meses se gestionan en una, mientras que los de mayor duración se gestionan en otra. Determine la fragmentación horizontal primaria de ASG utilizando la información anterior.

**Problema 2.3.** Considere las relaciones EMP y PAY en la figura 2.2. EMP y PAY están fragmentadas horizontalmente de la siguiente manera:

$$\begin{aligned} \text{EMP1} &= \sigma_{\text{TÍTULO}=\text{"Ing. Elect."}}(\text{EMP}) \\ \text{EMP2} &= \sigma_{\text{TÍTULO}=\text{"Análisis del sistema."}}(\text{EMP}) \\ \text{EMP3} &= \sigma_{\text{TITLE}=\text{"Ing. Mec."}}(\text{EMP}) \\ \text{EMP4} &= \sigma_{\text{TITLE}=\text{"Programador"}}(\text{EMP}) \\ \text{PAGO1} &= \sigma_{\text{SAL} \geq 30000}(\text{PAGO}) \\ \text{PAGO2} &= \sigma_{\text{SAL} < 30000}(\text{PAGO}) \end{aligned}$$

Dibuje el grafo de unión de EMPTITLE PAY. ¿Es el grafo simple o particionado? Si lo es, modifique la fragmentación de EMP o PAY para que el grafo de unión de EMP PAY sea simple.

#### TÍTULO

**Problema 2.4.** Dé un ejemplo de una matriz CA donde el punto de división no sea único y la partición esté en el centro de la matriz. Indique el número de operaciones de desplazamiento necesarias para obtener un único punto de división.

**Problema 2.5 (\*\*)** Dada la relación PAY de la Fig. 2.2, sean  $p1 : \text{SAL} < 30000$  y  $p2 : \text{SAL} \geq 30000$  dos predicados simples. Realice una fragmentación horizontal de PAY con respecto a estos predicados para obtener PAY1 y PAY2. Utilizando la fragmentación de PAY, realice una fragmentación horizontal derivada adicional para EMP.

Mostrar la integridad, reconstrucción y disyunción de la fragmentación del EMP.

**Problema 2.6 (\*\*)** Sea  $Q = \{q1, \dots, q5\}$  un conjunto de consultas,  $A = \{A1, \dots, A5\}$  un conjunto de atributos y  $S = \{S1, S2, S3\}$  un conjunto de sitios. La matriz de la Fig. 2.25a describe los valores de uso de los atributos y la matriz de la Fig. 2.25b proporciona las frecuencias de acceso de la aplicación. Suponga que  $\text{refi}(qk) = 1$  para todos los  $qk$  y  $S_i$ , y que  $A1$  es el atributo clave. Utilice los algoritmos de energía de enlace y partición vertical para obtener una fragmentación vertical del conjunto de atributos en A.

**Problema 2.7 (\*\*)** Escriba un algoritmo para la fragmentación horizontal derivada.

	A1	A2	A3	A4	A5		S1	S2	S3	
q1	0	1	1	0	1		q1	10	20	0
q2	1	1	1	0	1		q2	5	0	10
q3	1	0	0	1	1		q3	0	35	5
q4	0	0	1	0	0		q4	0	10	0
q5	1	1	1	0	0		q5	0	15	0

(a)

(b)

Fig. 2.25 Valores de uso de atributos y frecuencias de acceso a la aplicación en el Ejercicio 3.6

Problema 2.8 (\*\*\*) Suponga la siguiente definición de vista:

```
CREAR VISTA EMPVIEW(ENO, ENAME, PNO, RESP)
COMO      SELECCIONAR EMP.ENO, EMP.ENAME, ASG.PNO, ASG.RESP
DESDE EMP ÚNETE A ASG
DONDE DUR=24
```

Se accede mediante la aplicación q1, ubicada en los sitios 1 y 2, con frecuencias de 10 y 20, respectivamente. Supongamos además que existe otra consulta q2 definida como

```
SELECCIONE ENO, DUR
DE ASG
```

Que se ejecuta en los sitios 2 y 3 con frecuencias de 20 y 10, respectivamente. Con base en la información anterior, construya la matriz  $use(q_i, A_j)$  para los atributos de ambas relaciones EMP y ASG. También construya la matriz de afinidad que contenga todos los atributos de EMP y ASG. Finalmente, transforme la matriz de afinidad para que pueda usarse para dividir la relación en dos fragmentos verticales mediante heurística o BEA.

Problema 2.9 (\*\*\*) Defina formalmente los tres criterios de corrección para la fragmentación horizontal derivada.

Problema 2.10 (\*) Dada una relación R(K, A, B, C) (donde K es la clave) y la siguiente consulta:

```
SELECCIONAR *
DESDE R
DONDE RA=10 Y RB=15
```

(a) ¿Cuál será el resultado de ejecutar PHF en esta consulta? (b) ¿El algoritmo COM\_MIN produce en este caso un conjunto de predicados completo y mínimo?  
Justifique su respuesta.

Problema 2.11 (\*) Demuestre que el algoritmo de energía de enlace genera los mismos resultados utilizando la operación de fila o de columna.

Problema 2.12 (\*\*\*) Modifique el algoritmo SPLIT para permitir la partición de n vías y calcule la complejidad del algoritmo resultante.

Problema 2.13 (\*\*\*) Defina formalmente los tres criterios de corrección para la fragmentación híbrida.

Problema 2.14 Analice cómo el orden en que se aplican los dos esquemas básicos de fragmentación en la fragmentación híbrida afecta la fragmentación final.

Problema 2.15 (\*\*\*) Describa cómo se puede modelar adecuadamente lo siguiente en el problema de asignación de base de datos.

(a) Relaciones entre fragmentos (b)

Procesamiento de consultas

(c) Aplicación de la integridad (d)

Mecanismos de control de concurrencia

Problema 2.16 (\*\*\*) Considere los diversos algoritmos heurísticos para el problema de asignación de base de datos.

(a) ¿Cuáles son algunos de los criterios razonables para comparar estas heurísticas?  
Conversar.

(b) Compare los algoritmos heurísticos con respecto a estos criterios.

Problema 2.17 (\*) Elija uno de los algoritmos heurísticos utilizados para resolver el DAP y escriba un programa para él.

Problema 2.18 (\*\*\*) Suponga el entorno del Ejercicio 3.8. Suponga también que el 60% de los accesos a la consulta q1 son actualizaciones de PNO y RESP de la vista EMPVIEW, y que ASG.DUR no se actualiza a través de EMPVIEW. Además, suponga que la velocidad de transferencia de datos entre el sitio 1 y el sitio 2 es la mitad de la que existe entre el sitio 2 y el sitio 3.

Con base en la información anterior, encuentre una fragmentación razonable de ASG y EMP y una replicación y ubicación óptimas para los fragmentos, suponiendo que los costos de almacenamiento no importan aquí, pero las copias se mantienen consistentes.

Sugerencia: Considere la fragmentación horizontal para ASG basada en el predicado DUR = 24 y la fragmentación horizontal derivada correspondiente para EMP. Observe también la matriz de afinidad obtenida en el Ejemplo 2.7 para EMP y ASG conjuntamente, y considere si sería conveniente realizar una fragmentación vertical para ASG.

## Capítulo 3

### Control de datos distribuidos



Un requisito importante de un DBMS es la capacidad de soportar el control de datos, es decir, controlar cómo se accede a los datos mediante un lenguaje de alto nivel. El control de datos generalmente incluye la gestión de vistas, el control de acceso y el control de integridad semántica. Informalmente, estas funciones deben garantizar que los usuarios autorizados realicen operaciones correctas en la base de datos, contribuyendo así al mantenimiento de la integridad de la base de datos. Las funciones necesarias para mantener la integridad física de la base de datos en presencia de accesos concurrentes y fallos se estudian por separado en el capítulo 5 en el contexto de la gestión de transacciones. En los DBMS relacionales, el control de datos se puede lograr de manera uniforme. Las vistas, las autorizaciones y las restricciones de integridad semántica se pueden definir como reglas que el sistema aplica automáticamente. La violación de algunas reglas por parte de las operaciones de la base de datos generalmente implica el rechazo de los efectos de algunas operaciones (p. ej., deshacer algunas actualizaciones) o la propagación de algunos efectos (p. ej., actualizar datos relacionados) para preservar la integridad de la base de datos.

La definición de estas reglas forma parte de la administración de la base de datos, función generalmente desempeñada por un administrador de base de datos (DBA). Esta persona también se encarga de aplicar las políticas de la organización. Se han propuesto soluciones reconocidas para el control de datos en sistemas de gestión de bases de datos (SGBD) centralizados. En este capítulo, analizamos cómo estas soluciones pueden extenderse a los SGBD distribuidos. El coste de implementar el control de datos, que es elevado en términos de utilización de recursos en un SGBD centralizado, puede resultar prohibitivo en un entorno distribuido.

Dado que las reglas de control de datos deben almacenarse, la gestión de un directorio distribuido también es relevante en este capítulo. El directorio de un SGBD distribuido puede considerarse una base de datos distribuida. Existen varias maneras de almacenar las definiciones de control de datos, según la forma en que se gestione el directorio. La información del directorio puede almacenarse de forma diferente según su tipo; es decir, parte de la información puede replicarse completamente, mientras que otra puede distribuirse. Por ejemplo, la información útil en tiempo de compilación, como la información de control de acceso, podría replicarse. En este capítulo, se destaca el impacto de la gestión de directorios en el rendimiento de los mecanismos de control de datos.

Este capítulo está organizado de la siguiente manera: La gestión de vistas es el tema de la Sección 3.1. El control de acceso se presenta en la Sección 3.2. Finalmente, el control de integridad semántica se aborda en la Sección 3.3. Para cada sección, primero se describe la solución en un SGBD centralizado y luego se presenta la solución distribuida, que suele ser una extensión de la centralizada, aunque más compleja.

### 3.1 Gestión de vistas

Una de las principales ventajas del modelo relacional es que proporciona independencia lógica total de los datos. Como se introdujo en el capítulo 1, los esquemas externos permiten a los grupos de usuarios tener su propia vista de la base de datos. En un sistema relacional, una vista es una relación virtual, definida como el resultado de una consulta sobre relaciones base (o relaciones reales), pero no materializada como una relación base en la base de datos. Una vista es una ventana dinámica, en el sentido de que refleja todas las actualizaciones de la base de datos. Un esquema externo puede definirse como un conjunto de vistas y/o relaciones base. Además de su uso en esquemas externos, las vistas son útiles para garantizar la seguridad de los datos de forma sencilla. Al seleccionar un subconjunto de la base de datos, las vistas ocultan algunos datos. Si los usuarios solo pueden acceder a la base de datos a través de las vistas, no pueden ver ni manipular los datos ocultos, que, por lo tanto, son seguros.

En el resto de esta sección, analizaremos la gestión de vistas en sistemas centralizados y distribuidos, así como los problemas de actualización de vistas. Cabe destacar que, en un SGBD distribuido, una vista puede derivarse de relaciones distribuidas, y el acceso a una vista requiere la ejecución de la consulta distribuida correspondiente a su definición. Un aspecto importante en un SGBD distribuido es la eficiencia de la materialización de vistas. Veremos cómo el concepto de vistas materializadas ayuda a resolver este problema, entre otros, pero requiere técnicas eficientes para su mantenimiento.

#### 3.1.1 Vistas en sistemas DBMS centralizados

La mayoría de los SGBD relacionales utilizan un mecanismo de vista, donde una vista es una relación derivada de las relaciones base como resultado de una consulta relacional (esto se propuso inicialmente en los proyectos INGRES y System R). Se define asociando el nombre de la vista con la consulta de recuperación que la especifica.

Ejemplo 3.1 La vista de los analistas del sistema (SYSAN) derivada de la relación EMP se puede definir mediante la siguiente consulta SQL:

```
CREAR VISTA SYSAN(ENO, ENAME) COMO
SELECCIONAR ENO, ENAME
DESDE EMP
DONDE TÍTULO = "Sist. Anal."
```

## 3.1 Gestión de vistas

93

El único efecto de esta declaración es el almacenamiento de la definición de la vista en el catálogo. No es necesario registrar ninguna otra información. Por lo tanto, el resultado de la consulta que define la vista (es decir, una relación que tiene los atributos ENO y ENAME para la analistas de sistemas (como se muestra en la Fig. 3.1) no se produce. Sin embargo, la vista SYSAN puede manipularse como una relación base.

## Ejemplo 3.2 La consulta

"Encuentre los nombres de todos los analistas de sistemas con su número de proyecto y responsabilidad(es)"

La participación de la vista SYSAN y la relación ASG se puede expresar como

```
SELECCIONAR ENAME, PNO, RESP
DESDE SYSAN NATURAL ÚNETE A ASG
```

Mapeo de una consulta expresada en vistas a una consulta expresada en relaciones base

Se puede realizar mediante la modificación de la consulta. Con esta técnica, se cambian las variables. para abarcar las relaciones base y la calificación de la consulta se fusiona (AND) con la Ver calificación.

## Ejemplo 3.3 La consulta anterior se puede modificar para

```
SELECCIONAR ENAME, PNO, RESP
DESDE EMP NATURAL ÚNETE A ASG
DONDE TÍTULO = "Sist. Anal."
```

El resultado de esta consulta se ilustra en la figura 3.2.

La consulta modificada se expresa en relaciones base y, por lo tanto, se puede procesar. por el procesador de consultas. Es importante tener en cuenta que el procesamiento de la vista se puede realizar en Tiempo de compilación. El mecanismo de vista también se puede utilizar para refinar los controles de acceso.

SYSAN	
ENO	ENAME
E2	M. Smith
E5	B. Casey
E8	J. Jones

Fig. 3.1 Relación correspondiente a la vista SYSAN

ENAME	PNO	RESP
M. Smith	Analista	P1
M. Smith	Analista de	P2
B. Casey	Gerente de	P3
J. Jones	Gerente	P4

Fig. 3.2 Resultado de la consulta que involucra la vista SYSAN

Para incluir subconjuntos de objetos. Para especificar cualquier usuario al que se desee ocultar información, la palabra clave USER generalmente se refiere al identificador del usuario conectado.

Ejemplo 3.4 La vista ESAME restringe el acceso de cualquier usuario a aquellos empleados que tengan el mismo título:

```
CREAR VISTA IGUAL QUE
SELECCIONAR *
DE EMP E1, EMP E2 DONDE E1.ENO =
E2.ENO
Y           E1.ENO = USUARIO
```

En la definición de vista anterior, \* representa "todos los atributos" y las dos variables de tupla (E1 y E2) que abarcan la relación EMP son necesarias para expresar la unión de una tupla de EMP (la correspondiente al usuario conectado) con todas las tuplas de EMP basadas en el mismo título. Por ejemplo, la siguiente consulta emitida por el usuario J.

Gama

```
SELECCIONAR *
DE ESAME
```

Devuelve la relación de la Fig. 3.3. Nótese que el usuario J. Doe también aparece en el resultado.

Si el usuario que crea ESAME es un ingeniero eléctrico, como en este caso, la vista representa el conjunto de todos los ingenieros eléctricos.

Las vistas se pueden definir mediante consultas relacionales de complejidad arbitraria que incluyen funciones de selección, proyección, unión, agregación, etc. Todas las vistas pueden interrogarse como relaciones base, pero no todas pueden manipularse como tales. Las actualizaciones a través de las vistas solo se pueden gestionar automáticamente si se propagan correctamente a las relaciones base. Podemos clasificar las vistas como actualizables y no actualizables.

Una vista solo es actualizable si sus actualizaciones se pueden propagar a las relaciones base sin ambigüedad. La vista SYSAN mencionada anteriormente es actualizable; por ejemplo, la inserción de un nuevo analista de sistemas 201, Smith, se asignará a la inserción de un nuevo empleado 201, Smith, Syst. Anal.. Si la vista ocultara atributos distintos de TITLE, se les asignarían valores nulos.

Ejemplo 3.5 Sin embargo, la siguiente vista, que utiliza una unión natural (es decir, la unión equitativa de dos relaciones en un atributo común), no es actualizable:

```
CREAR VISTA EG(ENAME, RESP) COMO
SELECCIONAR NOMBRE DISTINTO, RESP
DESDE EMP NATURAL ÚNETE A ASG
```

ENO ENO	NOMBRE TÍTULO E1	J. Doe Electa.
Ing. E2 L. Chu	Ing. Elect.	

Fig. 3.3 Resultado de la consulta en la vista ESAME

Por ejemplo, la eliminación de la tupla Smith, Analista no se puede propagar, ya que es ambigua. Las eliminaciones de Smith en relación con EMP o de analista en relación con ASG son significativas, pero el sistema desconoce cuál es la correcta.

Los sistemas actuales son muy restrictivos en cuanto al soporte de actualizaciones a través de vistas. Las vistas solo se pueden actualizar si se derivan de una única relación mediante selección y proyección. Esto excluye las vistas definidas por uniones, agregados, etc. Sin embargo, en teoría es posible admitir automáticamente actualizaciones de una clase más amplia de vistas. Es interesante notar que las vistas derivadas por unión son actualizables si incluyen las claves de las relaciones base.

### 3.1.2 Vistas en DBMS distribuidos

La definición de una vista es similar en un DBMS distribuido y en sistemas centralizados. Sin embargo, una vista en un sistema distribuido puede derivarse de relaciones fragmentadas almacenadas en diferentes sitios. Cuando se define una vista, su nombre y su consulta de recuperación se almacenan en el catálogo.

Dado que las vistas pueden ser utilizadas como relaciones base por los programas de aplicación, su definición debe almacenarse en el directorio de la misma manera que las descripciones de las relaciones base. Según el grado de autonomía del sitio que ofrece el sistema, las definiciones de vista pueden centralizarse en un sitio, duplicarse parcial o totalmente. En cualquier caso, la información que asocia el nombre de una vista a su sitio de definición debe duplicarse. Si la definición de vista no está presente en el sitio donde se realiza la consulta, se requiere acceso remoto a dicho sitio.

La asignación de una consulta expresada en vistas a una consulta expresada en relaciones base (que potencialmente pueden fragmentarse) también puede realizarse mediante la modificación de consultas, de la misma manera que en los SGBD centralizados. Con esta técnica, la cualificación que define la vista se encuentra en el catálogo de la base de datos distribuida y se fusiona con la consulta para generar una consulta sobre relaciones base. Esta consulta modificada es una consulta distribuida, que puede ser procesada por el procesador de consultas distribuidas (véase el capítulo 4). El procesador de consultas asigna la consulta distribuida a una consulta sobre fragmentos físicos.

En el capítulo 2 presentamos métodos alternativos para fragmentar las relaciones de base. La definición de fragmentación es, de hecho, muy similar a la de las vistas particulares. Por lo tanto, es posible gestionar vistas y fragmentos mediante un mecanismo unificado. Además, los datos replicados pueden gestionarse de la misma manera. La utilidad de este mecanismo unificado reside en facilitar la administración distribuida de bases de datos.

Los objetos manipulados por el administrador de la base de datos pueden verse como una jerarquía donde las hojas son los fragmentos de los cuales se pueden derivar relaciones y vistas.

Por lo tanto, el DBA puede aumentar la localidad de referencia creando vistas en correspondencia biunívoca con los fragmentos. Por ejemplo, es posible implementar la vista SYSAN ilustrada en el Ejemplo 3.1 mediante un fragmento en un sitio determinado, siempre que la mayoría de los usuarios que acceden a la vista SYSAN se encuentren en el mismo sitio.

Evaluar vistas derivadas de relaciones distribuidas puede ser costoso. En una organización dada, es probable que muchos usuarios accedan a la misma vista, la cual debe recalcularse para cada usuario. Vimos en la Sección 3.1.1 que la derivación de vistas se realiza fusionando la calificación de la vista con la calificación de la consulta. Una solución alternativa es evitar la derivación de vistas manteniendo versiones reales de las vistas, llamadas vistas materializadas. Una vista materializada almacena las tuplas de una vista en una relación de base de datos, al igual que las demás tuplas de la base de datos, posiblemente con índices. Por lo tanto, el acceso a una vista materializada es mucho más rápido que derivarla, en particular, en un SGBD distribuido donde las relaciones base pueden ser remotas. Introducidas a principios de la década de 1980, las vistas materializadas han ganado mucho interés en el contexto del almacenamiento de datos para acelerar las aplicaciones de procesamiento analítico en línea (OLAP).

Las vistas materializadas en los almacenes de datos suelen incluir operadores de agregación (como SUM y COUNT) y de agrupación (GROUP BY), ya que proporcionan resúmenes compactos de la base de datos. Actualmente, todos los principales productos de bases de datos admiten vistas materializadas.

Ejemplo 3.6 La siguiente vista sobre la relación PROJ(PNO,PNAME,BUDGET,LOC) da, para cada ubicación, el número de proyectos y el presupuesto total.

```
CREAR VISTA PL(LOC, NBPROJ, TBUDGET) COMO
SELECCIONAR LOC, CONTAR(*),SUMA(PRESUPUESTO)
DEL PROYECTO
AGRUPAR POR LOC
```

### 3.1.3 Mantenimiento de vistas materializadas

Una vista materializada es una copia de datos base y, por lo tanto, debe mantenerse consistente con dichos datos, que pueden actualizarse. El mantenimiento de la vista es el proceso de actualizar (o refrescar) una vista materializada para reflejar los cambios realizados en los datos base. Los problemas relacionados con la materialización de vistas son similares a los de la replicación de bases de datos, que abordaremos en el capítulo 6. Sin embargo, una diferencia importante radica en que las expresiones de vista materializada, en particular para el almacenamiento de datos, suelen ser más complejas que las definiciones de réplica y pueden incluir operadores de unión, agrupación y agregación. Otra diferencia importante es que la replicación de bases de datos se centra en configuraciones de replicación más generales, por ejemplo, con múltiples copias de los mismos datos base en diferentes ubicaciones.

Una política de mantenimiento de vistas permite a un administrador de bases de datos (DBA) especificar cuándo y cómo debe actualizarse una vista. La primera pregunta (cuándo actualizar) está relacionada con la consistencia (entre la vista y los datos base) y la eficiencia. Una vista se puede actualizar de dos modos: inmediato o diferido. Con el modo inmediato, la vista se actualiza inmediatamente como parte de la transacción que actualiza los datos base utilizados por la vista. Si la vista y los datos base son administrados por diferentes SGBD, posiblemente en diferentes sitios, esto requiere el uso de una transacción distribuida, por ejemplo, mediante el protocolo de confirmación en dos fases (2PC) (véase el capítulo 5). Las principales ventajas de la actualización inmediata son que

La vista siempre es coherente con los datos base y las consultas de solo lectura pueden ser rápidas. Sin embargo, esto implica un mayor tiempo de transacción para actualizar tanto los datos base como las vistas dentro de las mismas transacciones. Además, el uso de transacciones distribuidas puede resultar difícil.

En la práctica, se prefiere el modo diferido porque la vista se actualiza en transacciones separadas (de actualización), sin penalización de rendimiento en las transacciones que actualizan los datos base. Las transacciones de actualización se pueden activar en diferentes momentos: de forma diferida, justo antes de que se evalúe una consulta en la vista; periódicamente, en momentos predefinidos, p. ej., todos los días; o de forma forzada, después de un número predefinido de actualizaciones a los datos base. La actualización diferida permite que las consultas vean el último estado consistente de los datos base, pero a expensas de un mayor tiempo de consulta para incluir la actualización de la vista. La actualización periódica y forzada permite que las consultas vean vistas cuyo estado no es consistente con el último estado de los datos base. Las vistas administradas con estas estrategias también se denominan instantáneas.

La segunda pregunta (cómo actualizar una vista) es un problema de eficiencia importante. La forma más sencilla de actualizar una vista es volver a calcularla desde cero utilizando los datos base. En algunos casos, esta puede ser la estrategia más eficiente, p. ej., si se ha modificado un subconjunto grande de los datos base. Sin embargo, hay muchos casos en los que solo es necesario modificar un subconjunto pequeño de la vista. En estos casos, una mejor estrategia es calcular la vista de forma incremental, calculando solo los cambios en ella. El mantenimiento incremental de la vista se basa en el concepto de relación diferencial. Sea  $u$  una actualización de la relación  $R$ .  $R+$  y  $R-$  son relaciones diferenciales de  $R$  por  $u$ , donde  $R+$  contiene las tuplas insertadas por  $u$  en  $R$ , y  $R-$  contiene las tuplas de  $R$  eliminadas por  $u$ . Si  $u$  es una inserción,  $R-$  está vacío. Si  $u$  es una eliminación,  $R+$  está vacío. Finalmente, si  $u$  es una modificación, la relación  $R$  se puede obtener calculando  $(V - V-) \cup V+$ . Para calcular los cambios en la vista, es decir,  $V+$  y  $V-$ , es posible que sea necesario utilizar las relaciones base además de las relaciones diferenciales.

Ejemplo 3.7. Considere la vista EG del Ejemplo 3.5 , que utiliza las relaciones EMP y ASG como datos base, y suponga que su estado se deriva del del Ejemplo 3.1, de modo que EG tiene 9 tuplas (véase la Fig. 3.4). Sea  $EMP+$  una tupla E9, B. Martin, Programador, que se insertará en  $EMP$ , y  $ASG+$  dos tuplas E4, P3, Programador, 12 y E9, P3, Programador, 12, que se insertarán en  $ASG$ . Los cambios en la vista EG se pueden calcular como:

```

 $EG+ = (\text{SELECCIONAR NOMBRE, RESP}$ 
       $\text{DESDE EMP NATURAL ÚNETE A ASG+})$ 
       $\text{UNIÓN}$ 
       $(\text{SELECCIONE NOMBRE, RESP}$ 
       $\text{DESDE EMP+ UNIÓN NATURAL ASG})$ 
       $\text{UNIÓN}$ 
       $(\text{SELECCIONE NOMBRE, RESP}$ 
       $\text{DESDE EMP+ UNIÓN NATURAL ASG+})$ 
```

Lo que produce las tuplas B. Martin, Programador, y J. Miller, Programador. Cabe destacar que las restricciones de integridad serían útiles en este caso para evitar trabajo innecesario (véase la sección 3.3.2). Suponiendo que las relaciones EMP y ASG están relacionadas por una restricción referencial que

dice que ENO en ASG debe existir en EMP, la segunda declaración SELECT es inútil ya que produce una relación vacía.

Se han ideado técnicas eficientes para realizar el mantenimiento incremental de vistas utilizando tanto las vistas materializadas como las relaciones base. Estas técnicas difieren esencialmente en la expresividad de sus vistas, el uso de restricciones de integridad y la forma en que gestionan la inserción y la eliminación. Se pueden clasificar, según la dimensión de expresividad de la vista, en vistas no recursivas, vistas con uniones externas y vistas recursivas. Para las vistas no recursivas, es decir, las vistas de selección-proyecto-unión (SPJ) que pueden tener eliminación, unión y agregación de duplicados, una solución elegante es el algoritmo de conteo. Un problema radica en que las tuplas individuales de la vista pueden derivarse de varias tuplas de las relaciones base, lo que dificulta la eliminación en la vista. La idea básica del algoritmo de conteo es mantener un recuento del número de derivaciones para cada tupla en la vista e incrementar (o decrementar) el recuento de tuplas en función de las inserciones (o deleciones); una tupla en cuya vista el recuento sea cero puede entonces eliminarse.

Ejemplo 3.8. Considere la vista EG de la Fig. 3.4. Cada tupla de EG tiene una derivación (es decir, un recuento de 1), excepto la tupla M. Smith, Analista, que tiene dos (es decir, un recuento de 2). Supongamos ahora que las tuplas E2, P1, Analista, 24 y E3, P3, Consultor, 10 se eliminan de ASG. En ese caso, solo la tupla A. Lee, Consultor debe eliminarse de EG.

A continuación, presentamos el algoritmo básico de conteo para actualizar una vista V, definida sobre dos relaciones R y S, como una consulta  $q(R, S)$ . Suponiendo que cada tupla en V tiene un conteo de derivación asociado, el algoritmo consta de tres pasos principales (véase el Algoritmo 3.1). En primer lugar, aplica la técnica de diferenciación de vistas para formular las vistas diferenciales  $V^+$  y  $V^-$  como consultas sobre la vista, las relaciones base y las relaciones diferenciales. En segundo lugar, calcula  $V^+$  y  $V^-$  y sus recuentos de tuplas. En tercer lugar, aplica los cambios de  $V^+$  y  $V^-$  en V sumando los recuentos positivos y restando los negativos, y eliminando las tuplas con un recuento de cero.

El algoritmo de conteo es óptimo, ya que calcula exactamente las tuplas de vista que se insertan o eliminan. Sin embargo, requiere acceso a las relaciones base. Esto implica que estas se mantengan (posiblemente como réplicas) en los sitios de la

ENAME RESP	J. Doe
Gerente M. Smith	Analista
A. Lee	Ingéniero Consultor
Programador	B. Casey Gerente
A. Lee	L. Chu
J. Miller	Gerente
Ingeniero	Gerente
R. Davis	
J. Jones	

Fig. 3.4 Estado de vista EG

---

Algoritmo 3.1: CONTEO

---

Entrada: V: vista definida como  $q(R, S)$ ; R, S: relaciones; R+, R-: comienzan los cambios en R

```

V+ = q+(V, R+, R, S)
V- = q-(V, R-, R, S)
calcula V+ con conteos positivos para tuplas
insertadas calcula V- con conteos negativos para
tuplas eliminadas calcula (V - V-)   V+ sumando conteos positivos y restando
conteos negativos eliminando cada tupla en V con conteo = 0;
fin

```

---

Vista materializada. Para evitar acceder a las relaciones base y almacenar la vista en un sitio diferente, esta debe ser mantenible utilizando únicamente la vista y las relaciones diferenciales. Estas vistas se denominan automantendibles.

Ejemplo 3.9 Considere la vista SYSAN del Ejemplo 3.1. Escribamos la definición de la vista como  $SYSAN = q(EMP)$ , lo que significa que la vista se define mediante una consulta  $q$  sobre  $EMP$ . Podemos calcular las vistas diferenciales utilizando únicamente las relaciones diferenciales, es decir,  $SYSAN+ = q(EMP+)$  y  $SYSAN- = q(EMP-)$ . Por lo tanto, la vista  $SYSAN$  es automantendible.

La automantendibilidad depende de la expresividad de las vistas y puede definirse en función del tipo de actualización (inserción, eliminación o modificación). La mayoría de las vistas SPJ no son automantendibles en cuanto a la inserción, pero sí suelen serlo en cuanto a la eliminación y modificación. Por ejemplo, una vista SPJ es automantendible en cuanto a la eliminación de la relación  $R$  si los atributos clave de  $R$  están incluidos en la vista.

Ejemplo 3.10 Considere la vista EG del Ejemplo 3.5. Añadamos el atributo  $ENO$  (que es la clave de  $EMP$ ) a la definición de la vista. Esta vista no es automantendible en cuanto a la inserción. Por ejemplo, tras insertar una tupla  $ASG$ , necesitamos realizar la unión con  $EMP$  para obtener el  $ENAME$  correspondiente que se insertará en la vista. Sin embargo, esta vista es automantendible en cuanto a la eliminación en  $EMP$ .

Por ejemplo, si se elimina una tupla  $EMP$ , se pueden eliminar las tuplas de vista que tengan el mismo  $ENO$ .

Analizamos dos optimizaciones que pueden reducir significativamente el tiempo de mantenimiento del algoritmo COUNTING. La primera optimización consiste en materializar las vistas que representan subconsultas de la consulta de entrada. Una vista se construye eliminando un subconjunto de relaciones de la consulta. Estas vistas son cada vez más pequeñas y construyen una jerarquía. El método F-IVM construye dicha jerarquía, denominada trie de vista, con la consulta de entrada en la parte superior, las relaciones en las hojas y las vistas internas definidas por consultas de proyecto, unión y agregación sobre sus hijos. Las actualizaciones de una relación se propagan de abajo a arriba en este trie de vista. Las vistas que se encuentran en la ruta desde la relación actualizada hasta la raíz se mantienen mediante el procesamiento delta del algoritmo COUNTING. Todas las demás vistas permanecen sin cambios; si se materializan, pueden acelerar este procesamiento delta. Para una clase restringida de consultas acíclicas, denominadas

q-jerárquicos, estos árboles de vista permiten actualizaciones en tiempo constante de cualquiera de las relaciones de entrada.

La segunda optimización aprovecha la asimetría de los datos. Los valores que aparecen con mucha frecuencia en la base de datos se consideran pesados, mientras que los demás se consideran ligeros. IVM utiliza estrategias de evaluación sensibles a la asimetría de los datos, tanto pesada como ligera, que utilizan vistas materializadas y cálculo delta, como todos los algoritmos de mantenimiento mencionados.

Ejemplificamos estas dos optimizaciones para una consulta que cuenta el número de triángulos en un grafo. Queremos actualizar este recuento de triángulos de forma inmediata e incremental con una actualización del grafo de datos, que puede ser la inserción o eliminación de una arista.

Consideremos tres copias R, S y T de la relación binaria de aristas de un grafo con N aristas.

Registraremos las multiplicidades de tuplas en las relaciones y vistas de entrada, es decir, el número de sus derivaciones, en una columna P aparte. Suponiendo que los esquemas de las relaciones son (A, B, P\_R), (B, C, P\_S) y (C, A, P\_T), la consulta de recuento de triángulos es

**CREAR VISTA Q(CNT) COMO**

```
SELECCIONAR SUMA(P_R * P_S * P_T) como CNT
DE R UNIÓN NATURAL S UNIÓN NATURAL T
```

La inserción o eliminación de una arista activa actualizaciones en cada una de las tres copias de la relación. Analizamos el caso de la actualización de R; los otros dos casos se tratan de forma similar. Modelamos esta actualización como una relación deltaR compuesta por una única tupla (a, b, p), donde (a, b) define la arista actualizada y p es la multiplicidad. Siguiendo el formalismo de las relaciones multiconjunto generalizadas, modelamos tanto las inserciones como las eliminaciones de forma uniforme, permitiendo que las multiplicidades sean números enteros, es decir, negativos y positivos. Luego, para insertar o eliminar la arista tres veces, fijamos la multiplicidad p en +3 o -3, respectivamente.

