Глава 13 Физическа организация на базите от данни

13.1. Процесът „проектиране на физическата база“

В базите от данни се съхранява голяма по обем информация, която се обработва постоянно и е устойчива във времето, т.е. данните се запазват от сеанс на работа с базата за следващ сеанс. За да може СУБД да изпълнява различните обработки, е необходимо, базата физически да се съхранява върху някакъв магнитен или оптичен носител. Тя не може да се съхранява изцяло в оперативната памет на компютъра, защото последната е енергозависимост и има по-висока цена. По този аспект базата от данни се различава от понятието "структури с данни" в традиционните езици за програмиране, където съдържанието на променливи, масиви и др. е определено само докато се изпълнява програмата. Напоследък обаче се предлагат решения – например HANA[1] на SAP и TimesTen[2] на Oracle, при които базата се разполага изцяло в оперативната памет на компютъра.

Всяка търговска СУБД предлага различни опции за организация на данните. Независимо от това, че проектантът на базата е ограничен в рамките на възможностите, заложени в конкретния продукт, съществуват начини за ускоряване на обработките.

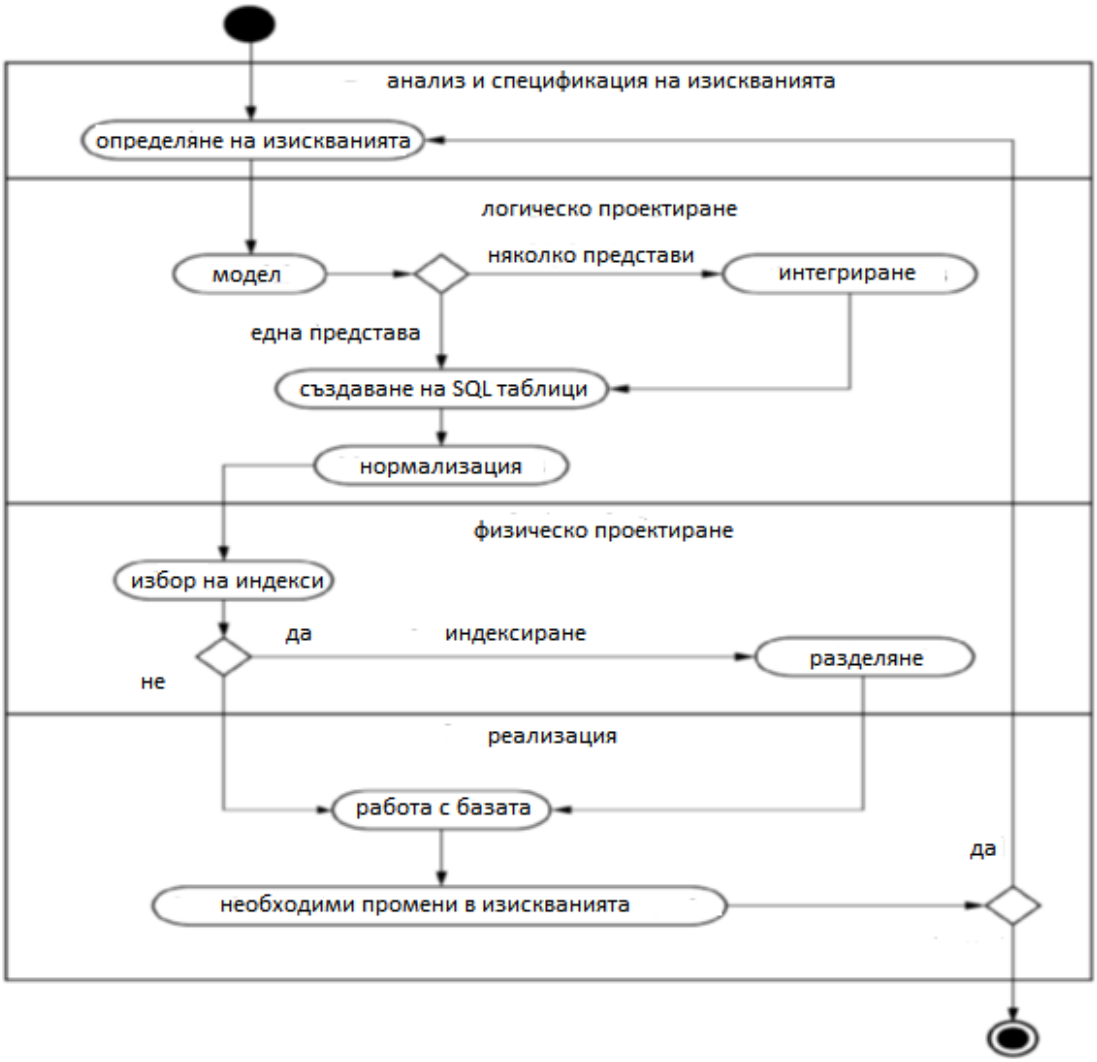
Под *проектиране на физическата база* се разбира избор на най-подходящите за дадено приложение методи за организация на данните. Този избор е важен, защото в голяма степен определя скоростта на обработките и производителността на системата като цяло. Проектът на физическата база е от особено значение при бази с голям обем, съдържащи различни видове данни: структурирани (релационни кортежи), полуструктурирани (например XML) и неструктурирани (например аудио, видео или уеб страници). Нарастването на обемите от данни до голяма степен се дължи на разпространението на различни персонални хардуерни устройства, с които се осъществяват съвременните комуникации. В резултат от приложението на информационните технологии в организациите са натрупани огромни информационни масиви, чиито обем се измерва в тера или пета байтове, т.е. в милиони гигабайтове. Освен това за нарастването на съхряваните обеми с данни допринася и поевтиняването на паметта. В резултат голяма част от хартиените архиви се дигитализират, понеже запазването им се оказва по-евтино на електронни носители. С нарастването на изчислителната мощност на компютрите (закон на Мур) стават възможни все по-сложни обработки на натрупаните големи обеми с оглед извличане на данни (data mining) или аналитична интерактивна обработка.

