Лекция 14. Организация на данни, складове данни и OLAP технология

14.1. Увод

Една от характеристики, която отличава извличане на закономерности от данни от други типове аналитични задачи, е количеството на данни. В много от ИЗД приложения матрицата от данни, представяща по стандартен начин данните, съдържа милиони редове и хиляди колонки, така че въпросите за *ефективността* на алгоритмите за анализ на данните са от много съществено значение. Един алгоритъм, чието време за изпълнение нараства експоненциално с нарастване на броя n на редовете, може да бъде неизползваем за всички практически случаи, освен за много малки множества от данни. Примерите на операции, които могат да бъдат изпълнени във времето като $O(n)$ или $O(n*\log n)$, са изчисляването на честотите на срещане на данни, намирането на модата на някоя дискретна променлива или атрибут, както и сортирането на данни. В общия случай тези изчисления са осъществими дори за много големи масиви от данни. Обаче, дори един линеен по времето на изпълнение алгоритъм може да има много голяма цена при използването, ако се налага няколкократното сканиране на целия масив от данни.

Ако броят на редове n на едно множество от данни оказва съществено влияние на сложността на алгоритъма, същото важи и за броя на променливи (атрибути) p. За някои приложения p е много малко (например, е по-малко от 10), но за други, като, например, анализ на потребителски кошници или анализ на текстови документи, то може да стига до 10^5 или дори 10^6 атрибута. В подобни ситуации не могат да бъдат използвани операции, изискващи $O(p^2)$ изчисления, като например, мерки за корелации между всички двойки атрибути.

При работа над всеки ИЗД проект е полезно да се прави разграничение между две фази. По време на първата се подготвят данните за ИЗД алгоритъма, а втората се състои в пускането на самият алгоритъм. Първата фаза изглежда тривиална, обаче често точно тя се оказва "тясното място" в проекта. Например, при анализа на данни често е необходимо избраният ИЗД алгоритъм да бъде прилаган към множество различни подмножества от данните. Това означава, че ние трябва да сме в състояние бързо да намираме и да идентифицираме членове на всяко подмножество, както и да натоварваме това подмножество в основната памет. Примерите на подобни алгоритми са класификационни и регресионни дървета, които последователно разбиват множеството от данни на все по-малки и по-малки подмножества, всяко от което трябва да бъде идентифицирано преди нарастването на дървото. Целта на *организацията на данни* е да бъдат намерени такива методи за съхраняване на данни, които позволяват колко може по-бърз достъп до различни подмножества от данни. Дори когато всички данни се намират в основната памет, организацията на данните остава от голямо значение.

Освен осигуряването на ефективен достъп до данни за ИЗД алгоритми, методите за организацията на данни са много важни за осигуряване на

итеративния и интерактивния характер на целия процес за извличане на закономерности от данни. В настоящата лекция ще разгледаме накратко йерархията на паметта на съвременните компютри, след което ще се запознаем със някои индексни структури, използвани от системите за управление на бази данни за ускоряване на оценката за заявки. След това ще дискутираме бази данни, складове данни и системи за тяхното управление.

14.2. Йерархията на паметта

Паметта на компютъра е разбита на няколко нива. Тези нива имат различно време за достъп, т.е. средното време за извличане на случайно избран байт от паметта. Различни структури на паметта могат да бъдат категоризирани по следния начин:

1. *Регистри на процесора.* Обикновено, те са по-малко от 100 и процесорът може да има директен достъп до данните, съхранявани в тях; т.е. няма никакво забавяне, асоциирано с достъпа към някой от регистрите.
2. *Кеш-памет на процесора (on-processor) или на дънната платка (on-board)*. Това е бърза полупроводникова памет, реализирана върху същия чип, като процесора, или разположена върху дънната платка. Типичният размер на тази памет е 16 – 2000 килобайт, а времето за достъп е около 20 ns.
3. *Основната памет.* Това е нормална полупроводникова памет с размер от 16 MB до няколко GB и времето за достъп от около 70 ns.
4. *Дискова кеш-памет.* Това е полупроводникова памет, реализирана като междинно хранилище между основната и дискова памети.
5. *Дискова памет.* Размерите са варират от 1 GB до стотици и дори хиляди гигабайта за големи масиви от дискове. Типичното време за достъп е около 10 ms.
6. *Памет на магнитна лента.* Магнитната лента може да съхранява до няколко терабайта данни. Времето за достъп се варира, но може да бъде от порядъка на минути.

Разликите между времената за достъп са действително големи: за тези 10 ms, необходими за достъп към диска, ние можем да изпълним до един милион достъпи към бързата кеш-памет. Друг начин да преценим това е да представим, че времето за достъп е линейно пропорционално на истинското разстояние. Тогава, ако представим разстоянието до основната памет да е 1 метър, то еквивалентното разстояние до дисковата памет е от порядъка на 100 км!

Другата основна разлика между основната и дисковата памети е че в основната памет имаме достъп до всеки отделен байт, докато при опит за достъп до един байт в дисковата памет в действителност в основната памет ще се товари цялата дискова страница – около 4 килобайта. Така че, ако се случи, че тази страница съдържа информацията, която трябва да бъде натоварена по-късно, тя в действителност вече се намира в бързата памет. Например, ако искаме да извлечем 1000 цели числа, всеки с размер по 4 байта, това ще изисква от 1 до 1000 достъпа към диска, в зависимост от това, дали тези числа са съхранявани върху една и съща дискова страница или всеки от тях – на отделна страница.

Физическите свойства на описаната йерархия на паметта водят до следното евристично правило:

- Ако е възможно, данните да се съхраняват в основната памет.
- В основната памет тези елементи от данни, които се използват заедно, да бъдат логически близки един до друг (т.е. ние трябва да можем бързо да намираме следващия елемент от някое подмножество).
- В дисковата памет тези елементи от данни, които се използват заедно, да бъдат също така физически близко разположени един до друг (т.е. ако е възможно, на една и съща дискова страница).

На практика, потребителят на една система практически има много малък контрол върху начина, по който данните се поместват в кеш-памети или върху действителното физическо разположение на данни върху диска. Обикновено, системите опитват да натоварят в основната памет колкото е възможно повече данни и сами решават как да поместват данни върху дискови страници. Потребителят може да влияе върху видове на допълнителни структури, които се създават за достъпа към подгрупи от данни. В следващия раздел ще разгледаме накратко някои от такива структури данни, използвани за достъп към големи масиви данни.

14.3. Индексни структури

Основната цел на методи за организация на данни е намирането на начини за бързото локализиране на всички точки (елементи) от данни, които удовлетворяват на зададеното условие за избор. Обикновено, условието за избор е конюнкцията от условия върху отделни атрибути, от типа на "възраст ≤ 40" и "доход ≤ 20000". Отначало, ще разгледаме структурите на данни, които са пригодени към ситуациите, съдържащи само един конюнкт.

Индексът на атрибут *A* е такава структура на данни, която позволява локализацията на точки с дадената стойност на атрибута да бъде извършена по-ефективно отколко при последователното сканиране на цялото множество от данни.

14.3.1. B-дървета

Дървото на търсене е най-простата индексна структура. Да предположим, че имаме множество *S* от векторни данни {**x**(1), ... , **x**(n)} и искаме да намерим колкото може по-бързо всички точки, които имат определена стойност на някой непрекъснат атрибут *A*. Дървото на търсене е едно двоично дърво, в който всеки възел има някаква запомнена стойност на атрибута A, а всяко листо има указател към някой елемент от *S*. Освен това, дървото е структурирано по такъв начин, че всички елементи от *S*, на които сочат указателите в листата от лявото поддърво на някой възел *u*, който съдържа стойността *a* на атрибута *A*, ще имат стойностите на *A*, които са по-малки или равни на *a*. Аналогично, всички

елементи от *S*, на които сочат указатели в листата от дясното поддърво на *u*, ще имат стойностите на *A* по-големи от *a*.

При създаденото двоично дърво на търсене за атрибута *A*, е лесно да бъде намерени данни от *S*, които имат желаната стойност *b* на атрибута *A*. За целта трябва да се започне търсенето от корена на дървото, избирайки лявото или дясното под-дърво чрез сравнение на *b* със стойностите, запомнени в съответните възли на дървото. При достигането на някое листо, ние или намираме указателят към запис(и) с $A = b$, или установяваме, че подобен указател не съществува.

По същия начин е лесно да се намерят всички точки от S, които удовлетворяват условието $b \leq A \leq c$ - така наречена "интервална заявка". За целта локализираме листото, където трябва да бъде *b* (по описания по-горе начин), локализираме листото, където трябва да бъде *c*, и желаните записи ще са тези, на които сочат указатели, разположени в листата между тези две позиции.