El algoritmo COUNTING calcula sobre la marcha una consulta delta, deltaQ, que representa el cambio en el resultado de la consulta. Esta consulta es igual a Q, donde se reemplaza R por deltaR. Este cálculo delta toma O(N) tiempo, ya que necesita interseccar dos listas de posiblemente O(N) valores C emparejados con b en S y con a en T (es decir, la multiplicidad de dichos pares en S y T es distinta de cero).

El enfoque DBToaster acelera el cálculo delta al precalcular tres vistas auxiliares que representan las partes independientes de la actualización de las consultas delta para las actualizaciones de las tres relaciones:

**CREAR VISTA V\_ST(B, A, CNT) COMO**

```
SELECCIONAR B, A, SUMA(P_S * P_T) como CNT
DE           S ARTICULACIÓN NATURAL T
GRUPO POR B, A
```

**CREAR VISTA V\_TR(C, B, CNT) COMO**

```
SELECCIONAR C, B, SUMA(P_T * P_R) como CNT
DE           T UNIÓN NATURAL R
GRUPO POR C, B
```

```

CREAR VISTA V_RS(A, C, CNT) COMO
SELECCIONE A, C, SUMA(P_R * P_S) como CNT
DE          R UNIÓN NATURAL S
AGRUPAR POR A, C

```

La vista V\_ST permite calcular la consulta deltaQ en tiempo O(1), ya que la unión de deltaR y V\_ST requiere una búsqueda en tiempo constante para a, b en V\_ST.

Sin embargo, mantener las vistas V\_RS y V\_TR, que se definen utilizando R, todavía requiere tiempo O(N).

El método F-IVM materializa solo una de las tres vistas, por ejemplo, V\_ST. En este caso, el mantenimiento de las actualizaciones de R toma O(1) tiempo, pero el mantenimiento de S y T también toma O(N) tiempo.

El algoritmo IVM partitiona los nodos del grafo en función de su grado, es decir, del número de nodos conectados directamente: los nodos pesados tienen grado mayor o igual a  $N^{1/2}$ , mientras que los nodos ligeros tienen grado menor que  $N^{1/2}$ . Esto lleva a una partición de cada una de las tres copias R, S y T de la relación de arista en una parte pesada R\_h (S\_h, T\_h) y una parte ligera R\_l (S\_l, T\_l): una tupla a, b, p está en R\_h si a es pesada y en R\_l en caso contrario; de forma similar, una tupla b, c, p está en S\_h si b es pesada y en S\_l en caso contrario; finalmente, c, a, p está en T\_h si c es pesada y en T\_l en caso contrario. Podemos reescribir Q reemplazando cada una de las tres relaciones con la unión de sus dos partes. La consulta Q es entonces equivalente a la unión de ocho vistas que tienen en cuenta la asimetría Q\_r,s,t, donde r, s, t ∈ {h, l}:

```

CREAR VISTA Q_r,s,t(CNT) COMO
SELECCIONAR SUMA(P_R * P_S * P_T) como CNT
DESDE R_r UNIÓN NATURAL S_s UNIÓN NATURAL T_t

```

Consideremos una actualización de una sola tupla  $\delta_{R_r} = \{(a, b, p)\}$  a la parte R\_r de la relación R para  $r ∈ \{h, l\}$ . El cálculo del delta para una vista Q\_r,s,t se obtiene mediante la siguiente consulta más simple:

```

CREAR VISTA deltaQ_r,s,t(CNT) COMO
SELECCIONAR SUMA(P_R * P_S * P_T) como CNT DE
deltaR_r UNIÓN NATURAL S_s UNIÓN NATURAL T_t DONDE S_s.A = a Y T_t.B = b

```

IVM adapta su estrategia de mantenimiento a cada vista con sesgo para lograr el tiempo de actualización sublineal. Si bien la mayoría de estas vistas alcanzan fácilmente el límite superior de  $O(N^{1/2})$ , existe una excepción. A continuación, explicamos cómo lograr este límite para el mantenimiento de cada una de estas vistas.

El cálculo delta para las cuatro vistas Q\_r,l,t (para  $r, t ∈ \{h, l\}$ ) es expresado de la siguiente manera:

```

CREAR VISTA deltaQ_r,l,t(CNT) COMO
SELECCIONAR SUMA(P_R * P_S * P_T) como CNT DE
deltaR_r UNIÓN NATURAL S_l UNIÓN NATURAL T_t DONDE S_l.A = a Y T_t.B = b

```

Une las partes  $S_I$  con  $T_t$  en  $C$ . Dado que la actualización  $\delta_{r,r}$  establece  $B$  en  $b$  en  $S_I$  y  $b$  solo puede ser un valor leve en  $S_I$ , existen como máximo  $N^{1/2}$  valores de  $C$  emparejados con  $b$  en  $S_I$ . La intersección del conjunto de valores de  $C$  en  $S_I$  y  $T_t$  puede entonces tardar como máximo  $O(N^{1/2})$  tiempo.

El cálculo delta para las vistas  $Q_r,h,h$  se expresa de forma similar. Dado que todos los valores  $C$  en  $T_h$  son pesados, cada uno de ellos tiene al menos  $N^{1/2}$  valores  $A$ . Esto también significa que hay como máximo  $N^{1/2}$  valores  $C$  pesados. La intersección del conjunto de los valores  $C$  pesados en  $T_h$  con los valores  $C$  en  $S_h$  puede tardar, como máximo,  $O(N^{1/2})$  tiempo.

Sin embargo, el cálculo delta para las vistas  $Q_r,h,I$  para  $r \in \{h, I\}$  requiere un tiempo lineal, ya que requiere iterar sobre todos los valores  $C$  emparejados con  $b$  en  $S_h$  y con  $a$  en  $T_I$ ; el número de estos valores  $C$  puede ser lineal en función del tamaño de la base de datos. En este caso, IVM precalcula las partes independientes de la actualización de las consultas delta como vistas materializadas auxiliares y luego las aprovecha para acelerar la evaluación delta:

```
CREAR VISTA V_ST(B, A, CNT) COMO
SELECCIONAR B, A, SUMA(P_S * P_T) como CNT
DESDE S_h UNIR NATURAL T_I
GRUPO POR B, A
```

Materializamos vistas similares  $V_RS$  y  $V_TR$  en caso de actualizaciones de  $T$  y, respectivamente, de  $S$ . Cada una de estas vistas requiere un espacio  $O(N^{3/2})$ . Ahora podemos calcular  $\delta_{r,h,I}$  usando  $V_ST$  como

```
CREAR VISTA deltaQ_r,h,I(CNT) COMO SELECCIONAR
SUMA(P_R * CNT) como CNT DE deltaR_r
UNIÓN NATURAL V_ST DONDE V_ST.B = b Y V_ST.A
= a
```

Esto toma  $O(1)$  tiempo ya que solo necesitamos una búsqueda en  $V_ST$  para obtener la multiplicidad del borde ( $a, b$ ) seguida de la multiplicación con  $p$  de  $\delta_{r,r}$ .

### 3.2 Control de acceso

El control de acceso es un aspecto importante de la seguridad de los datos, la función de un sistema de base de datos que protege los datos contra el acceso no autorizado. Otro aspecto importante es la protección de datos, que impide que usuarios no autorizados accedan al contenido físico de los datos. Esta función suele ser proporcionada por los sistemas de archivos en el contexto de sistemas operativos centralizados y distribuidos. El principal enfoque de protección de datos es el cifrado de datos.

El control de acceso debe garantizar que solo los usuarios autorizados realicen las operaciones permitidas en la base de datos. Muchos usuarios diferentes pueden tener acceso a una gran colección de datos bajo el control de un único sistema centralizado o distribuido. Por lo tanto, el SGBD centralizado o distribuido debe poder restringir el acceso.

De un subconjunto de la base de datos a un subconjunto de usuarios. El control de acceso se ha proporcionado desde hace tiempo por los sistemas operativos como servicios del sistema de archivos. En este contexto, se ofrece un control centralizado. De hecho, el controlador central crea objetos y puede permitir que usuarios específicos realicen operaciones específicas (lectura, escritura, ejecución) en estos objetos. Además, los objetos se identifican por sus nombres externos.

El control de acceso en los sistemas de bases de datos difiere en varios aspectos del de los sistemas de archivos tradicionales. Las autorizaciones deben refinarse para que los distintos usuarios tengan distintos derechos sobre los mismos objetos de la base de datos. Este requisito implica la capacidad de especificar subconjuntos de objetos con mayor precisión que por nombre y de distinguir entre grupos de usuarios. Además, el control descentralizado de las autorizaciones es de especial importancia en un contexto distribuido. En los sistemas relacionales, los administradores de bases de datos pueden controlar las autorizaciones de forma uniforme mediante construcciones de alto nivel.

Por ejemplo, los objetos controlados pueden especificarse mediante predicados de la misma manera que una calificación de consulta.

Existen dos enfoques principales para el control de acceso a bases de datos. El primer enfoque se denomina control de acceso discrecional (DAC) y los sistemas de gestión de bases de datos (DBMS) lo han proporcionado desde hace tiempo. El DAC define los derechos de acceso en función de los usuarios, el tipo de acceso (p. ej., SELECT, UPDATE) y los objetos a los que se accederá. El segundo enfoque, denominado control de acceso obligatorio (MAC), aumenta aún más la seguridad al restringir el acceso a datos clasificados a usuarios autorizados. La compatibilidad con MAC en los principales DBMS es más reciente y se debe al aumento de las amenazas a la seguridad procedentes de Internet. Otros enfoques profundizan en la incorporación de más semántica al control de acceso, en particular, el control de acceso basado en roles, que considera a los usuarios con diferentes roles, y el control de acceso basado en el propósito, p. ej., las bases de datos hipocráticas, que asocian la información del propósito con los datos, es decir, los motivos de la recopilación y el acceso a los datos.

A partir de las soluciones para el control de acceso en sistemas centralizados, derivamos las de los SGBD distribuidos. Sin embargo, existe una complejidad adicional derivada de la distribución de objetos y usuarios. A continuación, presentamos primero el control de acceso discrecional y obligatorio en sistemas centralizados y, posteriormente, los problemas adicionales y sus soluciones en sistemas distribuidos.

### 3.2.1 Control de acceso discrecional

En el DAC intervienen tres actores principales: el sujeto (p. ej., usuarios, grupos de usuarios) que desencadena la ejecución de los programas de aplicación; las operaciones, integradas en los programas de aplicación; y los objetos de la base de datos, sobre los que se realizan las operaciones. El control de autorización consiste en comprobar si una terna determinada (sujeto, operación, objeto) puede proceder (es decir, si el usuario puede ejecutar la operación sobre el objeto). Una autorización puede considerarse una terna (sujeto, tipo de operación, definición de objeto) que especifica que los sujetos tienen derecho a realizar una operación de tipo operación sobre un objeto. Para controlar las autorizaciones correctamente, el SGBD requiere la definición de sujetos, objetos y derechos de acceso.

La introducción de un tema en el sistema se realiza generalmente mediante un par de credenciales (nombre de usuario y contraseña). El nombre de usuario identifica de forma única a los usuarios con ese nombre en el sistema, mientras que la contraseña, conocida solo por ellos, los autentica .

Para iniciar sesión en el sistema, es necesario proporcionar tanto el nombre de usuario como la contraseña. Esto evita que quienes desconozcan la contraseña accedan al sistema solo con el nombre de usuario.

Los objetos a proteger son subconjuntos de la base de datos. Los sistemas relacionales proporcionan una granularidad de protección más fina y general que los sistemas anteriores. En un sistema de archivos, el gránulo de protección es el archivo. En un sistema relacional, los objetos pueden definirse por su tipo (vista, relación, tupla, atributo), así como por su contenido mediante predicados de selección. Además, el mecanismo de vista, presentado en la sección 3.1 , permite la protección de objetos simplemente ocultando subconjuntos de relaciones (atributos o tuplas) de usuarios no autorizados.

Un derecho expresa una relación entre un sujeto y un objeto para un conjunto específico de operaciones. En un SGBD relacional basado en SQL, una operación es una instrucción de alto nivel, como SELECT, INSERT, UPDATE o DELETE, y los derechos se definen (conceden o revocan) mediante las siguientes instrucciones:

Tipo(s) de operación GRANT ON objeto TO sujeto(s)

Tipo(s) de operación REVOKE DESDE objeto HASTA sujeto(s)

La palabra clave " público" puede utilizarse para referirse a todos los usuarios. El control de autorización se puede caracterizar según quién (los otorgantes) puede otorgar los derechos. Para facilitar la administración de bases de datos, es conveniente definir grupos de usuarios, como en los sistemas operativos, para fines de autorización. Una vez definidos, un grupo de usuarios puede utilizarse como sujeto en las sentencias GRANT y REVOKE .

En su forma más simple, el control está centralizado: un solo usuario o clase de usuario, los administradores de la base de datos, tiene todos los privilegios sobre los objetos de la base de datos y es el único autorizado para usar las sentencias GRANT y REVOKE . Una forma más flexible de control es el descentralizado: el creador de un objeto se convierte en su propietario y se le otorgan todos los privilegios sobre él. En particular, existe el tipo de operación adicional GRANT, que transfiere todos los derechos del otorgante que realiza la sentencia a los sujetos especificados. Por lo tanto, quien recibe el derecho (el cessionario) puede posteriormente otorgar privilegios sobre ese objeto. Por lo tanto, el control de acceso es discrecional, en el sentido de que los usuarios con privilegios de concesión pueden tomar decisiones sobre la política de acceso. El proceso de revocación es complejo, ya que debe ser recursivo. Por ejemplo, si A, quien otorgó a B, quien a su vez otorgó a C el privilegio GRANT sobre el objeto O, desea revocar todos los privilegios de B sobre O, también deben revocarse todos los privilegios de C sobre O. Para realizar la revocación, el sistema debe mantener una jerarquía de concesiones por objeto, donde el creador del objeto es la raíz.

Los privilegios de los sujetos sobre los objetos se registran en el catálogo (directorio) como reglas de autorización. Existen varias maneras de almacenar las autorizaciones. El enfoque más conveniente es considerar todos los privilegios como una matriz de autorización, en la que una fila define un sujeto, una columna un objeto y una entrada de la matriz (para un par sujeto-objeto), las operaciones autorizadas. Las operaciones autorizadas se especifican por su tipo (p. ej., SELECT, UPDATE). También es habitual asociar

con el tipo de operación un predicado que restringe aún más el acceso al objeto.

Esta última opción se proporciona cuando los objetos deben ser relaciones base y no pueden ser vistas. Por ejemplo, una operación autorizada para el par Jones, relación EMP, podría ser

**SELECCIONAR DONDE TÍTULO = "Syst.Anal."**

que autoriza a Jones a acceder únicamente a las tuplas de empleados para analistas de sistemas. La figura 3.5 ofrece un ejemplo de una matriz de autorización donde los objetos son relaciones (EMP y ASG) o atributos (ENAME).

La matriz de autorizaciones se puede almacenar de tres maneras: por fila, por columna o por elemento. Cuando la matriz se almacena por fila, cada sujeto se asocia a la lista de objetos accesibles, junto con sus derechos de acceso. Este enfoque optimiza la aplicación de autorizaciones, ya que todos los derechos del usuario conectado se encuentran agrupados (en el perfil de usuario). Sin embargo, la manipulación de los derechos de acceso por objeto (por ejemplo, hacer público un objeto) no es eficiente, ya que se requiere el acceso a todos los perfiles de los sujetos. Cuando la matriz se almacena por columna, cada objeto se asocia a la lista de sujetos que pueden acceder a él con sus derechos de acceso correspondientes.

Las ventajas y desventajas de este enfoque son las inversas del enfoque anterior.

Las ventajas de ambos enfoques se combinan en el tercer enfoque, en el que la matriz se almacena por elemento, es decir, por relación (sujeto, objeto, derecho). Esta relación puede tener índices tanto en el sujeto como en el objeto, lo que facilita la manipulación rápida de los derechos por sujeto y por objeto.

Gestionar directamente las relaciones entre muchos sujetos y muchos objetos resulta complicado para los administradores de bases de datos. El control de acceso basado en roles (RBAC) soluciona este problema añadiendo roles, lo que proporciona un nivel de independencia entre sujetos y objetos. Los roles corresponden a diversas funciones laborales (p. ej., administrativo, analista, gerente, etc.), a los usuarios se les asignan roles específicos y las autorizaciones sobre los objetos se asignan a roles específicos. Por lo tanto, los usuarios ya no obtienen autorizaciones directamente, sino solo a través de sus roles. Dado que no hay tantos roles, el RBAC simplifica considerablemente el control de acceso, en particular al añadir o modificar cuentas de usuario.

		ENAME ASG
		Punto electromagnético
		Casey ACTUALIZACIÓN ACTUALIZACIÓN ACTUALIZACIÓN
Jones		SELECCIONAR SELECCIONAR SELECCIONAR
		DONDE RESP = "Gerente"
Casey		NINGUNO SELECCIONAR NINGUNO

Fig. 3.5 Ejemplo de matriz de autorización

### 3.2.2 Control de acceso obligatorio

DAC tiene algunas limitaciones. Un problema es que un usuario malintencionado puede acceder a datos no autorizados a través de un usuario autorizado. Por ejemplo, considere al usuario A, que tiene acceso autorizado a las relaciones R y S, y al usuario B, que solo tiene acceso autorizado a la relación S. Si B logra modificar un programa de aplicación utilizado por A para que escriba datos de R en S, entonces B podrá leer datos no autorizados sin infringir las reglas de autorización.

MAC resuelve este problema y mejora aún más la seguridad al definir diferentes niveles de seguridad tanto para los sujetos como para los objetos de datos. Además, a diferencia de DAC, las decisiones sobre políticas de acceso están bajo el control de un único administrador; es decir, los usuarios no pueden definir sus propias políticas ni conceder acceso a los objetos. MAC en bases de datos se basa en el conocido modelo Bell-LaPadula, diseñado para la seguridad de sistemas operativos. En este modelo, los sujetos son procesos que actúan en nombre de un usuario; un proceso tiene un nivel de seguridad, también llamado autorización , derivado del del usuario. En su forma más simple, los niveles de seguridad son Alto Secreto (TS), Secreto (S), Confidencial (C) y Sin Clasificar (U), y se ordenan como TS > S > C > U, donde ">" significa "más seguro". El acceso en modo lectura y escritura por parte de los sujetos está restringido por dos reglas simples:

1. A un sujeto T se le permite leer un objeto de nivel de seguridad I sólo si  $\text{nivel}(T) \geq I$ .
2. A un sujeto T se le permite escribir un objeto de nivel de seguridad I sólo si  $\text{clase}(T) \leq I$ .

La regla 1 (denominada "no lectura") protege los datos de la divulgación no autorizada; es decir, un sujeto con un nivel de seguridad determinado solo puede leer objetos con el mismo nivel de seguridad o uno inferior. Por ejemplo, un sujeto con autorización secreta no puede leer datos de alto secreto.

La regla 2 (denominada "no escritura") protege los datos contra cambios no autorizados. Es decir, un sujeto con un nivel de seguridad determinado solo puede escribir objetos con el mismo nivel de seguridad o uno superior. Por ejemplo, un sujeto con autorización de alto secreto solo puede escribir datos de alto secreto, pero no puede escribir datos secretos (que podrían contener datos de alto secreto).

En el modelo relacional, los objetos de datos pueden ser relaciones, tuplas o atributos. Por lo tanto, una relación se puede clasificar en diferentes niveles: relación (es decir, todas las tuplas en la relación tienen el mismo nivel de seguridad), tupla (es decir, cada tupla tiene un nivel de seguridad) o atributo (es decir, cada valor de atributo distinto tiene un nivel de seguridad). Una relación clasificada se denomina relación multinivel para reflejar que aparecerá de manera diferente (con diferentes datos) para sujetos con diferentes autorizaciones. Por ejemplo, una relación multinivel clasificada a nivel de tupla se puede representar agregando un atributo de nivel de seguridad a cada tupla. De manera similar, una relación multinivel clasificada a nivel de atributo se puede representar agregando un nivel de seguridad correspondiente a cada atributo. La Figura 3.6 ilustra una relación multinivel PROJ\* basada en la relación PROJ que se clasifica a nivel de atributo. Tenga en cuenta que los atributos de nivel de seguridad adicionales pueden aumentar significativamente el tamaño de la relación.

Toda la relación también tiene un nivel de seguridad que es el nivel de seguridad más bajo de todos los datos que contiene. Por ejemplo, la relación PROJ\* tiene un nivel de seguridad C. Cualquier sujeto con un nivel de seguridad igual o superior a C puede acceder a una relación.

## PROY

PNO	SL1	PNAME	PRES	UPUESTO	SL2	SL3	LOC	C	SL4
P1	do	Instrumentación	150000	C Desarrollo de base	de datos	do			do
P2	do	de Montreal	135000	S Nueva York	S Nueva York				S
P3	S CAD/CAM		S	250000					S

Fig. 3.6 Relación multinivel PROJ\* clasificada a nivel de atributo

## PROY C

PNO	SL1	PNAME	P1	PRES	UPUESTO	SL2	SL3	LOC	SL4
Desarrollo	de base	de datos de		do	150000	do	Montreal	C	
P2	do	instrumentación.		do	Nulo	S	Nulo	S	

Fig. 3.7 Relación confidencial PROJ\*C

## PROY

PNO	SL1	PNAME	SL2	BUDGET	SL3	LOC	P1	Instrumentación	C 150000	C	SL4
Montreal	P2	Desarrollo	de base	de datos	C 135000	S Nueva	York	P3	S Nueva	York	P3
C París										do	
		Desarrollo	web	CAD/	\$250000					S	
		C CCS.		C	C 200000					S	

Fig. 3.8 Relación multinivel con poliinstanciación

superior. Sin embargo, un sujeto solo puede acceder a los datos para los que tiene autorización. Por lo tanto, Los atributos para los cuales un sujeto no tiene autorización aparecerán como nulos para el sujeto.

Valores con un nivel de seguridad asociado igual al del sujeto. Figura 3.7

muestra una instancia de la relación PROJ\* a la que accede un sujeto en un nivel confidencial nivel de seguridad.

MAC tiene un fuerte impacto en el modelo de datos porque los usuarios no ven lo mismo datos y tienen que lidiar con efectos secundarios inesperados. Un efecto secundario importante se llama poliinstanciación, que permite que el mismo objeto tenga diferentes valores de atributos Dependiendo del nivel de seguridad de los usuarios. La Figura 3.8 ilustra una multirrelación con Tuplas poliinstanciadas. La tupla de clave primaria P3 tiene dos instancias, cada una con un nivel de seguridad diferente. Esto puede resultar de un sujeto T con nivel de seguridad C insertando una tupla con clave="P3" en relación PROJ\* en la Fig. 3.6. Porque T (con nivel de autorización confidencial) debe ignorar la existencia de una tupla con clave="P3" (clasificado como secreto), la única solución práctica es agregar una segunda tupla con el mismo clave y clasificación diferente. Sin embargo, un usuario con autorización secreta vería ambas tuplas con clave="E3" y deberían interpretar este efecto inesperado.

### 3.2.3 Control de acceso distribuido

Los problemas adicionales del control de acceso en un entorno distribuido se derivan de que los objetos y sujetos están distribuidos y que los mensajes con datos confidenciales pueden ser leídos por usuarios no autorizados. Estos problemas incluyen la autenticación remota de usuarios, la gestión de reglas de acceso discrecional, la gestión de vistas y grupos de usuarios, y la aplicación de la MAC.

La autenticación remota de usuarios es necesaria, ya que cualquier sitio de un SGBD distribuido puede aceptar programas iniciados y autorizados en sitios remotos. Para evitar el acceso remoto de usuarios o aplicaciones no autorizados (por ejemplo, desde un sitio ajeno al SGBD distribuido), los usuarios también deben identificarse y autenticarse en el sitio al que acceden. Además, en lugar de usar contraseñas que podrían obtenerse rastreando mensajes, se podrían usar certificados cifrados.

Hay tres soluciones posibles para gestionar la autenticación:

1. La información de autenticación se mantiene en un sitio central para usuarios globales que luego pueden autenticarse solo una vez y luego acceder a ellos desde múltiples sitios.
2. La información para la autenticación de usuarios (nombre de usuario y contraseña) se replica en todos los sitios del catálogo. Los programas locales, iniciados en un sitio remoto, también deben indicar el nombre de usuario y la contraseña.
3. Todos los sitios del SGBD distribuido se identifican y autentican de forma similar a como lo hacen los usuarios. De esta forma, la comunicación entre sitios está protegida mediante la contraseña del sitio. Una vez autenticado el sitio de inicio, no es necesario autenticar a sus usuarios remotos.

La primera solución simplifica considerablemente la administración de contraseñas y permite la autenticación única (también llamada inicio de sesión único). Sin embargo, el sitio central de autenticación puede ser un punto único de fallo y un cuello de botella. La segunda solución es más costosa en términos de gestión de directorios, dado que la incorporación de un nuevo usuario es una operación distribuida. No obstante, los usuarios pueden acceder a la base de datos distribuida desde cualquier sitio. La tercera solución es necesaria si la información del usuario no está replicada. Sin embargo, también se puede utilizar si existe replicación de la información del usuario. En este caso, la autenticación remota es más eficiente. Si los nombres de usuario y las contraseñas no se replican, deben almacenarse en los sitios donde los usuarios acceden al sistema (es decir, el sitio principal). Esta última solución se basa en el supuesto realista de que los usuarios son más estáticos, o al menos siempre acceden a la base de datos distribuida desde el mismo sitio.

Las reglas de autorización distribuida se expresan de la misma manera que las centralizadas. Al igual que las definiciones de vista, deben almacenarse en el catálogo. Pueden replicarse completamente en cada sitio o almacenarse en los sitios de los objetos referenciados. En este último caso, las reglas se duplican solo en los sitios donde se distribuyen los objetos referenciados. La principal ventaja del enfoque de replicación completa es que la autorización puede procesarse mediante la modificación de consultas en tiempo de compilación. Sin embargo, la gestión de directorios resulta más costosa debido a la duplicación de datos. La segunda solución es mejor si la localidad de referencia es muy alta. Sin embargo, la autorización distribuida no puede controlarse en tiempo de compilación.

El mecanismo de autorización puede considerar las vistas como objetos. Las vistas son objetos compuestos, es decir, están compuestas por otros objetos subyacentes. Por lo tanto, otorgar acceso a una vista implica otorgar acceso a los objetos subyacentes. Si la definición de la vista y las reglas de autorización para todos los objetos se replican completamente (como en muchos sistemas), esta traducción es bastante sencilla y puede realizarse localmente. La traducción es más compleja cuando la definición de la vista y sus objetos subyacentes se almacenan por separado, como ocurre con el supuesto de autonomía del sitio. En este caso, la traducción es una operación totalmente distribuida. Las autorizaciones otorgadas a las vistas dependen de los derechos de acceso del creador de la vista a los objetos subyacentes. Una solución es registrar la información de asociación en el sitio de cada objeto subyacente.

La gestión de grupos de usuarios para la autorización simplifica la administración distribuida de bases de datos. En un SGBD centralizado, "todos los usuarios" se pueden denominar públicos. En un SGBD distribuido, el mismo concepto es útil: público designa a todos los usuarios del sistema. Sin embargo, a menudo se introduce un nivel intermedio para especificar el público en un sitio específico, por ejemplo, public@site\_s. Se pueden definir grupos más precisos mediante el comando

```
DEFINIR GRUPO group_id COMO lista de subject_ids
```

La gestión de grupos en un entorno distribuido presenta algunos problemas, ya que los sujetos de un grupo pueden estar ubicados en varios sitios y el acceso a un objeto puede otorgarse a varios grupos, que a su vez están distribuidos. Si la información del grupo y las reglas de acceso se replican completamente en todos los sitios, la aplicación de los derechos de acceso es similar a la de un sistema centralizado. Sin embargo, mantener esta replicación puede resultar costoso. El problema se agrava si se debe mantener la autonomía del sitio (con control descentralizado). Una solución aplica los derechos de acceso mediante una consulta remota a los nodos que contienen la definición del grupo. Otra solución replica una definición de grupo en cada nodo que contenga un objeto al que puedan acceder los sujetos de ese grupo. Estas soluciones tienden a reducir el grado de autonomía del sitio.

La aplicación de MAC en un entorno distribuido se dificulta por la posibilidad de medios indirectos, denominados canales ocultos, para acceder a datos no autorizados. Por ejemplo, considere una arquitectura de SGBD distribuida simple con dos sitios, cada uno gestionando su base de datos con un único nivel de seguridad; por ejemplo, un sitio es confidencial y el otro, secreto. Según la regla de "no escribir hacia abajo", una operación de actualización de un sujeto con autorización secreta solo podría enviarse al sitio secreto. Sin embargo, según la regla de "no leer hacia arriba", una consulta de lectura del mismo sujeto secreto podría enviarse tanto al sitio secreto como al confidencial. Dado que la consulta enviada al sitio confidencial puede contener información secreta (por ejemplo, en un predicado de selección), se trata potencialmente de un canal oculto. Para evitar estos canales ocultos, una solución consiste en replicar parte de la base de datos para que un sitio con nivel de seguridad I contenga todos los datos a los que un sujeto con nivel de seguridad I puede acceder. Por ejemplo, el sitio secreto replicaría los datos confidenciales para poder procesar completamente las consultas secretas. Un problema con esta arquitectura es la sobrecarga que supone mantener la consistencia de las réplicas (véase el capítulo 6 sobre replicación). Además, aunque no existen canales encubiertos para consultas, aún puede haber c

Para operaciones de actualización, ya que los retrasos en la sincronización de transacciones pueden ser explotados. Por lo tanto, la compatibilidad total con MAC en sistemas de bases de datos distribuidas requiere ampliaciones significativas en las técnicas de gestión de transacciones y procesamiento distribuido de consultas.

### 3.3 Control de integridad semántica

Otro problema importante y complejo para un sistema de bases de datos es cómo garantizar su consistencia. Se dice que un estado de una base de datos es consistente si cumple un conjunto de restricciones, denominadas restricciones de integridad semántica. Mantener una base de datos consistente requiere diversos mecanismos, como el control de concurrencia, la confiabilidad, la protección y el control de integridad semántica, que se incluyen en la gestión de transacciones. El control de integridad semántica garantiza la consistencia de la base de datos rechazando las transacciones de actualización que generan estados incoherentes o activando acciones específicas sobre el estado de la base de datos que compensan los efectos de las transacciones de actualización. Cabe destacar que la base de datos actualizada debe cumplir el conjunto de restricciones de integridad.

En general, las restricciones de integridad semántica son reglas que representan el conocimiento sobre las propiedades de una aplicación. Definen propiedades estáticas o dinámicas de la aplicación que no pueden ser capturadas directamente por los conceptos de objeto y operación de un modelo de datos. Por lo tanto, el concepto de regla de integridad está estrechamente vinculado al de modelo de datos, ya que permite capturar más información semántica sobre la aplicación mediante estas reglas.

Se pueden distinguir dos tipos principales de restricciones de integridad: restricciones estructurales y restricciones de comportamiento. Las restricciones estructurales expresan propiedades semánticas básicas inherentes a un modelo. Ejemplos de estas restricciones son las restricciones de clave única en el modelo relacional o las asociaciones de uno a muchos entre objetos en el modelo orientado a objetos. Las restricciones de comportamiento, por otro lado, regulan el comportamiento de la aplicación. Por lo tanto, son esenciales en el proceso de diseño de bases de datos. Pueden expresar asociaciones entre objetos, como la dependencia de inclusión en el modelo relacional, o describir propiedades y estructuras de objetos. La creciente variedad de aplicaciones de bases de datos y el desarrollo de herramientas de ayuda al diseño de bases de datos exigen potentes restricciones de integridad que puedan enriquecer el modelo de datos.

El control de integridad surgió con el procesamiento de datos y evolucionó desde los métodos procedimentales (en los que los controles se integraban en los programas de aplicación) hasta los métodos declarativos. Los métodos declarativos surgieron con el modelo relacional para mitigar los problemas de dependencia entre programas y datos, la redundancia de código y el bajo rendimiento de los métodos procedimentales. La idea es expresar las restricciones de integridad mediante aserciones de cálculo de predicados. Por lo tanto, un conjunto de aserciones de integridad semántica define la consistencia de la base de datos. Este enfoque permite declarar y modificar fácilmente restricciones de integridad complejas.

El principal problema al soportar el control automático de integridad semántica es que el costo de verificar la violación de restricciones puede ser prohibitivo. Aplicar restricciones de integridad es costoso porque generalmente requiere acceso a una gran cantidad de datos que no están directamente involucrados en las actualizaciones de la base de datos. El problema se agrava cuando las restricciones se definen en una base de datos distribuida.

Se han investigado diversas soluciones para diseñar un gestor de integridad mediante la combinación de estrategias de optimización. Su objetivo es (1) limitar el número de restricciones que deben aplicarse, (2) disminuir el número de accesos a datos para aplicar una restricción dada en presencia de una transacción de actualización, (3) definir una estrategia preventiva que detecte inconsistencias para evitar deshacer las actualizaciones, y (4) realizar el máximo control de integridad posible en tiempo de compilación. Se han implementado algunas de estas soluciones, pero presentan una falta de generalidad. O bien se limitan a un pequeño conjunto de aserciones (restricciones más generales tendrían un coste de comprobación prohibitivo) o bien solo admiten programas restringidos (por ejemplo, actualizaciones de tuplas individuales).

En esta sección, presentamos las soluciones para el control de integridad semántica, primero en sistemas centralizados y luego en sistemas distribuidos. Dado que nuestro contexto es el modelo relacional, consideramos únicamente métodos declarativos.

### 3.3.1 Control de integridad semántica centralizado

Un administrador de integridad semántica tiene dos componentes principales: un lenguaje para expresar y manipular restricciones de integridad y un mecanismo de aplicación que realiza acciones específicas para aplicar la integridad de la base de datos en las transacciones de actualización.