Всички тези фактори определят важността на задачата за проектиране на физическата база. Независимо, че потребителят разполага само със средствата, предоставени му от разработчика на избраната СУБД, правилното им използване може да намали значително времето за обработване на данните.

Фазите, през които преминава проектирането на базата от данни и съответните им дейности са представени в Глава 7 и на Фиг.13.1. Проектирането на физическата база се извършва след проектирането на схемата и обхваща определяне на пътища за ефективен достъп до данните, задаване на системни параметри като размер на блока,

[1] https://www.sap.com/products/hana.html, достъпен на 27.07.2017 г.
[2] https://www.oracle.com/database/timesten-in-memory-database/index.html, достъпен на 27.07.2017 г.

буферите, протокол за възстановяване на данните при софтуерни и хардуерни повреди и др.



Фиг.13.1 Жизнен цикъл на базата от данни

13.2. Елементи на физическата база от данни

В даден момент потребителят работи само една малка част от базата. Тези данни трябва да се намерят, да се прехвърлят в паметта за обработка и след това, при промяна, да се запишат отново на диска.

Обикновено данните се съхраняват като файлове със записи. Физическата база от данни представлява множество от взаимно свързани записи от различен тип, групирани във файлове. Операциите върху базата (заявки или транзакции) са ефективни благодарение на реализирани в конкретната СУБД методи за търсене. Записът е съвкупност от свързани полета с данни, които описват обектите с техните атрибути и връзки. Например, да разгледаме обекта "Студент" с атрибути номер, име, адрес и т.н. На всеки атрибут отговаря поле от определен тип (числов, символен, логически, от тип "дата" и др.) както при езиците за програмиране. Множеството от имената на полетата заедно със съответстващия им тип образува формàта на записа. Освен специфичните за дадено приложение данни, записът съдържа указатели и служебна информация, необходима за идентифицирането и обработването му от СУБД.

От разположението на записите върху диска зависи ефективен ли е достъпът на СУБД до тях.

При обработка на данни съществена е разликата между физически и логически аспект на данните. Логическият аспект отразява естеството на данните, независимо от физическите подробности на съхраняването и представянето им. Физическият аспект изразява начина на съхраняването и представянето им върху определен носител.

Логически запис се нарича множеството от данни, което представлява едно цяло за потребителя или приложната програма, т.е. това е съвкупността от полетата на записа. *Физически запис* се нарича единицата информация, която реално се чете от едно устройство или се записва на него. Това е най-малкият обем данни, който се предава между паметта и диска при обработката на данните. Синоними на понятието "физически блок" са понятията "блок" или "страница".

Файловете биват два вида:

- с неблокувани записи; при тях физическият запис съвпада със логическия;
- с блокувани записи; при тях физическият запис се състои от няколко логически записи.

Освен това във файла записите могат да бъдат:

- с постоянна или с фиксирана дължина;
- с променлива дължина.

Файловете, изграждащи дадена база от данни, представляват множества от еднотипни записи с постоянна или променлива дължина. Записите се групират съобразно блоковете на диска, като в един блок се поместват по няколко записа в зависимост от дължината им. *Блоков фактор* се нарича числото *bfr = [B/R]*, където *B* е размерът на блока, а *R* - дължина на записа. В общия случай R не дели точно *B*, затова във всеки блок остава *B - bfr*R* байта неизползувано място. Съществуват начини то да се използува като се допуска един запис да се разполага в два блока.