Времето, необходимо за намиране на записите със зададената стойност на атрибута *A*, е пропорционално на височината на дървото плюс броя на всички такива записи. В най-лошия случай височината на дървото е *n* - броя на точките в множеството *S*, но винаги съществуват начини, да осигуриш, че височината на дървото ще бъде $O(\log n)$. На практика, двоичните дървета на търсене се използват относително рядко, тъй като съществуват други B^+-дървета, които са доказано по-добри за достъпа към данни, разположени върху диска.

Основната идея на B^+-дървета е същата, като за дървета на търсене: указателите към обектите на данни се пазят в листата на дървото, а междинните възли съдържат стойности на атрибута *A*, които указват, къде трябва да се намират определени указатели. Обаче, вместо двата наследника на една стойност на *A* за всеки възел, едно B^+-дърво обикновено има стотици наследници и стойности.

По-точно, едно B^+-дърво от степен *M* за някое множество от стойности е дървото, в който:

- Всички листа се намират на една и съща дълбочина;
- Всяко листо съдържа между $M/2$ и M ключове (възможни целеви стойности);
- Всеки междинен възел (с възможно изключение на корена) имат *K* наследника $C_1, \ldots, C_K$, където $M/2 \leq K \leq M$ и $K - 1$ стойности $a_1, \ldots, a_{K-1}$; за всички *i,* ключовите стойности, съхранявани в листата на поддърво C_i, са по-големи от a_{i-1} и са по-малки или равни на a_i.

Търсенето в B^+-дървото се осъществява по същия начин, както и в двоичното дърво на търсене: при всеки междинен възел на дървото стойностите a_i се ползват за избор на коректното поддърво.

B^+-дърво се отличава от базисното двоично дърво на търсене по това, че неговата височина е гарантирано $O(\log n)$, тъй като всички листа са на една и съща дълбочина. В действителност, дълбочината на дървото е ограничена от $\log_{M/2} n$. Обикновено стойността на *M* се избира по такъв начин, че всеки възел в

B^+-дървото отговаря на една дискова страница. Ако M = 100, то $(M/2)^5$ е около 300 милиона, така че за най-реалистични стойности на n – броя на елементи в множеството, дървото ще има най-много 5 нива. Това означава, че намирането на някаква точка данни измежду 300 милиона точки на базата на стойността на един единствен атрибут, може да се осъществи чрез три достъпа до диска, като се смята, че коренът и второто ниво на дървото могат да се пазят в основната памет. Повечето от системи за управление на бази данни използват на B^+-дърво като една от техните индексни структури.

14.3.2. Хеш-индекси

Да предположим отново, че имаме множеството от S точки данни и искаме да намерим всички точки, за които стойността на атрибута *A* е *a*. Ако множеството от възможни стойности на *A* е малко, можем да направим следното: за всяка възможна стойност *a* да конструираме списък от указатели към точки от данни, за които $A = a$. Тогава, за да отговорим на заявката "Намери точки с $A = a$" трябва само да осъществим достъп към списъка за *a*.

Този метод, обаче, не е приложим, ако съществува голям брой потенциални стойности на *A*: например, ние не можем да поддържаме по един списък за всяко от възможни 2^{32} цели числа, които могат да бъдат представени чрез 32 бита. Това, което можем да направим, е да приложим към стойностите на *A* една трансформация, която намалява диапазон на възможни стойности.

Нека *Dom(A)* е множеството от възможни стойности на *A*. *Хеш-функция* е една функция $h: Dom(A) \rightarrow \{1, ..., M\}$, където *M* е размерът на хеш-таблицата *r*. За всеки $j \in \{1,...,M\}$ ще пазим в *r[j]* списъка от указатели към тези записи $\mathbf{x}_i$ от *S*, за които стойността a_i на атрибута *A* удовлетворява условието $h(a_i) = j$. Когато искаме да намерим всички точки данни с $A = a$, трябва просто да изчислим *h(a)*, да отидем в клетката *r[h(a)]* и да прегледаме посочения там списък от точки, проверявайки за всяка от тях дали стойността на *A* е действително равна на *a* или тя е равна на някакво друга стойност *b* със свойството $h(b) = h(a)$ (това се нарича *колизия*).

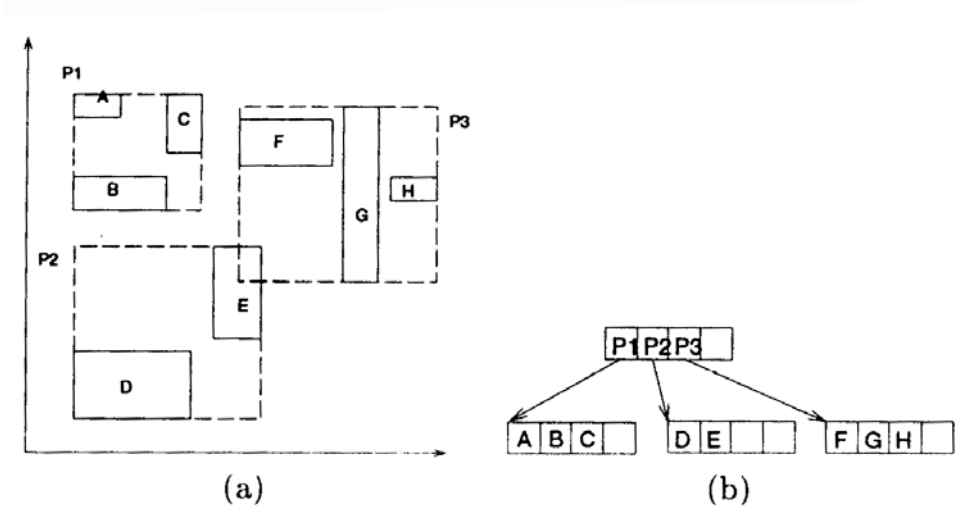
Една типична хеш-функция е $a \bmod M$, където *M* се избира да е подходящо просто число по-голямо от *n* – брой на точки данни. Ако хеш-функцията е избрана добре и хеш-таблицата е достатъчно голяма, то колизиите се случват рядко и търсенето на точки със зададената стойност на *A* може да се осъществи за времето пропорционално на броя на такива точки. Обаче, хеш-индексите не поддържат директно интервалните заявки.

14.3.4. Многомерно индексиране

Традиционните индексни структури, такива като хеширане и B^+-дървета, осигуряват бърз достъп към редове на таблици на базата на стойности на зададен атрибут или набор от атрибути. Обаче, в някои приложения е необходимо да се изразят условията за избор на базата на няколко атрибута – в този случай нормалните индексни структури не помагат. Да разгледаме, като пример, случай със съхраняване на географската информация за градове. Да

предположим, че искаме да намерим всички населени места, разположени между 30 и 40 градуса северната ширина и между 60 и 70 градуса западната дължина, които имат население не по-малко от 1000 души. Подобната заявка се нарича *заявка с правоъгълен диапазон (rectangular range query)*. Да предположим, че таблицата на градове е голяма и съдържа милиони имена. Как трябва да бъде изпълнена такава заявка? Индексът за стойностите на ширина, представен във вид на B^+-дърво, позволява да бъдат намерени всички градове, удовлетворяващи условието за този атрибут, обаче намирането сред тях на редове, удовлетворяващи на останалите условия от заявката, трябва да бъде реализирано само чрез последователно сканиране. Аналогично, индексните структури за географска дължина или броя на населението, също няма много да помогнат. Това, което ни е трябва, е такава индексна структура, която позволява директно използване на условията върху всички атрибути.

Многомерното индексиране се използва за означаване на точно такива техники за намиране на редовете от таблици на базата, отговарящи на условията върху няколко атрибута. Един от най-широко разпространени методи е R-дърво. При този метод всеки многомерен обект (точка данни) се представя чрез свой *най-малък ограничаващ правоъгълник (MBR – minimum bounding rectangle)*. Всеки възел в такова дърво съответства на една област в многомерното пространство (MBR на всички наследници на възела) и съдържа указатели към свои наследници в дървото. Листата на дървото съдържат указатели към съответния обект и неговият MBR. Най-голямото достижение на R-дърво се състои в това, че се допуска препокриване на междинните възли. По този начин дървото може да гарантира добро използване на пространството и да остава добре балансирано. На Фиг. 14-1 е показано примерното R-дърво с фактор на разклоняване 4 (т.е. 4 е максимален брой на непосредствени наследници на възела) и неговото представяне върху диска.