#### 3.3.1.1 Especificación de restricciones de integridad

El administrador de la base de datos manipula las restricciones de integridad mediante un lenguaje de alto nivel. En esta sección, ilustramos un lenguaje declarativo para especificar restricciones de integridad. Este lenguaje se asemeja bastante al lenguaje SQL estándar, pero con mayor generalidad. Permite especificar, leer o eliminar restricciones de integridad. Estas restricciones pueden definirse al crear la relación o en cualquier momento, incluso si la relación ya contiene tuplas. Sin embargo, en ambos casos, la sintaxis es prácticamente la misma. Para simplificar y no perder generalidad, asumimos que el efecto de la violación de una restricción de integridad es la cancelación de las transacciones infractoras.

Sin embargo, el estándar SQL proporciona medios para expresar la propagación de acciones de actualización para corregir inconsistencias mediante la cláusula CASCADING dentro de la declaración de la restricción. De forma más general, se pueden usar desencadenadores (reglas de evento-condición-acción) para propagar actualizaciones automáticamente y, por lo tanto, mantener la integridad semántica. Sin embargo, los desencadenadores son bastante potentes y, por lo tanto, más difíciles de mantener eficientemente que las restricciones de integridad específicas.

En los sistemas de bases de datos relacionales, las restricciones de integridad se definen como afirmaciones. Una aserción es una expresión particular del cálculo relacional de tuplas, en la que cada variable se cuantifica universalmente ( ) o existencialmente ( ). Por lo tanto, una aserción puede considerarse como una calificación de consulta que es verdadera o falsa para cada tupla en el producto cartesiano de las relaciones determinadas por las variables de la tupla. Podemos distinguir tres tipos de restricciones de integridad: predefinidas, de precondition y generales.

Las restricciones predefinidas se basan en palabras clave simples. Mediante ellas, es posible expresar de forma concisa las restricciones más comunes del modelo relacional, como atributos no nulos, claves únicas, claves foráneas o dependencias funcionales.

Los ejemplos 3.11 a 3.14 demuestran restricciones predefinidas.

Ejemplo 3.11 El número de empleado en la relación EMP no puede ser nulo.

ENO NO NULO EN EMP

Ejemplo 3.12 El par (ENO, PNO) es la clave única en relación ASG.

(ENO, PNO) ÚNICO EN ASG

Ejemplo 3.13 El número de proyecto PNO en la relación ASG es una clave foránea que coincide con la clave principal PNO de la relación PROJ. En otras palabras, un proyecto al que se hace referencia en la relación ASG debe existir en la relación PROJ.

PNO EN REFERENCIAS ASG PNO EN PROJ

Ejemplo 3.14 El número de empleado determina funcionalmente el nombre del empleado.

ENO EN EMP DETERMINA ENAME

Las restricciones de precondition expresan las condiciones que deben cumplir todas las tuplas de una relación para un tipo de actualización determinado. El tipo de actualización, que puede ser INSERT, DELETE o MODIFY, permite restringir el control de integridad. Para identificar en la definición de la restricción las tuplas sujetas a actualización, se definen implícitamente dos variables, NEW y OLD. Estas abarcan las tuplas nuevas (que se insertarán) y las tuplas antiguas (que se eliminarán), respectivamente. Las restricciones de precondition se pueden expresar con la sentencia SQL CHECK , que permite especificar el tipo de actualización.

La sintaxis de la instrucción CHECK es

VERIFICAR EN el nombre de la relación CUANDO cambie el tipo (calificación  
sobre el nombre de la relación)

Algunos ejemplos de restricciones de precondición son los siguientes:

Ejemplo 3.15 El presupuesto de un proyecto está entre 500K y 1000K.

```
VERIFICAR PROYECTO (PRESUPUESTO+ >= 500000 Y PRESUPUESTO <= 1000000)
```

Ejemplo 3.16 Sólo se pueden eliminar las tuplas cuyo presupuesto sea 0.

```
VERIFICAR PROYECTO AL BORRAR (PRESUPUESTO = 0)
```

Ejemplo 3.17 El presupuesto de un proyecto sólo puede aumentar.

```
VERIFICAR PROYECTOS (NUEVO PRESUPUESTO > PRESUPUESTO ANTIGUO Y  
NUEVO.PNO = VIEJO.PNO)
```

Las restricciones generales son fórmulas de cálculo relacional de tuplas donde se cuantifican todas las variables. El sistema de base de datos debe garantizar que estas fórmulas sean siempre verdaderas. Las restricciones generales son más concisas que las precompiladas, ya que las primeras pueden implicar más de una relación. Por ejemplo, se necesitan al menos tres restricciones precompiladas para expresar una restricción general en tres relaciones. Una restricción general puede expresarse con la siguiente sintaxis:

```
MARCAR EN lista de nombre de variable:nombre de relación,  
(calificación)
```

A continuación se dan ejemplos de restricciones generales.

Ejemplo 3.18 La restricción del Ejemplo 3.8 también puede expresarse como

```
COMPROBAR EN e1:EMP, e2:EMP  
(e1.ENAME = e2.ENAME SI e1.ENO = e2.ENO)
```

Ejemplo 3.19 La duración total de todos los empleados en el proyecto CAD es menor a 100.

```
COMPROBAR EN g:ASG, j:PROJ (SUMA(g.DUR DONDE  
g.PNO=j.PNO)<100 SI j.PNAME="CAD/CAM")
```

### 3.3.1.2 Cumplimiento de la integridad

Ahora nos centramos en el cumplimiento de la integridad semántica, que consiste en rechazar las transacciones de actualización que violan ciertas restricciones de integridad. Una restricción se viola cuando se vuelve falsa en el nuevo estado de la base de datos generado por la transacción de actualización. Una dificultad importante en el diseño de un gestor de integridad reside en encontrar algoritmos de cumplimiento eficientes. Dos métodos básicos permiten rechazar transacciones de actualización inconsistente. El primero se basa en la detección de inconsistencias. Se ejecuta la transacción de actualización u, lo que provoca un cambio del estado de la base de datos D a Du. El algoritmo de cumplimiento verifica, mediante pruebas derivadas de estas restricciones, que se cumplan todas las restricciones relevantes en el estado Du. Si el estado Du es inconsistente, el SGBD puede intentar modificar Du con acciones alcanzar otro estado consistente, D<sub>tú</sub>, de compensación o restaurar el estado D deshaciendo u. Dado que estas pruebas se aplican después de haber cambiado el estado de la base de datos, generalmente se denominan pruebas posteriores. Este enfoque puede ser ineficiente si se debe deshacer una gran cantidad de trabajo (la actualización de D) en caso de un fallo de integridad.

El segundo método se basa en la prevención de inconsistencias. Una actualización se ejecuta solo si cambia el estado de la base de datos a un estado consistente. Las tuplas sujetas a la transacción de actualización están disponibles directamente (en el caso de una inserción) o deben recuperarse de la base de datos (en el caso de una eliminación o modificación). El algoritmo de cumplimiento verifica que se cumplan todas las restricciones relevantes tras actualizar dichas tuplas. Esto se realiza generalmente aplicando a dichas tuplas pruebas derivadas de las restricciones de integridad. Dado que estas pruebas se aplican antes de que se modifique el estado de la base de datos, se denominan pruebas previas. El enfoque preventivo es más eficiente que el de detección, ya que las actualizaciones no necesitan deshacerse debido a una violación de la integridad.

El algoritmo de modificación de consultas es un ejemplo de método preventivo particularmente eficiente para aplicar restricciones de dominio. Añade la calificación de aserción a la calificación de la consulta mediante un operador AND para que la consulta modificada pueda garantizar la integridad.

Ejemplo 3.20 La consulta para aumentar el presupuesto del proyecto CAD/CAM en un 10%, que se especificaría como

```

ACTUALIZACIÓN DEL PROYECTO
ESTABLECER PRESUPUESTO =
PRESUPUESTO*1.1 DONDE PNAME= "CAD/CAM"

```

se transformará en la siguiente consulta para hacer cumplir la restricción de dominio discutida en el Ejemplo 3.9.

```

ACTUALIZACIÓN DEL PROYECTO
COLOCAR      PRESUPUESTO = PRESUPUESTO * 1.1
DONDE PNAME= "CAD/CAM"
Y           NUEVO PRESUPUESTO ≥ 500000
Y           NUEVO PRESUPUESTO ≤ 1000000

```

El algoritmo de modificación de consultas, conocido por su elegancia, genera preuebas en tiempo de ejecución mediante la combinación AND de los predicados de aserción con los predicados de actualización de cada instrucción de la transacción. Sin embargo, el algoritmo solo se aplica a fórmulas de cálculo de tuplas y se puede especificar de la siguiente manera. Considere la aserción  $(\exists x \in R)F(x)$ , donde  $F$  es una expresión de cálculo de tuplas en la que  $x$  es la única variable libre. Una actualización de  $R$  se puede escribir como  $(\exists x \in R)(Q(x) \rightarrow update(x))$ , donde  $Q$  es una expresión de cálculo de tuplas cuya única variable libre es  $x$ . En términos generales, la modificación de consultas consiste en generar la actualización  $(\exists x \in R)((Q(x) \wedge F(x)) \rightarrow update(x))$ . Por lo tanto,  $x$  debe cuantificarse universalmente.

Ejemplo 3.21 La restricción de clave externa del Ejemplo 3.13 que se puede reescribir como

$g \text{ ASG, } j \text{ PROJ : } g.PNO = j.PNO$

No se pudo procesar mediante la modificación de la consulta porque la variable  $j$  no está cuantificada universalmente.

Para gestionar restricciones más generales, se pueden generar preuebas al definir la restricción y aplicarlas en tiempo de ejecución cuando se producen actualizaciones. En el resto de esta sección, presentamos un método general. Este método se basa en la generación, al definir la restricción, de preuebas que se utilizan posteriormente para evitar la introducción de inconsistencias en la base de datos. Se trata de un método preventivo general que gestiona todas las restricciones introducidas en la sección anterior. Reduce significativamente la proporción de la base de datos que debe comprobarse al aplicar aserciones en presencia de actualizaciones. Esta es una gran ventaja cuando se aplica a un entorno distribuido.

La definición de preueba utiliza relaciones diferenciales, como se define en la sección 3.1.3. Una preueba es una tripleta  $(R, U, C)$  en la que  $R$  es una relación,  $U$  es un tipo de actualización y  $C$  es una aserción que abarca las relaciones diferenciales involucradas en una actualización de tipo  $U$ . Cuando se define una restricción de integridad  $I$ , se puede generar un conjunto de preuebas para las relaciones utilizadas por  $I$ . Siempre que una relación involucrada en  $I$  se actualiza mediante una transacción  $u$ , las preuebas que deben verificarse para aplicar  $I$  son solo las definidas en  $I$  para el tipo de actualización de  $u$ . Este enfoque ofrece dos ventajas de rendimiento. En primer lugar, se minimiza el número de aserciones a aplicar, ya que solo se deben verificar las preuebas de tipo  $u$ . En segundo lugar, el costo de aplicar una preueba es menor que el de aplicar  $I$ , ya que las relaciones diferenciales son, en general, mucho menores que las relaciones base.

Se pueden obtener pruebas preliminares aplicando reglas de transformación a la afirmación original. Estas reglas se basan en un análisis sintáctico de la afirmación y en permutaciones de cuantificadores. Permiten sustituir relaciones diferenciales por relaciones base.

Como las pruebas previas son más simples que las originales, el proceso que las genera se llama simplificación.

Ejemplo 3.22 Considere la expresión modificada de la restricción de clave externa del Ejemplo 3.15. Las pruebas previas asociadas con esta restricción son

$(ASG, INSERTAR, C1), (PROY, ELIMINAR, C2) \text{ y } (PROY, MODIFICAR, C3),$

donde  $C1$  es

NUEVO ASG+, j PROJ: NUEVO.PNO = j .PNO

C2 es

g ASG, VIEJO PROJ- : g.PNO = VIEJO.PNO

y C3 es

g ASG, ANTIGUO PROJ- NUEVO PROJ+ : g.PNO = ANTIGUO.PNO O  
ANTIGUO.PNO = NUEVO.PNO

La ventaja que ofrecen estas pruebas previas es obvia. Por ejemplo, una eliminación en La relación ASG no incurre en ninguna comprobación de afirmación.

El algoritmo de cumplimiento utiliza pruebas previas y se especializa según la clase de las afirmaciones. Se distinguen tres clases de restricciones: restricciones de relación única, restricciones de relación múltiple y restricciones que involucran funciones agregadas.

Resumamos ahora el algoritmo de cumplimiento. Recordemos que una transacción de actualización actualiza todas las tuplas de la relación R que cumplen alguna condición. El algoritmo funciona en dos pasos. El primero genera las relaciones diferenciales R+ y R- a partir de R. El segundo paso consiste simplemente en recuperar las tuplas de R+ y R- que no cumplen las pruebas previas. Si no se recuperan tuplas, la restricción es válida. De lo contrario, se viola.

Ejemplo 3.23 Supongamos que hay una eliminación en PROJ. La ejecución de (PROJ, DELETE, C2) consiste en generar la siguiente sentencia:

resultado ← recupera todas las tuplas de PROJ- donde  $\neg(C2)$

Luego, si el resultado está vacío, la afirmación se verifica mediante la actualización y se preserva la consistencia.

### 3.3.2 Control de integridad semántica distribuida

En esta sección, presentamos algoritmos para garantizar la integridad semántica de bases de datos distribuidas. Estos algoritmos son extensiones del método de simplificación descrito anteriormente. A continuación, asumimos capacidades de gestión de transacciones globales, como las proporcionadas para sistemas homogéneos o sistemas multibase de datos. Por lo tanto, los dos principales problemas al diseñar un gestor de integridad para un SGBD distribuido de este tipo son la definición y el almacenamiento de restricciones, y su aplicación. También analizaremos los problemas relacionados con la comprobación de restricciones de integridad cuando no se admite la gestión de transacciones globales.

### 3.3.2.1 Definición de restricciones de integridad distribuida

Se supone que una restricción de integridad se expresa en el cálculo de predicados. Cada aserción se considera una calificación de consulta, verdadera o falsa, para cada tupla en el producto cartesiano de las relaciones determinadas por las variables de la tupla. Dado que las aserciones pueden involucrar datos almacenados en diferentes ubicaciones, el almacenamiento de las restricciones debe decidirse de forma que se minimice el costo de la comprobación de integridad. Existe una estrategia basada en una taxonomía de restricciones de integridad que distingue tres clases:

1. Restricciones individuales: restricciones de una sola relación y una sola variable. Se refieren únicamente a las tuplas que se actualizan independientemente del resto de la base de datos. Por ejemplo, la restricción de dominio del Ejemplo 3.15 es una aserción individual.
2. Restricciones orientadas a conjuntos: incluyen restricciones multivariables de relación única, como la dependencia funcional (Ejemplo 3.14) y restricciones multivariables de relación múltiple, como las restricciones de clave externa (Ejemplo 3.13).
3. Restricciones que involucran agregados: requieren un procesamiento especial debido al costo de evaluarlos. La afirmación del Ejemplo 3.19 es representativa de una restricción de esta clase.

La definición de una nueva restricción de integridad puede iniciarse en uno de los sitios que almacenan las relaciones involucradas en la aserción. Recuerde que las relaciones pueden fragmentarse. Un predicado de fragmentación es un caso particular de aserción de clase 1. Diferentes fragmentos de la misma relación pueden ubicarse en diferentes sitios. Por lo tanto, definir una aserción de integridad se convierte en una operación distribuida, que se realiza en dos pasos. El primer paso consiste en transformar las aserciones de alto nivel en pruebas, utilizando las técnicas descritas en la sección anterior. El siguiente paso consiste en almacenar las pruebas según la clase de restricciones. Las restricciones de clase 3 se tratan como las de clase 1 o 2, dependiendo de si son individuales o están orientadas a conjuntos.

#### Restricciones individuales

La definición de la restricción se envía a todos los demás sitios que contienen fragmentos de la relación implicada en la restricción. La restricción debe ser compatible con los datos de la relación en cada sitio. La compatibilidad se puede comprobar en dos niveles: predicado y datos. Primero, la compatibilidad del predicado se verifica comparando el predicado de la restricción con el predicado del fragmento. Una restricción C no es compatible con un predicado del fragmento p si "C es verdadero" implica que "p es falso", y es compatible con p en caso contrario. Si se encuentra incompatibilidad en uno de los sitios, la definición de la restricción se rechaza globalmente porque las tuplas de ese fragmento no satisfacen las restricciones de integridad. Segundo, si se ha encontrado compatibilidad del predicado, la restricción se prueba contra la instancia del fragmento. Si esa instancia no la satisface, la restricción también se rechaza globalmente. Si se encuentra compatibilidad, la restricción se almacena en cada sitio. Tenga en cuenta que las comprobaciones de compatibilidad se realizan solo para pruebas previas cuyo tipo de actualización es "insertar" (las tuplas en los fragmentos se consideran "insertadas").

Ejemplo 3.24 Considere la relación EMP, fragmentada horizontalmente en tres sitios utilizando los predicados

$$\begin{aligned} p1 : 0 \leq ENO < "E3" \quad p2 : "E3" \\ \leq ENO \leq "E6" \quad p3 : ENO > "E6" \end{aligned}$$

y la restricción de dominio C:  $ENO < "E4"$ . La restricción C es compatible con p1 (si C es verdadero, p1 es verdadero) y p2 (si C es verdadero, p2 no es necesariamente falso), pero no con p3 (si C es verdadero, p3 es falso). Por lo tanto, la restricción C debe rechazarse globalmente porque las tuplas en el sitio 3 no pueden satisfacer C y, por lo tanto, la relación EMP no satisface C.

#### Restricciones orientadas a conjuntos

Las restricciones orientadas a conjuntos son multivariables; es decir, implican predicados de unión. Aunque el predicado de aserción puede ser multirrelación, una prueba previa se asocia a una sola relación. Por lo tanto, la definición de restricción puede enviarse a todos los sitios que almacenan un fragmento referenciado por estas variables. La comprobación de compatibilidad también involucra fragmentos de la relación utilizados en el predicado de unión. La compatibilidad de predicados es inútil en este caso, ya que es imposible inferir que un predicado de fragmento p es falso si la restricción C (basada en un predicado de unión) es verdadera. Por lo tanto, se debe comprobar la compatibilidad de C con los datos. Esta comprobación de compatibilidad básicamente requiere unir cada fragmento de la relación, por ejemplo, R, con todos los fragmentos de la otra relación, por ejemplo, S, involucrados en el predicado de restricción. Esta operación puede ser costosa y, como cualquier unión, debe ser optimizada por el procesador de consultas distribuidas. Pueden darse tres casos, dados el costo creciente de la comprobación:

1. La fragmentación de R se deriva (ver Cap. 2) de la de S basada en una semiunión en el atributo utilizado en el predicado de unión de la aserción.
2. S está fragmentado en el atributo de unión.
3. S no está fragmentado en el atributo de unión.

En el primer caso, la comprobación de compatibilidad es económica, ya que la tupla de S que coincide con una tupla de R se encuentra en el mismo sitio. En el segundo caso, cada tupla de R debe compararse con, como máximo, un fragmento de S, ya que el valor del atributo de unión de la tupla de R permite encontrar el sitio del fragmento correspondiente de S. En el tercer caso, cada tupla de R debe compararse con todos los fragmentos de S. Si se encuentra compatibilidad para todas las tuplas de R, la restricción puede almacenarse en cada sitio.

Ejemplo 3.25 Considere la prueba previa orientada a conjuntos (ASG, INSERT, C1) definida en el Ejemplo 3.16, donde C1 es

NUEVO ASG+, j PROJ: NUEVO.PNO = j .PNO

Consideremos los tres casos siguientes:

1. ASG se fragmenta utilizando el predicado

#### Proyecto ASGPNO

Donde PROJ1 es un fragmento de la relación PROJ. En este caso, cada tupla NEW de ASG se ha ubicado en el mismo sitio que la tupla j, de modo que NEW.PNO = j .PNO. Dado que el predicado de fragmentación es idéntico al de C1, la comprobación de compatibilidad no implica comunicación.

2. PROJ está fragmentado horizontalmente en función de los dos predicados

p1 : PNO < "P3" p2 :

PNO ≥ "P3"

En este caso, cada tupla NUEVA de ASG se compara con el fragmento PROJ1, si NEW.PNO < "P3", o con el fragmento PROJ2, si NEW.PNO ≥ "P3".

3. PROJ está fragmentado horizontalmente en función de los dos predicados

p1 : PNAME = "CAD/CAM" p2 :

PNAME = "CAD/CAM"

En este caso cada tupla de ASG debe compararse con ambos fragmentos PROJ1 y PROJ2.

#### 3.3.2.2 Aplicación de restricciones de integridad distribuida

Aplicar restricciones de integridad distribuidas es más complejo que en sistemas de gestión de bases de datos centralizados, incluso con soporte para la gestión global de transacciones. El principal problema radica en decidir dónde (en qué sitio) aplicar las restricciones de integridad. La elección depende de la clase de restricción, el tipo de actualización y la naturaleza del sitio donde se emite la actualización (denominado sitio maestro de consultas). Este sitio puede, o no, almacenar la relación actualizada o algunas de las relaciones implicadas en las restricciones de integridad. El parámetro crítico que consideramos es el coste de transferir datos, incluidos los mensajes, de un sitio a otro. A continuación, analizamos los diferentes tipos de estrategias según estos criterios.

#### Restricciones individuales

Se consideran dos casos. Si la transacción de actualización es una instrucción de inserción, el usuario proporciona explícitamente todas las tuplas que se insertarán. En este caso, todas las restricciones individuales se pueden aplicar en el sitio donde se envía la actualización. Si la actualización es una actualización cualificada (instrucciones de eliminación o modificación), se envía a los sitios que almacenan la relación que se actualizará. El procesador de consultas ejecuta la cualificación de la actualización para cada fragmento. Las tuplas resultantes en cada sitio se combinan en un archivo

Relación en el caso de una instrucción de eliminación, o dos, en el caso de una instrucción de modificación (es decir, R+ y R-). Cada sitio involucrado en la actualización distribuida aplica las aserciones relevantes en ese sitio (por ejemplo, las restricciones de dominio cuando se trata de una eliminación).

#### Restricciones orientadas a conjuntos

Primero estudiaremos las restricciones de relación única mediante un ejemplo. Consideraremos la dependencia funcional del Ejemplo 3.14. La prueba previa asociada con el tipo de actualización INSERT es

(EMP, INSERTAR, C)

donde C es

$$(e \in EMP) \wedge (NUEVO1 \in EMP) \wedge (NUEVO2 \in EMP) \quad (1)$$

$$(NUEVO1.ENO = e.ENO \wedge NUEVO1.ENOMBRE = e.ENOMBRE) \quad (2)$$

$$(NUEVO1.ENO = NUEVO2.ENO \wedge NUEVO1.ENOMBRE = NUEVO2.ENOMBRE) \quad (3)$$

La segunda línea de la definición de C verifica la restricción entre las tuplas insertadas (NUEVA1) y las existentes (e), mientras que la tercera la verifica entre las tuplas insertadas. Por ello, se declaran dos variables (NUEVA1 y NUEVA2) en la primera línea.

Consideremos ahora una actualización de EMP. Primero, el procesador de consultas ejecuta la cualificación de la actualización y devuelve una o dos relaciones temporales, como en el caso de las restricciones individuales. Estas relaciones temporales se envían a todos los sitios que almacenan EMP. Supongamos que la actualización es una instrucción INSERT . Cada sitio que almacene un fragmento de EMP aplicará la restricción C descrita anteriormente. Dado que e en C se cuantifica universalmente, C debe ser satisfecha por los datos locales de cada sitio. Esto se debe a que  $x \in \{a_1, \dots, a_n\} f(x)$  es equivalente a  $[f(a_1) \wedge f(a_2) \wedge \dots \wedge f(a_n)]$ .

Por lo tanto, el sitio donde se envía la actualización debe recibir, para cada sitio, un mensaje que indique que se cumple esta restricción y que es una condición para todos los sitios. Si la restricción no se cumple en un sitio, este envía un mensaje de error indicando que se ha violado. La actualización se considera inválida, y es responsabilidad del administrador de integridad decidir si se debe rechazar toda la transacción mediante el administrador de transacciones global.

Consideremos ahora las restricciones multirrelacionales. Para mayor claridad, asumimos que las restricciones de integridad no tienen más de una variable de tupla que abarque la misma relación. Cabe destacar que este es probablemente el caso más frecuente. Al igual que con las restricciones de una sola relación, la actualización se calcula en el sitio donde se envió. La aplicación se realiza en el sitio maestro de consultas, utilizando el algoritmo ENFORCE descrito en el Algoritmo 3.2.

---

Algoritmo 3.2: ENFORCE

---

Entrada: U: tipo de actualización; R: relación

```

comienza a recuperar todas las aserciones
compiladas (R, U, Ci)

inconsistente ← falso para cada aserción compilada
|   resultado ← todas las nuevas (respectivamente, antiguas), tuplas de R donde
|   - (Ci) fin

para si card(resultado) = 0
|   entonces inconsistente ←

verdadero fin si si
|   |   ¬inconsistente entonces envía las tuplas para actualizar a todos los sitios que

almacenan fragmentos de R
de lo contrario rechaza la actualización fin si

fin

```

---

Ejemplo 3.26 Ilustramos este algoritmo con un ejemplo basado en la restricción de clave externa del Ejemplo 3.13. Sea u una inserción de una nueva tupla en ASG. El algoritmo anterior utiliza la prueba previa (ASG, INSERT, C), donde C es

NEW ASG+, j PROJ: NEW.PNO = j .PNO Para esta

restricción, la instrucción de recuperación recupera todas las tuplas nuevas en ASG+, donde C no es verdadero. Esta instrucción se puede expresar en SQL como

```

SELECCIONAR NUEVO.*
DESDE ASG+ NUEVO, PROY
DONDE CONTAR(PROY.PNO DONDE NUEVO.PNO = PROY.PNO)=0

```

Tenga en cuenta que NUEVO.\* denota todos los atributos de ASG+.

Por lo tanto, la estrategia consiste en enviar nuevas tuplas a los sitios que almacenan la relación PROY para realizar las uniones y, posteriormente, centralizar todos los resultados en el sitio maestro de consultas. Por cada sitio que almacena un fragmento de PROY, el sitio une el fragmento con ASG+ y envía el resultado al sitio maestro de consultas, que realiza la unión de todos los resultados.

Si la unión está vacía, la base de datos es consistente. De lo contrario, la actualización genera un estado inconsistente y debe rechazarse mediante el gestor de transacciones global.

También se pueden idear estrategias más sofisticadas que notifiquen o compensen las inconsistencias.

#### Restricciones que involucran agregados

Estas restricciones se encuentran entre las más costosas de probar, ya que requieren el cálculo de las funciones agregadas. Las funciones agregadas que generalmente se manipulan son MIN, MAX, SUM y COUNT. Cada función agregada contiene una parte de proyección y

Una parte de selección. Para aplicar estas restricciones eficientemente, es posible generar pruebas previas que aíslen los datos redundantes, los cuales pueden almacenarse en cada sitio que almacena la relación asociada. Estos datos son lo que llamamos vistas materializadas en la Sección 3.1.2.

### 3.3.2.3 Resumen del control de integridad distribuida

El principal problema del control de integridad distribuido es que los costos de comunicación y procesamiento para aplicar restricciones distribuidas pueden ser prohibitivos. Los dos aspectos principales al diseñar un gestor de integridad distribuida son la definición de las aserciones distribuidas y de los algoritmos de aplicación que minimizan el costo de la comprobación de integridad distribuida. En este capítulo, hemos demostrado que el control de integridad distribuida puede lograrse completamente mediante la extensión de un método preventivo basado en la compilación de restricciones de integridad semántica en pruebas previas. El método es general, ya que puede manejar todos los tipos de restricciones expresadas en la lógica de predicados de primer orden. Es compatible con la definición de fragmentos y minimiza la comunicación entre sitios. Se puede obtener un mejor rendimiento de la aplicación de integridad distribuida si los fragmentos se definen con cuidado. Por lo tanto, la especificación de las restricciones de integridad distribuida es un aspecto importante del proceso de diseño de bases de datos distribuidas.

El método descrito anteriormente asume compatibilidad con transacciones globales. Sin esta compatibilidad, como ocurre en algunos sistemas multibase de datos débilmente acoplados, el problema se complica. En primer lugar, la interfaz entre el gestor de restricciones y el SGBD de componentes es diferente, ya que la comprobación de restricciones ya no puede formar parte de la validación de transacciones globales. En su lugar, los SGBD de componentes deben notificar al gestor de integridad para que realice la comprobación de restricciones después de ciertos eventos, por ejemplo, como resultado de la ejecución de transacciones locales. Esto puede realizarse mediante activadores cuyos eventos son actualizaciones de las relaciones implicadas en las restricciones globales. En segundo lugar, si se detecta una violación de una restricción global, dado que no es posible especificar abortos globales, se deben proporcionar transacciones correctoras específicas para generar estados de base de datos globales consistentes. La solución consiste en contar con una familia de protocolos para la comprobación de integridad global. La raíz de esta familia es una estrategia simple, basada en el cálculo de relaciones diferenciales (como en el método anterior), que se ha demostrado segura (identifica correctamente las violaciones de restricciones) pero imprecisa (puede generar un evento de error aunque no haya violación de restricciones). La inexactitud se debe a que la generación de relaciones diferenciales en diferentes momentos y sitios puede generar estados fantasma para la base de datos global, es decir, estados que nunca existieron. Se proponen extensiones del protocolo básico, ya sea con marca de tiempo o mediante comandos de transacción locales, para resolver este problema.

## 3.4 Conclusión

El control de datos incluye la gestión de vistas, el control de acceso y el control de integridad semántica. En los SGBD relacionales, estas funciones se pueden lograr de forma uniforme mediante la aplicación de reglas que especifican el control de la manipulación de datos. Las soluciones inicialmente diseñadas para gestionar estas funciones en sistemas centralizados se han ampliado y enriquecido significativamente para sistemas distribuidos, en particular, la compatibilidad con vistas materializadas y el control de acceso discrecional basado en grupos. El control de integridad semántica ha recibido menos atención y, por lo general, no cuenta con un buen soporte en los SGBD distribuidos.

El control total de datos es más complejo y costoso en términos de rendimiento en sistemas distribuidos. Los dos aspectos principales para un control de datos eficiente son la definición y el almacenamiento de las reglas (selección de sitio) y el diseño de algoritmos de cumplimiento que minimicen los costos de comunicación. El problema es complejo, ya que una mayor funcionalidad (y generalidad) tiende a incrementar la comunicación entre sitios. El problema se simplifica si las reglas de control se replican completamente en todos los sitios y se complica si se desea preservar la autonomía de los sitios. Además, se pueden realizar optimizaciones específicas para minimizar el costo del control de datos, pero con una sobrecarga adicional, como la gestión de vistas materializadas o datos redundantes. Por lo tanto, la especificación del control de datos distribuidos debe incluirse en el diseño de la base de datos distribuida para que también se considere el costo de control de los programas de actualización.

## 3.5 Notas bibliográficas

El control de datos es bien conocido en sistemas centralizados y todos los principales SGBD ofrecen un amplio soporte para él. La investigación sobre el control de datos en sistemas distribuidos comenzó a mediados de la década de 1980 con el proyecto R\* en IBM Research y ha aumentado considerablemente desde entonces para abordar nuevas aplicaciones importantes, como el almacenamiento o la integración de datos.

La mayor parte del trabajo sobre gestión de vistas se ha centrado en las actualizaciones mediante vistas y la compatibilidad con vistas materializadas. Los dos artículos fundamentales sobre gestión centralizada de vistas son [Chamberlin et al., 1975] y [Stonebraker , 1975]. La primera referencia presenta una solución integrada para la gestión de vistas y autorizaciones en el proyecto System R de IBM Research. La segunda referencia describe la técnica de modificación de consultas propuesta en el proyecto INGRES de UC Berkeley para la gestión uniforme de vistas, autorizaciones y control de integridad semántica. Este método se presentó en la sección 3.1.

Se ofrecen soluciones teóricas al problema de las actualizaciones de vistas en [Bancilhon y Spyros 1981, Dayal y Bernstein 1978, Keller 1982]. En el artículo fundamental sobre la semántica de actualización de vistas [Bancilhon y Spyros 1981], los autores formalizan la propiedad de invariancia de las vistas tras la actualización y muestran cómo se puede actualizar una amplia clase de vistas, incluyendo uniones. La información semántica sobre las relaciones base es

Particularmente útil para encontrar la propagación única de actualizaciones. Sin embargo, los sistemas comerciales actuales son muy restrictivos a la hora de admitir actualizaciones mediante vistas.

Las vistas materializadas han recibido mucha atención en el contexto del almacenamiento de datos. El concepto de instantánea para optimizar la derivación de vistas en sistemas de bases de datos distribuidas se debe a [Adiba y Lindsay, 1980] y se generalizó en Adiba [1981] como un mecanismo unificado para gestionar vistas e instantáneas, así como datos fragmentados y replicados. La automantenibilidad de las vistas materializadas con respecto al tipo de actualizaciones (inserción, eliminación o modificación) se aborda en [Gupta et al., 1996].

En Gupta y Mumick [1999] se puede encontrar un artículo completo sobre la gestión de vistas materializadas , que incluye las principales técnicas para realizar el mantenimiento incremental de las vistas materializadas. El algoritmo de conteo que presentamos en la sección 3.1.3 se propuso en [Gupta et al., 1993]. Presentamos dos optimizaciones importantes recientes que se han propuesto para reducir significativamente el tiempo de mantenimiento del algoritmo de conteo, siguiendo el formalismo de las relaciones multiconjunto generalizadas [Koch, 2010].

La primera optimización consiste en materializar vistas que representan subconsultas de la consulta de entrada [Koch et al., 2014; Berkholz et al. , 2017; Nikolic y Olteanu , 2018]. La segunda optimización aprovecha la desviación estándar de los datos [Kara et al., 2019].