Физически файлът може да се съхранява на диска по следните начини:

- в последователни блокове;
- в свързани чрез указатели блокове;
- в клъстери (clusters).

Когато файлът се съхранява на диска в последователни блокове, четенето на целия файл е много бързо, но разширяването му е трудно. При съхраняване на файла в свързани чрез указатели блокове е вярно обратното. Клъстерите представляват комбинация от двата начина на съхранение.

Всеки файл е снабден с челен блок (хедър), който съдържа информация за адресите на блоковете и формàта на записа (дължина на полетата, наредба, начин на запис в блока, с фиксирана дължина ли е, или не и др.). При търсене на даден запис в буферите на оперативната памет се копират един или повече блока от диска. За се намери необходимия запис се използва информацията от челния блок на файла като се провежда пълно претърсване на отделните блокове. При файлове с голям размер процесът може да продължи дълго. Затова при организацията на файла се цели локализиране на блока, съдържащ търсения запис като се минимизира прехвърлянето на блокове в паметта.

От гледна точка на базите операциите върху файлове се делят на две големи групи:

1. Операции от тип "търсене". Те не променят данните, а само намират определени записи, така че техните полета да се изследват и съдържанието им да се извежда.
2. Операции от тип "обновяване". Те променят съдържанието на файла. Такива са въвеждане на нов запис, изтриване на съществуващ запис, модификация стойностите на някои полета.

И при двата вида операции характерното е, че се избира запис или група от записи съобразно някакъв критерий за търсене. Последният определя условията, които записите трябва да удовлетворяват. Например `s# = 155` или `Salary = 900` са прости условия за избор. В общия случай може да се формулира сложно условие, съдържащо булев израз.

Файловата система е елемент от дадена операционна система, която осигурява управление на файловете, свързани с отделните приложения. Тя предоставя контролиран достъп до файл по име, многопотребителска работа и стандартизирани входно/изходни процедури. Повечето файлови системи могат да намират запис, само ако критерия за търсене е просто условие. Сложното условие за избор обикновено се разлага от СУБД на прости подусловия, които се използуват за намиране на необходимите записи. Ако повече от един запис удовлетворява даден критерий за търсене, то се намира само първият (има се предвид физическата им наредба върху диска). Достъпът до останалите записи, които удовлетворяват същото условие се осъществява чрез допълнителни операции. Последният намерен запис се нарича "текущ". Следващите операции към базата започват от последния текущ запис.

Действителните операции върху файлове зависят от конкретната операционна система. Най-общо те биват:

1. Файлово-ориентирани операции - работят с файла като логическа единица: open, close, create, copy, rename, delete, type.
2. Операции за достъп до запис - работят върху запис в рамките на даден файл: read, write, update, insert, delete.

СУБД осигурява достъп до записите в базата чрез следните видове команди:

- Open - подготвя файла за четене или запис; запазва поне два буфера за прехвърляне на негови блокове; намира челния блок на файла и установява неговия указател към началото му;
- Reset – установява указателя на отворен файл към началото му;
- Find (или Locate) – намира първия запис във файла, който удовлетворява условието за търсене; прехвърля при необходимост съответния блок в буфер от главната памет и го прави текущ запис;
- Read (или Get) – копира текущия запис от буфера в програмни променливи;
- FindNext - намира следващия запис във файла, който удовлетворява условието за тъсене; прехвърля при необходимост съответния блок в буфер от главната памет и го прави текущ запис.
- Delete – изтрива текущия запис;
- Modify – актуализира съдържанието на някои от полетата в записа;
- Insert – въвежда нов запис във файла;

- Close – прекратява достъпа до файла чрез освобождаване на буферите и извършване на необходимите почистващи операции.

Начинът на съхранение и достъпът до записите в базата зависят от файловата организация.

13.3. Видове файлови организации

Файлова организация се нарича физическата организация на данните от файла в записи, блокове и структури за достъп. Тя включва начина, по който записите са разпределени върху даден носител и са свързани помежду си. С всяка файлова организация се свързват определени методи за достъп.

Метод за достъп се нарича множество от програми, които позволяват изброените по-горе операции да бъдат приложени към даден файл. Методът за достъп се състои от два интегрирани компонента: структура от данни и механизъм за търсене. Структурата от данни определя начина на съхранение на блоковете с данни и свързаните с тях допълнителни структури за достъп в паметта. Механизмът за търсене определя пътя за достъп, т.е. начина за достигане до записите с нужната информация. Той може да бъде различен: последователно търсене, двоично търсене, директно търсене. Всяка комбинация на структура от данни с механизъм за търсене дава различен метод за достъп. Следователно методът за достъп определя как се съхраняват и намират записите от файл с определена файлова структура. При създаването му се прилагат различни техники: сортиране, хеширане или индексиране. Да отбележим, че някои от методите за достъп са приложими само към файлове, организирани по определен начин.