Фиг.14-1. а) Данни (правоъгълници, оградени с плътни линии), организирани в R –дърво. b) Файловата структура на същото дърво, в който възлите съответстват на дискови страници.

За бази данни до 10 размерности, многомерното индексиране позволява значително да ускори търсенето на отговори на заявки с правоъгълен диапазон. Бързото изпълнение на подобни заявки за бази данни с голям брой размерности (от порядъка на стотици) е все още открит въпрос на научните изследвания.

14.4. Какво е склад на данни?

Методите за създаване на складове данни (data warehousing) предоставят на ръководителите на организации архитектури и инструменти за систематичната организация, разбиране и използване на техните данни за приемане на стратегически решения. Понятието "склад данни" означава една база данни, която се обслужва отделно от операционни бази данни на дадената организация. Съгласно W.H. Inmon, един от водещите архитекти в конструирането на системи за управление на складове данни, "*склад данни е една тематично ориентирана, интегрирана, отчитаща време и сравнително неизменна колекция от данни, предназначена за поддръжка на процеса на взимане на управленческите решение*".

- *Тематично ориентиран*. Един склад данни е организиран около няколко основни теми, като например, купувач, производител, продукт и продажби. Вместо да се концентрира на ежедневни операции и обработка на транзакции (продажби) в една организация, складът данни се фокусира върху моделиране и анализ на данни, предназначени за взимането на решения. Следователно, складовете данни обикновено предоставят един ясен и сбит поглед на определени аспекти от дадената тема чрез отхвърляне на данни, които не се използват в процеса на приемане на решения.
- *Интегриран.* Един склад данни обикновено се конструира чрез интегриране на множество хетерогенни източници, такива, като релационни бази данни, обикновени "плоски" файлове и записи за интерактивни транзакции. Към тях се прилагат техниките за изчистване и интегриране на данни, за да осигури непротиворечивостта на използвани имена, кодиращи структури, мерителните мерки на атрибути и т.н.
- *Отчитащ време.* Данните се съхраняват, за да осигури информацията в историческата перспектива (например, за последните 5 – 10 години). Всяка ключова структура в склада данни съдържа, явно или неявно, някой времеви елемент.
- *Сравнително неизменен.* Един склад данни винаги е направен като отделно хранилище за данни, получени чрез трансформиране на някои приложни данни, намерени в операционната среда. Поради тази отделеност складът данни не изисква наличието на механизми за обработка на транзакции, възстановяване и управление на конкурентен достъп. Обикновено той изисква само две операции при достъпа към данни: *начално натоварване на данни* и *достъп към данни*.

И така, складът данни е едно хранилище на семантично непротиворечиви данни, които служат като една физическа реализация на модела на данни за вземане на решения и съхранява информация, необходима за една организация за взимане на стратегически решения. Повечето организации използват съхраняваната в складове данни информация за поддръжка на дейности, свързани с приемането на решения, целящи:

- По-голямо фокусиране върху покупателя, което включва анализ на покупателните шаблони на клиенти (такива, като предпочитания при купуване, времето на пазаруване, бюджетни цикли, диапазон на харчене и т.н.).
- Преразпределяне на продукти и управление на предлагания диапазон на продукти чрез сравнение на поведението на продажби по квартали, години и по географски райони, с цел подобряване на производствените стратегии.
- Анализиране на операции и търсене на източници на печалба.

Създаване и използване на складове данни е също така много удобно от гледната точка на *интеграция на хетерогенни бази данни*. Повечето организации обикновено събират множество разнообразни данни и поддържат големи бази данни от различни, хетерогенни, автономни и разпределени източници на информация. Интегриране на такива данни и осигуряване на лесен и ефективен достъп до тях е една много предизвикателна задача за решаване на която бяха хвърлени много усилия както от страна на изследователска, така и от страна на индустриалната общност от специалисти, работещи по бази данни.

Традиционният подход към интегрирането на хетерогенни бази данни се състои в създаване на така наречени обвивки (wrappers) и интегратори (или медиатори – посредници) на върха на множествени, хетерогенни бази данни (например IBM Data Joiner и Informix DataBlade). При формулиране на някоя заявка към сайта на клиента се използва речникът на метаданни, за транслиране на заявката в колекцията от заявки, подходящи за отделни хетерогенни сайтове. След това тези заявки се изпращат към локални процесори на заявките. Резултатите, върнати от различни сайтове, се интегрират в едно глобално множество от отговори. Този *воден от заявката подход* изисква реализация на сложни процеси по филтриране и интегриране на информация и конкурирането на локални процеси по обработка на информация. Той е неефективен и потенциално много скъп за често използвани заявки, особено за заявки, изискващи агрегацията.

Една интересна алтернатива към описания по-горе подход предлага използването на складовете данни. Вместо използване на водения от заявката подход, складовете данни прилагат *подход, воден от обновяване* (update-driven), в който информацията от множество хетерогенни източници се интегрира предварително и се запазва в склада данни за формиране на директен отговор на заявки и анализиране. В отличие от бази данни, интерактивно обработващи транзакции, складовете данни не съдържат текуща (“най-прясна”) информация. Обаче, складовете данни осигуряват на интегрирани хетерогенни системи за бази данни едно високоефективно поведение, тъй като данните се копират, подлагат се на предварителна обработка, интеграция, анотация и резюмиране, както и се преструктурират, превръщайки се в едно семантично хранилище на данни. Освен това, обработката на заявки в складовете данни не се намесва в обработката, извършвана в локални източници. Повече от това, складовете данни могат да съхраняват и интегрират историческата информация и да поддържат сложни многомерни заявки. Като резултат, складовете данни станаха много популярни в съвременната индустрия.

14.4.1. Разлики между оперативни системи за управление на бази данни и складове данни

Главната задача за интерактивни (on-line) оперативни системи за управление на бази данни е изпълнение на интерактивни транзакции и обработка на заявки. Такива системи се наричат *системи за обработка на интерактивни транзакции* (*OLTP – on-line transaction processing* systems). Системи за управление на складове данни служат на инженери по знания за анализ на данни и вземане на стратегически управленски решения. Такива системи се наричат *системи за интерактивна аналитична обработка* (*OLAP – on-line analytical processing* systems). Основни разлики между OLTP и OLAP системи могат да бъдат резюмирани по следния начин:

- *Ориентация на потребители и система*: OLTP-система е *ориентирана към покупателя* и се използва за обработка на транзакции и заявки от чиновници, клиенти и ИТ-професионалисти. OLAP-система е ориентирана към пазара и се използва за анализ на данни от менъджери, управители и анализатори.
- *Съдържание на данни*: OLTP-система управлява текущи данни, които, обикновено, са прекалено подробни, за да бъдат лесно използвани за вземане на решения. OLAP-система управлява голям обем исторически данни, предоставяйки възможности за обобщение и агрегация, както и съхранява и управлява информация на различни нива на абстракцията. Тези характеристики облекчават използване на данни в процеса на вземането на решения.
- *Дизайн на база данни:* OLTP-система обикновено използва обектно-ориентиран модел на данни и ориентиран на приложения дизайн на бази данни. OLAP-система обикновено използва или модел *звезда*, или модел *снежинка* (ще ги разгледаме по-късно в тази лекция) и тематично-ориентиран дизайн на бази данни.
- *Изглед:* OLTP-система се фокусира основно на текущите данни на някоя организация или отдел, без справки към исторически данни или данни от други организации. OLAP-система често обхваща няколко версии на една схема да база данни, което отразява процеса на развитие на организацията. OLAP-системи също така работят с информация, организирана от различни организации или интегрира информация от различни хранилища на данни. Поради своят огромен обем OLAP-данни се съхраняват на множество носители на данни.
- *Шаблони за достъп:* за OLTP-системи те са състоят основно от къси, прости транзакции. Подобни системи се нуждаят от механизми за управление на конкурентен достъп и възстановяване. Шаблоните за достъп при OLAP-системи са основно операции само за четене (тъй като повечето от складове данни съхраняват не текущата, а историческата информация), макар че много от тях могат да представляват сложни заявки.

14.5. Модел на многомерни данни

14.5.1. Кубоиди и кубове данни

Складовете данни и OLAP-инструменти се базират върху определен *модел на многомерни данни*. Този модел разглежда данни като един *куб на данни*. Кубът на данни позволява данни да бъдат моделирани и разглеждани по няколко размерности. Той се дефинира чрез размерности и факти.