La seguridad en sistemas informáticos en general se presenta en [Hoffman , 1977]. La seguridad en sistemas de bases de datos centralizadas se presenta en [Lunt y Fernández, 1990; Castano et al., 1995]. El control de acceso discrecional (DAC) en sistemas distribuidos recibió mucha atención inicialmente en el contexto del proyecto R\*. El mecanismo de control de acceso del Sistema R [Griffiths y Wade, 1976] se amplía en [Wilms y Lindsay, 1981] para gestionar grupos de usuarios y ejecutarse en un entorno distribuido.

El control de acceso obligatorio (MAC) para sistemas de gestión de bases de datos distribuidos ha despertado gran interés recientemente. El artículo fundamental sobre MAC es el modelo de Bell y LaPadula, diseñado originalmente para la seguridad de sistemas operativos [Bell y Lapuda, 1976]. El MAC para bases de datos se describe en [Lunt y Fernández, 1990; Jajodia y Sandhu, 1991]. Una buena introducción a la seguridad multinivel en sistemas de gestión de bases de datos relacionales se puede encontrar en [Rjaibi, 2004]. La gestión de transacciones en sistemas de gestión de bases de datos seguros multinivel se aborda en [Ray et al., 2000; Jajodia et al., 2001]. Se proponen extensiones de MAC para sistemas de gestión de bases de datos distribuidos en [Thuraisingham, 2001]. El control de acceso basado en roles (RBAC) [Ferraiolo y Kuhn, 1992] extiende el DAC y el MAC añadiendo roles, como un nivel de independencia entre sujetos y objetos. Las bases de datos hipocráticas [Sandhu et al. [1996] asocian la información de propósito con los datos, es decir, las razones para la recopilación y el acceso a los datos.

El contenido de la sección 3.3 proviene en gran parte del trabajo sobre control de integridad semántica descrito en [Simon y Valduriez 1984, 1986, 1987]. En particular, [Simon y Valduriez 1986] extienden una estrategia preventiva para el control de integridad centralizado basada en pruebas previas para ejecutarse en un entorno distribuido, asumiendo soporte de transacciones globales. La idea inicial de los métodos declarativos, es decir, usar aserciones de lógica de predicados para especificar restricciones de integridad, se debe a [Florentin 1974]. Los métodos declarativos más importantes se encuentran en [Bernstein et al. 1980a, Blaustein 1981, Nicolas 1982, Simon y Valduriez 1984, Stonebraker 1975]. El concepto de vistas concretas para almacenar datos redundantes se describe en [Bernstein y Blaustein 1982].

Cabe destacar que las vistas concretas son útiles para optimizar la aplicación de restricciones que involucran agregados. Civelek et al. [1988], Sheth et al. [1988b] y Sheth et al.

[1988a] describen sistemas y herramientas para el control de datos, particularmente la gestión de vistas. En [Grefen y Widom 1997] se aborda la comprobación de la integridad semántica en sistemas multibase de datos acoplados de forma flexible sin soporte de transacciones globales .

## Ceremonias

Problema 3.1 Defina en sintaxis similar a SQL una vista de la base de datos de ingeniería V(ENO, ENAME, PNO, RESP), donde la duración es 24. ¿La vista V es actualizable?

Supongamos que las relaciones EMP y ASG están fragmentadas horizontalmente en función de las frecuencias de acceso de la siguiente manera:

Sitio 1	Sitio 2	Sitio 3
EMP1	EMP2	
ASG1	ASG2	

dónde

$$\text{EMP1} = \sigma_{\text{TÍTULO}=\text{"Ingeniero"}(\text{EMP})}$$

$$\text{EMP2} = \sigma_{\text{TÍTULO} = \text{"Ingeniero"} (\text{EMP})}$$

$$\text{ASG1} = \sigma_{0 < \text{DUR} < 36}(\text{ASG})$$

$$\text{ASG2} = \sigma_{\text{DUR} \geq 36}(\text{ASG})$$

¿En qué sitio(s) debe almacenarse la definición de V sin replicarla completamente, para aumentar la localidad de referencia?

Problema 3.2 Exprese la siguiente consulta: nombres de los empleados en la vista V que trabajan en el proyecto CAD/CAM.

Problema 3.3 (\*) Supóngase que la relación PROJ está fragmentada horizontalmente como

$$\text{PROJ1} = \sigma_{\text{PNAME} = \text{"CAD/CAM"} (\text{PROJ})}$$

$$\text{PROJ2} = \sigma_{\text{PNAME} = \text{"CAD/CAM"} (\text{PROJ})}$$

Modificar la consulta obtenida en el problema 3.2 a una consulta expresada en los fragmentos.

Problema 3.4 (\*\*) Proponga un algoritmo distribuido para actualizar eficientemente una instantánea en un sitio derivada por proyección de una relación fragmentada horizontalmente en otros dos sitios. Dé un ejemplo de consulta sobre las relaciones de vista y base que produzca un resultado inconsistente.

Problema 3.5 (\*) Considere la vista EG del Ejemplo 3.5 , que utiliza las relaciones EMP y ASG como datos base, y suponga que su estado se deriva del del Ejemplo 3.1, de modo que EG tiene 9 tuplas (véase la Fig. 3.4). Suponga que la tupla E3, P3, Consultor, 10 de ASG se actualiza a E3, P3, Ingeniero, 10. Aplique el algoritmo de conteo básico para

Actualizando la vista EG. ¿Qué atributos proyectados se deben agregar a la vista EG para que sea automantenible?

Problema 3.6 Proponga un esquema de relación para almacenar los derechos de acceso asociados con grupos de usuarios en un catálogo de base de datos distribuida y proporcione un esquema de fragmentación para esa relación, asumiendo que todos los miembros de un grupo están en el mismo sitio.

Problema 3.7 (\*\*\*) Proporcione un algoritmo para ejecutar la declaración REVOKE en un DBMS distribuido, asumiendo que el privilegio GRANT se puede otorgar sólo a un grupo de usuarios donde todos sus miembros están en el mismo sitio.

Problema 3.8 (\*\*) Considere la relación multinivel PROJ\*\* en la Fig. 3.8. Suponiendo que solo existen dos niveles de clasificación para los atributos (S y C), proponga una asignación de PROJ\*\* en dos sitios mediante fragmentación y replicación que evite los canales ocultos en las consultas de lectura. Analice las restricciones de las actualizaciones para que esta asignación funcione.

Problema 3.9 Utilizando el lenguaje de especificación de restricciones de integridad de este capítulo, exprese una restricción de integridad que establezca que la duración invertida en un proyecto no puede exceder los 48 meses.

Problema 3.10 (\*) Defina las pruebas previas asociadas con las restricciones de integridad cubiertas en los ejemplos 3.11–3.14.

Problema 3.11 Suponga la siguiente fragmentación vertical de las relaciones EMP, ASG y PROJ:



dónde

EMP1 =	ENO, ENAME(EMP)
EMP2 =	ENO, TÍTULO(EMP)
PROY1 =	PNO, PNAME(PROY)
PROY2 =	PNO, PRESUPUESTO(PROY)
ASG1 =	ENO, PNO, RESP(ASG)
ASG2 =	ENO, PNO, DUR(ASG)

¿Dónde deben almacenarse las pruebas previas obtenidas en el problema 3.9 ?

Problema 3.12 (\*\*) Considere la siguiente restricción orientada a conjuntos:

COMPROBAR EN e:EMP, a:ASG  
 (e.ENO = a.ENO Y (e.TITLE = "Programador")  
 SI a.RESP = "Programador")

¿Qué significa? Suponiendo que EMP y ASG están asignados como en el ejercicio anterior, defina las pruebas previas correspondientes y su almacenamiento. Aplique el algoritmo ENFORCE para una actualización de tipo INSERT en ASG.

Problema 3.13 (\*\*\*) Suponga un sistema multibase de datos distribuido sin soporte para transacciones globales. Suponga también que existen dos sitios, cada uno con una relación EMP (diferente) y un gestor de integridad que se comunica con el SGBD del componente. Supongamos que queremos una restricción de clave única global para EMP. Proponga una estrategia sencilla que utilice relaciones diferenciales para comprobar esta restricción. Analice las posibles acciones cuando se viola una restricción.

## Capítulo 4

### Procesamiento de consultas distribuidas



Al ocultar los detalles básicos sobre la organización física de los datos, los lenguajes de bases de datos relacionales permiten la expresión de consultas complejas de forma concisa y sencilla. En particular, para construir la respuesta a la consulta, el usuario no especifica con precisión el procedimiento a seguir; este procedimiento lo diseña un módulo llamado procesador de consultas. Esto libera al usuario de la optimización de consultas, una tarea laboriosa que el procesador de consultas gestiona mejor, ya que puede aprovechar una gran cantidad de información útil sobre los datos.

Debido a que se trata de un problema crítico de rendimiento, el procesamiento de consultas ha recibido (y continúa recibiendo) considerable atención en el contexto de sistemas de gestión de bases de datos (SGBD) tanto centralizados como distribuidos. Sin embargo, el problema del procesamiento de consultas es mucho más complejo en entornos distribuidos, ya que un mayor número de parámetros afecta el rendimiento de las consultas distribuidas. En particular, las relaciones involucradas en una consulta distribuida pueden fragmentarse o replicarse, lo que genera costos de comunicación.

Además, al haber muchos sitios a los que acceder, el tiempo de respuesta a las consultas puede llegar a ser muy alto.

En este capítulo, presentamos detalladamente el procesamiento de consultas en sistemas de gestión de bases de datos (SGBD) distribuidos. El contexto elegido es el del cálculo relacional y el álgebra relacional, debido a su generalidad y amplio uso en SGBD distribuidos. Como vimos en el capítulo 2, las relaciones distribuidas se implementan mediante fragmentos, con el objetivo de aumentar la localidad de referencia y, en ocasiones, la ejecución en paralelo de las consultas más importantes. La función de un procesador de consultas distribuidas es mapear una consulta de alto nivel (que se asume expresada en cálculo relacional) en una base de datos distribuida (es decir, un conjunto de relaciones globales) en una secuencia de operadores de base de datos (de álgebra relacional) en fragmentos de relación. Varias funciones importantes caracterizan esta asignación. En primer lugar, la consulta de cálculo debe descomponerse en una secuencia de operadores relacionales denominada consulta algebraica. En segundo lugar, los datos a los que accede la consulta deben localizarse para que los operadores en las relaciones se traduzcan para que se apliquen a los datos locales (fragmentos).

Finalmente, la consulta algebraica sobre fragmentos debe extenderse con operadores de comunicación y optimizarse con respecto a una función de coste que se desea minimizar. Esto

La función de costo generalmente se refiere a recursos computacionales como E/S de disco, CPU y redes de comunicación.

Este capítulo se organiza de la siguiente manera: la Sección 4.1 ofrece una visión general del procesamiento de consultas distribuidas. En la Sección 4.2, se describe la localización de datos, con énfasis en las técnicas de reducción y simplificación para los cuatro tipos de fragmentación: horizontal, vertical, derivada e híbrida. En la Sección 4.3, se aborda el principal problema de optimización, que trata sobre la ordenación de las uniones en consultas distribuidas. También se examinan estrategias de unión alternativas basadas en semijoin. En la Sección 4.4, se analiza el modelo de coste distribuido. En la Sección 4.5, se ilustra el uso de las técnicas en tres enfoques básicos de optimización de consultas distribuidas: dinámica, estática e híbrida.

En la sección 4.5, analizamos el procesamiento de consultas adaptativo.

Suponemos que el lector está familiarizado con las nociones básicas de procesamiento de consultas en DBMS centralizados que se tratan en la mayoría de los cursos y libros de texto de bases de datos de pregrado.

## 4.1 Descripción general

Esta sección presenta el procesamiento distribuido de consultas. Primero, en la sección 4.1.1, analizamos el problema del procesamiento de consultas. Luego, en la sección 4.1.2, presentamos la optimización de consultas. Finalmente, en la Sección 4.1.3, presentamos las diferentes capas de procesamiento de consultas desde una consulta distribuida hasta la ejecución de operadores en sitios locales.

### 4.1.1 Problema de procesamiento de consultas

La función principal de un procesador de consultas es transformar una consulta de alto nivel (normalmente, en cálculo relacional) en una consulta equivalente de bajo nivel (normalmente, en alguna variante del álgebra relacional). La consulta de bajo nivel implementa la estrategia de ejecución de la consulta. La transformación debe ser correcta y eficiente. Es correcta si la consulta de bajo nivel tiene la misma semántica que la consulta original, es decir, si ambas consultas producen el mismo resultado. La correspondencia bien definida entre el cálculo relacional y el álgebra relacional facilita la resolución de la cuestión de la corrección. Sin embargo, generar una estrategia de ejecución eficiente es más difícil. Una consulta de cálculo relacional puede tener muchas transformaciones equivalentes y correctas en álgebra relacional.

Dado que cada estrategia de ejecución equivalente puede dar lugar a consumos muy diferentes de recursos informáticos, la principal dificultad es seleccionar la estrategia de ejecución que minimice el consumo de recursos.

Ejemplo 4.1 Consideraremos el siguiente subconjunto del esquema de base de datos de ingeniería:

```
EMP(ENO, ENAME, TÍTULO)
ASG(ENO, PNO, RESP, DUR)
```

y la siguiente consulta de usuario simple:

“Encuentre los nombres de los empleados que gestionan un proyecto”

La expresión de la consulta en cálculo relacional utilizando la sintaxis SQL (con unión natural) es

```
SELECCIONAR NOMBRE
DESDE EMP NATURAL ÚNETE A ASG
DONDE RESP = "Gerente"
```

Dos consultas de álgebra relacional equivalentes que son transformaciones correctas de la consulta anterior son

$$\text{ENAME}(\sigma_{\text{RESP}=\text{"Administrador"}}(\text{EMP} \times \text{ASG}))$$

y

$$\text{ENAME}(\text{ENO}(\sigma_{\text{RESP}=\text{"Administrador"}}(\text{ASG})))$$

Es intuitivamente obvio que la segunda consulta, que evita el producto cartesiano de EMP y ASG, consume muchos menos recursos computacionales que la primera y, por lo tanto, debería conservarse.

En un sistema distribuido, el álgebra relacional no basta para expresar estrategias de ejecución. Debe complementarse con operadores para el intercambio de datos entre sitios. Además de la elección de los operadores de álgebra relacional para ordenar, el procesador de consultas distribuidas también debe seleccionar los mejores sitios para procesar los datos y, posiblemente, la forma en que deben transformarse. Esto amplía el margen de soluciones para elegir la estrategia de ejecución distribuida, lo que dificulta considerablemente el procesamiento de consultas distribuidas en comparación con el procesamiento centralizado.

Ejemplo 4.2. Este ejemplo ilustra la importancia de la selección del sitio y la comunicación para una consulta de álgebra relacional seleccionada en una base de datos fragmentada. Consideremos la siguiente consulta del Ejemplo 4.1:

$$\text{ENAME}(\text{EMP ENO}(\sigma_{\text{RESP}=\text{"Administrador"}}(\text{ASG})))$$

Suponemos que las relaciones EMP y ASG están fragmentadas horizontalmente de la siguiente manera:

$$\text{EMP1} = \sigma_{\text{ENO} \leq \text{"E3}}(\text{EMP})$$

$$\text{EMP2} = \sigma_{\text{ENO} > \text{"E3}}(\text{EMP})$$

$$\text{ASG1} = \sigma_{\text{ENO} \leq \text{"E3}}(\text{ASG})$$

$$\text{ASG2} = \sigma_{\text{ENO} > \text{"E3}}(\text{ASG})$$

Los fragmentos ASG1, ASG2, EMP1 y EMP2 se almacenan en los sitios 1, 2, 3 y 4, respectivamente, y se espera el resultado en el sitio 5.

Para simplificar, ignoramos el operador de proyecto en lo sucesivo. En la figura 4.1 se muestran dos estrategias de ejecución distribuida equivalentes para la consulta anterior.

Una flecha del sitio i al sitio j, etiquetada con R, indica que la relación R se transfiere del sitio i al sitio j. La estrategia A aprovecha que las relaciones EMP y ASG se fragmentan de la misma manera para ejecutar los operadores de selección y unión en paralelo.

La estrategia B es una estrategia predeterminada (siempre funciona) que simplemente centraliza todos los datos de los operandos en el sitio de resultados antes de procesar la consulta.

Para evaluar el consumo de recursos de estas dos estrategias, utilizamos un modelo de costos muy simple.

Suponemos que el acceso a una tupla, denotado por  $tup_{acc}$ , es de 1 unidad (sin especificar) y la transferencia de una tupla, denotada por  $tup_{trans}$ , es de 10 unidades.

Suponemos que las relaciones EMP y ASG tienen 400 y 1000 tuplas, respectivamente, y que hay 20 administradores en la relación ASG. También asumimos que los datos se distribuyen uniformemente entre los sitios. Finalmente, asumimos que las relaciones ASG y EMP están agrupadas localmente en los atributos RESP y ENO, respectivamente. Por lo tanto, existe acceso directo a las tuplas de ASG (EMP, respectivamente) según el valor del atributo RESP (ENO, respectivamente).

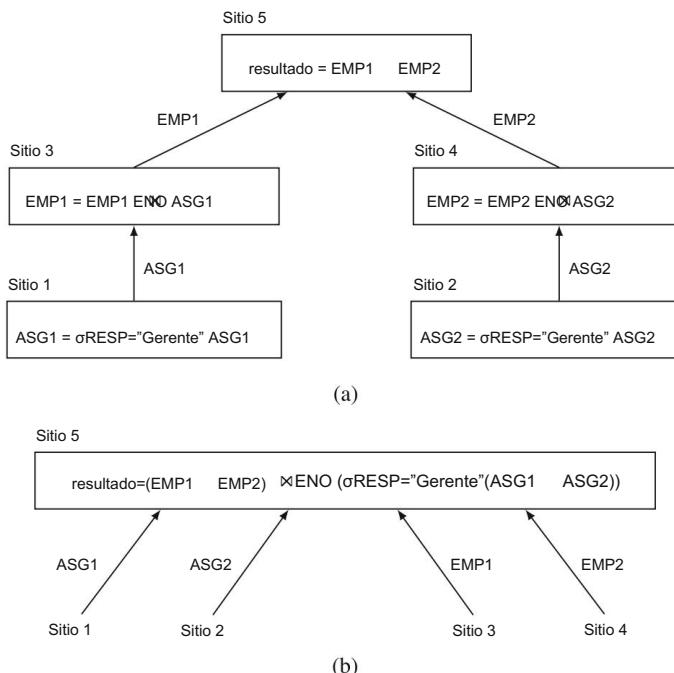


Figura 4.1 Estrategias de ejecución distribuida equivalentes. (a) Estrategia A. (b) Estrategia B

El coste total de la estrategia A se puede derivar de la siguiente manera:

1. Producir ASG seleccionando ASG requiere (10 + 10)	=	20
Tupac	=	20
2. Transferir ASG a los sitios de EMP requiere (10 + 10)	=	200
tuptrans	=	200
3. Producir EMP uniendo ASG y EMP requiere (10 + 10) tupacc 2	=	40
4. Transferir EMP al sitio de resultados requiere (10 + 10)	=	200
tuptrans	=	200
El costo total es		<u>460</u>

El costo de la estrategia B se puede derivar de la siguiente manera:

1. Transferir EMP al sitio 5 requiere 400 tuptrans	2. Transferir ASG al sitio 5 requiere 1000 tuptrans	= 4.000
3. Producir ASG seleccionando ASG requiere 1000 tupacc	= 10.000	
4. Unirse a EMP y ASG requiere 400 20 tupacc	= 8.000	
El costo total es		<u>23.000</u>

En la estrategia A, la unión de ASG y EMP (paso 3) puede explotar el índice agrupado

En ENO de EMP. Por lo tanto, se accede a EMP solo una vez por cada tupla de ASG. En estrategia B, asumimos que los métodos de acceso a las relaciones EMP y ASG se basan en atributos RESP y ENO se pierden debido a la transferencia de datos. Esta es una suposición razonable. En la práctica. Suponemos que la unión de EMP y ASG en el paso 4 se realiza mediante el algoritmo de bucle anidado predeterminado (que simplemente realiza el producto cartesiano de los dos relaciones de entrada). La estrategia A es mejor por un factor de 50, lo cual es bastante significativo. Además, proporciona una mejor distribución del trabajo entre sitios. La diferencia Sería aún mayor si asumimos una comunicación más lenta y/o un mayor grado de fragmentación.

#### 4.1.2 Optimización de consultas

La optimización de consultas se refiere al proceso de producir un plan de ejecución de consultas (QEP), que representa una estrategia de ejecución para la consulta. Este QEP minimiza una Función de coste objetivo. Un optimizador de consultas, el módulo de software que realiza las consultas La optimización suele considerarse como compuesta por tres componentes: un espacio de búsqueda, un modelo de costos y una estrategia de búsqueda.

##### 4.1.2.1 Espacio de búsqueda

El espacio de búsqueda es el conjunto de planes de ejecución alternativos que representan la entrada consulta. Estos planes son equivalentes, en el sentido de que producen el mismo resultado,

Pero difieren en el orden de ejecución de los operadores y su implementación, y, por lo tanto, en su rendimiento. El espacio de búsqueda se obtiene aplicando reglas de transformación, como las del álgebra relacional.

#### 4.1.2.2 Modelo de costos

El modelo de costos se utiliza para predecir el costo de cualquier plan de ejecución y comparar planes equivalentes para elegir el mejor. Para ser preciso, el modelo de costos debe comprender bien el entorno de ejecución distribuido, utilizando estadísticas sobre los datos y las funciones de costos.

En una base de datos distribuida, las estadísticas suelen referirse a fragmentos e incluyen la cardinalidad y el tamaño de estos, así como el tamaño y el número de valores distintos de cada atributo. Para minimizar la probabilidad de error, a veces se utilizan estadísticas más detalladas, como histogramas de valores de atributos, a costa de un mayor coste de gestión. La precisión de las estadísticas se consigue mediante actualizaciones periódicas.

Una buena medida del costo es el costo total que se incurrirá en procesar la consulta. El costo total es la suma de todos los tiempos invertidos en procesar los operadores de la consulta en varios sitios y en la comunicación entre sitios. Otra buena medida es el tiempo de respuesta de la consulta, que es el tiempo transcurrido para ejecutarla. Dado que los operadores pueden ejecutarse en paralelo en diferentes sitios, el tiempo de respuesta de una consulta puede ser significativamente menor que su costo total.

En un sistema de base de datos distribuida, el coste total que debe minimizarse incluye los costes de CPU, E/S y comunicación. El coste de CPU se genera al realizar operaciones con los datos en la memoria principal. El coste de E/S es el tiempo necesario para acceder al disco. Este coste se puede minimizar reduciendo el número de accesos al disco mediante métodos de acceso rápido a los datos y un uso eficiente de la memoria principal (gestión de buffer). El coste de comunicación es el tiempo necesario para intercambiar datos entre los sitios que participan en la ejecución de la consulta. Este coste se produce al procesar los mensajes (formatear/desformatear) y al transmitir los datos a través de la red de comunicación.

El componente de coste de comunicación es probablemente el factor más importante a considerar en las bases de datos distribuidas. La mayoría de las primeras propuestas para la optimización de consultas distribuidas asumían que el coste de comunicación predominaba en gran medida sobre el coste de procesamiento local (E/S y coste de CPU), por lo que ignoraban este último. Sin embargo, los entornos de procesamiento distribuido modernos cuentan con redes de comunicación mucho más rápidas, cuyo ancho de banda es comparable al de los discos. Por lo tanto, la solución consiste en una combinación ponderada de estos tres componentes de coste, ya que todos contribuyen significativamente al coste total de la evaluación de una consulta.

En este capítulo, consideraremos el álgebra relacional como base para expresar el resultado del procesamiento de consultas. Por lo tanto, la complejidad de los operadores del álgebra relacional, que afecta directamente su tiempo de ejecución, dicta algunos principios útiles para un procesador de consultas. Estos principios pueden ayudar a elegir la estrategia de ejecución final.

La forma más sencilla de definir la complejidad es en términos de cardinalidades de relación independientes de los detalles de implementación física, como la fragmentación y el almacenamiento.

estructuras. La complejidad es  $O(n)$  para operadores unarios, donde  $n$  denota la cardinalidad de la relación, si las tuplas resultantes pueden obtenerse independientemente unas de otras.

La complejidad es  $O(n \log n)$  para los operadores binarios si cada tupla de una relación debe compararse con cada tupla de la otra basándose en la igualdad de los atributos seleccionados.

Esta complejidad supone que las tuplas de cada relación deben ordenarse según los atributos de comparación. Sin embargo, el uso de hash y memoria suficiente para almacenar una relación hash puede reducir la complejidad de los operadores binarios a  $O(n)$ . Los proyectos con operadores de eliminación de duplicados y agrupación requieren que cada tupla de la relación se compare con las demás y, por lo tanto, también tienen una complejidad  $O(n \log n)$ .

Finalmente, la complejidad es  $O(n^2)$  para el producto cartesiano de dos relaciones porque cada tupla de una relación debe combinarse con cada tupla de la otra.

#### 4.1.2.3 Estrategia de búsqueda

La estrategia de búsqueda explora el espacio de búsqueda y selecciona el mejor plan, utilizando el modelo de costos. Define qué planes se examinan y en qué orden. Los detalles del entorno distribuido se capturan mediante el espacio de búsqueda y el modelo de costos.

Un método inmediato para optimizar consultas consiste en explorar el espacio de soluciones, predecir exhaustivamente el coste de cada estrategia y seleccionar la estrategia con el mínimo coste. Si bien este método es eficaz para seleccionar la mejor estrategia, puede suponer un coste de procesamiento significativo para la optimización en sí. El problema radica en que el espacio de soluciones puede ser amplio; es decir, puede haber muchas estrategias equivalentes, incluso cuando la consulta implica un número reducido de relaciones. El problema se agrava a medida que aumenta el número de relaciones o fragmentos (por ejemplo, supera 10). Un coste de optimización elevado no es necesariamente negativo, sobre todo si la optimización de la consulta se realiza una sola vez para varias ejecuciones posteriores.

La estrategia de búsqueda más popular utilizada por los optimizadores de consultas es la programación dinámica, propuesta por primera vez en el proyecto System R de IBM Research. Esta consiste en construir planes, comenzando con las relaciones base y uniendo una relación más en cada paso hasta obtener planes completos. La programación dinámica construye todos los planes posibles, priorizando la amplitud, antes de elegir el mejor. Para reducir el coste de optimización, los planes parciales que probablemente no conduzcan al plan óptimo se eliminan (es decir, se descartan) lo antes posible.

Para consultas muy complejas, al ampliar el espacio de búsqueda, se pueden utilizar estrategias aleatorias como la Mejora Iterativa y el Recocido Simulado. Estas estrategias buscan encontrar una solución óptima, no necesariamente la mejor, pero con un buen equilibrio entre el tiempo de optimización y el tiempo de ejecución.

Otra solución complementaria consiste en restringir el espacio de soluciones para que solo se consideren unas pocas estrategias. Tanto en sistemas centralizados como distribuidos, una heurística común consiste en minimizar el tamaño de las relaciones intermedias. Esto puede lograrse ejecutando primero los operadores unarios y ordenando los operadores binarios según el tamaño creciente de sus relaciones intermedias.

Una consulta puede optimizarse en diferentes momentos con respecto a su ejecución. La optimización puede realizarse estáticamente antes de ejecutar la consulta o dinámicamente durante su ejecución. La optimización estática de consultas se realiza durante la compilación. Por lo tanto, el coste de la optimización puede amortizarse en múltiples ejecuciones. Por lo tanto, este momento es adecuado para su uso con el método de búsqueda exhaustiva. Dado que los tamaños de las relaciones intermedias de una estrategia no se conocen hasta el tiempo de ejecución, deben estimarse utilizando estadísticas de bases de datos. Los errores en estas estimaciones pueden llevar a la elección de estrategias subóptimas.

La optimización dinámica de consultas se realiza durante la ejecución de la consulta. En cualquier punto de la ejecución, la elección del mejor operador siguiente puede basarse en el conocimiento preciso de los resultados de los operadores ejecutados previamente. Por lo tanto, no se necesitan estadísticas de la base de datos para estimar el tamaño de los resultados intermedios. Sin embargo, pueden ser útiles para elegir los primeros operadores. La principal ventaja sobre la optimización estática de consultas es que el procesador de consultas dispone de los tamaños reales de las relaciones intermedias, lo que minimiza la probabilidad de una mala elección. La principal desventaja es que la optimización de consultas, una tarea costosa, debe repetirse para cada ejecución. Por lo tanto, este enfoque es el más adecuado para consultas ad hoc.

La optimización de consultas híbrida busca ofrecer las ventajas de la optimización de consultas estáticas, evitando los problemas generados por estimaciones inexactas. El enfoque es básicamente estático, pero la optimización de consultas dinámicas puede implementarse en tiempo de ejecución cuando se detecta una gran diferencia entre los tamaños predichos y el tamaño real de las relaciones intermedias.

#### 4.1.3 Capas de procesamiento de consultas

El problema del procesamiento de consultas puede descomponerse en varios subproblemas, correspondientes a diversas capas. En la figura 4.2 se muestra un esquema genérico de capas para el procesamiento de consultas, donde cada capa resuelve un subproblema bien definido. Para simplificar el análisis, supongamos un procesador de consultas estático que no explota fragmentos replicados. La entrada es una consulta sobre datos globales expresados en cálculo relacional. Esta consulta se plantea sobre relaciones globales (distribuidas), lo que significa que la distribución de datos está oculta. Cuatro capas principales intervienen en el procesamiento distribuido de consultas. Las tres primeras capas asignan la consulta de entrada a un plan de ejecución de consultas distribuidas (QEP distribuido). Realizan las funciones de descomposición de consultas, localización de datos y optimización global de consultas. La descomposición de consultas y la localización de datos corresponden a la reescritura de consultas. Las tres primeras capas son ejecutadas por un sitio de control central y utilizan la información del esquema almacenada en el directorio global. La cuarta capa realiza la ejecución distribuida de consultas mediante la ejecución del plan y la devolución de la respuesta a la consulta. Esta tarea la realizan los sitios locales y el sitio de control. En el resto de esta sección, se presentarán estas cuatro capas.

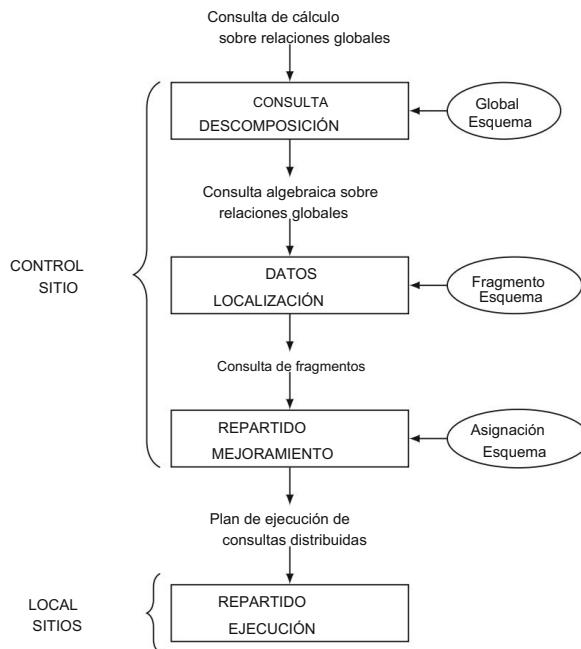


Fig. 4.2 Esquema de capas genérico para el procesamiento de consultas distribuidas

#### 4.1.3.1 Descomposición de consultas

La primera capa descompone la consulta de cálculo en una consulta algebraica sobre relaciones globales. La información necesaria para esta transformación se encuentra en el esquema conceptual global que describe las relaciones globales. Sin embargo, la información sobre la distribución de datos no se utiliza aquí, sino en la siguiente capa. Las técnicas empleadas por esta capa son las de un SGBD centralizado, por lo que en este capítulo solo las recordaremos brevemente.

La descomposición de consultas puede considerarse en cuatro pasos sucesivos. Primero, la consulta de cálculo se reescribe en una forma normalizada que facilita su posterior manipulación. La normalización de una consulta generalmente implica la manipulación de los cuantificadores de la consulta y de la calificación de la consulta mediante la aplicación de la prioridad del operador lógico.

En segundo lugar, la consulta normalizada se analiza semánticamente para detectar y rechazar las consultas incorrectas lo antes posible. Una consulta es semánticamente incorrecta si sus componentes no contribuyen de ninguna manera a la generación del resultado. En el contexto del cálculo relacional, no es posible determinar la corrección semántica de las consultas generales. Sin embargo, sí es posible hacerlo para una amplia gama de consultas relacionales, es decir, aquellas que no contienen disyunción ni negación. Esto se basa en la representación de la consulta como un grafo, denominado grafo de consulta o grafo de conexión. Definimos este grafo para los tipos de consultas más útiles que involucran operadores de selección, proyección y unión. En un grafo de consulta, un nodo indica el resultado.

Relación, y cualquier otro nodo indica una relación de operandos. Una arista entre dos nodos que no son resultados representa una unión, mientras que una arista cuyo nodo de destino es el resultado representa un proyecto. Además, un nodo no resultante puede etiquetarse mediante un predicado de selección o de autounión (unión de la relación consigo misma). Un subgrafo importante del grafo de consulta es el grafo de unión, en el que solo se consideran las uniones.

En tercer lugar, se simplifica la consulta correcta (aún expresada en cálculo relacional) . Una forma de simplificar una consulta es eliminar predicados redundantes. Tenga en cuenta que es probable que surjan consultas redundantes cuando una consulta es el resultado de transformaciones del sistema aplicadas a la consulta del usuario. Como se vio en el capítulo 3, dichas transformaciones se utilizan para realizar control de datos distribuidos (vistas, protección y control de integridad semántica).