Размерностите са множеството от перспективи или обекти, от гледна точка на които една организация иска да съхранява своята информация. Например, една фирма за търговия с електроника AllElectronics, която ще използваме за примери в лекциите, може да създаде склад за данни за *продажби (sales)*, за да пази в него записи за продажби по отношение на такива размерности като *време, артикул, клон и местоположение (time, item, branch, location)*. Всяка размерност може да има асоциираната с нея таблица, наречена таблицата на размерност, която описва дадената размерност. Например, таблицата на размерност "артикул" може да съдържа атрибути "име-на-артикула" (item_name), "вид" (brand) и "тип" (type). Таблиците на размерностите могат да бъдат описани от потребители или експерти, или да се генерират автоматично на базата на разпределението на данни.

Един модел на многомерни данни обикновено е организиран около някоя централна тема (например "продажби"). Тази тема е представена чрез така наречена *таблица на факти*. *Фактите* са определени числени мерки. Те могат да се разглеждат като количествата, чрез които искаме да анализираме съществуващи релации между размерностите. Примерите на факти за складът на данни за продажби са "продажби-в-долари" (amount_sold) и "продадени-бройки" (unit_sold). *Таблицата на факти* съдържа имената на факти или мерки, както и ключове към всяка от съответни таблици на размерностите.

location = "Vancouver"

	item (type)			
time (quarter)	home entertainment	computer	phone	security
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

Фиг. 14.2. 2-мерен поглед на данните за продажби на фирмата AllElectronix в съответствие с размерностите "време" и "артикул", като продажбите са от клонове, разположени във Ванкувер. Използваната мярка е "продажби-в-долари" (в хиляди).

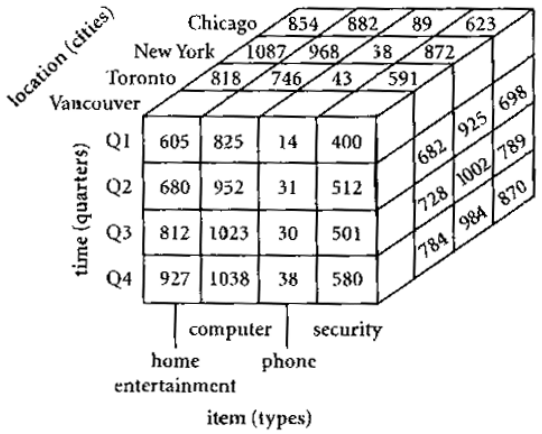
Като пример да разгледаме един двумерен (2-D) куб данни, който на практика е една таблица за продажби на "нашата" фирма (Фиг. 14.2). В частност, тя съдържа данни за артикули, продадени във Ванкувер (размерност "артикул"), организирани съгласно своите типове, и разбити по тримесечия (quarters) – размерност "време". Показаният факт или мярка е "продажби-в-долари" (в хиляди).

Да предположим, че искаме да разгледаме данните за продажби с използване на третата размерност – "местоположение" за такива градове, като Чикаго, Ню Йорк, Торонто и Ванкувер. Този тримерен куб на данни е показан в таблицата на Фиг. 14.3.

	location = *"Chicago"*				***location*** = *"New York"*				***location*** = *"Toronto"*				***location*** = *"Vancouver"*			
	item				***item***				***item***				***item***			
time	*home ent.*	*comp.*	*phone*	*sec*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp.*	*phone*	*sec.*	*home ent.*	*comp*	*phone*	*sec.*
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

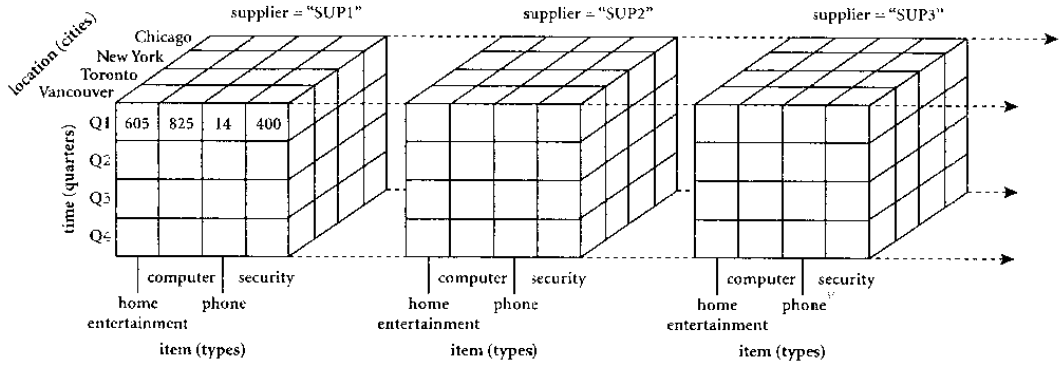
Фиг. 14.3. 3-мерен поглед на данните за продажби на фирмата AllElectronix в съответствие с размерностите "време", "артикул" и "местоположение". Използваната мярка е "продажби-в-долари" (в хиляди).

Тримерните данни от тази таблица могат да бъдат представени като една серия от двумерни таблици. По-принцип, същите данни могат да бъдат представени като един тримерен куб, показан на Фиг. 14.4.



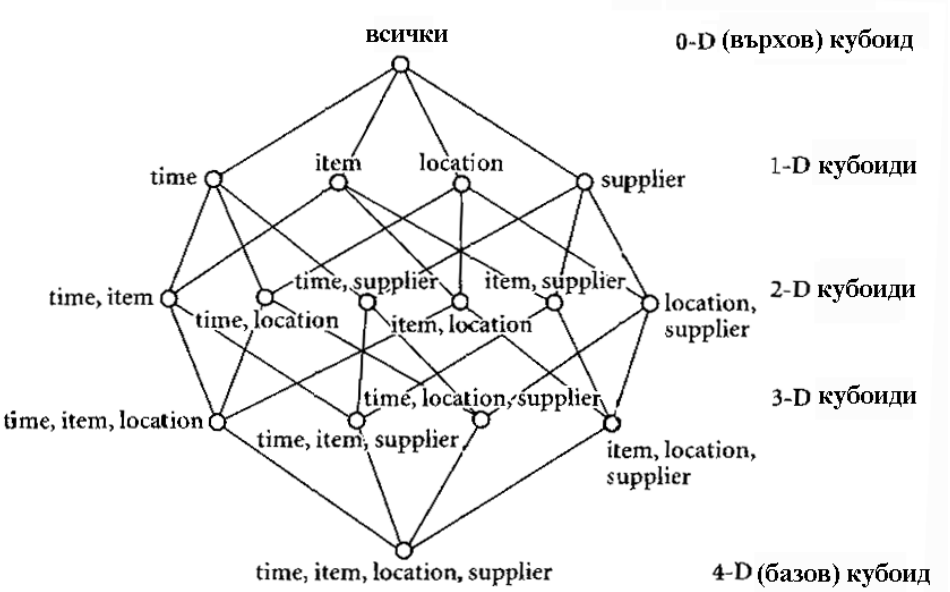
Фиг. 14.4. Представянето на табличните данни от Фиг. 2 във вид на тримерен куб данни с размерностите "време", "артикул" и "местоположение". Използваната мярка е "продажби-в-долари" (в хиляди).

Сега да предположим, че искаме да разгледаме нашити данни за продажби с допълнителната четвърта размерност – "снабдител" (supplier). Визуалното представяне на данни в четири размерности е доста сложно, обаче можем да представим един четиримерен куб като определена серия от тримерни кубове, както това е показано на Фиг. 14.5. По същия начин, един n-мерен куб може да се представи като серия от (n-1)-мерни кубове. Кубът на данни е една удобна метафора за хранилища на многомерни данни. Действителното физическо хранилище на такива данни може да се отличава от тяхното логическо представяне.



Фиг. 14.5. Представянето на 4-мерен куб данните за продажби в съответствие с размерностите "време", "артикул", "местоположение" и "снабдител". Използваната мярка е "продажби-в-долари" (в хиляди). За да подобри разбираемостта, на рисунката са показани само някои от стойностите на куба.

На показаните по-горе таблици данните са представени на различни нива на обобщение. В изследователската литература подобни кубове данни се наричат *кубоиди*. По зададеното множество от размерности е възможно да бъде построена една *решетка (lattice) от кубоиди*, всеки от които показва данни на различно ниво на детайлизацията (т.е. обобщени чрез различно подмножество от размерностите). Тази решетка от кубоиди се нарича куб данни. На Фиг. 14.6. е показана решетката от кубоиди за размерностите "време", "артикул", "местоположение" и "снабдител".



Фиг. 14.6. Решетка от кубоиди, формираща четиримерен куб данни с размерности "време", "артикул", "местоположение" и "снабдител". Всеки кубоид представлява различна степен на обобщение.