En cuarto lugar, la consulta de cálculo se reestructura como una consulta algebraica. Recordemos de la sección 4.1.1 que varias consultas algebraicas pueden derivarse de la misma consulta de cálculo, y que algunas consultas algebraicas son "mejores" que otras. La calidad de una consulta algebraica se define en términos del rendimiento esperado. La forma tradicional de realizar esta transformación hacia una "mejor" especificación algebraica es comenzar con una consulta algebraica inicial y transformarla para encontrar una "buena". La consulta algebraica inicial se deriva inmediatamente de la consulta de cálculo traduciendo los predicados y la sentencia de destino en operadores relacionales tal como aparecen en la consulta. Esta consulta algebraica traducida directamente se reestructura luego mediante reglas de transformación. La consulta algebraica generada por esta capa es buena en el sentido de que generalmente se evitan las peores ejecuciones. Por ejemplo, se accederá a una relación solo una vez, incluso si hay varios predicados de selección. Sin embargo, esta consulta generalmente está lejos de proporcionar una ejecución óptima, ya que la información sobre la distribución de datos y la asignación de fragmentos no se utiliza en esta capa.

#### 4.1.3.2 Localización de datos

La entrada a la segunda capa es una consulta algebraica sobre relaciones globales. Su función principal es localizar los datos de la consulta utilizando la información de distribución de datos del esquema de fragmentos. En el capítulo 2, vimos que las relaciones se fragmentan y se almacenan en subconjuntos disjuntos, denominados fragmentos, cada uno de los cuales se almacena en un sitio diferente. Esta capa determina los fragmentos que intervienen en la consulta y transforma la consulta distribuida en una consulta sobre fragmentos. La fragmentación se define mediante reglas de fragmentación que pueden expresarse como operadores relacionales. Una relación global puede reconstruirse aplicando las reglas de fragmentación y derivando un programa, llamado programa de materialización, de operadores de álgebra relacional que actúa sobre los fragmentos. La localización consta de dos pasos. Primero, la consulta se convierte en una consulta de fragmentos sustituyendo cada relación por su programa de materialización.

En segundo lugar, la consulta de fragmentos se simplifica y reestructura para generar otra consulta "buena". La simplificación y la reestructuración pueden realizarse según las mismas reglas utilizadas en la capa de descomposición. Al igual que en la capa de descomposición, la consulta de fragmentos final suele ser deficiente porque no se utiliza la información relativa a los fragmentos.

#### 4.1.3.3 Optimización distribuida

La entrada a la tercera capa es una consulta algebraica sobre fragmentos, es decir, una consulta de fragmentos. El objetivo de la optimización de consultas es encontrar una estrategia de ejecución para la consulta cercana a la óptima. Una estrategia de ejecución para una consulta distribuida puede describirse con operadores de álgebra relacional y primitivas de comunicación (operadores de envío/recepción) para la transferencia de datos entre sitios. Las capas anteriores ya han optimizado la consulta, por ejemplo, eliminando expresiones redundantes. Sin embargo, esta optimización es independiente de las características de los fragmentos, como la asignación de fragmentos y las cardinalidades. Además, los operadores de comunicación aún no se han especificado. Al permutar el orden de los operadores dentro de una consulta en fragmentos, se pueden encontrar múltiples consultas equivalentes.

La optimización de consultas consiste en encontrar la mejor ordenación de los operadores en la consulta, incluyendo los operadores de comunicación, que minimice una función de coste. Esta función, a menudo definida en unidades de tiempo, se refiere al uso de recursos informáticos como disco, CPU y red. Generalmente, es una combinación ponderada de los costes de E/S, CPU y comunicación. Para seleccionar la ordenación de los operadores, es necesario predecir los costes de ejecución de las alternativas de ordenación.

La determinación de los costos de ejecución antes de la ejecución de la consulta (es decir, la optimización estática) se basa en las estadísticas de fragmentos y las fórmulas para estimar las cardinalidades de los resultados de los operadores relationales. Por lo tanto, las decisiones de optimización dependen de la asignación de fragmentos y de las estadísticas disponibles sobre los fragmentos, registradas en el esquema de asignación.

Un aspecto importante de la optimización de consultas es el orden de las uniones, ya que las permutaciones de las uniones dentro de la consulta pueden generar mejoras de órdenes de magnitud.

La salida de la capa de optimización de consultas es una consulta algebraica optimizada con operadores de comunicación incluidos en los fragmentos. Normalmente se representa y se guarda (para futuras ejecuciones) como un QEP distribuido.

#### 4.1.3.4 Ejecución distribuida

La última capa la realizan todos los sitios con fragmentos involucrados en la consulta. Cada subconsulta que se ejecuta en un sitio, denominada consulta local, se optimiza utilizando el esquema local del sitio y se ejecuta. En este punto, se pueden elegir los algoritmos para ejecutar los operadores relationales. La optimización local utiliza los algoritmos de sistemas centralizados.

La implementación clásica de operadores relationales en sistemas de bases de datos se basa en el modelo iterador, que proporciona paralelismo segmentado dentro de los árboles de operadores. Se trata de un modelo pull simple que ejecuta operadores desde el nodo raíz (que produce el resultado) hasta los nodos hoja (que acceden a las relaciones base). Por lo tanto, no es necesario materializar los resultados intermedios de los operadores, ya que las tuplas se generan bajo demanda y pueden ser utilizadas por operadores posteriores. Sin embargo, requiere que los operadores se implementen en modo segmentado, utilizando una interfaz de apertura-siguiente-cierre. Cada operador debe implementarse como un iterador con tres funciones:

1. Open(): inicializa el estado interno del operador, por ejemplo, asignar una tabla hash; 2. Next(): produce y devuelve la siguiente tupla de resultados o nulo; 3. Close(): limpia todos los recursos asignados, después de que se hayan procesado todas las tuplas.

Por lo tanto, un iterador proporciona el componente de iteración de un bucle while, es decir, inicialización, incremento, condición de terminación del bucle y limpieza final. La ejecución de un QEP se realiza de la siguiente manera. Primero, se inicializa la ejecución llamando a Open() en el operador raíz del árbol de operadores, que luego reenvía la llamada a Open() a través de todo el plan utilizando los propios operadores. A continuación, el operador raíz produce iterativamente su siguiente registro de resultado reenviando la llamada a Next() a través del árbol de operadores según sea necesario. La ejecución finaliza cuando la última llamada Open() devuelve "fin" al operador raíz.

Para ilustrar la implementación de un operador relacional utilizando la interfaz abrir-siguiente-cerrar, consideremos el operador de unión de bucle anidado que realiza RS en el atributo A. Las funciones Open() y Next() son las siguientes:

Función Open()

```
R.Abrir();
S.Abrir(); r :=
R.Siguiente();
```

Función Next()

```
mientras (r = nulo) hacer
  (mientras (s:=S.Next()) = null) hacer
    si rA=sA entonces devuelve(r,s);
  S.cerrar();
  S.abrir();
  r:=R.siguiente();
devuelve nulo;
```

No siempre es posible implementar un operador en modo pipeline. Estos operadores son bloqueantes, es decir, necesitan materializar sus datos de entrada en memoria o disco antes de poder generar una salida. Ejemplos de operadores bloqueantes son la ordenación y la unión hash. Si los datos ya están ordenados, se pueden implementar la combinación de datos, la agrupación y la eliminación de duplicados en modo pipeline.

## 4.2 Localización de datos

La localización de datos traduce una consulta algebraica sobre relaciones globales en una consulta de fragmentos utilizando la información almacenada en el esquema de fragmentos. Una forma sencilla de hacerlo es generar una consulta donde cada relación global se sustituye por su programa de materialización. Esto puede entenderse como la sustitución de las hojas del trie de operadores de la consulta distribuida por subárboles correspondientes a los programas materializados. En general, este enfoque es ineficiente, ya que aún se pueden realizar importantes reestructuraciones y simplificaciones de la consulta de fragmentos. En el resto de esta sección, para cada tipo de fragmentación, presentaremos técnicas de reducción que generan resultados más simples y

Consultas optimizadas. Utilizamos las reglas de transformación y las heurísticas, como la inserción de operadores unarios en el trie.

## 4.2.1 Reducción por fragmentación horizontal primaria

La función de fragmentación horizontal distribuye una relación según predicados de selección. El siguiente ejemplo se utiliza en las discusiones posteriores.

Ejemplo 4.3 La relación EMP(ENO, ENAME, TITLE) se puede dividir en tres fragmentos horizontales EMP1, EMP2 y EMP3, definidos de la siguiente manera:

$$\text{EMP1} = \sigma_{\text{ENO} \leq "E3"}(\text{EMP})$$

$$\text{EMP2} = \sigma_{"E3" < \text{ENO} \leq "E6"}(\text{EMP})$$

$$\text{EMP3} = \sigma_{\text{ENO} > "E6"}(\text{EMP})$$

El programa de materialización para una relación fragmentada horizontalmente es la unión de los fragmentos. En nuestro ejemplo, tenemos

$$\text{EMP} = \text{EMP1} \cup \text{EMP2} \cup \text{EMP3}$$

De esta forma, la forma materializada de cualquier consulta especificada en EMP se obtiene mediante reemplazándolo por (EMP1  $\cup$  EMP2  $\cup$  EMP3).

La reducción de consultas sobre relaciones fragmentadas horizontalmente consiste principalmente en determinar, tras la reestructuración de los subárboles, aquellas que generarán relaciones vacías y eliminarlas. La fragmentación horizontal puede aprovecharse para simplificar los operadores de selección y unión.

### 4.2.1.1 Reducción con selección

Las selecciones en fragmentos con una calificación que contradice la calificación de la regla de fragmentación generan relaciones vacías. Dada una relación R fragmentada horizontalmente como R1, R2, ..., R<sub>w</sub>, donde R<sub>j</sub> = σ<sub>pj</sub>(R), la regla puede enunciarse formalmente de la siguiente manera:

$$\text{Regla 1 } \sigma_{pi}(R_j) = \emptyset \text{ si } \exists x \in R : \neg(pi(x) \wedge pj(x)) \text{ donde } pi \text{ y } pj$$

son predicados de selección, x denota una tupla y p(x) denota "el predicado p se cumple para x".

Por ejemplo, el predicado de selección ENO="E1" entra en conflicto con los predicados de los fragmentos EMP2 y EMP3 del Ejemplo 4.3 (es decir, ninguna tupla en EMP2 y EMP3 puede satisfacer este predicado). Determinar los predicados contradictorios requiere el teorema...

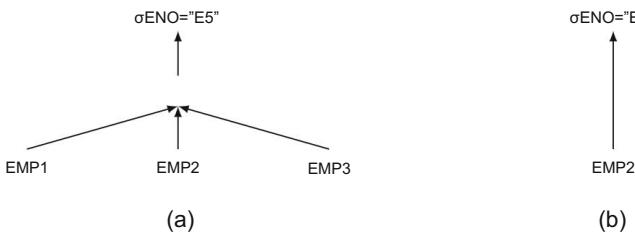


Fig. 4.3 Reducción para fragmentación horizontal (con selección). (a) Consulta de fragmentos. (b) Consulta reducida.

Técnicas de comprobación si los predicados son bastante generales. Sin embargo, los SGBD suelen simplificar la comparación de predicados al admitir únicamente predicados simples para definir reglas de fragmentación (por parte del administrador de la base de datos).

Ejemplo 4.4 Ahora ilustramos la reducción por fragmentación horizontal utilizando la siguiente consulta de ejemplo:

```
SELECCIONAR *
DESDE EMP
DONDE ENO = "E5"
```

La aplicación del enfoque ingenuo para localizar EMP usando EMP1, EMP2 y EMP3 da como resultado la consulta de fragmento de la Fig. 4.3a. Al conmutar la selección con el operador de unión, es fácil detectar que el predicado de selección contradice los predicados de EMP1 y EMP3, lo que produce relaciones vacías. La consulta reducida se aplica simplemente a EMP2, como se muestra en la Fig. 4.3b.

## 4.2.2 Reducción con unión

Las uniones en relaciones fragmentadas horizontalmente se pueden simplificar fragmentando las relaciones unidas según el atributo de unión. La simplificación consiste en distribuir las uniones entre las uniones y eliminar las uniones innecesarias. La distribución de las uniones entre las uniones se puede expresar como:

$$(R1 \cup R2) S = (R1 S) \cup (R2 S)$$

donde  $R_i$  son fragmentos de  $R$  y  $S$  es una relación.

Con esta transformación, las uniones pueden desplazarse hacia arriba en el operador trie para que se muestren todas las posibles uniones de fragmentos. Se pueden determinar uniones de fragmentos inútiles cuando las calificaciones de los fragmentos unidos son contradictorias, lo que produce un resultado vacío. Suponiendo que los fragmentos  $R_i$  y  $R_j$  están definidos, respectivamente...

tivamente, según los predicados  $\pi_i$  y  $\pi_j$  sobre el mismo atributo, la regla de simplificación puede enunciarse de la siguiente manera:

$$\text{Regla 2 } R_i \cap R_j = \emptyset \text{ si } x \in R_i, y \in R_j : \neg(\pi_i(x) \wedge \pi_j(y))$$

La determinación de uniones inútiles y su eliminación mediante la regla 2 puede, por lo tanto, realizarse observando únicamente los predicados de fragmentos. La aplicación de esta regla permite implementar la unión de dos relaciones como uniones parciales paralelas de fragmentos. No siempre se da el caso de que la consulta reducida sea mejor (es decir, más simple) que la consulta de fragmentos. La consulta de fragmentos es mejor cuando hay un gran número de uniones parciales en la consulta reducida. Este caso surge cuando hay pocos predicados de fragmentación contradictorios. El peor caso ocurre cuando cada fragmento de una relación debe unirse con cada fragmento de la otra relación. Esto equivale al producto cartesiano de los dos conjuntos de fragmentos, donde cada conjunto corresponde a una relación. La consulta reducida es mejor cuando el número de uniones parciales es pequeño. Por ejemplo, si ambas relaciones se fragmentan utilizando los mismos predicados, el número de uniones parciales es igual al número de fragmentos de cada relación. Una ventaja de la consulta reducida es que las uniones parciales pueden realizarse en paralelo y, por lo tanto, aumentar el tiempo de respuesta.

Ejemplo 4.5 Supongamos que la relación EMP está fragmentada como EMP1, EMP2, EMP3, como se indicó anteriormente, y que la relación ASG está fragmentada como

$$ASG1 = \sigma_{ENO \leq "E3"}(ASG)$$

$$ASG2 = \sigma_{ENO > "E3"}(ASG)$$

EMP1 y ASG1 se definen mediante el mismo predicado. Además, el predicado que define ASG2 es la unión de los predicados que definen EMP2 y EMP3. Consideraremos ahora la consulta de unión.

SELECCIONAR \*  
DESDE EMP NATURAL ÚNETE A ASG

La consulta de fragmento equivalente se muestra en la Fig. 4.4a. La consulta, reducida mediante la distribución de uniones entre uniones y la aplicación de la regla 2, puede implementarse como una unión de tres uniones parciales que pueden ejecutarse en paralelo (Fig. 4.4b).

### 4.2.3 Reducción por fragmentación vertical

La función de fragmentación vertical distribuye una relación basándose en atributos de proyección. Dado que el operador de reconstrucción para la fragmentación vertical es la unión, el programa de materialización para una relación fragmentada verticalmente consiste en la unión de los fragmentos en el atributo común. Para la fragmentación vertical, utilizamos el siguiente ejemplo.

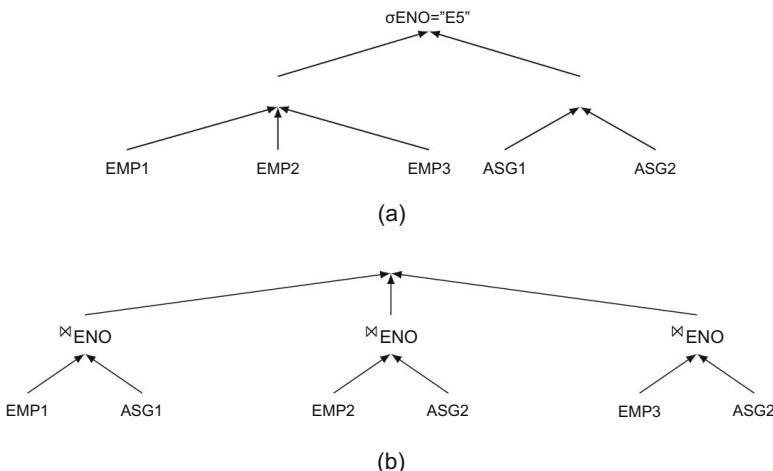


Fig. 4.4 Reducción por fragmentación horizontal (con unión). (a) Consulta de fragmento. (b) Consulta reducida.

Ejemplo 4.6 La relación EMP se puede dividir en dos fragmentos verticales donde el atributo clave ENO está duplicado:

$$\text{EMP1} = \text{ENO}, \text{ENAME}(\text{EMP})$$

$$\text{EMP2} = \text{ENO}, \text{TÍTULO}(\text{EMP})$$

El programa de materialización es

$$\text{EMP} = \text{EMP1 ENO EMP2}$$

De forma similar a la fragmentación horizontal, las consultas sobre fragmentos verticales pueden reducirse determinando las relaciones intermedias inútiles y eliminando los subárboles que las generan. Las proyecciones sobre un fragmento vertical que no comparte atributos con los de la proyección (excepto la clave de la relación) producen relaciones inútiles, aunque no vacías. Dada una relación R, definida sobre los atributos  $A = \{A_1, \dots, A_n\}$ , que está fragmentada verticalmente como  $R_i = A_i(R)$ , donde  $A_i \subseteq A$ , la regla puede enunciarse formalmente de la siguiente manera:

Regla 3  $D, K(R_i)$  es inútil si el conjunto de atributos de proyección D no está en  $A_i$

Ejemplo 4.7 Ilustremos la aplicación de esta regla utilizando la siguiente consulta de ejemplo en SQL:

```
SELECCIONAR NOMBRE
DESDE EMP
```

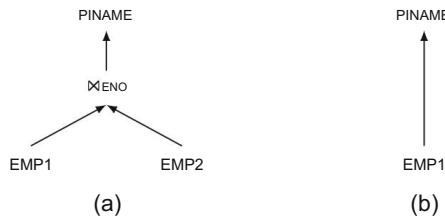


Fig. 4.5 Reducción por fragmentación vertical. (a) Consulta de fragmentos. (b) Consulta reducida.

La consulta de fragmento equivalente en EMP1 y EMP2 (obtenida en el Ejemplo 4.4) se muestra en la Fig. 4.5a. Al conmutar la proyección con la unión (es decir, proyectando en ENO, ENAME), podemos ver que la proyección en EMP2 es inútil porque ENAME no está en EMP2. Por lo tanto, la proyección solo debe aplicarse a EMP1, como se muestra en la Fig. 4.5b.

#### 4.2.4 Reducción por fragmentación derivada

Como vimos en secciones anteriores, el operador de unión, probablemente el más importante por su frecuencia y alto coste, puede optimizarse mediante la fragmentación horizontal primaria cuando las relaciones unidas se fragmentan según los atributos de unión. En este caso, la unión de dos relaciones se implementa como una unión de uniones parciales. Sin embargo, este método impide que una de las relaciones se fragmente según un atributo diferente utilizado para la selección. La fragmentación horizontal derivada es otra forma de distribuir dos relaciones, mejorando así el procesamiento conjunto de la selección y la unión. Normalmente, si la relación R está sujeta a fragmentación horizontal derivada debido a la relación S, los fragmentos de R y S que comparten los mismos valores de atributo de unión se ubican en el mismo sitio. Además, S puede fragmentarse según un predicado de selección.

Dado que las tuplas de R se colocan según las tuplas de S, la fragmentación derivada solo debe usarse para relaciones uno a muchos (jerárquicas) de la forma  $S \rightarrow R$ , donde una tupla de S puede coincidir con n tuplas de R, pero una tupla de R coincide con exactamente una tupla de S. Cabe destacar que la fragmentación derivada podría usarse para relaciones muchos a muchos, siempre que las tuplas de S (que coincidan con n tuplas de R) se repliquen. Para simplificar, asumimos y recomendamos que la fragmentación derivada se use solo para relaciones jerárquicas.

Ejemplo 4.8 Dada una relación de uno a muchos de EMP a ASG, la relación ASG(ENO, PNO, RESP, DUR) se puede fragmentar indirectamente de acuerdo con las siguientes reglas:

$$ASG1 = ASG \text{ ENO EMP1}$$

$$ASG2 = ASG \text{ ENO EMP2}$$

Recordemos del Cap. 2 que EMP1 y EMP2 están fragmentados de la siguiente manera:

$$\text{EMP1} = \sigma_{\text{TÍTULO}=\text{"Programador"}(\text{EMP})}$$

$$\text{EMP2} = \sigma_{\text{TÍTULO}=\text{"Programador"}(\text{EMP})}$$

El programa de materialización para una relación fragmentada horizontalmente es la unión de los fragmentos. En nuestro ejemplo, tenemos

$$\text{ASG} = \text{ASG1} \cup \text{ASG2}$$

También se pueden reducir las consultas sobre fragmentos derivados. Dado que este tipo de fragmentación es útil para optimizar las consultas de unión, una transformación útil consiste en distribuir las uniones entre las uniones (usadas en los programas de materialización) y aplicar la regla 2 presentada anteriormente. Dado que las reglas de fragmentación indican cuáles son las tuplas coincidentes, ciertas uniones producirán relaciones vacías si los predicados de fragmentación entran en conflicto. Por ejemplo, los predicados de ASG1 y EMP2 entran en conflicto; por lo tanto, tenemos

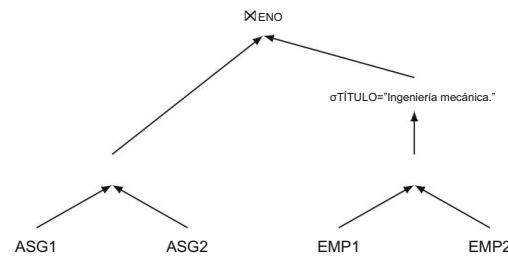
$$\text{ASG1 EMP2} = \emptyset$$

A diferencia de la reducción con unión discutida anteriormente, la consulta reducida siempre es preferible a la consulta de fragmentos porque el número de uniones parciales generalmente es igual al número de fragmentos de R.

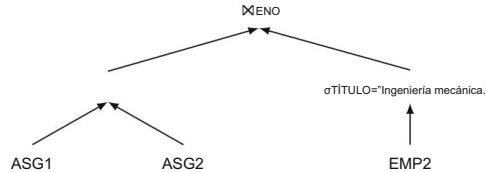
Ejemplo 4.9 La reducción por fragmentación derivada se ilustra aplicándola a la siguiente consulta SQL, que recupera todos los atributos de las tuplas de EMP y ASG que tienen el mismo valor de ENO y el título "Mech. Eng.":

```
SELECCIONAR *
DESDE EMP NATURAL ÚNETE A ASG
DONDE TÍTULO = "Ingeniería mecánica"
```

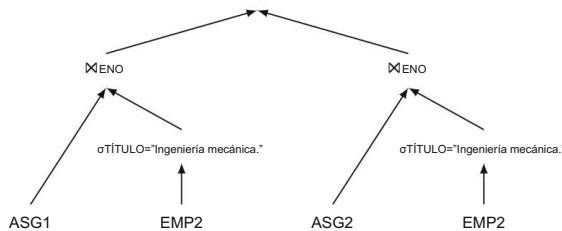
La consulta de fragmentos sobre los fragmentos EMP1, EMP2, ASG1 y ASG2 , definida previamente, se muestra en la Fig. 4.6a. Al reducir la selección a los fragmentos EMP1 y EMP2, la consulta se reduce a la de la Fig. 4.6b. Esto se debe a que el predicado de selección entra en conflicto con el de EMP1, por lo que EMP1 puede eliminarse. Para descubrir predicados de unión conflictivos, distribuimos las uniones entre las uniones. Esto produce el trie de la Fig. 4.6c. El subárbol izquierdo une dos fragmentos, ASG1 y EMP2, cuyas calificaciones entran en conflicto debido a los predicados TITLE = "Programmer" en ASG1 y TITLE = "Programmer" en EMP2. Por lo tanto, el subárbol izquierdo que produce una relación vacía puede eliminarse, y se obtiene la consulta reducida de la Fig. 4.6d . La consulta resultante se simplifica, lo que ilustra el valor de la fragmentación para mejorar el rendimiento de las consultas distribuidas.



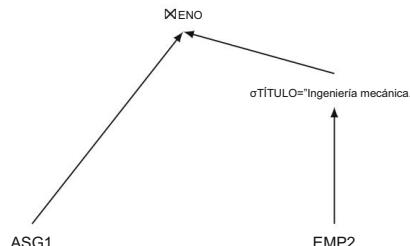
(a)



(b)



(c)



(d)

Fig. 4.6 Reducción por fragmentación indirecta. (a) Consulta de fragmentos. (b) Consulta tras desplazar la selección hacia abajo. (c) Consulta tras desplazar las uniones hacia arriba. (d) Consulta reducida tras eliminar el subárbol izquierdo.

## 4.2.5 Reducción por fragmentación híbrida

La fragmentación híbrida se obtiene combinando las funciones de fragmentación descritas anteriormente. El objetivo de la fragmentación híbrida es admitir, de forma eficiente, consultas que involucran proyección, selección y unión. Cabe destacar que la optimización de un operador o de una combinación de operadores siempre se realiza a expensas de otros operadores. Por ejemplo, la fragmentación híbrida basada en selección-proyección hará que la selección o la proyección sean menos eficientes que la fragmentación horizontal (o vertical). El programa de materialización para una relación fragmentada híbrida utiliza uniones y uniones de fragmentos.

Ejemplo 4.10 A continuación se muestra un ejemplo de fragmentación híbrida de la relación EMP:

$$\text{EMP1} = \sigma_{\text{ENO} \leq "E4"}(\text{ENO}, \text{ENOMBRE}(\text{EMP}))$$

$$\text{EMP2} = \sigma_{\text{ENO} > "E4"}(\text{ENO}, \text{ENOMBRE}(\text{EMP}))$$

$$\text{EMP3} = \text{ENO}, \text{TÍTULO}(\text{EMP})$$

En nuestro ejemplo, el programa de materialización es

$$\text{EMP} = (\text{EMP1} \quad \text{EMP2}) \text{ ENO } \text{EMP3}$$

Las consultas sobre fragmentos híbridos se pueden reducir combinando las reglas utilizadas, respectivamente, en la fragmentación horizontal primaria, vertical y horizontal derivada. Estas reglas se pueden resumir de la siguiente manera:

1. Eliminar relaciones vacías generadas por selecciones contradictorias en horizontal fragmentos.
2. Eliminar las relaciones inútiles generadas por proyecciones sobre fragmentos verticales.
3. Distribuya las uniones entre sí para aislar y eliminar las uniones inútiles.

Ejemplo 4.11 La siguiente consulta de ejemplo en SQL ilustra la aplicación de las reglas (1) y (2) a la fragmentación horizontal-vertical de la relación EMP en EMP1, EMP2 y EMP3 dadas anteriormente:

```
SELECCIONAR NOMBRE
DESDE EMP
DONDE ENO="E5"
```

La consulta de fragmentos de la Fig. 4.7a puede reducirse reduciendo primero la selección, eliminando el fragmento EMP1, y luego reduciendo la proyección, eliminando el fragmento EMP3. La consulta reducida se muestra en la Fig. 4.7b.

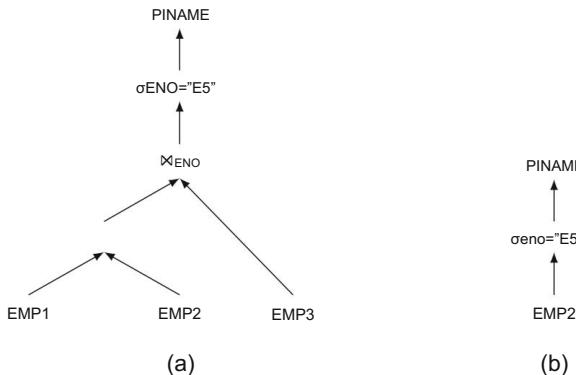


Fig. 4.7 Reducción para fragmentación híbrida. (a) Consulta de fragmento. (b) Consulta reducida.

### 4.3 Ordenamiento de uniones en consultas distribuidas

Ordenar las uniones es un aspecto importante de la optimización de consultas centralizadas. En un contexto distribuido, ordenar las uniones es aún más importante, ya que las uniones entre fragmentos pueden aumentar el tiempo de comunicación. Por lo tanto, el espacio de búsqueda investigado por un optimizador de consultas distribuidas se centra en los árboles de unión (véase la siguiente sección). Existen dos enfoques básicos para ordenar las uniones en consultas distribuidas. Uno intenta optimizar el orden de las uniones directamente, mientras que el otro las sustituye por combinaciones de semiuniones para minimizar los costes de comunicación.

#### 4.3.1 Unir árboles

Los QEP se suelen abstraer mediante árboles de operadores, que definen el orden de ejecución de los operadores. Se enriquecen con información adicional, como el mejor algoritmo elegido para cada operador. Para una consulta dada, el espacio de búsqueda puede definirse como el conjunto de árboles de operadores equivalentes que pueden generarse mediante reglas de transformación. Para caracterizar los optimizadores de consultas, conviene centrarse en los árboles de unión, cuyos operadores son de unión o producto cartesiano. Esto se debe a que las permutaciones del orden de unión tienen el mayor impacto en el rendimiento de las consultas relacionales.

Ejemplo 4.12 Considere la siguiente consulta:

```
SELECCIONAR NOMBRE, RESP
DESDE EMP NATURAL ÚNETE A ASG NATURAL ÚNETE A PROJ
```

La Figura 4.8 ilustra tres árboles de unión equivalentes para esa consulta, que se obtienen aprovechando la asociatividad de los operadores binarios. Cada uno de estos árboles de unión puede ser

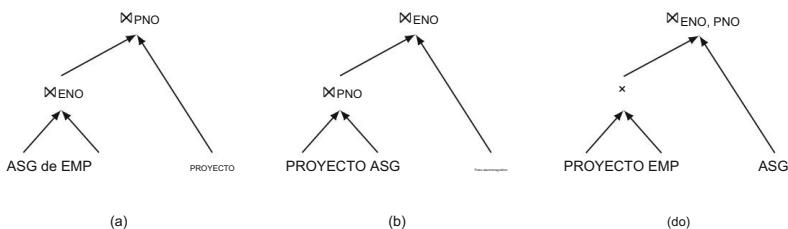


Fig. 4.8 Árboles de unión equivalentes

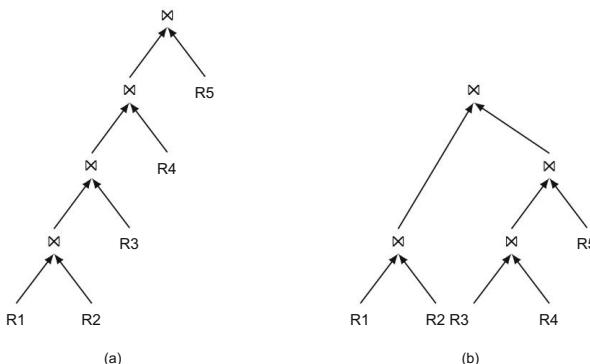


Fig. 4.9 Las dos formas principales de árboles de unión. (a) Árbol de unión lineal. (b) Árbol de unión arbustivo.

Se le asigna un costo basado en el costo estimado de cada operador. El trie de unión (c), que comienza con un producto cartesiano, puede tener un costo mucho mayor que los demás árboles de unión.

Para una consulta compleja (que involucra muchas relaciones y operadores), el número de árboles de operadores equivalentes puede ser muy elevado. Por ejemplo, el número de árboles de unión alternativos que se pueden generar aplicando las reglas de commutatividad y asociatividad es  $O(N!)$  para  $N$  relaciones. Investigar un espacio de búsqueda amplio puede hacer que el tiempo de optimización sea prohibitivo, a veces mucho más costoso que el tiempo de ejecución real.

Por lo tanto, los optimizadores de consultas suelen restringir el tamaño del espacio de búsqueda que consideran. La primera restricción es el uso de heurísticas. La heurística más común consiste en realizar selección y proyección al acceder a las relaciones base. Otra heurística común consiste en evitar productos cartesianos innecesarios para la consulta.

Por ejemplo, en la Fig. 4.8, el operador trie (c) no sería parte del espacio de búsqueda considerado por el optimizador.

Otra restricción importante se refiere a la forma del árbol de unión. Se suelen distinguir dos tipos de árboles de unión: lineales y arbustivos (véase la figura 4.9).

Un trie lineal es un trie tal que al menos un operando de cada nodo de operador es una relación base.

Un trie lineal izquierdo es un trie lineal donde el subárbol derecho de un nodo de unión siempre es un nodo hoja correspondiente a una relación base. Un trie arbustivo es más general y puede tener operadores sin relaciones base como operandos (es decir, ambos operandos son relaciones intermedias). Al considerar solo árboles lineales, el tamaño del espacio de búsqueda

se reduce a  $O(2N)$ . Sin embargo, en un entorno distribuido, los árboles frondosos son útiles para mostrar paralelismo. Por ejemplo, en el trie de unión (b) de la Fig. 4.9, los operadores R1, R2 y R3, R4 pueden ejecutarse en paralelo.

### 4.3.2 Unirse a Pedidos

Algunos algoritmos optimizan el orden de las uniones directamente sin utilizar semiuniones. El propósito de esta sección es enfatizar la dificultad que presenta el ordenamiento de uniones y motivar la sección posterior, que trata el uso de semiuniones para optimizar las consultas de unión.

Es necesario partir de una serie de supuestos para poder centrarse en las cuestiones principales. Dado que la consulta se expresa en fragmentos, no es necesario distinguir entre fragmentos de la misma relación y fragmentos de relaciones diferentes. Para simplificar la notación, utilizamos el término «relación» para designar un fragmento almacenado en un sitio específico. Además, para centrarnos en la ordenación de las uniones, ignoramos el tiempo de procesamiento local, asumiendo que los reductores (selección, proyección) se ejecutan localmente antes o durante la unión (recuerde que realizar la selección primero no siempre es eficiente). Por lo tanto, solo consideraremos consultas de unión cuyas relaciones de operandos se almacenan en sitios diferentes. Suponemos que las transferencias de relaciones se realizan en modo "conjunto a la vez" en lugar de "tupla a la vez". Finalmente, ignoramos el tiempo de transferencia para generar los datos en un sitio de resultados.