Кубоидът, който съдържа данни на най-низкото ниво на обобщение, се нарича *базов кубоид*. Например, четиримерен кубод от Фиг. 14.5 е базовият кубоид за размерностите "време", "артикул", "местоположение" и "снабдител". Фиг. 14.4 представлява един тримерен (небазов) кубоид за размерности "време", "артикул" и "местоположение", обобщаващ данни за всички снабдители. Нуламерен кубоид, който съдържа данни за най-високо ниво на обобщение, се

нарича *върхов (apex) кубоид*. В нашия пример това са всички продажби или "продажби-в-долари", обобщени по всички четири размерности. Върховият кубоид обикновено се означава чрез "*всичко" (all)*.

14.5.2. Модели на многомерни бази данни

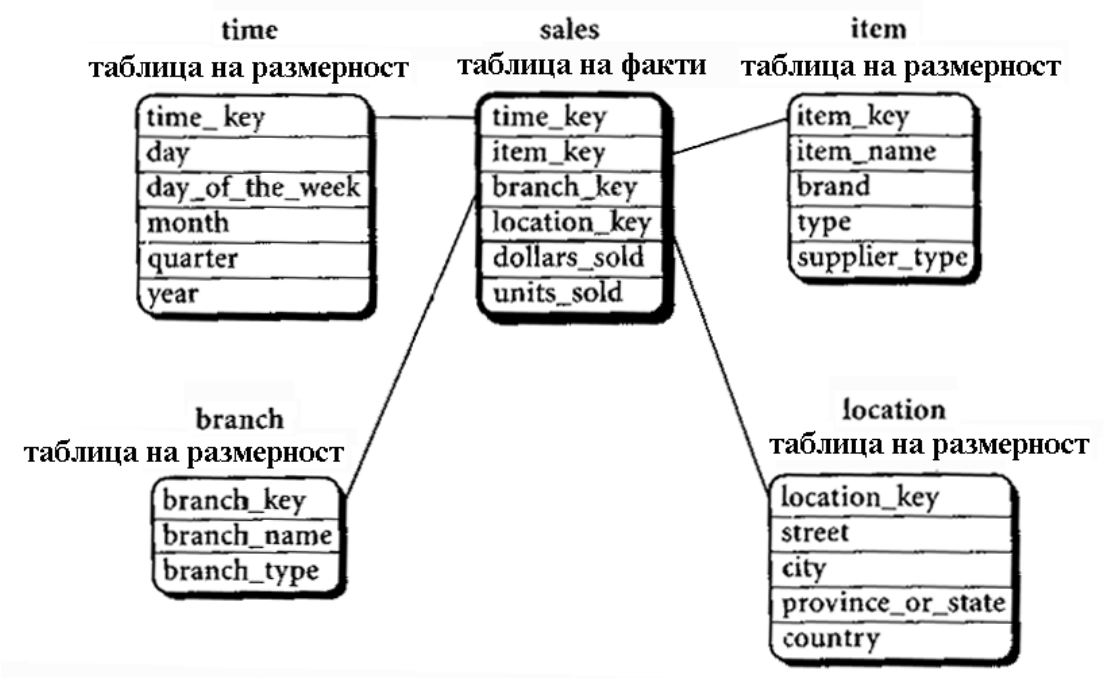
При дизайна на релационни бази данни най-често се използва моделът на данни от типа "обекти-релации", в който схемата на базата данни се състои от множество обекти и релации между тях. Този модел данни е подходящ за интерактивна обработка на транзакции. Дизайнът на склад данни изисква една сбита, тематично-ориентирана схема, улесняваща интерактивен анализ на данни.

Най-популярният модел данни за складове данни е така наречените *многомерен модел*. Този модел може да съществува във вид на *схема "звезда", схема "снежинка" или схема "съзвездие от факти"*.

Схема "звезда": Най-често използвана моделна парадигма е схемата "звезда", в която складът на данни се състои от:

1. Една голяма централна таблица (*таблицата на факти*), описваща обема на данни без никакво повторение.
2. Едно множество от по-малки съпътстващи таблици (*таблици на размерности*) – по една за всяка размерност.

Графът на схемата прилича на една звезда, в която таблиците на размерности са разположени като лъчи на звездата около централно разположена таблица на факти.

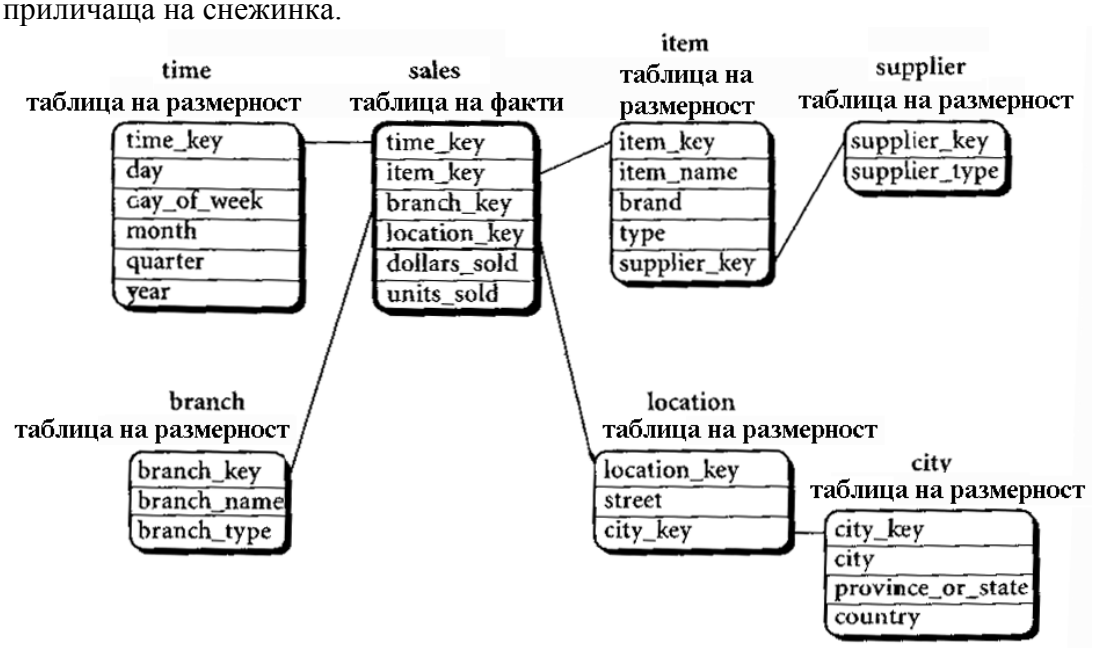


Фиг. 14.7. Схемата "звезда" за един склад с данни за продажби.

Един пример на схемата “звезда” за продажбите на фирмата All Electronix е показан на Фиг. 14.7. Продажбите се разглеждат относително четири размерности – време, артикул, клон и местоположение. Схемата съдържа една централна таблица на факти за *продажби,* която съдържа ключове към всяка от четири таблици на размерности, както и две мерки: “продажби-в-долари” и “продадени-бройки”. За да минимизира размера на таблицата на факти, идентификатори на размерностите (такива, като “ключ-време” (time_key) и “ключ-артикул” (item_key) се генерират от системата.

Обърнете внимание, в схемата “звезда” всяка размерност е представена само от една таблица, която съдържа по едно множество от атрибути. Например, таблицата на размерност “местоположение” (location) съдържа множеството от атрибути {location_key, street, city, province_or_state, country}. Атрибутите в една таблица на размерност могат да бъдат организирани в йерархия (т.е. пълна наредба) или в решетка (т.е. непълна подредба).

Схемата “снежинка”: Тази схема е вариант на схемата “звезда”, в която някои от таблиците на размерности са *нормализирани*, разбивайки по този начин данните на допълнителни таблици. Полученият граф на схемата има форма, приличаща на снежинка.



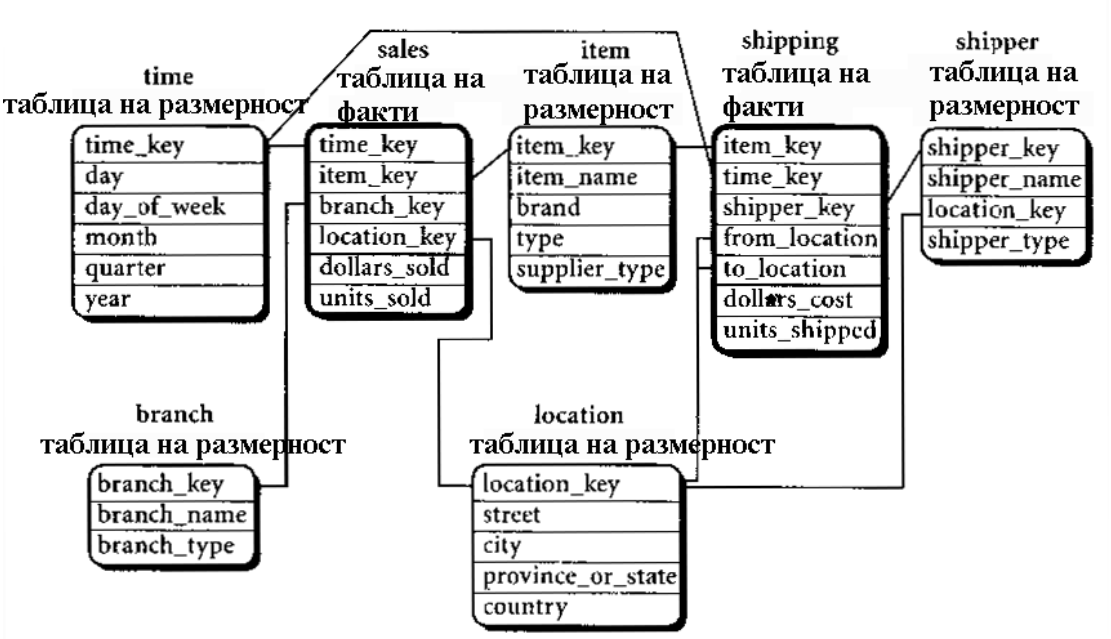
Фиг. 14.8. Схема “снежинка” за склад на данни за продажби.

Основната разлика между схемните модели “снежинка” и “звезда” е, че таблиците на размерности на модела “звезда” могат да бъдат съхранени в нормализирана форма с цел намаляване на излишествата. Такава една таблица е по-лесна за обработка и запазва пространство в паметта, тъй като една голяма таблица на размерности може да стане ненормално голяма, когато структурата на размерност се включва като колонки. Обаче, обем на спестеното пространство е много малък в сравнение типичния размер на таблицата на факти. Освен това, тази структура може да намали ефективността на търсене, тъй като за изпълнение на една заявка ще се потрябва изпълнение на повече операции по съединение и с това да намали общата ефективност на системата.

По тази причина, схемата “снежинка” не е толкова популярна при дизайн на складове данни като схемата “звезда”.

Един пример на схемата “снежинка” за данните за продажби на AllElectronics е показан на Фиг. 14.8. Тука таблицата на факти “продажби” е същата, както и при схемата “снежинка”. Основната разлика между двете схеми е в дефинициите на таблиците за размерности. Една единствена таблица за размерност “артикул” в схемата “звезда” е нормализирана в схемата “снежинка”, водейки до създаване на новите таблици “артикул” и “снабдител”. Новата таблица за размерност “артикул” сега съдържа атрибутите item_key, item_name, brand, type и supplier_key, където supplier_key е съединен с таблицата на размерността “снабдител”, съдържаща атрибути supplier_key и supplier_type. По същия начин, единствената таблица за размерност “местоположение” в схемата “звезда” сега е разбита на две нови таблици: “местоположение” и “град”. Атрибутът city_key в новата таблица за размерност “местоположение” съединява тази таблица с новата размерност “град”.

Схема “съзвездие от факти”: Някои сложни приложения могат да изискват няколко таблици от факти да имат *общи* таблици на размерности. Този вид на схема може да се разглежда като една колекция от звезди и, по тази причина, е наречена *схемата на галактика* или “*съзвездие от факти*”.



Фиг. 14.9. Схема “съзвездие на факти” за склада на данни за продажби и превоз.

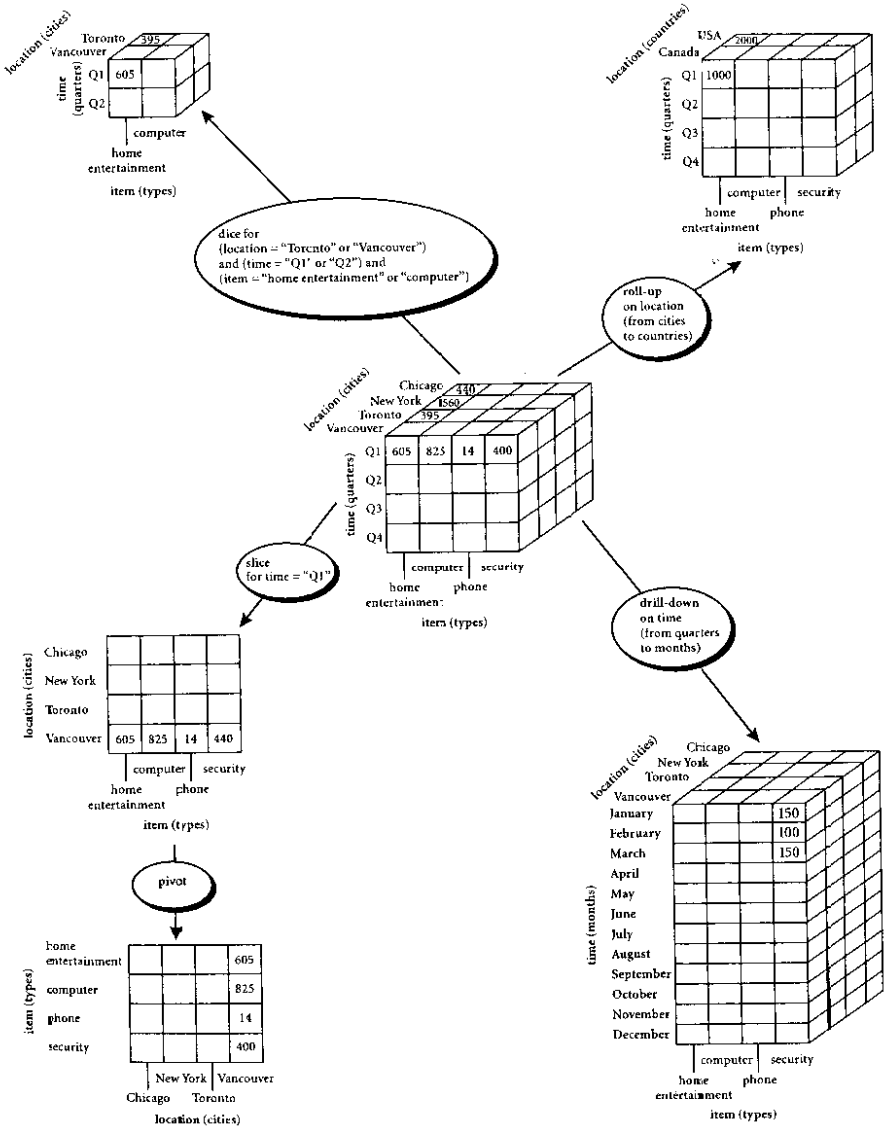
Един пример на схемата “съзвездие от факти” е показан на Фиг. 14.9. Тя специфицира две таблици от факти – “продажби” и “превоз” (shipping). Дефинициите на таблицата за продажби са същите, както и в по-рано разгледаната схема “звезда”. Таблицата за превози има пет размерности или ключа: *item_key, time_key, shipper_key, from_location* и *to_location* и две мерки: *“цената-в-долари” (dollars_cost)* и *“превозвани-бройки” (units_shipped)*. Схемата “съзвездие от факти” позволява таблиците на размерности да бъдат поделени между таблиците на факти. Например, таблиците на размерности

“време”, “артикул” и “местоположение” са поделени между двете таблици на факти – “продажби” и “превоз”.

14.5.3. OLAP операции върху модела на многомерни данни

Описаните модели на многомерни данни позволяват използване на различни OLAP операции, осигуряващи на потребителя различни погледи на данни от различни гледни точки. Тези операции позволяват една интерактивна обработка на заявки и анализ на данни.

Да разгледаме няколко типични OLAP операции на примера на вече познатия ни куб данни за продажби. Те са илюстрирани с помощта на един кубоид (наречен *централен кубоид*), в който стойностите на размерностите “местоположение” са обобщени до значение “град”, “време” – до “тримесечие”, “артикул” – до “типа на артикула” (Фиг. 14.10). Използваната мярка е “продажби-в-долари” (dollars_sold).



Фиг. 14.10. Примери на типични OLAP операции върху многомерни данни.

Свиване (roll-up): Операцията на свиване изпълнява обобщаване върху данни или чрез качване по йерархията на понятия за определена размерност, или чрез намаляване на размерността. На Фиг. 14.10 е показан резултатът на операцията на свиване върху централен кубоид, извършена чрез изкачване по йерархия на понятия за размерност "местоположение", определена като street < city < province_or_state < country. Показаната операция на свиване агрегира данни от ниво "град" на ниво "страна", т.е. вместо да групира данни по градове, полученият кубоид ги групира по страни.