Centrémonos primero en el problema más sencillo de la transferencia de operandos en una sola unión. La consulta es RS, donde R y S son relaciones almacenadas en sitios diferentes. La elección obvia de la relación a transferir es enviar la relación más pequeña al sitio de la más grande, lo que da lugar a dos posibilidades, como se muestra en la figura 4.10. Para realizar esta elección, necesitamos evaluar los tamaños de R y S (suponemos que existe una función `size()`). Consideremos ahora el caso en el que hay más de dos relaciones para unir. Al igual que en el caso de una sola unión, el objetivo del algoritmo de ordenación de uniones es transmitir operandos más pequeños. La dificultad radica en que los operadores de unión pueden reducir o aumentar el tamaño de los resultados intermedios. Por lo tanto, estimar el tamaño de los resultados de la unión es obligatorio, pero también difícil. Una solución consiste en estimar los costes de comunicación de todas las estrategias alternativas y elegir la mejor. Sin embargo, como se mencionó anteriormente, el número de estrategias crece rápidamente con el número de relaciones. Este enfoque encarece la optimización, aunque esta sobrecarga se amortiza rápidamente si la consulta se ejecuta con frecuencia.

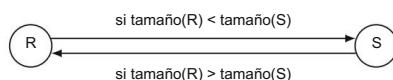


Fig. 4.10 Transferencia de operandos en operador binario

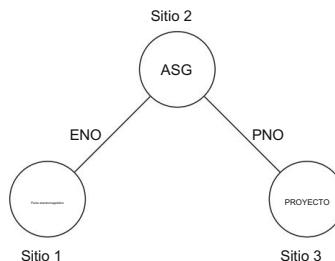


Fig. 4.11 Gráfico de unión de consulta distribuida

Ejemplo 4.13 Considere la siguiente consulta expresada en álgebra relacional:

PROYECTO PNO ASG ENO EMP

Esta consulta se puede representar mediante su grafo de unión en la Fig. 4.11. Nótese que hemos hecho ciertas suposiciones sobre las ubicaciones de las tres relaciones. Esta consulta puede ejecutarse de al menos cinco maneras diferentes. Describimos estas estrategias mediante los siguientes programas, donde  $(R \rightarrow \text{sitio } j)$  significa que la relación  $R$  se transfiere al sitio  $j$ .

1.  $\text{EMP} \rightarrow \text{sitio 2}; \text{el}$   
sitio 2 calcula  $\text{EMP} = \text{EMP ASG}; \text{EMP} \rightarrow \text{sitio}$   
3; el sitio 3 calcula  
 $\text{EMP PROJ}.$
2.  $\text{ASG} \rightarrow \text{sitio 1}; \text{el}$   
sitio 1 calcula  $\text{EMP} = \text{EMP ASG}; \text{EMP} \rightarrow \text{sitio}$   
3; el sitio 3 calcula  
 $\text{EMP PROJ}.$
3.  $\text{ASG} \rightarrow \text{sitio 3}; \text{el}$   
sitio 3 calcula  $\text{ASG} = \text{ASG PROJ}; \text{ASG} \rightarrow \text{sitio}$   
1; el sitio 1 calcula  
 $\text{ASG EMP}.$
4.  $\text{PROY} \rightarrow \text{sitio 2};$   
El sitio 2 calcula  $\text{PROJ} = \text{PROJ ASG}; \text{PROJ} \rightarrow$   
sitio 1; El sitio 1  
calcula  $\text{PROJ EMP}.$
5.  $\text{EMP} \rightarrow \text{sitio 2};$   
 $\text{PROJ} \rightarrow \text{sitio 2}; \text{El}$   
sitio 2 calcula  $\text{EMP PROJ ASG}$

Para seleccionar uno de estos programas, se deben conocer o predecir los siguientes tamaños: tamaño( $\text{EMP}$ ), tamaño( $\text{ASG}$ ), tamaño( $\text{PROJ}$ ), tamaño( $\text{EMP ASG}$ ) y tamaño( $\text{ASG PROJ}$ ).

Además, si lo que se considera es el tiempo de respuesta, la optimización debe tener en cuenta el hecho de que las transferencias se pueden realizar en paralelo con la estrategia 5.

Una alternativa a enumerar todas las soluciones es utilizar heurísticas que consideren sólo

los tamaños de las relaciones de operandos asumiendo, por ejemplo, que la cardinalidad de La unión resultante es el producto de las cardinalidades de los operandos. En este caso, las relaciones son ordenados por tamaños crecientes y el orden de ejecución está dado por este ordenamiento y El gráfico de unión. Por ejemplo, la orden (EMP, ASG, PROJ) podría usar la estrategia 1, mientras que La orden (PROJ, ASG, EMP) podría utilizar la estrategia 4.

#### 4.3.3 Algoritmos basados en semiuniones

El operador de semiunión tiene la importante propiedad de reducir el tamaño del operando. relación. Cuando el principal componente de costo considerado por el procesador de consultas es En la comunicación, una semiunión es particularmente útil para mejorar el procesamiento de operadores de unión distribuidos, ya que reducen el tamaño de los datos intercambiados entre sitios. Sin embargo, el uso de semiuniones puede resultar en un aumento en el número de mensajes. y en el tiempo de procesamiento local. Los primeros SGBD distribuidos, como SDD-1, que fueron diseñados para redes de área amplia lenta, hacen un uso extensivo de semiuniones. Sin embargo, las semiuniones siguen siendo beneficiosas en el contexto de redes rápidas cuando Inducir una fuerte reducción del operando de unión. Por lo tanto, algunos algoritmos buscan seleccionar una combinación óptima de uniones y semiuniones.

En esta sección, mostramos cómo se puede utilizar el operador de semiunión para disminuir la Tiempo total de las consultas de unión. Se parte de las mismas premisas que en la sección 4.3.2. La principal deficiencia del enfoque de unión descrito en la sección anterior es que todas las relaciones de operandos deben transferirse entre sitios. La semiunión actúa como un reductor de tamaño para una relación, tal como lo hace una selección.

La unión de dos relaciones R y S sobre el atributo A, almacenadas en los sitios 1 y 2, respectivamente, se pueden calcular reemplazando una o ambas relaciones de operandos por una semijoin con la otra relación, utilizando las siguientes reglas:

$$\begin{array}{ccc} R & \text{A} & S \\ & \diagdown & \diagup \\ & (R \Delta S) A & \\ & \diagup & \diagdown \\ R & & A (S A R) \\ & \diagdown & \diagup \\ & (R A S) A (S A R) & \end{array}$$

La elección entre una de las tres estrategias de semiunión requiere estimar su costos respectivos.

El uso de la semiunión es beneficioso si el costo de producirlo y enviarlo a la El otro sitio es menor que el costo de enviar toda la relación de operandos y de hacerlo La unión real. Para ilustrar el beneficio potencial de la semiunión, comparemos los costos de las dos alternativas: RS versus  $(R \Delta S) A S$ , suponiendo que tamaño(R) < tamaño(S).

El siguiente programa, que utiliza la notación de la Secc. 4.3.2, utiliza la semiunión operador:

1.  $A(S) \rightarrow$  sitio 1

2. El sitio 1 calcula  $R = R$

$\Delta$  S

3.  $R \rightarrow$  sitio 2 4.

El sitio 2 calcula R

A S

Para simplificar, ignoremos la constante TMSG en el tiempo de comunicación, suponiendo que el término  $TT_R \cdot size(R)$  es mucho mayor. Podemos entonces comparar las dos alternativas en términos del tamaño de los datos transmitidos. El costo del algoritmo basado en unión es el de transferir la relación R al sitio 2. El costo del algoritmo basado en semiunión es el costo de los pasos 1 y 3 anteriores. Por lo tanto, el enfoque de semiunión es mejor si

$$tamaño(A(S)) + tamaño(R A S) < tamaño(R)$$

El enfoque de semiunión es mejor si actúa como un reductor suficiente, es decir, si unas pocas tuplas de R participan en la unión. El enfoque de unión es mejor si casi todas las tuplas de R participan en la unión, ya que requiere la transferencia adicional de una proyección sobre el atributo de unión. El coste del paso de proyección se puede minimizar codificando el resultado de la proyección en matrices de bits, lo que reduce el coste de transferir los valores de los atributos unidos. Es importante destacar que ninguno de los dos enfoques es sistemáticamente el mejor; deben considerarse complementarios.

En términos más generales, la semiunión puede ser útil para reducir el tamaño de las relaciones de operandos involucradas en consultas de unión múltiple. Sin embargo, la optimización de consultas se vuelve más compleja en estos casos. Considere nuevamente el grafo de unión de las relaciones EMP, ASG y PROJ, mostrado en la Fig. 4.11. Podemos aplicar el algoritmo de unión anterior usando semiuniones a cada unión individual. Por lo tanto, un ejemplo de un programa para calcular EMP ASG PROJ es EMP ASG PROJ, donde  $EMP = EMP ASG$  y  $ASG = ASG PROJ$ .

Sin embargo, podemos reducir aún más el tamaño de una relación de operandos utilizando más de una semiunión. Por ejemplo, EMP puede reemplazarse en el programa anterior por EMP derivado como

$$EMP = EMP (PROYECCIÓN ASG)$$

Dado que si  $tamaño(ASG PROJ) \leq tamaño(ASG)$ , tenemos  $tamaño(EMP) \leq tamaño(EMP)$ . De esta manera, EMP puede reducirse mediante la secuencia de semiuniones:  $EMP(ASG PROJ)$ .

Esta secuencia de semiuniones se denomina programa de semiunión para EMP. De igual forma, se pueden encontrar programas de semiunión para cualquier relación en una consulta. Por ejemplo, PROJ podría reducirse mediante el programa de semiunión PROJ (ASG EMP). Sin embargo, no es necesario reducir todas las relaciones involucradas en una consulta; en particular, podemos ignorar aquellas relaciones que no participan en las uniones finales.

Para una relación dada, existen varios programas de semiunión potenciales. De hecho, el número de posibilidades es exponencial en relación con el número de relaciones. Pero existe un programa de semiunión óptimo, llamado reductor completo, que reduce cada relación R más que las demás. El problema es encontrar el reductor completo. Un método sencillo es...

Evaluar la reducción de tamaño de todos los programas semijoin posibles y seleccionar el mejor. El método enumerativo presenta dos problemas:

1. Existe una clase de consultas, llamadas consultas cíclicas, que tienen ciclos en su gráfico de unión y para las cuales no se pueden encontrar reductores completos.
2. Para otras consultas, llamadas consultas de árbol, existen reductores completos, pero la cantidad de programas semijoin candidatos es exponencial en la cantidad de relaciones, lo que hace que el enfoque enumerativo sea NP-difícil.

A continuación analizamos soluciones a estos problemas.

Ejemplo 4.14 Considere las siguientes relaciones, donde el atributo CITY se ha añadido a las relaciones EMP (renombrada ET), PROJ (renombrada PT) y ASG (renombrada AT) de la base de datos de ingeniería. El atributo CITY de AT corresponde a la ciudad donde reside el empleado identificado por ENO.

```
ET(ENO, NOMBRE, TÍTULO, CIUDAD)
EN(ENO, PNO, RESP, DUR, CIUDAD)
PT(PNO, PNAME, PRESUPUESTO, CIUDAD)
```

La siguiente consulta SQL recupera los nombres de todos los empleados que viven en la ciudad en la que se encuentra su proyecto junto con el nombre del proyecto.

```
SELECCIONAR ENAME, PNAME
DESDE ET NATURAL ÚNETE A NATURAL ÚNETE PT
UNIÓN NATURAL ET
```

Como se ilustra en la figura 4.12a, esta consulta es cíclica.

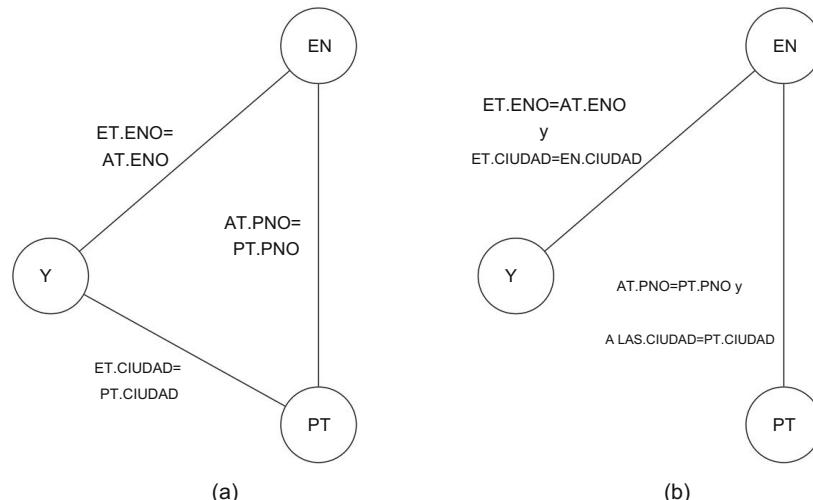


Fig. 4.12 Transformación de una consulta cíclica. (a) Consulta cíclica. (b) Consulta acíclica equivalente.

No existe un reductor completo para la consulta del Ejemplo 4.14. De hecho, es posible derivar programas de semiunión para reducirla, pero el número de operadores se multiplica por el número de tuplas en cada relación, lo que hace que el enfoque sea ineficiente. Una solución consiste en transformar el grafo cíclico en un trie eliminando un arco del grafo y añadiendo predicados apropiados a los demás arcos, de modo que el predicado eliminado se conserve por transitividad. En el ejemplo de la Fig. 4.12b, donde se elimina el arco (ET, PT), los predicados adicionales ET.CITY = AT.CITY y AT.CITY = PT.CITY implican ET.CITY = PT.CITY por transitividad. Por lo tanto, la consulta acíclica es equivalente a la consulta cíclica.

Aunque existen reductores completos para consultas trie, encontrarlos es un problema NP-hard. Sin embargo, existe una clase importante de consultas, llamadas consultas encadenadas, para las cuales existe un algoritmo polinomial. Una consulta encadenada tiene un grafo de uniones donde las relaciones se pueden ordenar, y cada relación se une solo con la siguiente en el orden. Además, el resultado de la consulta se encuentra al final de la cadena. Por ejemplo, la consulta de la figura 4.11 es una consulta en cadena. Debido a la dificultad de implementar un algoritmo con reductores completos, la mayoría de los sistemas utilizan semiuniones simples para reducir el tamaño de la relación.

#### 4.3.4 Unión versus semiunión

En comparación con la unión, la semiunión induce más operadores, pero posiblemente en operandos más pequeños. La Figura 4.13 ilustra estas diferencias con un par equivalente de estrategias de unión y semiunión para la consulta cuyo grafo de unión se muestra en la Figura 4.11. La unión EMP ASG se realiza enviando una relación, por ejemplo, ASG, al sitio de la otra, por ejemplo, EMP, para completar la unión localmente. Sin embargo, cuando se utiliza una semiunión,

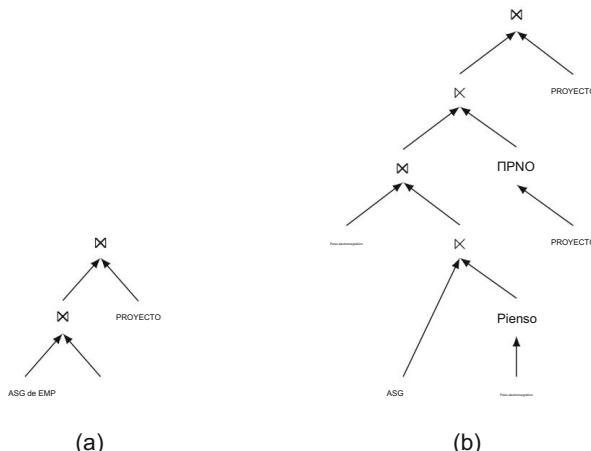


Fig. 4.13 Enfoques de unión versus semiunión. (a) Enfoque de unión. (b) Enfoque de semiunión

Se evita la transferencia de la relación ASG. En su lugar, se transfieren los valores del atributo de unión de la relación EMP al sitio de la relación ASG, seguido de la transferencia de las tuplas coincidentes de la relación ASG al sitio de la relación EMP, donde se completa la unión. Si la semiunión presenta una buena selectividad, este enfoque puede generar un ahorro significativo en el tiempo de comunicación. Además, puede reducir el tiempo de procesamiento local al aprovechar los índices del atributo de unión.

Consideremos de nuevo la unión EMP ASG, suponiendo que existe una selección en ASG y un índice en el atributo de unión de ASG. Sin semiunión, primero se realizaría la selección de ASG y luego se enviaría la relación resultante al sitio de EMP para completar la unión. Por lo tanto, no se puede usar el índice del atributo de unión de ASG (ya que la unión se realiza en el sitio de EMP). Con el enfoque de semiunión, tanto la selección como la semiunión ASG EMP se realizarían en el sitio de ASG y pueden realizarse eficientemente mediante índices.

Las semiuniones pueden ser beneficiosas en redes rápidas si tienen una selectividad muy alta y se implementan con matrices de bits. Una matriz de bits BA[1: n] es útil para codificar los valores de los atributos de unión presentes en una relación. Consideremos la semiunión R S. BA[i] se establece en 1 si existe un valor de atributo de unión A = val en la relación S tal que  $h(val) = i$ , donde  $h$  es una función hash. En caso contrario, BA[i] se establece en 0. Una matriz de bits de este tipo es mucho más pequeña que una lista de valores de atributos de unión.

Por lo tanto, transferir la matriz de bits en lugar de los valores de los atributos de unión al sitio de la relación R ahorra tiempo de comunicación. La semiunión se puede completar de la siguiente manera.

Cada tupla de la relación R, cuyo valor de atributo de unión es val, pertenece a la semiunión si  $BA[h(val)] = 1$ .

## 4.4 Modelo de costos distribuidos

El modelo de costos de un optimizador incluye funciones de costos para predecir el costo de los operadores, estadísticas y datos base, y fórmulas para evaluar los tamaños de los resultados intermedios.

El costo está en términos de tiempo de ejecución, por lo que una función de costo representa el tiempo de ejecución de una consulta.

### 4.4.1 Funciones de costos

El coste de una estrategia de ejecución distribuida puede expresarse en función del tiempo total o del tiempo de respuesta. El tiempo total es la suma de todos los componentes de tiempo (también denominados coste), mientras que el tiempo de respuesta es el tiempo transcurrido desde el inicio hasta la finalización de la consulta. Una fórmula general para determinar el tiempo total puede especificarse de la siguiente manera:

$$Total\_time = TCPU \quad \#insts + TI/O \quad \#I/Os + TMSG \quad \#msgs + TT R \quad \#bytes$$

Los dos primeros componentes miden el tiempo de procesamiento local, donde TCPU es el tiempo de una instrucción de CPU y TI/O es el tiempo de una E/S de disco. El tiempo de comunicación se representa mediante los dos últimos componentes. TMSG es el tiempo fijo de inicio y recepción de un mensaje, mientras que TT R es el tiempo que tarda en transmitirse una unidad de datos de un sitio a otro. La unidad de datos se expresa en bytes (#bytes es la suma del tamaño de todos los mensajes), pero podría expresarse en unidades diferentes (por ejemplo, paquetes). Se suele suponer que TT R es constante. Esto podría no ser cierto para redes de área extensa (WAN), donde algunos sitios están más alejados que otros. Sin embargo, esta suposición simplifica enormemente la optimización de consultas. Por lo tanto, se supone que el tiempo de comunicación para transferir #bytes de datos de un sitio a otro es una función lineal de #bytes:

$$CT (\#bytes) = TMSG + TT R \quad \#bytes$$

Los costos generalmente se expresan en términos de unidades de tiempo, que a su vez pueden traducirse a otras unidades (por ejemplo, dólares).

Los valores relativos de los coeficientes de coste caracterizan el entorno de bases de datos distribuidas. La topología de la red influye considerablemente en la relación entre estos componentes. En una red de área extensa como Internet, el tiempo de comunicación suele ser el factor dominante. Sin embargo, en las redes de área local, existe un mayor equilibrio entre los componentes. Por lo tanto, la mayoría de los primeros DBMS distribuidos diseñados para redes de área extensa ignoraron el coste de procesamiento local y se centraron en minimizar el coste de comunicación. Los DBMS distribuidos diseñados para redes de área local, por otro lado, consideran los tres componentes de coste. Las nuevas redes más rápidas (tanto de área extensa como de área local) han mejorado las relaciones anteriores en favor del coste de comunicación en igualdad de condiciones. Sin embargo, la comunicación sigue siendo el factor tiempo dominante en redes de área extensa como Internet debido a las mayores distancias desde las que se recuperan (o envían) los datos.

Cuando el tiempo de respuesta de la consulta es la función objetivo del optimizador, también se deben considerar el procesamiento local paralelo y las comunicaciones paralelas. Una fórmula general para el tiempo de respuesta es

$$\begin{aligned} \text{Tiempo de respuesta} = & \text{TCPU seq\_#insts} + \text{TI/O seq\_#/Os} \\ & + \text{TMSG seq\_#msgs} + \text{TT R seq\_#bytes} \end{aligned}$$

Donde seq\_x, donde x puede ser instrucciones (insts), E/S, mensajes (msgs) o bytes, es el número máximo de x que debe ejecutarse secuencialmente para la ejecución de la consulta. Por lo tanto, se ignora cualquier procesamiento y comunicación en paralelo.

Ejemplo 4.15 Ilustraremos la diferencia entre el costo total y el tiempo de respuesta usando el ejemplo de la Figura 4.14, que calcula la respuesta a una consulta en el sitio 3 con datos de los sitios 1 y 2. Para simplificar, suponemos que solo se considera el costo de comunicación.

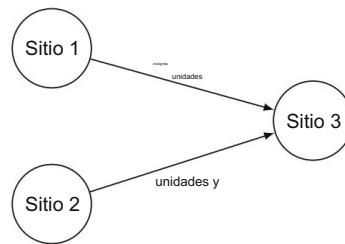


Fig. 4.14 Ejemplo de transferencias de datos para una consulta

Suponga que TMSG y TT R se expresan en unidades de tiempo. El tiempo total de transferencia de  $x$  unidades de datos del sitio 1 al sitio 3 e  $y$  unidades de datos del sitio 2 al sitio 3 es

$$\text{Tiempo\_total} = 2 \text{TMSG} + \text{TT R} \quad (x + y)$$

El tiempo de respuesta de la misma consulta se puede aproximar como

$$\text{Tiempo de respuesta} = \max\{\text{TMSG} + \text{TT R} - x, \text{TMSG} + \text{TT R} - y\}$$

ya que las transferencias se pueden realizar en paralelo.

Se minimiza el tiempo de respuesta aumentando el grado de ejecución paralela. Sin embargo, esto no implica que el tiempo total también se minimice. Al contrario, se puede aumentar el tiempo total, por ejemplo, con más procesamiento y transmisiones locales en paralelo. Minimizar el tiempo total implica una mejora en la utilización de los recursos, lo que aumenta el rendimiento del sistema. En la práctica, se busca un equilibrio entre ambos. En la sección 4.5 presentamos algoritmos que pueden optimizar la combinación del tiempo total y el tiempo de respuesta, priorizando uno de ellos.

#### 4.4.2 Estadísticas de la base de datos

El principal factor que afecta el rendimiento de una estrategia de ejecución es el tamaño de las relaciones intermedias que se generan durante la ejecución. Cuando un operador posterior se ubica en un sitio diferente, la relación intermedia debe transmitirse por la red. Por lo tanto, resulta fundamental estimar el tamaño de los resultados intermedios de los operadores de álgebra relacional para minimizar la cantidad de transferencias de datos. Esta estimación se basa en información estadística sobre las relaciones base y fórmulas para predecir las cardinalidades de los resultados de los operadores relacionales. Existe una relación directa entre la precisión de las estadísticas y el coste de su gestión: cuanto más precisas sean las estadísticas, más costosas serán. Para una relación  $R$  fragmentada como  $R1, R2, \dots, Rr$ , los datos estadísticos suelen ser los siguientes:

1. Para cada atributo A de la relación R, su longitud (en bytes), denotada por  $\text{length}(A)$ , la cardinalidad de su dominio, denotada por  $\text{card}(\text{dom}[A])$ , que da el número de valores únicos en  $\text{dom}[A]$ , y en el caso de que el dominio se defina en un conjunto de valores que se puedan ordenar (por ejemplo, enteros o reales), los valores mínimo y máximo posibles, denotados por  $\text{min}(A)$  y  $\text{max}(A)$ .
2. Para cada atributo A de cada fragmento  $R_i$ , el número de valores distintos de A, con la cardinalidad de la proyección del fragmento  $R_i$  sobre A, denotada por  $\text{card}(A(R_i))$ .
3. El número de tuplas en cada fragmento  $R_i$ , denotado por  $\text{card}(R_i)$ .

Además, para cada atributo A, puede haber un histograma que se aproxima a la distribución de frecuencia del atributo dentro de varios grupos, cada uno correspondiente a un rango de valores.

Estas estadísticas son útiles para predecir el tamaño de las relaciones intermedias. Recuerde que en el Cap. 2 definimos el tamaño de una relación intermedia R de la siguiente manera:

$$\text{tamaño}(R) = \text{tarjeta}(R) \cdot \text{longitud}(R)$$

donde  $\text{length}(R)$  es la longitud (en bytes) de una tupla de R, y  $\text{card}(R)$  es el número de tuplas en R.

La estimación de  $\text{card}(R)$  requiere el uso de fórmulas. Se suelen hacer dos suposiciones simplificadoras sobre la base de datos. Se supone que la distribución de los valores de los atributos en una relación es uniforme y que todos los atributos son independientes, lo que significa que el valor de un atributo no afecta el valor de ningún otro.

Estas dos suposiciones suelen ser erróneas en la práctica, pero hacen que el problema sea abordable. Con base en estas suposiciones, podemos usar fórmulas simples para estimar las cardinalidades de los resultados de los operadores básicos del álgebra relacional, según su selectividad. El factor de selectividad de un operador, es decir, la proporción de tuplas de una relación de operandos que participan en el resultado de esa operación, se denota por  $SF(\text{op})$ , donde op es la operación. Es un valor real entre 0 y 1. Un valor bajo (p. ej., 0,001) corresponde a una selectividad buena (o alta), mientras que un valor alto (p. ej., 0,5) a una selectividad mala (o baja).

Ilustremos esto con los dos operadores principales, es decir, selección y unión.

La cardinalidad de la selección es

$$\text{carta}(\sigma F(R)) = SF(\sigma F(R)) \cdot \text{carta}(R)$$

donde  $SF(\sigma F(R))$  se puede calcular de la siguiente manera para los predicados básicos:

$$\begin{aligned} SF(\sigma A = \text{valor}(R)) &= \frac{1}{\text{carta}(A(R)) \cdot \text{máx}(A) -} \\ SF(\sigma A > \text{valor}(R)) &= \frac{\text{valor}}{\text{máx}(A) - \text{mín}(A)} \\ SF(\sigma A < \text{valor}(R)) &= \frac{\text{valor} - \text{mín}(A)}{\text{máx}(A) - \text{mín}(A)} \end{aligned}$$

La cardinalidad de la unión es

$$\text{tarjeta}(RS) = SF(R \ A \ S) \cdot \text{tarjeta}(R) \cdot \text{tarjeta}(S)$$

No existe una forma general de estimar  $SF(R \ A \ S)$  sin información adicional.

Por lo tanto, una aproximación sencilla consiste en utilizar una constante, por ejemplo, 0,01, que refleja la selectividad de unión conocida. Sin embargo, existe un caso frecuente en el que la estimación es precisa. Si la relación R se une equitativamente con la relación S sobre el atributo A, donde A es una clave de R y una clave foránea de S, el factor de selectividad de unión puede aproximarse como

$$SF(R \ A \ S) = \frac{1}{\text{carta}(R)}$$

porque cada tupla de S coincide con como máximo una tupla de R.

## 4.5 Optimización de consultas distribuidas

En esta sección, ilustramos el uso de las técnicas presentadas en secciones anteriores en el contexto de tres algoritmos básicos de optimización de consultas. Primero, presentamos los enfoques dinámico y estático. Luego, presentamos un enfoque híbrido.

### 4.5.1 Enfoque dinámico

Ilustramos el enfoque dinámico con el algoritmo de INGRES Distribuido.

La función objetivo del algoritmo es minimizar la combinación del tiempo de comunicación y el tiempo de respuesta. Sin embargo, estos dos objetivos pueden ser contradictorios. Por ejemplo, aumentar el tiempo de comunicación (mediante paralelismo) puede reducir el tiempo de respuesta. Por lo tanto, la función puede otorgar mayor importancia a uno u otro. Cabe destacar que este algoritmo de optimización de consultas ignora el costo de transmisión de los datos al sitio de resultados. El algoritmo también aprovecha la fragmentación, pero solo maneja la fragmentación horizontal para simplificar.

Dado que se consideran tanto redes generales como de difusión, el optimizador tiene en cuenta la topología de la red. En las redes de difusión, la misma unidad de datos puede transmitirse de un sitio a todos los demás en una sola transferencia, y el algoritmo aprovecha explícitamente esta capacidad. Por ejemplo, la difusión se utiliza para replicar fragmentos y, posteriormente, para maximizar el grado de paralelismo.

La entrada del algoritmo es una consulta expresada en cálculo relacional de tuplas (en forma normal conjuntiva) e información del esquema (el tipo de red, así como la ubicación y el tamaño de cada fragmento). Este algoritmo lo ejecuta el sitio maestro, donde se inicia la consulta. El algoritmo, denominado Dynamic-QOA, se describe en el Algoritmo 4.1.

---

Algoritmo 4.1: QOA dinámico Entrada: MRQ:

consulta multirelación Salida: resultado de la última consulta multirelación begin para cada ORQi

```

separable en MRQ do run(ORQi) end para {MRQ}                                {ORQ es una consulta de monorrelación}
|   reemplazado                                                               (1)
por n
consultas irreducibles
MRQ _list ← REDUCE(MRQ) mientras n =                                         (2)
0 do {n es el número de consultas irreducibles} {elija la siguiente consulta irreducible que
    involucra los fragmentos más pequeños}
    MRQ ← SELECT_QUERY(MRQ _list) {determinar
        fragmentos a transferir y sitio de procesamiento para MRQ}
        Fragment-site-list ← SELECT_STRATEGY(MRQ ) {mover los
            fragmentos seleccionados a los sitios seleccionados} para cada
            par (F , S) en Fragment-site-list mover el fragmento F al
            |   sitio S fin para ejecutar MRQ n                                         (3.1)
← n - 1
fin mientras {la
salida es el
resultado del
último MRQ } fin                                                               (3.4)

```

---

Todas las consultas de monorrelación (p. ej., selección y proyección) que se pueden separar se procesan primero localmente (paso 1). A continuación, se aplica el algoritmo de reducción a la consulta original (paso 2). La reducción es una técnica que aísla todas las subconsultas irreducibles y las de monorrelación mediante separación. Las subconsultas de monorrelación se ignoran porque ya se procesaron en el paso 1. Por lo tanto, el procedimiento REDUCE produce una secuencia de subconsultas irreducibles  $q_1 \rightarrow q_2 \rightarrow \dots \rightarrow q_n$ , con como máximo una relación en común entre dos subconsultas consecutivas.

Con base en la lista de consultas irreducibles aisladas en el paso 2 y el tamaño de cada fragmento, la siguiente subconsulta, MRQ, que tiene al menos dos variables, se selecciona en el paso 3.1 y se le aplican los pasos 3.2 a 3.4. Los pasos 3.1 y 3.2 se describen a continuación. El paso 3.2 selecciona la mejor estrategia para procesar la consulta MRQ. Esta estrategia se describe mediante una lista de pares  $(F,S)$ , en la que  $F$  es un fragmento a transferir al sitio de procesamiento  $S$ . El paso 3.3 transfiere todos los fragmentos a sus sitios de procesamiento. Finalmente, el paso 3.4 ejecuta la consulta MRQ. Si quedan subconsultas, el algoritmo regresa al paso 3 y realiza la siguiente iteración. De lo contrario, finaliza.

La optimización se realiza en los pasos 3.1 y 3.2. El algoritmo ha generado subconsultas con varios componentes y su orden de dependencia (similar al dado por un trie de álgebra relacional). En el paso 3.1, una opción sencilla para la siguiente subconsulta es tomar la siguiente sin predecesores y que incluye los fragmentos más pequeños. Esto minimiza el tamaño de los resultados intermedios. Por ejemplo, si una consulta  $q$  tiene las subconsultas  $q_1, q_2$  y  $q_3$ , con dependencias  $q_1 \rightarrow q_3, q_2 \rightarrow q_3$ , y si los fragmentos a los que hace referencia  $q_1$  son más pequeños que los de  $q_2$ , entonces  $q_1$  es

Seleccionado. Dependiendo de la red, esta elección también puede verse afectada por el número de sitios con fragmentos relevantes.

La subconsulta seleccionada debe ejecutarse. Dado que la relación involucrada en una subconsulta puede almacenarse en diferentes sitios e incluso fragmentarse, la subconsulta puede subdividirse.

Ejemplo 4.16 Consideremos la siguiente consulta:

"Nombres de los empleados que trabajan en el proyecto CAD/CAM"

Esta consulta se puede expresar en SQL mediante la siguiente consulta q1 en la base de datos de ingeniería:

```
q1 : SELECCIONAR EMP.ENAME DE EMP
      NATURAL JOIN ASG NATURAL JOIN PROJ DONDE PNAME="CAD/CAM"
```

Supongamos que las relaciones EMP, ASG y PROJ se almacenan de la siguiente manera, donde la relación EMP está fragmentado.

Sitio 1	Sitio 2
EMP1	EMP2
PROYECTO	ASG

Existen varias estrategias posibles, entre ellas las siguientes:

1. Ejecute la consulta completa (EMP ASG PROJ) moviendo EMP1 y ASG a sitio 2.
2. Ejecute (EMP ASG) PROJ moviendo (EMP1 ASG) y ASG al sitio 2, etcétera.