Разтегляне (drill-down): Операцията на разтегляне е обратна на операцията по свиване на данни. Тя довежда до получаване на по-детайлни данни от по-общи такива. Тази операция се реализира или чрез слизането надолу по йерархията на понятия за някоя от размерности, или чрез добавяне на новите размерности. На Фиг. 14.10 е показан резултат от операцията по разтегляне, извършена върху централен кубоид чрез слизане надолу по йерархията на понятия за размерност "време", определена като day < month < quarter < year. По-точно, показаният кубоид илюстрира разтеглянето по време от "тримесечие" (quarter) към "месец", т.е. данните се агрегират не по квартали, а по месеци.

Разрязване (slice) и изрязване (dice): Операцията на разрязване води до създаване на един под-кубоид чрез избиране на една от размерностите на даденият кубоид и фиксиране на някоя от нейните стойности. На Фиг. 14.10 разрязването на централен кубоид стана по размерността "време" при условието за избор "време" = Q1, т.е. резултиращите данни са за първото тримесечие. Операцията по изрязване определя един под-кубоид чрез избор върху две или повече размерности. На Фиг. 14.10 тази операция е илюстрирана чрез критерий за избор, включващ три размерности (location = "Toronto" **or** "Vancouver") **and** (time = "Q1") **and** (item = "home entertainment" **or** "computer").

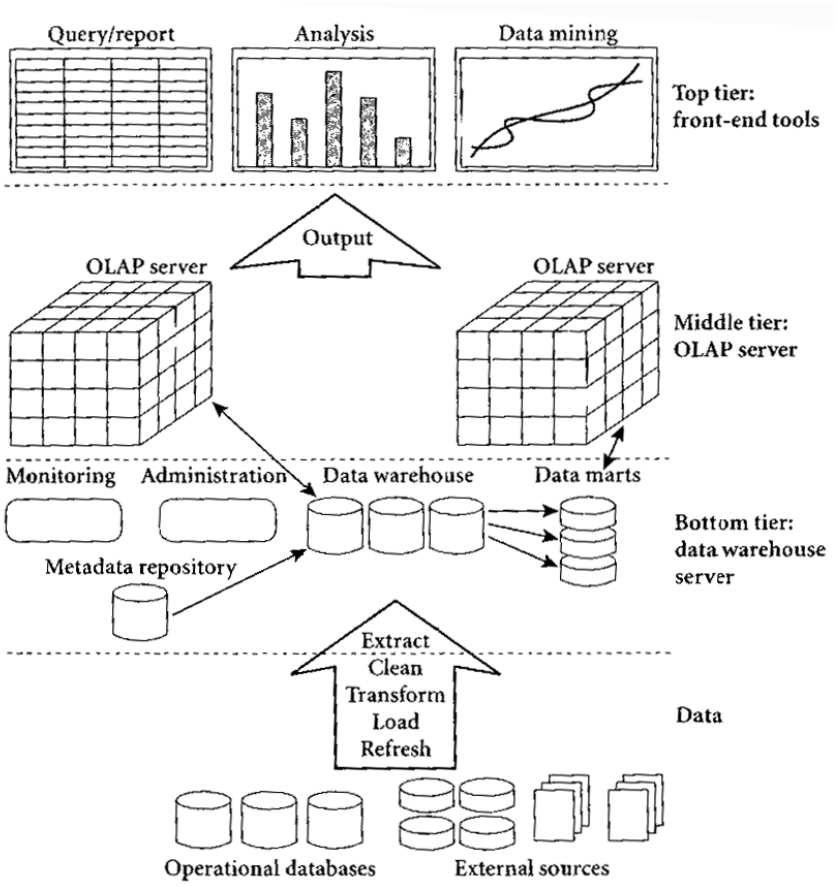
Завъртване (pivot или rotate): Завъртването е една операция по визуализация, която завърта осите на данни при показване, за да осигури определено алтернативно показване на данни. На Фиг. 14.10 е показана операция по завъртване, в която са завъртени осите "артикул" и "време" на един двумерен кубоид, получен чрез разрязване.

14.6. Три-нивова архитектура на складове данни

Най-често складове данни са построени съгласно три-нивова архитектура, представена на Фиг. 14.11.

1. Най-долното ниво е представено от сървъра на бази данни на склада, който почти винаги е реализиран като някаква система за управление на релационни бази данни (DBMS – database management system). Данните от операционни бази данни и външни източници (като информация за профили на потребители, предоставена от външни консултанти) се извличат чрез използване на интерфейсите на приложни програми известни като *врати (gateways)*. Една врата се поддържа от основната DBMS и позволява на клиентските програми да генерират SQL код, подлежащ на изпълнение от сървъра. Примерите на вратите са ODBC (Open Database Connection) и OLE-DB (Open Linking and Embedding for Databases) на Microsoft и JDBC (JavaDatabase Connection).

2. Средното ниво представлява един OLAP сървър, който обикновено е имплементиран чрез или
    a. някой релационен OLAP (ROLAP) модел, т.е. чрез една разширена релационна DBMS, която изобразява операциите върху многомерни данни в стандартни релационни операции; или чрез
    b. някой многомерен OLAP (MOLAP) модел, т.е. един специализиран сървър, който директно имплементира многомерни данни и операции върху тях.
3. Най-горното ниво е клиентът, който съдържа инструменти за обработка на заявки и генериране на отчети, инструменти за анализ и/или инструменти за извличане на закономерности от данни (например, предсказване, клъстеризация и т.н.).



Фиг. 14.11. Три-нивова архитектура на склад за данни

14.7. Имплементация на складове данни

14.7.1. Създаване на куба на данни

Как могат да бъдат изчислена решетка от кубоиди – например, тази от Фиг. 14.7? Да предположим, че схема на релационната база данни за продажби на AllElectronics има следния вид:

time(time_key, day, day_of_week, month, quarter, year).

item(item_key, item_name, brand, type, supplier_type).
branch(branch_key, branch_name, branch_type).
location(location_key, street, city, province_or_state, country).
sales(time_key, item_key, branch_key, location_key, number_of_units_sold, price).

За създаване на куба на данни за продажби с името sales_star можем да използваме следната SQL заявка, в която функцията **sum** се използва за изчисляването на мерки dollars_sold и units_sold:

```
select  s.time_key, s.item_key, s.branch_key, s.location_key,
        sum(s.number_of_units_sold*s.price), sum(s.number_of_units_sold)
from    time t, item I, branch b, location l, sales s,
where   s.time_key = t.time_key and s.item_key = i.item_key
         and s.branch_key = b.branch_key and s.location_key = l.location_key
group by s.time_key, s.item_key, s.branch_key, s.location_key
```

Създаденият от тази заявка кубоид ще представлява базов кубоид на куба данни sales_star. Той ще съдържа всички зададени четири размерности, като степен на детайлизацията на всяка от тях ще бъде на така наречено *ниво на съединителния ключ (joint key)*. Съединителният ключ е такъв ключ, който съединява една таблица на факти с една таблица на размерност.

Чрез промяна на клаузите **group by** можем да генерираме други кубоди за куба данни sales_star. Например, ако вместо да групираме по s.time_key ще групираме по t.month, ще изчислим използваните мерки за всяка група по месеци. Или чрез премахването на оператор **group by** s.branch_key ще генерираме кубоид от по-горното ниво на обобщение (в който продажбите са сумирани за всички клонове, а не са разбити по всеки от тях). Ако изобщо премахнем всички клаузи **group by**, то като резултат ще получим общата сума на продажби (dollars_sold) и общия брой на продадените артикули (units_sold). Този нуламерен кубоид ще представлява върхов (apex) кубоид. Други кубоиди, представени на Фиг. 14.5, могат да се получат чрез прилагане на операции по избор и/или проекция към базовият кубоид, като всеки кубоид ще съответства на различни нива на обобщаване на данни.

Един от подходи към изчисляване на кубове данни е чрез разширяване на SQL чрез оператори за дефиниране **define cube** и изчисляване на кубове **compute cube**. Оператор за дефиниране на куба има следния синтаксис:

define cube <име_на_куба> [<списък_на_размерности>] : <списък_на _мерки>

Например, описаният по-горе куб данни за продажби може да се дефинира по следния начин:

```
define cube sales_star [time, item, branch, location] :
                dollars_sold = sum(sales_in_dollars), units_sold = count(*)
```

Операторът **compute cube** изчислява всички обобщения върху всички възможни подмножества от размерности, указани в оператора. Например, описаният по-горе куб данни за продажби може да се изчисли по следния начин:

compute cube sales_star

Този оператор ще създаде (*материализира*) решетка от кубоиди, съдържаща 2^4 = 16 кубоида.

Интерактивната аналитична обработка на данни (OLAP) в общия случай изисква достъп до различни кубоиди за изпълняване на различни заявки. По тази причина идеята за материализация на всички (или поне на няколко) кубоиди предварително изглежда примамливо, тъй като това води до кратко време за отговор и до избягване на излишни изчисления. На практика, почти всички OLAP системи се базират върху предварително изчислени многомерни модели на данни.

Основният проблем с този подход свързан с експоненциалното нарастване на необходим обем от паметта за запазване на кубоидите в случаи, когато има много размерности, които от своя страна имат множествени нива на йерархия на описваните от тях понятия. Когато размерности нямат асоциираните с тях атрибути, то общият брой на кубоиди, представящи един n-мерен куб данни е 2^n. Обаче, на практика повечето от размерности имат йерархии. Например, размерност "време" обикновено има не само едно ниво, например "година", а определена йерархия или решетка от стойностите. Например, ден < седмица < месец < тримесечие < година. Така общ брой на кубоиди, които могат да бъдат създадени (включително тези, които се създават чрез "изкачване" по йерархия на всяка една размерност) за един n-мерния куб данни, става:

$$T = \prod_{i=1}^{n} (L_i + 1)$$

където L_i е броят на нива, асоциирани със размерността i (без най-горното ниво – "всички", тъй като обобщение до това ниво е еквивалентно на премахването на тази размерност). Например, за един куб данни с 10 размерности, всяка от които има 4 нива, общият брой на кубоиди ще бъде $5^{10} \approx 9.8 \times 10^6$.

Всичко това показва, че на практика не е реалистично да бъдат предварително изчислени и материализирани всички кубоиди, които потенциално могат да бъдат генерирани за един куб данни (или от един базов кубоид). Така че, един по-разумен избор е *частична материализация*, т.е. да бъдат материализирани само няколко от възможни кубоида.

14.7.2. Частична материализация

Има три възможности за материализацията на куба на данни при зададен базов кубоид:

1. Да не се изчисляват предварително никакви "небазови" кубоиди (*никаква материализация*).
2. Предварително изчисляване на всички кубоиди (*пълна материализация*).
3. Избирателно да бъдат предварително изчислени определено подмножество от всички възможни кубоиди (*частична материализация*).

Първият избор води до големи изчисления на сложни многомерни обобщения, които трябва да се правят в режима на реалното време. Такива изчисления могат да изискват много време, което съществено забавя времето за отговор на системата.

Вторият избор може да изисква огромното пространство от паметта за съхраняване на всички предварително изчислени кубоиди. Третият избор представлява определен баланс между необходимата обем на паметта и времето за отговор.

Частичната материализация на кубоиди трябва да отчита следните три фактора:

1. Идентификация на подмножеството от кубоиди, подлежащи на материализация.
2. Използване на материализирани кубоиди при обработка на заявки.
3. Ефективно обновяване на материализираните кубоиди при начално натоварване и обновяване на данни.

Изборът на подмножеството от кубоиди, които ще бъдат материализирани, трябва се базира на анализа на заявките, техните честоти, както и на тяхната цена за достъп. Освен това, трябва да бъдат отчитани цената на инкрементални обновявания и изисквания към размера на общата памет. При избора се отчита и по-широк контекст на физическия дизайн на бази данни, като, например, създаване и избор на индекси. Няколко OLAP продукта използват за избор на кубоиди, подлежащи на материализация, евристичните подходи. Един разпространен подход е да бъдат материализирано това множество от кубоиди, на които се базират други често използвани кубоиди.

След като избраните кубоиди са материализирани, е важно те да бъдат използвани ефективно за обработката на заявки. Това включва определяне на релевантен кубоид (кубоиди) от множеството на материализираните кубоиди, които ще се ползват за обработката на конкретната заявка, начина на използване на съществуващи индексни структури върху материализирани кубоиди, и как да бъдат трансформирани OLAP операции върху избрани кубоиди.

Освен това, материализираните кубоиди трябва да бъдат ефективно обновявани при операциите на натоварване и обновяване на данни. За тази цел могат да се ползват техниките за паралелно и инкрементално обновяване.

14.7.3. Хранилище на метаданни

Метаданни са данни за самите данни. В случая на складове данни метаданни са данни, които дефинират обектите на склада. Едно хранилище на метаданни следва да съдържа следната информация:

- Описание на *структурата на склада на данни*, която включва схемата на склада, изглед, размерности, йерархии и дефиниции на изведените данни.
- *Операционни метаданни*, които включват история на предвижването на данни и последователността на приложени към тях трансформации, степен

на актуалността на данни (активни, архивирани или изчистени), мониторната информация (статистиките по използване на склада, доклади за грешки и т.н.).

- *Алгоритми, използвани за обобщаване на данни*, които включват алгоритми за дефиниция на мерки и размерности, данните за нивата на детайлизация, разбивания, тематични области, агрегация, обобщение, както и за предварително дефинирани заявки и отчети.
- *Изображението от операционната среда към склада на данни*, което включва базите данни на източници и тяхното съдържание, описание на врати, начини на извличани и изчистване на данни, трансформационни правила и стойности по премълчаване, правилата за сигурност (авторизацията на потребители и управление на достъпа).
- *Данни, отнасящи се за поведението на системата*, които включват индекси и профили, които подобряват поведението при достъпа и извличане на данни, както и правила за периодите на обновяване на данни.
- *Бизнес метаданни,* които включват бизнес термини и дефиниции, информация за притежателя на данни и ценовата политика.

14.8. От складове данни към извличане на закономерности от данни

Съществуват три типа приложения на складове данни: обработка на информация, аналитична обработка и извличане на закономерности от данни:

- *Обработка на информация* поддържа заявки, основен статистически анализ и отчетност чрез използване на таблици, диаграми или графици. Едно съвременно направление в обработката на информация чрез складове данни се състои в конструиране на евтини Web базирани инструменти за достъп, които се интегрират с Web браузери.
- *Аналитична обработка* поддържа основни OLAP операции, такива като свиване, разтегляне, разрязване, изрязване и завъртване. В общият случай те са работят върху исторически данни, представени както в детайлен, така и в обобщен вид. Основната сила на интерактивна аналитична обработка във сравнение с обработката на информация се състои в многомерен анализ на данни, представени в склада на данни.
- *Извличане на закономерности от данни* поддържа откриване на нови знания чрез намиране на скрити шаблони и асоциации в данни, създаване на аналитични модели, изпълнение на класификацията и предсказване и представянето на резултата чрез използване на средства за визуализацията.

Как се отнасят извличането на закономерности от данни с обработка на информация и интерактивна аналитична обработка? Обработка на информация, базирана върху заявки, може да намери полезна информация. Обаче, отговори на такива заявки отразяват информацията, която директно е запазена в базите данни или може да бъде изчислена чрез функциите за агрегацията. Тя не отразява сложни шаблони или регулярности, скрити в базата данни. По тази причина, обработката на информация не едно и също с извличане на закономерности от данни.

Функционалностите на OLAP и ИЗД могат да се разглеждат като непресичащи се: OLAP е инструмент за агрегация/обобщение, което помага да опрости анализ на данни, докато ИЗД позволява автоматично откриване на неявни шаблони и интересни закономерности, скрити в голям обем данни. В този смисъл ИЗД е една следваща стъпка след традиционната интерактивна аналитична обработка на данни.

Друг по-широк поглед на ИЗД, разглежда този процес като дейностите, покриващи както процеси на описание, така и на моделиране на данни. OLAP системи могат да представят общите описания на данни от складове данни, а OLAP функции са насочени към управлявано от потребителя сравнение и анализ на данни. От тази гледна точка те изпълняват, макар и ограничени, ИЗД функции. Съгласно тази гледна точка ИЗД покрива значително по-голям спектър от колко прости OLAP операции, тъй като не само обобщава и сравнява данни, но и намира асоциации, изпълнява класификация, предсказване, клъстеризация и други задачи.

ИЗД не се ограничава само до анализа на данни, съхранявани в един склад от данни. То може да анализира данни, съществуващи на по-детайлно ниво, отколко обобщени данни, присъстващи в склада на данни. То също така може да анализира текстови, пространствени и мултимедийни данни, които трудно се моделират с съществуваща технология на многомерни бази данни. С други думи, ИЗД покрива значително по-голям от OLAP спектър както по своята функционалност, така и по сложността на обработваните данни.