La elección entre las posibles estrategias requiere una estimación del tamaño de los resultados intermedios. Por ejemplo, si tamaño(EMP ASG) > tamaño(EMP1), se prefiere la estrategia 1 a la 2. Por lo tanto, se requiere una estimación del tamaño de las uniones.

En el paso 3.2, el siguiente problema de optimización consiste en determinar cómo ejecutar la subconsulta seleccionando los fragmentos que se moverán y los sitios donde se realizará el procesamiento. Para una subconsulta de n relaciones, los fragmentos de n-1 relaciones deben moverse al sitio o sitios de los fragmentos de la relación restante, por ejemplo, Rp, y luego replicarse allí. Además, la relación restante puede dividirse en k fragmentos "igualados" para aumentar el paralelismo. Este método se denomina fragmentar y replicar y realiza una sustitución de fragmentos en lugar de tuplas. La selección de la relación restante y del número de sitios de procesamiento k en los que debe dividirse se basa en la función objetivo y la topología de la red. Recuerde que la replicación es más económica en redes de difusión que en redes punto a punto. Además, la elección del número de sitios de procesamiento implica un equilibrio entre el tiempo de respuesta y el tiempo total. Un mayor

El número de sitios disminuye el tiempo de respuesta (mediante el procesamiento paralelo) pero aumenta el tiempo total, lo que en particular aumenta los costos de comunicación.

Las fórmulas para minimizar el tiempo de comunicación o de procesamiento utilizan como entrada la ubicación de los fragmentos, su tamaño y el tipo de red. Permiten minimizar ambos costos, pero priorizando uno. Para ilustrar estas fórmulas, se presentan las reglas para minimizar el tiempo de comunicación. La regla para minimizar el tiempo de respuesta es aún más compleja. Se parte de las siguientes premisas: existen n relaciones  $R_j, R_1, R_2, \dots, R_n$  involucradas en la consulta. Denota el fragmento de  $R_i$  almacenado en el sitio  $j$ .

i

Hay m sitios en la red. Finalmente,  $CT_k(\#bytes)$  denota el tiempo de comunicación para transferir #bytes a k sitios, con  $1 \leq k \leq m$ . La regla para minimizar el tiempo de comunicación considera los tipos de red por separado. Centrémonos primero en una red de difusión. En este caso, tenemos

$$CT_k(\#bytes) = CT_1(\#bytes)$$

La regla puede enunciarse así:

si  $\max_{j=1}^m (\sum_{i=1}^{n_j} \text{tamaño}(R_{ij})) > \max_{i=1}^n (\text{tamaño}(R_{i1}))$   
entonces

El sitio de procesamiento es el j con la mayor cantidad de datos

demás

$R_p$  es la relación más grande y el sitio de  $R_p$  es el  
sitio de procesamiento

Si se cumple el predicado de desigualdad, un sitio contiene una cantidad de datos útiles para la consulta mayor que el tamaño de la relación más grande. Por lo tanto, este sitio debe ser el sitio de procesamiento. Si no se cumple el predicado, una relación es mayor que la cantidad máxima de datos útiles en un sitio. Por lo tanto, esta relación debe ser el  $R_p$ , y los sitios de procesamiento son aquellos que contienen sus fragmentos.

Consideremos ahora el caso de las redes punto a punto. En este caso tenemos

$$CT_k(\#bytes) = k \cdot CT_1(\#bytes)$$

La elección de  $R_p$  que minimiza la comunicación es obviamente la relación más grande.

Suponiendo que los sitios están organizados en orden decreciente de cantidades de datos útiles para la consulta, es decir,

$$\sum_{i=1}^{n_j} \text{tamaño}(R_{ij}) > \sum_{i=1}^{n_{j+1}} \text{tamaño}(R_{(j+1)i})$$

La elección de  $k$ , el número de sitios en los que se debe realizar el procesamiento, está dada

como

```

si i=p(tamaño(Ri) – tamaño(R1 ))> tamaño(R1 p)
entonces
    k = 1
demás

k es el j más grande tal que i=p(tamaño(Ri) – tamaño(Rj )) ≤ tamaño(Rj p) i

```

Esta regla selecciona un sitio como sitio de procesamiento solo si la cantidad de datos que debe recibir es menor que la cantidad adicional que tendría que enviar si no fuera un sitio de procesamiento.

Obviamente, la parte "entonces" de la regla supone que el sitio 1 almacena un fragmento de Rp.

**Ejemplo 4.17** Consideremos la consulta PROJ ASG, donde PROJ y ASG están fragmentados. Supongamos que la asignación de fragmentos y sus tamaños son los siguientes (en kilobytes):

	Sitio 1	Sitio 2	Sitio 3	Sitio 4	
PROYECTO	1000	1000	1000	1000	
ASG				2000	

Con una red punto a punto, la mejor estrategia es enviar cada PROJi al sitio 3, lo que requiere una transferencia de 3000 kbytes, frente a los 6000 kbytes que se envían si ASG se envía a los sitios 1, 2 y 4. Sin embargo, con una red de difusión, la mejor estrategia es enviar ASG (en una sola transferencia) a los sitios 1, 2 y 4, lo que supone una transferencia de 2000 kbytes. Esta última estrategia es más rápida y maximiza el tiempo de respuesta, ya que las uniones pueden realizarse en paralelo.

Este algoritmo de optimización de consultas dinámicas se caracteriza por una búsqueda limitada del espacio de soluciones, donde se toma una decisión de optimización para cada paso sin preocuparse por las consecuencias de esa decisión en la optimización global. Sin embargo, el algoritmo es capaz de corregir una decisión local que resulte incorrecta.

#### 4.5.2 Enfoque estático

Ilustramos el enfoque estático con el algoritmo R\*, que ha sido la base de muchos optimizadores de consultas distribuidas. Este algoritmo realiza una búsqueda exhaustiva de todas las estrategias alternativas para elegir la de menor coste.

Aunque predecir y enumerar estas estrategias puede ser costoso, la sobrecarga de una búsqueda exhaustiva se amortiza rápidamente si la consulta se ejecuta con frecuencia. La compilación de consultas es una tarea distribuida, coordinada por un sitio maestro, donde se inicia la consulta. El optimizador del sitio maestro toma todas las decisiones entre sitios, como la selección de los sitios de ejecución y los fragmentos, así como el método de transferencia de datos. Los sitios aprendices, que son los otros sitios que tienen relaciones involucradas en la consulta, toman las decisiones locales restantes (como el orden de...).

---

Algoritmo 4.2: QOA estático\*

```

Entrada: QT : consulta trie
Salida: strat: estrategia de costo mínimo begin
para
    cada relación Ri   QT do para cada
        ruta de acceso APij a Ri do calcular el
            |   costo (APij ) end para
            best_APi
        ← APij con costo mínimo end para para cada
    orden
    (Ri1, Ri2, … , Rin) con i = 1, … construir la estrategia (... , in!
        ((best APi1 Ri2) Ri3) calcular el costo de la estrategia     hacer ... Rin)
    end para strat ← estrategia con
    costo
    mínimo para cada sitio k almacenar una
    relación involucrada en QT do LSk ← estrategia local
        (estrategia, k) enviar (LSk , sitio k) end
    para
                                            {Cada estrategia local está optimizada en el sitio k}
fin

```

---

Se une a un sitio) y genera planes de acceso local para la consulta. La función objetivo del optimizador es la función de tiempo total general, incluyendo los costos de procesamiento y comunicación locales.

A continuación, resumimos este algoritmo de optimización de consultas. La entrada del algoritmo es una consulta fragmentada expresada como un trie de álgebra relacional (el trie de consulta), la ubicación de las relaciones y sus estadísticas. El algoritmo se describe mediante el procedimiento Static-QOA en el Algoritmo 4.2.

El optimizador debe seleccionar el orden de unión, el algoritmo de unión (bucle anidado o combinación de fusión) y la ruta de acceso para cada fragmento (p. ej., índice agrupado, escaneo secuencial, etc.). Estas decisiones se basan en estadísticas y fórmulas utilizadas para estimar el tamaño de los resultados intermedios y la información de la ruta de acceso. Además, el optimizador debe seleccionar los sitios de los resultados de la unión y el método de transferencia de datos entre ellos. Para unir dos relaciones, existen tres sitios candidatos: el sitio de la primera relación, el sitio de la segunda relación o un tercer sitio (por ejemplo, el sitio de la tercera relación con la que se realizará la unión). Se admiten dos métodos para las transferencias de datos entre sitios.

1. Envío completo. La relación completa se envía al sitio de unión y se almacena en una relación temporal antes de unirse. Si el algoritmo de unión es de fusión, no es necesario almacenar la relación y el sitio de unión puede procesar las tuplas entrantes en modo pipeline a medida que llegan.
2. Recuperación según sea necesario. La relación externa se escanea secuencialmente y, para cada tupla, el valor de unión se envía al sitio de la relación interna, que selecciona las tuplas internas que coinciden con el valor y las envía al sitio de la relación externa. Este método, también llamado bindjoin, es equivalente a la semiunión de la relación interna con cada tupla externa.

La compensación entre estos dos métodos es obvia. El envío completo genera una mayor transferencia de datos, pero menos mensajes que la recuperación según sea necesario. Intuitivamente, es mejor enviar las relaciones completas cuando son pequeñas. Por el contrario, si la relación es grande y la unión tiene buena selectividad (solo unas pocas tuplas coincidentes), las tuplas relevantes deben recuperarse según sea necesario. El optimizador no considera todas las combinaciones posibles de métodos de unión con métodos de transferencia, ya que algunos no son útiles. Por ejemplo, sería inútil transferir la relación externa mediante la recuperación según sea necesario en el algoritmo de unión de bucle anidado, ya que todas las tuplas externas deben procesarse de todos modos y, por lo tanto, deben transferirse como un todo.

Dada la unión de una relación externa R con una relación interna S en el atributo A, existen cuatro estrategias de unión. A continuación, describimos cada estrategia en detalle y proporcionamos una fórmula de coste simplificada para cada una, donde LT representa el tiempo de procesamiento local (E/S + tiempo de CPU) y CT el tiempo de comunicación. Para simplificar, ignoramos el coste de generar el resultado. Por conveniencia, denotamos con s el número medio de tuplas de S que coinciden con una tupla de R:

$$s = \frac{\text{tarjeta}(S \text{ A } R)}{\text{tarjeta}(R)}$$

Estrategia 1. Enviar la relación externa completa al sitio de la relación interna. En este caso, las tuplas externas se pueden unir con S a medida que llegan. Por lo tanto, tenemos

$$\begin{aligned} T_{\text{total\_time}} &= LT \text{ (recuperar tuplas de tarjeta(R) de R)} \\ &\quad + CT \text{ (tamaño(R))} \\ &\quad + LT \text{ (recuperar } s \text{ tuplas de S)} \quad \text{tarjeta}(R) \end{aligned}$$

Estrategia 2. Enviar la relación interna completa al sitio de la relación externa. En este caso, las tuplas internas no pueden unirse al llegar, y deben almacenarse en una relación temporal T. Por lo tanto, tenemos

$$\begin{aligned} T_{\text{total\_time}} &= LT \text{ (recuperar tuplas de tarjeta(s) de S)} \\ &\quad + CT \text{ (talla (S))} \\ &\quad + LT \text{ (tuplas de tarjeta(s) de la tienda en T)} \\ &\quad + LT \text{ (recuperar tuplas de tarjeta(R) de R)} \\ &\quad + LT \text{ (recuperar } s \text{ tuplas de T)} \quad \text{tarjeta}(R) \end{aligned}$$

Estrategia 3. Obtener las tuplas de la relación interna según sea necesario para cada tupla de la relación externa. En este caso, para cada tupla en R, se envía el valor del atributo de unión (A).

al sitio de S. Luego, las tuplas de S que coinciden con ese valor se recuperan y se envían al sitio de R para unirse a medida que llegan. Por lo tanto, tenemos

$$\begin{aligned}
 T_{\text{total\_time}} = & LT(\text{recuperar tuplas de tarjeta}(R) \text{ de } R) \\
 & + CT(\text{longitud}(A)) \quad \text{tarjeta}(R) \\
 & + LT(\text{recuperar s tuplas de } S) \quad \text{tarjeta}(R) \\
 & + CT(s \quad \text{longitud}(S)) \quad \text{tarjeta}(R)
 \end{aligned}$$

Estrategia 4. Mover ambas relaciones a un tercer sitio y calcular la unión allí. En este caso, la relación interna se mueve primero a un tercer sitio y se almacena en una relación temporal T. Luego, la relación externa se mueve al tercer sitio y sus tuplas se unen con T a medida que llegan. Por lo tanto, tenemos

$$\begin{aligned}
 T_{\text{total\_time}} = & LT(\text{recuperar tuplas de tarjeta}(s) \text{ de } S) \\
 & + CT(\text{talla }(S)) \\
 & + LT(\text{tuplas de tarjeta}(s) \text{ de la tienda en } T) \\
 & + LT(\text{recuperar tuplas de tarjeta}(R) \text{ de } R) \\
 & + CT(\text{tamaño}(R)) \\
 & + LT(\text{recuperar s tuplas de } T) \quad \text{tarjeta}(R)
 \end{aligned}$$

Ejemplo 4.18 Consideremos una consulta que consiste en la unión de las relaciones PROJ (la relación externa) y ASG (la relación interna) sobre el atributo PNO. Suponemos que PROJ y ASG se almacenan en dos ubicaciones diferentes y que existe un índice en el atributo PNO para la relación ASG. Las posibles estrategias de ejecución de la consulta son las siguientes:

1. Enviar todo el PROJ al sitio de ASG.
2. Enviar el ASG completo al sitio del PROJ.
3. Obtenga tuplas ASG según sea necesario para cada tupla de PROJ.
4. Trasladar ASG y PROJ a un tercer sitio.

El algoritmo de optimización predice el tiempo total de cada estrategia y selecciona la más económica. Dado que no hay ningún operador después de la unión PROJ ASG, la estrategia 4 obviamente tiene el mayor costo, ya que ambas relaciones deben transferirse.

Si el tamaño (PROJ) es mucho mayor que el tamaño (ASG), la estrategia 2 minimiza el tiempo de comunicación y es probable que sea la mejor si el tiempo de procesamiento local no es demasiado alto en comparación con las estrategias 1 y 3. Tenga en cuenta que el tiempo de procesamiento local de las estrategias 1 y 3 es probablemente mucho mejor que el de la estrategia 2, ya que explotan el índice en el atributo de unión.

Si la estrategia 2 no es la mejor, la elección es entre las estrategias 1 y 3. Los costos de procesamiento local en ambas alternativas son idénticos. Si PROJ es grande y solo coinciden unas pocas tuplas de ASG, la estrategia 3 probablemente incurre en el menor tiempo de comunicación y es la mejor. De lo contrario, es decir, si PROJ es pequeño o coinciden muchas tuplas de ASG, la estrategia 1 debería ser la mejor.

Conceptualmente, el algoritmo puede considerarse una búsqueda exhaustiva entre todas las alternativas definidas por la permutación del orden de unión de la relación, los métodos de unión (incluida la selección del algoritmo de unión), el sitio de resultados, la ruta de acceso a la relación interna y el modo de transferencia entre sitios. Este algoritmo presenta una complejidad combinatoria en el número de relaciones involucradas. De hecho, el algoritmo reduce significativamente el número de alternativas mediante el uso de programación dinámica y heurística. Con la programación dinámica, el árbol de alternativas se construye y se poda dinámicamente eliminando las opciones ineficientes.

La evaluación del rendimiento del algoritmo en el contexto de redes de alta velocidad (similares a las redes locales) y de redes de área amplia de velocidad media confirma la contribución significativa de los costos de procesamiento local, incluso para redes de área amplia.

Se muestra en particular que, para la unión distribuida, la transferencia de toda la relación interna supera al método de búsqueda según sea necesario.

### 4.5.3 Enfoque híbrido

Tanto la optimización de consultas dinámicas como la estáticas tienen ventajas y desventajas.

La optimización dinámica de consultas combina optimización y ejecución, lo que permite tomar decisiones de optimización precisas en tiempo de ejecución. Sin embargo, la optimización de consultas se repite con cada ejecución. Por lo tanto, este enfoque es ideal para consultas ad hoc. La optimización estática de consultas, realizada en tiempo de compilación, amortiza el coste de la optimización en múltiples ejecuciones. Por lo tanto, la precisión del modelo de costes es crucial para predecir los costes de los posibles QEP. Este enfoque es ideal para consultas integradas en procedimientos almacenados y ha sido adoptado por todos los SGBD comerciales.

Sin embargo, incluso con un modelo de costos sofisticado, existe un problema importante que impide realizar una estimación precisa de los costos y la comparación de los QEP en el momento de la compilación. El problema es que las vinculaciones reales de los valores de los parámetros en las consultas integradas no se conocen hasta el tiempo de ejecución. Considere, por ejemplo, el predicado de selección WHERE RA=\$a, donde \$a es un valor de parámetro. Para estimar la cardinalidad de esta selección, el optimizador debe confiar en el supuesto de una distribución uniforme de los valores de A en R y no puede usar histogramas. Dado que existe una vinculación en tiempo de ejecución del parámetro a, la selectividad precisa de  $\sigma_{A=\$a}(R)$  no se puede estimar hasta el tiempo de ejecución. Por lo tanto, puede cometer errores de estimación importantes que pueden llevar a la elección de QEP subóptimos. Además de las vinculaciones desconocidas de los valores de los parámetros en las consultas integradas, los sitios pueden no estar disponibles o sobrecargarse en tiempo de ejecución. Además, las relaciones (o fragmentos de relación) pueden replicarse en varios sitios. Por lo tanto, la selección de sitios y copias debe realizarse en tiempo de ejecución para aumentar la disponibilidad y el equilibrio de carga del sistema.

La optimización de consultas híbrida busca ofrecer las ventajas de la optimización de consultas estáticas, evitando los problemas generados por estimaciones inexactas. El enfoque es básicamente estático, pero se pueden tomar decisiones de optimización adicionales en tiempo de ejecución. Una solución general consiste en generar QEP dinámicos que incluyan decisiones de optimización cuidadosamente seleccionadas para su ejecución mediante operadores de "selección de plan". Este operador vincula dos o más subplanes equivalentes de un QEP que no son comparables en tiempo de compilación debido a la falta de información importante en tiempo de ejecución (p. ej., vinculación de parámetros) para estimar los costos. La ejecución de un operador de selección de plan permite comparar los subplanes según los costos reales y seleccionar el mejor. Los nodos de selección de plan pueden insertarse en cualquier parte de un QEP.

Este enfoque es lo suficientemente general como para incorporar decisiones de selección de sitio y copia. Sin embargo, el espacio de búsqueda de subplanes alternativos vinculados por operadores choose-plan se vuelve mucho mayor, lo que puede resultar en planes estáticos pesados y un tiempo de inicio mucho mayor. Por lo tanto, se han propuesto varias técnicas híbridas para optimizar las consultas en sistemas distribuidos. Estas se basan esencialmente en el siguiente enfoque de dos pasos:

1. En tiempo de compilación, generar un plan estático que especifique el orden de los operadores y los métodos de acceso, sin considerar dónde se almacenan las relaciones.
2. En el momento de inicio, genere un plan de ejecución realizando la selección del sitio y de la copia y asignando los operadores a los sitios.

Ejemplo 4.19 Considere la siguiente consulta expresada en álgebra relacional:

$$\sigma ( R1 ) R2R3$$

La Figura 4.15 muestra un plan de dos pasos para esta consulta. El plan estático muestra la ordenación de los operadores relacionales generada por un optimizador de consultas centralizado. El plan de ejecución extiende el plan estático con la selección de sitios y copias, y la comunicación entre ellos. Por ejemplo, la primera selección se asigna al sitio S1 de la copia R11 de la relación R1 y envía su resultado al sitio S3 para su unión con R23, y así sucesivamente.

El primer paso puede realizarse mediante un optimizador de consultas centralizado. También puede incluir operadores choose-plan para que las vinculaciones en tiempo de ejecución se puedan usar al inicio para realizar estimaciones de costos precisas. El segundo paso realiza la selección del sitio y la copia, posiblemente además de la ejecución del operador choose-plan. Además, puede optimizar el balanceo de carga del sistema. En el resto de esta sección, ilustramos este segundo paso.

Consideramos un sistema de base de datos distribuida con un conjunto de sitios  $S = \{S1, \dots, Sn\}$ . Una consulta  $Q$  se representa como una secuencia ordenada de subconsultas  $Q = \{q1, \dots, qm\}$ . Cada subconsulta  $qi$  es la unidad de procesamiento máxima que accede a una única relación de base y se comunica con sus subconsultas vecinas. Por ejemplo, en la Fig. 4.15, hay tres subconsultas: una para R1, otra para R2 y otra para R3. Cada sitio  $Si$  tiene una carga, denotada por  $load(Si)$ , que refleja el número de consultas enviadas actualmente. La carga puede expresarse de diferentes maneras, por ejemplo, como el número de consultas limitadas por E/S y CPU en el sitio. La carga promedio del sistema se define como:

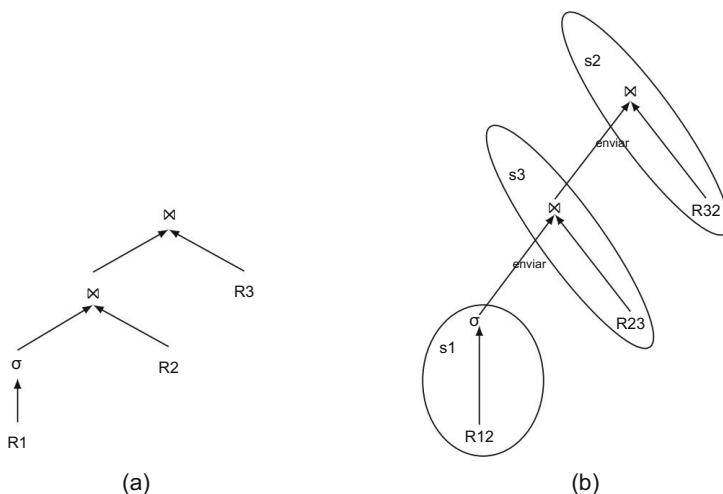


Fig. 4.15 Un plan de dos pasos. (a) Plan estático. (b) Plan de tiempo de ejecución.

$$\text{Carga promedio}(S) = \frac{\sum_{i=1}^n \text{carga}(S_i)}{n}$$

El saldo del sistema para una asignación dada de subconsultas a sitios puede ser medido como la varianza de las cargas del sitio utilizando el siguiente factor de desequilibrio:

$$UF(S) = \frac{1}{n_{note}} (carga(S_i) - Carga\_promedio(S))^2$$

A medida que el sistema se equilibra, su factor de desequilibrio se acerca a 0 (equilibrio perfecto). El factor de desequilibrio de  $\{S_1, S_2\}$  = que con carga( $S_1$ ) y carga( $S_1$ ), es 0.

El problema abordado en el segundo paso de la optimización de consultas en dos pasos se puede formalizar como el siguiente problema de asignación de subconsultas. Dado

1. un conjunto de sitios  $S = \{S_1, \dots, S_n\}$  con la carga de cada sitio; 2. una consulta  $Q = \{q_1, \dots, q_m\}$ ; y 3. para cada subconsulta  $q_i$  en  $Q$ , un conjunto de asignación factible de sitios  $S_q = \{S_{1,i}, \dots, S_{n,i}\}$  donde cada sitio almacena una copia de la relación involucrada en  $q_i$ ;

El objetivo es encontrar una asignación óptima de Q a S tal que

1. Se minimiza la UF (S) y 2. Se minimiza el coste total de comunicación.

Hay un algoritmo que encuentra soluciones casi óptimas en una cantidad razonable del tiempo. El algoritmo, que describimos en el Algoritmo 4.3 para árboles de unión lineal, Utiliza varias heurísticas. La primera (paso 1) consiste en comenzar asignando subconsultas. con la menor flexibilidad de asignación, es decir, con los conjuntos de sitios de asignación más pequeños posibles. Por lo tanto, las subconsultas con pocos sitios candidatos se asignan antes. Otra heurística El paso 2 consiste en considerar los sitios con menor carga y mayor beneficio. El beneficio de un sitio se define como el número de subconsultas ya asignadas al sitio y mide El ahorro en costos de comunicación al asignar la subconsulta al sitio. Finalmente, en Paso 3 del algoritmo, la información de carga de cualquier subconsulta no asignada que tenga una Se vuelve a calcular el sitio seleccionado en su conjunto de asignación factible.

Ejemplo 4.20 Considere la siguiente consulta Q expresada en álgebra relacional:

$$\sigma ( R1 ) R2R3R4$$

La figura 4.16 muestra la ubicación de las copias de las 4 relaciones en los 4 sitios, y el sitio carga. Suponemos que Q se descompone como  $Q = \{q_1, q_2, q_3, q_4\}$ , donde  $q_1$  está asociado con  $R1$ ,  $q_2$  con  $R2$  unido con el resultado de  $q_1$ ,  $q_3$  con  $R3$  unido con el resultado de  $q_2$ , y  $q_4$  con  $R4$  unido con el resultado de  $q_3$ . La SQAllocation

---

#### Algoritmo 4.3: SQAllocation

---

Entrada:  $Q: q_1, \dots, q_m$

Conjuntos de asignación factibles:  $F_{q_1}, \dots, F_{q_m}$

Cargas:  $carga(F_1), \dots, carga(F_m)$

Salida: una asignación de  $Q$  a  $S$

comenzar

para cada  $q$  en  $Q$  hacer  
|   calcular(cargar( $F_q$  ))  
fin para

mientras  $Q$  no esté vacío

| {seleccionar subconsulta a para asignación}  
a  $\leftarrow q \in Q$  con menor flexibilidad de asignación (1)  
{seleccione el mejor sitio b para a}

b  $\leftarrow f \in Fa$  con menor carga y mejor beneficio (2)

Q  $\leftarrow Q - a$

| {recalcular cargas de conjuntos de asignación factibles restantes si es necesario} para (3)

cada  $q \in Q$  donde  $b \in F_q$  hacer  
|   calcular(cargar( $F_q$  ))

fin para

terminar mientras

fin

---

Carga de sitios	R1	R2	R3	R4
s1	1	R11		R31
s2	2		R22	
s3	2	R13		R33
s4	2	R14	R24	

Fig. 4.16 Ejemplo de colocación y carga de datos

El algoritmo realiza 4 iteraciones. En la primera, selecciona q4 , que tiene la menor flexibilidad de asignación, la asigna a S1 y actualiza la carga de S1 a 2. En la segunda iteración, las siguientes subconsultas que se seleccionarán son q2 o q3, ya que tienen la misma flexibilidad de asignación. Elijamos q2 y supongamos que se asigna a S2 (podría asignarse a S4, que tiene la misma carga que S2). La carga de S2 se incrementa a 3. En la tercera iteración, la siguiente subconsulta seleccionada es q3 y se asigna a S1 , que tiene la misma carga que S3, pero un beneficio de 1 (en comparación con 0 para S3) como resultado de la asignación de q4. La carga de S1 se incrementa a 3. Finalmente, en la última iteración, q1 se asigna a S3 o S4 , que tienen las menores cargas. Si en la segunda iteración q2 se hubiera asignado a S4 en lugar de a S2, la cuarta iteración habría asignado q1 a S4 debido a un beneficio de 1. Esto habría generado un mejor plan de ejecución con menos comunicación. Esto demuestra que la optimización en dos pasos aún puede no alcanzar los planes óptimos.

Este algoritmo tiene una complejidad razonable. Considera cada subconsulta por turno, considerando cada sitio potencial, selecciona el actual para su asignación y ordena la lista de subconsultas restantes. Por lo tanto, su complejidad se puede expresar como  $O(\max.(m n, m^2 \log^2 m))$ .

Finalmente, el algoritmo incluye una fase de refinamiento para optimizar aún más el procesamiento de uniones y decidir si se utilizan semiuniones. Si bien minimiza la comunicación con un plan estático, la optimización de consultas en dos pasos puede generar planes de ejecución con un mayor coste de comunicación que el plan óptimo. Esto se debe a que el primer paso se realiza ignorando la ubicación de los datos y su impacto en el coste de comunicación. Por ejemplo, considere el plan de ejecución de la Fig. 4.15 y suponga que la tercera subconsulta en R3 se asigna al sitio S1 (en lugar del sitio S2). En este caso, el plan que realiza la unión (o producto cartesiano) del resultado de la selección de R1 con R3 primero en el sitio S1 puede ser mejor, ya que minimiza la comunicación. Una solución a este problema es realizar la reorganización del plan mediante transformaciones de operador trié al inicio.

## 4.6 Procesamiento de consultas adaptativo

Hasta ahora, se ha asumido que el procesador de consultas distribuido posee suficiente conocimiento sobre las condiciones de ejecución de las consultas para generar un QEP eficiente y que las condiciones de ejecución se mantienen estables durante la ejecución. Esta es una suposición válida para consultas con pocas relaciones de base de datos que se ejecutan en un entorno controlado. Sin embargo, esta suposición es inadecuada para entornos cambiantes con un gran número de relaciones y condiciones de ejecución impredecibles.

Ejemplo 4.21. Considere el QEP de la Fig. 4.17 con las relaciones EMP, ASG, PROJ y PAY en los sitios S1, S2, S3 y S4, respectivamente. La flecha cruzada indica que, por alguna razón (p. ej., un fallo), el sitio S2 (donde se almacena ASG) no está disponible al inicio de la ejecución. Supongamos, para simplificar, que la consulta se ejecuta según el modelo de ejecución del iterador, de modo que las tuplas fluyen desde la relación situada más a la izquierda.

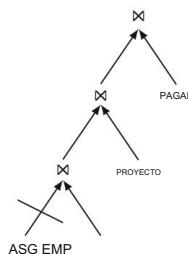


Fig. 4.17 Plan de ejecución de consulta con una relación bloqueada

Debido a la indisponibilidad de S2, todo el pipeline está bloqueado, a la espera de que se produzcan las tuplas ASG. Sin embargo, con una reorganización del plan, se podrían evaluar otros operadores mientras se espera a S2; por ejemplo, para evaluar la unión de EMP y PAY.

Este sencillo ejemplo ilustra que un plan estático típico no puede gestionar la indisponibilidad impredecible de las fuentes de datos. Ejemplos más complejos implican consultas continuas, predicados costosos y sesgo de datos. La principal solución es contar con un comportamiento adaptativo durante el procesamiento de consultas, es decir, un procesamiento adaptativo de consultas. El procesamiento adaptativo de consultas es una forma de procesamiento dinámico de consultas, con un bucle de retroalimentación entre el entorno de ejecución y el optimizador de consultas para reaccionar ante variaciones imprevistas en las condiciones de ejecución. Un sistema de procesamiento de consultas se define como adaptativo si recibe información del entorno de ejecución y determina su comportamiento en función de dicha información de forma iterativa.

En esta sección, primero ofrecemos una presentación general del proceso de procesamiento adaptativo de consultas. A continuación, presentamos el enfoque de remolinos, que proporciona un marco sólido para dicho procesamiento.

#### 4.6.1 Proceso de procesamiento de consultas adaptativo

El procesamiento adaptativo de consultas añade al procesamiento tradicional las siguientes actividades: monitorización, evaluación y reacción. Estas actividades se implementan lógicamente en el sistema de procesamiento de consultas mediante sensores, componentes de evaluación y componentes de reacción, respectivamente. La monitorización consiste en medir ciertos parámetros del entorno dentro de un intervalo de tiempo y reportarlos al componente de evaluación. Este último analiza los informes y considera los umbrales para elaborar un plan de reacción adaptativo. Finalmente, el plan de reacción se comunica al componente de reacción, que aplica las reacciones a la ejecución de la consulta.

Normalmente, un proceso adaptativo especifica la frecuencia con la que se ejecutará cada componente. Existe un equilibrio entre la reactividad, donde valores más altos provocan reacciones impulsivas, y la sobrecarga causada por el proceso adaptativo. Una representación genérica del proceso adaptativo viene dada por la función

$fadapt(E, T) \rightarrow Ad$ , donde E es un conjunto de parámetros ambientales monitoreados, T es un conjunto de valores umbral y Ad es un conjunto posiblemente vacío de reacciones adaptativas.

Los elementos de E, T y Ad, denominados elementos adaptativos, pueden variar de diversas maneras según la aplicación. Los elementos más importantes son los parámetros de monitorización y las reacciones adaptativas. A continuación, los describiremos.

#### 4.6.1.1 Parámetros de monitoreo

El monitoreo de los parámetros de ejecución de consultas implica colocar sensores en lugares clave del QEP y definir ventanas de observación, durante las cuales los sensores recopilan información.

También requiere la especificación de un mecanismo de comunicación para transmitir la información recopilada al componente de evaluación. Ejemplos de candidatos para el seguimiento. son:

- Tamaño de la memoria. Monitorear el tamaño de la memoria disponible permite, por ejemplo, a los operadores para reaccionar ante la falta de memoria o el aumento de la memoria.

- Tasas de llegada de datos. Monitorear la variación en las tasas de llegada de datos puede permitir que el procesador de consultas realice un trabajo útil mientras espera una fuente de datos bloqueada. •

Estadísticas reales. Las estadísticas de bases de datos en un entorno distribuido tienden a ser inaccesibles.

Curar, si es que está disponible. Monitorear el tamaño real de las relaciones y los resultados intermedios puede llevar a modificaciones importantes en el QEP. Además, se pueden abandonar las suposiciones habituales sobre los datos, en las que la selectividad de los predicados sobre los atributos en una relación es mutuamente independiente, y se pueden calcular valores reales de selectividad. • Coste de ejecución del operador. Monitorear

el coste real de la ejecución del operador, incluyendo las tasas de producción, es útil para una mejor programación del operador. Además, monitorear el tamaño de las colas colocadas antes de los operadores puede evitar la sobrecarga. • Rendimiento de la red. Monitorear el rendimiento de la red puede ser útil para definir el tamaño de bloque para recuperar datos. En una red de bajo rendimiento, el sistema puede reaccionar con tamaños de bloque mayores para reducir la penalización de la red.

#### 4.6.1.2 Reacciones adaptativas

Las reacciones adaptativas modifican el comportamiento de ejecución de consultas según las decisiones tomadas por el componente de evaluación. Entre las reacciones adaptativas más importantes se encuentran:

- Cambiar la programación: modifica el orden en que se programan los operadores en el QEP. La codificación de consultas reacciona modificando la programación del plan para evitar el bloqueo en una fuente de datos bloqueada durante la evaluación de la consulta. Eddy adopta una reacción más precisa, donde la programación de operadores puede decidirse por tuplas. • Reemplazo de operadores: reemplaza un operador físico por uno equivalente. Por ejemplo, dependiendo de la memoria disponible, el sistema puede elegir entre una unión de bucle anidado o una unión hash. El reemplazo de operadores también puede cambiar el plan introduciendo un nuevo operador para unir los resultados intermedios producidos por

Una reacción adaptativa previa. La codificación de consultas, por ejemplo, puede introducir nuevos operadores para evaluar las uniones entre los resultados de las reacciones de programación de cambios . • Refragmentación de datos: considera la fragmentación dinámica de una relación. La partición estática de una relación tiende a generar un desequilibrio de carga entre sitios. Por ejemplo, la información particionada según su región geográfica asociada puede presentar diferentes tasas de acceso durante el día debido a las diferencias horarias en la ubicación de los usuarios.

Reformulación del plan: calcula un nuevo QEP para reemplazar uno ineficiente. El optimizador considera las estadísticas reales y la información de estado, recopiladas sobre la marcha, para generar un nuevo plan.

#### 4.6.2 Enfoque de remolino

Eddy es un marco general para el procesamiento adaptativo de consultas sobre relaciones distribuidas. Para simplificar, solo consideramos consultas select-project-join (SPJ). Los operadores de selección pueden incluir predicados costosos. El proceso de generar un QEP a partir de una consulta SPJ de entrada comienza con la producción de un trie de operadores del grafo de unión G de la consulta de entrada. La elección entre algoritmos de unión y métodos de acceso a la relación favorece la adaptabilidad. Un QEP puede modelarse como una tupla  $Q = D, P, C$ , donde D es un conjunto de relaciones de base de datos, P es un conjunto de predicados de consulta con algoritmos asociados y C es un conjunto de restricciones de ordenación que deben seguirse durante la ejecución. Observe que se pueden derivar múltiples árboles de operadores válidos de G que obedecen las restricciones en C, explorando el espacio de búsqueda con diferentes órdenes de predicados. No es necesario encontrar un QEP óptimo durante la compilación de la consulta. En su lugar, la ordenación de los operadores se realiza sobre la marcha, tupla por tupla (es decir, enrutamiento de tuplas). El proceso de compilación de QEP se completa agregando el operador eddy , que es un operador n-ario ubicado entre las relaciones en D y los predicados de consulta en P.

Ejemplo 4.22 Considere una consulta de tres relaciones  $Q = (\text{op}(R) ST)$ , donde las uniones son equijoins. Suponga que el único método de acceso a la relación T es a través de un índice en el atributo de unión TA; es decir, la segunda unión solo puede ser una unión de índice sobre TA. Supongamos también que op es un predicado costoso (p. ej., un predicado sobre los resultados de ejecutar un programa con valores de TB). Bajo estos supuestos, el QEP se define como  $D = \{R, S, T\}$ ,  $P = \{\text{op}(R), RS, ST\}$  y  $C = \{S \rightarrow T\}$ . La restricción  $\rightarrow$  impone S tuplas para sondar T tuplas, basándose en el índice de TA.

La Figura 4.18 muestra un QEP generado mediante la compilación de la consulta Q con eddy. Una elipse corresponde a un operador físico (es decir, un operador eddy o un algoritmo que implementa un predicado  $p \rightarrow P$ ). Como es habitual, la parte inferior del plan presenta las relaciones de origen. En ausencia de un método de acceso de escaneo, el acceso a la relación T se envuelve mediante la unión ST, por lo que no aparece como una relación de origen. Las flechas especifican el flujo de datos de la tubería siguiendo una relación productor-consumidor. Finalmente, una flecha que parte del eddy modela la producción de tuplas de salida.

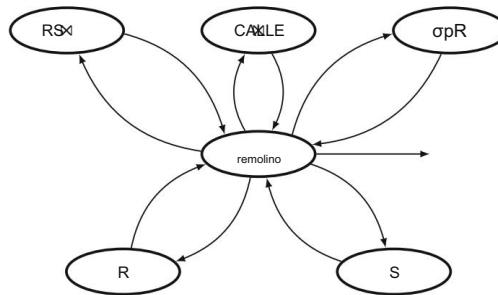


Fig. 4.18 Un plan de ejecución de consulta con eddy

Eddy proporciona adaptabilidad detallada al decidir sobre la marcha cómo enrutar las tuplas a través de predicados según una política de programación. Durante la ejecución de la consulta, las tuplas de las relaciones de origen se recuperan y se almacenan en un búfer de entrada gestionado por el operador eddy. Eddy responde a la indisponibilidad de una relación simplemente leyendo de otra relación y almacenando las tuplas en el búfer.

La flexibilidad para elegir la relación fuente disponible actualmente se obtiene flexibilizando el orden fijo de predicados en un QEP. En eddy, no existe un QEP fijo y cada tupla sigue su propia ruta a través de los predicados, según las restricciones del plan y su propio historial de evaluación de predicados.

La estrategia de enrutamiento basada en tuplas produce una nueva topología QEP. El operador eddy, junto con sus predicados gestionados, forma un flujo de datos circular en el que las tuplas salen del operador eddy para ser evaluadas por los predicados, quienes, a su vez, devuelven las tuplas de salida al operador eddy. Una tupla abandona el flujo de datos circular cuando es eliminada por la evaluación de un predicado o cuando el operador eddy detecta que ha pasado por todos los predicados de su lista. La falta de un QEP fijo requiere que cada tupla registre el conjunto de predicados para el que es elegible. Por ejemplo, en la figura 4.18, las tuplas S son elegibles para los dos predicados de unión, pero no para el predicado  $\sigma p(R)$ .

## 4.7 Conclusión

En este capítulo, presentamos detalladamente el procesamiento de consultas en sistemas de gestión de bases de datos distribuidos. Primero, presentamos el problema del procesamiento distribuido de consultas. El supuesto principal es que la consulta de entrada se expresa en cálculo relacional, como ocurre con la mayoría de los SGBD distribuidos actuales. La complejidad del problema es proporcional a la capacidad expresiva y de abstracción del lenguaje de consulta.

El problema del procesamiento de consultas es muy difícil de comprender en entornos distribuidos debido a la gran cantidad de elementos involucrados. Sin embargo, el problema puede dividirse en varios subproblemas que son más fáciles de resolver individualmente. Por lo tanto, hemos propuesto un esquema genérico de capas para describir las consultas distribuidas.

Procesamiento. Se han aislado cuatro funciones principales: descomposición de consultas, localización de datos, optimización distribuida y ejecución distribuida. Estas funciones refinan la consulta añadiendo más detalles sobre el entorno de procesamiento.

A continuación, describimos la localización de datos, con énfasis en las técnicas de reducción y simplificación para los cuatro tipos de fragmentación siguientes: horizontal, vertical, derivada e híbrida. La consulta generada por la capa de localización de datos es eficaz, ya que evita las ejecuciones deficientes. Sin embargo, las capas posteriores suelen realizar optimizaciones importantes, ya que añaden a la consulta mayor detalle sobre el entorno de procesamiento.

A continuación, analizamos el principal problema de optimización, que trata del ordenamiento de uniones en consultas distribuidas, incluidas estrategias de unión alternativas basadas en semiuniones y la definición de un modelo de costos distribuidos.

Ilustramos el uso de las técnicas de unión y semiunión en tres algoritmos básicos de optimización de consultas distribuidas: dinámico, estático e híbrido. Los enfoques de optimización distribuida estático y dinámico presentan las mismas ventajas y desventajas que en los sistemas centralizados. El enfoque híbrido es óptimo en los entornos dinámicos actuales, ya que retrasa decisiones importantes como la selección de copias y la asignación de subconsultas a sitios al inicio de la consulta. Por lo tanto, puede mejorar la disponibilidad y el equilibrio de carga del sistema. Ilustramos el enfoque híbrido con una optimización de consultas en dos pasos que primero genera un plan estático que especifica el orden de los operadores, como en un sistema centralizado, y luego genera un plan de ejecución al inicio, mediante la selección de sitios y copias y la asignación de operadores a los sitios.

Finalmente, analizamos el procesamiento adaptativo de consultas para gestionar el comportamiento dinámico de los SGBD locales. El procesamiento adaptativo de consultas aborda este problema con un enfoque dinámico mediante el cual el optimizador de consultas se comunica con el entorno de ejecución en tiempo de ejecución para reaccionar ante variaciones imprevistas de las condiciones de ejecución.

## 4.8 Notas bibliográficas

Existen varios estudios sobre el procesamiento y la optimización de consultas en el contexto del modelo relacional. Graefe [1993] ofrece un estudio detallado.

El modelo de ejecución iterativa, que ha servido de base para la implementación de numerosos procesadores de consultas, se propuso en el contexto del sistema de evaluación de consultas extensible Volcano [Graefe , 1994]. El artículo fundamental sobre optimización de consultas basada en costes [Selinger et al., 1979] fue el primero en proponer un modelo de costes con estadísticas de bases de datos (véase la sección 4.4.2) y el uso de una estrategia de búsqueda de programación dinámica (véase la sección 4.1.1). Se han propuesto estrategias aleatorias, como la Mejora Iterativa [Swami, 1989] y el Recocido Simulado [Ioannidis y Wong, 1987] , para lograr un buen equilibrio entre el tiempo de optimización y el tiempo de ejecución.

El estudio más completo sobre el procesamiento distribuido de consultas es el de Kossmann [2000] y trata tanto de sistemas de gestión de bases de datos distribuidos como de sistemas multibase de datos.

Este artículo presenta las fases tradicionales del procesamiento de consultas en sistemas centralizados y distribuidos, y describe las diversas técnicas para el procesamiento distribuido de consultas. Los modelos de costes distribuidos se analizan en varios artículos, como [Lohman et al., Khoshafian y Valduriez, 1987].

Ceri y Pelagatti [1983] tratan en detalle la localización de datos para relaciones particionadas horizontalmente, denominadas multirelaciones. Ceri et al. [1986] utilizan las propiedades formales de la fragmentación horizontal y vertical para caracterizar las uniones distribuidas sobre relaciones fragmentadas.

La teoría de las semiuniones y su utilidad para el procesamiento de consultas distribuidas se ha abordado en [Bernstein y Chiu, 1981], [Chiu y Ho, 1980] y [Kambayashi et al., 1982]. El enfoque basado en semiuniones para la optimización de consultas distribuidas fue propuesto por Bernstein et al. [1981] para el sistema SDD-1 [Wong, 1977]. Chiu y Ho [1980] y Kambayashi et al. investigan programas de semiuniones con reductor completo.

[1982]. El problema de encontrar reductores completos es NP-hard. Sin embargo, para consultas encadenadas, existe un algoritmo polinomial [Chiu y Ho 1980, Ullman 1982]. El costo de las semiuniones puede minimizarse mediante el uso de matrices de bits [Valduriez 1982]. Otros algoritmos de procesamiento de consultas buscan seleccionar una combinación óptima de uniones y semiuniones [Özsoyoglu y Zhou 1987, Wah y Lien 1985].

El enfoque dinámico para la optimización de consultas distribuidas se propuso por primera vez para INGRES distribuido [Epstein et al., 1978]. El algoritmo aprovecha la topología de la red (redes generales o de difusión) y utiliza el algoritmo de reducción [Wong y Youssef, 1976], que aísla todas las subconsultas irreducibles y monorrelacionales mediante desapego.

El enfoque estático para la optimización de consultas distribuidas se propuso por primera vez para R\* [Selinger y Adiba, 1980]. Es uno de los primeros artículos en reconocer la importancia del procesamiento local en el rendimiento de las consultas distribuidas. La validación experimental de Lohman et al. y Mackert y Lohman [1986a,b] ha confirmado esta importante afirmación. El método fetch-as-needed de R\* se denomina bindjoin en [Haas et al., 1997a].

Un enfoque híbrido general para la optimización de consultas consiste en utilizar operadores de elección de plan [Cole y Graefe, 1994]. Se han propuesto varios enfoques híbridos basados en la optimización de consultas en dos pasos para sistemas distribuidos [Carey y Lu , 1986 ; Du et al. , 1995; Evrendilek et al., 1997]. El contenido de la sección 4.5.3 se basa en el artículo fundamental sobre optimización de consultas en dos pasos de Carey y Lu [1986]. Du et al. [1995] proponen operadores eficientes para transformar árboles de unión lineal (generados en el primer paso) en árboles frondosos con mayor paralelismo. Evrendilek et al. [1997] proponen una solución para maximizar el paralelismo de las uniones entre sitios en el segundo paso.

El procesamiento de consultas adaptativo se analiza en [Hellerstein et al. 2000, Gounaris et al. 2002]. El artículo fundamental sobre el enfoque de remolinos, que utilizamos para ilustrar el procesamiento adaptativo de consultas en la sección 4.6, es [Avnur y Hellerstein, 2000]. Otras técnicas importantes para el procesamiento adaptativo de consultas son la codificación de consultas [Amsaleg et al., 1996; Urhan et al., 1998], las uniones de ondulación [Haas y Hellerstein, 1999b], el particionamiento adaptativo [Shah et al., 2003] y la selección de datos [Porto et al., 2003]. Las principales extensiones del enfoque de remolinos son los módulos de estado [Raman et al., 2003] y los remolinos distribuidos [Tian y DeWitt, 2003].

## Ceremonias

Problema 4.1 Suponga que la relación PROJ de la base de datos muestra está fragmentada horizontalmente de la siguiente manera:

$$\text{PROJ1} = \sigma_{\text{PNO} \leq "P2"}(\text{PROJ})$$

$$\text{PROJ2} = \sigma_{\text{PNO} > "P2"}(\text{PROJ})$$

Transforme la siguiente consulta en una consulta reducida sobre fragmentos:

```
SELECCIONAR ENO, PNAME
DESDE PROJ NATURAL ÚNETE A ASG
DONDE PNO = "P4"
```

Problema 4.2 (\*) Supóngase que la relación PROJ está fragmentada horizontalmente como en el Problema 4.1, y que la relación ASG está fragmentada horizontalmente como

$$\text{ASG1} = \sigma_{\text{PNO} \leq "P2"}(\text{ASG})$$

$$\text{ASG2} = \sigma_{\text{P2} < \text{PNO} \leq "P3"}(\text{ASG})$$

$$\text{ASG3} = \sigma_{\text{PNO} > "P3"}(\text{ASG})$$

Transforme la siguiente consulta en una consulta reducida sobre fragmentos y determine si es mejor que la consulta de fragmentos:

```
SELECCIONAR RESP, PRESUPUESTO
DESDE ASG NATURAL ÚNETE AL PROYECTO
DONDE PNAME = "CAD/CAM"
```

Problema 4.3 (\*\*) Suponga que la relación PROJ está fragmentada como en el problema 4.1.

Además, la relación ASG está fragmentada indirectamente como

$$\text{ASG1} = \text{ASG PNO PROJ1}$$

$$\text{ASG2} = \text{ASG PNO PROJ2}$$

y la relación EMP está fragmentada verticalmente como

$$\text{EMP1} = \text{ENO,ENAME(EMP)}$$

$$\text{EMP2} = \text{ENO,TÍTULO(EMP)}$$

vnueve

Transforme la siguiente consulta en una consulta reducida sobre fragmentos:

```
SELECCIONAR NOMBRE
DESDE EMP NATURAL ÚNETE A ASG NATURAL ÚNETE A PROJ
DONDE PNAME = "Instrumentación"
```

**Problema 4.4** Considere el gráfico de unión de la figura 4.11 y la siguiente información: tamaño(EMP) = 100, tamaño(ASG) = 200, tamaño(PROJ) = 300, tamaño(EMP ASG) = 300 y tamaño (ASG PROJ) = 200. Describa un programa de unión óptimo basado en la función objetivo del tiempo total de transmisión.

**Problema 4.5** Considere el gráfico de unión de la figura 4.11 y haga las mismas suposiciones. Como en el problema 4.4. Describa un programa de unión óptimo que minimice el tiempo de respuesta. (considerar solo comunicación).

**Problema 4.6** Considere el gráfico de unión de la figura 4.11 y proporcione un programa (posiblemente no óptimo) que reduce cada relación completamente mediante semiuniones.

**Problema 4.7 (\*)** Considere el gráfico de unión de la Fig. 4.11 y la fragmentación Se muestra en la Fig. 4.19. Suponga también que tamaño (EMP ASG) = 2000 y tamaño (ASG PROJ) = 1000. Aplique el algoritmo de optimización de consultas distribuidas dinámicas. en la Secc. 4.5.1 en dos casos, red general y red de difusión, de modo que El tiempo de comunicación se minimiza.

**Problema 4.8 (\*\*)** Considere la siguiente consulta en nuestra base de datos de ingeniería:

```
SELECCIONAR NOMBRE,SAL
DESDE PAGAR NATURAL UNIRSE EMP NATURAL UNIRSE ASG
    PROYECTO DE UNIÓN NATURAL
DONDE (PRESUPUESTO>200000 O DUR>24)
Y          (DUR>24 O PNAME = "CAD/CAM")
```

Supongamos que las relaciones EMP, ASG, PROJ y PAY se han almacenado en los sitios 1, 2, y 3 según la tabla de la Fig. 4.20. Supóngase también que la tasa de transferencia entre Cualquier par de sitios son iguales y la transferencia de datos es 100 veces más lenta que el procesamiento de datos. Realizado por cualquier sitio. Finalmente, suponga que tamaño(RS) = máx.(tamaño(R), tamaño(S)) para cualesquiera dos relaciones R y el factor de selectividad de la selección disyuntiva de la consulta es 0,5. Redacte un programa distribuido que calcule la respuesta a la consulta y minimiza el tiempo total.

Relación Sitio 1 Sitio 2 Sitio 3			
	1000	1000	1000
ASG		2000	
PROYECTO	1000		

Figura 4.19 Fragmentación

Relación Sitio 1 Sitio 2 Sitio 3			
	2000	3000	1000
ASG			
PROYECTO			500
PAGAR			

Fig. 4.20 Estadísticas de fragmentación

Problema 4.9 (\*\*\*) En la Sección 4.5.3, describimos el algoritmo 4.3 para árboles de unión lineal.

Extienda este algoritmo para que admita árboles de unión arbustiva. Aplíquelo al árbol de unión arbustiva de la Fig. 4.9 utilizando la ubicación de datos y las cargas del sitio que se muestran en la Fig. 4.16.

Problema 4.10 (\*\*\*) Consideremos tres relaciones  $R(A,B)$ ,  $S(B,D)$  y  $T(D,E)$ , y

Consulta  $Q(\sigma_1 p(R) \leq S \leq T)$ , donde 1 y 2 son uniones naturales. Suponga que  $S$  tiene un índice en el atributo  $B$  y  $T$  tiene un índice en el atributo  $D$ . Además,  $\sigma_1$  es un predicado costoso (es decir, un predicado sobre los resultados de ejecutar un programa sobre valores de  $R$ ). Utilizando el enfoque de remolinos para el procesamiento adaptativo de consultas, responda las siguientes preguntas:

(a) Proponga el conjunto  $C$  de restricciones sobre  $Q$  para producir un QEP basado en remolinos. (b) Dé un grafo de unión  $G$  para  $Q$ . (c) Utilizando  $C$  y

$G$ , proponga un QEP basado en remolinos. (d) Proponga un segundo QEP que utilice módulos de estado. Analice las ventajas.

obtenido mediante el uso de módulos estatales en este QEP.

Problema 4.11 (\*\*\*) Proponga una estructura de datos para almacenar tuplas en el grupo de buffers de remolino para ayudar a elegir rápidamente la siguiente tupla a evaluar según la preferencia especificada por el usuario, por ejemplo, producir los primeros resultados antes.

# Capítulo 5

## Procesamiento de transacciones distribuidas



El concepto de transacción se utiliza en los sistemas de bases de datos como unidad básica de computación consistente y fiable. Por lo tanto, las consultas se ejecutan como transacciones una vez que se determinan sus estrategias de ejecución y se traducen en operaciones primitivas de base de datos. Las transacciones garantizan el mantenimiento de la consistencia y la durabilidad de la base de datos cuando se accede simultáneamente al mismo dato (al menos uno de estos es una actualización) y cuando se producen fallos.

Es necesario definir con mayor precisión los términos "consistente" y "confiable" en la definición de transacciones. Distinguimos entre consistencia de base de datos y consistencia de transacciones.

Una base de datos se encuentra en estado consistente si cumple con todas las restricciones de consistencia (integridad) definidas (véase el capítulo 3). Los cambios de estado se producen debido a modificaciones, inserciones y eliminaciones (denominadas en conjunto actualizaciones). Por supuesto, queremos asegurar que la base de datos nunca entre en un estado inconsistente. Cabe destacar que la base de datos puede ser (y suele serlo) temporalmente inconsistente durante la ejecución de una transacción. El punto importante es que la base de datos debe ser consistente cuando la transacción finaliza (Fig. 5.1).

La consistencia de las transacciones, por otro lado, se refiere a las operaciones de transacciones concurrentes. Queremos que la base de datos se mantenga consistente incluso si varias solicitudes de usuario acceden a ella (leen o actualizan) simultáneamente.

La confiabilidad se refiere tanto a la resiliencia de un sistema ante diversos tipos de fallos como a su capacidad para recuperarse de ellos. Un sistema resiliente tolera fallos y puede seguir prestando servicios incluso cuando estos ocurren. Un SGBD recuperable es aquel que puede alcanzar un estado consistente (regresando a un estado consistente anterior o avanzando a uno nuevo) tras diversos tipos de fallos.

La gestión de transacciones aborda la necesidad de mantener la base de datos en un estado consistente, incluso ante accesos concurrentes y fallos. Los problemas de la gestión de transacciones concurrentes son bien conocidos en los SGBD centralizados y se pueden encontrar en muchos libros de texto. En este capítulo, investigamos estos problemas.

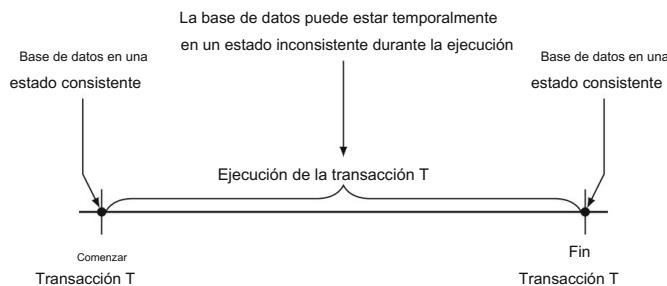


Fig. 5.1 Un modelo de transacción

El contexto de los SGBD distribuidos, centrado en el control de concurrencia distribuido, la confiabilidad y la recuperación distribuidas. Esperamos que el lector esté familiarizado con los conceptos y técnicas básicas de gestión de transacciones, comúnmente abordados en cursos y libros de bases de datos de pregrado. Ofrecemos un breve repaso en la Sección 5.1.

En el Apéndice C se ofrece una descripción más detallada de los conceptos fundamentales del procesamiento de transacciones. Por ahora, ignoraremos los problemas de replicación de datos; el siguiente capítulo se dedica a este tema. Los sistemas de gestión de bases de datos (SGBD) se clasifican generalmente como Procesamiento de Transacciones en Línea (OLTP) o Procesamiento Analítico en Línea (OLAP). Las aplicaciones de Procesamiento de Transacciones en Línea , como las de reservas de aerolíneas o los sistemas bancarios, están orientadas a transacciones de alto rendimiento. Requieren un amplio control y disponibilidad de datos, un alto rendimiento multiusuario y tiempos de respuesta rápidos y predecibles. Por el contrario, las aplicaciones de Procesamiento Analítico en Línea , como el análisis de tendencias o la previsión, necesitan analizar datos históricos resumidos procedentes de diversas bases de datos operativas. Utilizan consultas complejas sobre tablas potencialmente muy grandes. La mayoría de las aplicaciones OLAP no requieren las versiones más recientes de los datos y, por lo tanto, no requieren acceso directo a los datos operativos más actualizados. En este capítulo, nos centramos en los sistemas OLTP y los consideramos en el Capítulo 7.

Este capítulo se organiza de la siguiente manera. En la Sección 5.1, se ofrece una breve introducción a la terminología básica utilizada en este capítulo y se revisa el modelo arquitectónico definido en el Capítulo 1 para destacar las modificaciones necesarias para la gestión de transacciones. La Sección 5.2 profundiza en las técnicas de control de concurrencia distribuida basadas en la serialización, mientras que la Sección 5.3 aborda el control de concurrencia con aislamiento de instantáneas. La Sección 5.4 analiza las técnicas de confiabilidad distribuida, centrándose en los protocolos de confirmación, terminación y recuperación distribuidos.

## 5.1 Antecedentes y terminología

Nuestro objetivo en esta sección es proporcionar una breve introducción a los conceptos y la terminología que utilizaremos en el resto del capítulo. Como se mencionó anteriormente, nuestro

El objetivo no es proporcionar una descripción detallada de los conceptos fundamentales (que se pueden encontrar en el Apéndice C), sino introducir la terminología básica que será útil en el resto del capítulo. También analizamos cómo debe revisarse la arquitectura del sistema para adaptarla a las transacciones.

Como se indicó anteriormente, una transacción es una unidad de cálculo consistente y confiable. Cada transacción comienza con el comando `Begin_transaction`, incluye una serie de operaciones de lectura y escritura, y finaliza con una confirmación o una cancelación. La confirmación, al procesarse, garantiza que las actualizaciones realizadas por la transacción en la base de datos sean permanentes a partir de ese momento, mientras que la cancelación deshace las acciones de la transacción, por lo que, en lo que respecta a la base de datos, es como si la transacción nunca se hubiera ejecutado. Cada transacción se caracteriza por su conjunto de lectura (RS), que incluye los datos que lee, y su conjunto de escritura (WS), que incluye los datos que escribe. Los conjuntos de lectura y escritura de una transacción no tienen por qué ser mutuamente excluyentes. La unión de ambos constituye su conjunto base (BS = RS ∪ WS).

Los servicios de transacciones DBMS típicos proporcionan propiedades ACID:

1. La atomicidad garantiza que las ejecuciones de transacciones sean atómicas, es decir, todas las acciones de una transacción se reflejan en la base de datos o ninguna de ellas.
2. La consistencia se refiere a que una transacción sea una ejecución correcta (es decir, el código de la transacción es correcto y cuando se ejecuta en una base de datos que es consistente, la dejará en un estado consistente).
3. El aislamiento indica que los efectos de las transacciones concurrentes están protegidos entre sí hasta que se confirman: así es como se garantiza la exactitud de la ejecución concurrente de transacciones (es decir, la ejecución concurrente de transacciones no rompe la consistencia de la base de datos).
4. La durabilidad se refiere a aquella propiedad de las transacciones que garantiza que los efectos de las transacciones confirmadas en la base de datos sean permanentes y sobrevivan a las fallas del sistema.

Los algoritmos de control de concurrencia que analizamos en la Sección 5.2 aplican la propiedad de aislamiento para que las transacciones concurrentes vean un estado consistente en la base de datos y la mantengan en un estado consistente. Por otro lado, las medidas de confiabilidad que analizamos en la Sección 5.4 aplican la atomicidad y la durabilidad. La consistencia, en términos de garantizar que una transacción dada no afecte negativamente a la base de datos, se gestiona generalmente mediante el cumplimiento de la integridad, como se explica en el Capítulo 3.

Los algoritmos de control de concurrencia implementan el concepto de "ejecución concurrente correcta". El concepto de corrección más común es la serialización, que requiere que el historial generado por la ejecución concurrente de transacciones sea equivalente a un historial serial (es decir, una ejecución secuencial de estas transacciones). Dado que una transacción asigna un estado consistente de la base de datos a otro, cualquier orden de ejecución serial es, por definición, correcto; si el historial de ejecución concurrente es equivalente a uno de estos órdenes, también debe ser correcto. En la sección 5.3, presentamos un concepto de corrección más flexible denominado aislamiento de instantáneas (SI).

Los algoritmos de control de concurrencia se ocupan básicamente de imponer diferentes niveles de aislamiento entre transacciones concurrentes de manera muy eficiente.

Cuando se confirma una transacción, sus acciones deben hacerse permanentes. Esto requiere la gestión de registros de transacciones , donde se registra cada acción. Los protocolos de confirmación garantizan que las actualizaciones de la base de datos, así como los registros, se guarden en un almacenamiento persistente para que sean permanentes. Los protocolos de cancelación, por otro lado, utilizan los registros para borrar de la base de datos todas las acciones de la transacción cancelada. Cuando se requiere la recuperación tras fallos del sistema, se consultan los registros para que la base de datos alcance un estado consistente.

La introducción de transacciones en la carga de trabajo del DBMS junto con consultas de solo lectura requiere revisar el modelo arquitectónico presentado en el Capítulo 1. La revisión es una expansión del rol del monitor de ejecución distribuida.

El monitor de ejecución distribuida consta de dos módulos: un gestor de transacciones (TM) y un programador (SC). El gestor de transacciones se encarga de coordinar la ejecución de las operaciones de la base de datos en nombre de una aplicación.

El programador, por otro lado, es responsable de la implementación de un algoritmo de control de concurrencia específico para sincronizar el acceso a la base de datos.

Un tercer componente que participa en la gestión de transacciones distribuidas son los gestores de recuperación local (LRM) existentes en cada sitio. Su función es implementar los procedimientos locales que permiten recuperar la base de datos local a un estado consistente tras un fallo.

Cada transacción se origina en un sitio, al que llamaremos su sitio de origen.

La ejecución de las operaciones de base de datos de una transacción es coordinada por la TM en el sitio de origen de dicha transacción. Nos referimos a la TM en el sitio de origen como el coordinador o la TM coordinadora.

Un gestor de transacciones implementa una interfaz para los programas de aplicación con los comandos de transacción identificados anteriormente: Begin\_transaction, Read, Write, Commit y Abort. El procesamiento de cada uno de estos comandos en un SGBD distribuido no replicado se describe a continuación a nivel abstracto. Para simplificar, nos centraremos en la interfaz con la MT; los detalles se presentan en las siguientes secciones.

1. Inicio\_transacción. Esto indica a la TM coordinadora que se está iniciando una nueva transacción.

La TM lleva la contabilidad registrando el nombre de la transacción, la aplicación de origen, etc., en un registro de memoria principal (denominado registro volátil).

2. Lectura. Si el dato se almacena localmente, su valor se lee y se devuelve a la transacción. De lo contrario, la TM coordinadora busca dónde está almacenado y solicita la devolución de su valor (tras implementar las medidas de control de concurrencia adecuadas). El sitio donde se lee el dato inserta un registro en el registro volátil.

3. Escritura. Si el dato se almacena localmente, su valor se actualiza (en coordinación con el procesador de datos). De lo contrario, la TM coordinadora localiza el dato y solicita que la actualización se realice en ese sitio (tras implementar las medidas de control de concurrencia adecuadas). De nuevo, el sitio que ejecuta la escritura inserta un registro en el registro volátil.

4. Confirmación. La TM coordina los sitios involucrados en la actualización de datos en nombre de esta transacción para que las actualizaciones sean duraderas en cada sitio. El protocolo WAL se ejecuta para mover los registros volátiles a un registro en disco (denominado registro estable).
5. Abortar. La TM garantiza que ningún efecto de la transacción se refleje en ninguna de las bases de datos de los sitios donde actualizó los datos. El registro se utiliza para ejecutar el protocolo de deshacer (reversión).

Al proporcionar estos servicios, una TM puede comunicarse con los SC y los procesadores de datos. En el mismo sitio o en sitios diferentes. Esta disposición se muestra en la Fig. 5.2.

Como indicamos en el capítulo 1, el modelo arquitectónico descrito es solo una abstracción con fines pedagógicos. Permite separar muchos de los problemas de gestión de transacciones y analizarlos de forma independiente y aislada. En la sección 5.2, al analizar el algoritmo de planificación, nos centramos en la interfaz entre una MT y un SC, y entre un SC y un procesador de datos. En la sección 5.4, consideraremos las estrategias de ejecución de los comandos de confirmación y cancelación en un entorno distribuido, además de los algoritmos de recuperación necesarios para el gestor de recuperación. En el capítulo 6, ampliamos esta discusión al caso de bases de datos replicadas. Cabe destacar que el modelo computacional descrito aquí no es único. Se han propuesto otros modelos, como, por ejemplo, el uso de un espacio de trabajo privado para cada transacción.

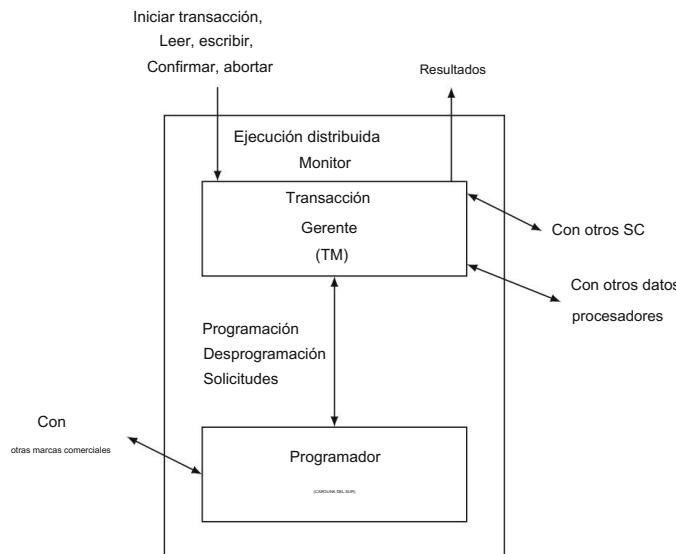


Fig. 5.2 Modelo detallado del monitor de ejecución distribuida