Най-общо командите за обработка на базата попадат в някоя от следните групи:

- извличане на всички записи от даден тип (последователен достъп);
- избор на един запис от даден тип (директен достъп);
- избор на група от записи, удовлетворяващи някакво условие (достъп с булева заявка).

С всеки тип обработка се свързват ефективни методи за достъп до данните.

Някои от условията за търсене се използват по-често от други. Освен това някои от файловете се обновяват рядко и са статични доколкото съдържанието им главно се чете - така е при хранилищата с данни. Сполучливата организация на файла спомага за ефективното изпълнение на често изпълняваните операции: търсене по определено условие, добавяне, актуализиране или изтриване на записи. В зависимост от тези операции проектантът на базата избира файловата организация, така че те да се изпълняват лесно. Например ако често се търси по ЕГН на служител може да се извърши сортиране на файла по това поле. За съжаление избраната файлова организация не позволява ефективното изпълнение на всички необходими операции. Затова АБД изследва условията за търсене при обработките и операциите за обновяване на файла, за да определи подходящата организация на файла.

Съществуват три основни типа файлови организации:

1. Неструктуриран файл (heap, sequential).
2. Подреден файл (ordered).
3. Директен файл (hash, random).

Неструктуриран файл

Неструктурираният, или последователният файл, е най-простия тип на файлова организация. При него записите се поместват така както са въведени физически и не е установена наредба върху тях. Нови записи се добавят винаги в края на блока, а ако няма място, се добавя нов блок. Тази организация се използува, когато данните в базата се натрупват първоначално и още не е ясно какви обработки ще се изискват. Добавянето на нов запис е изключително ефективно. Търсенето обаче на определен запис е бавно, защото се преглеждат последователно всички записи (линейно търсене), докато се открие необходимият. Ако файлът заема пространство от b блока, времето за намиране на даден запис е $b/2$ достъпи до блок. Изтриването на произволен запис става или като се отбележи за изтрит, или като се изтрие физически. И в двата случая се налага периодична реорганизация на файла от АБД, за да се използува празното пространство на диска и за да се ускорят обработките. За да се прочете файлът по определен начин, се налага или сортиране, или създаване на допълнителни структури за достъп.

Последователният файл е приемлива структура за съхранение данните при следните ситуации:

- първоначално зареждане на таблиците в базата с данни;
- малки по размер таблици, заемащи само няколко блока;
- извличане на всички редове от дадена таблица при всеки достъп до нейните данни;
- създадена е допълнителна структура за ускоряване на достъпа.

Подреден (сортиран) файл

Записите с еднаква дължина физически могат да се подредят по стойностите на едно или няколко от полетата на записа, наречени сортиращи полета. Тогава търсенето на записи в реда, наложен от сортиращото поле се извършва бързо. Също така може да се провежда двоично търсене - когато в условието за избор участвува полето, по което е сортиран файлът. Въвеждането и изтриването на записи в подреден файл е трудно, поради спазването на наложения от стойностите на сортиращото поле ред. Сортирани файлове не се използуват често при базите от данни, тъй като се прави копие на първоначалнит данни. Освен че копието заема допълнително място, има опасност от получаване на противоречива база. Ако файлът заема пространство от b блока, времето за намиране на даден запис е $b/2$ достъпи до блок при линейно търсене или $log_2 b$ - при двоично търсене.

Директен файл

При директните файловете записите не се въвеждат последователно един след друг. Местоположението им се изчислява с помощта на хеш-функция, която трансформира стойността на определено поле във физически адрес на блок (Knuth, 1998). Обикновено се извършва хеширане чрез деление с верижно препълване. Трансформирането на адреса става чрез деление на стойността на дадено поле от записа (обикновено се избира поле с уникални стойности – хеш поле) на части и прилагане на аритметична функция към всяка част. Така се получава единствено число - адрес, който след това се преобразува във физически адрес на диска. Достъпът до този физически адрес позволява извличането на съществуващ запис или записването на нов. При опит да се въведе запис в зает блок се получава колизия и съответно препълване. На новопостъпващия запис трябва да се намери място. Съществуват различни подходи, но най-често се заделят блокове за препълване, свързани помежду си с указатели -оттам и

терминът "верижно препълване". Хеширането от своя страна може да бъде статично или динамично (Larson, 1978). Записите в един хеш файл са произволно разпределени върху блокове от диска и затова понякога хеш-файловете се наричат файлове с директен достъп. Търсенето на записи в хеш-файл е ефективно само ако в критерия за търсене участвува хеш-полето. Директният файл не е приемлива структура за съхранение данните при следните ситуации:

- условието за търсене съдържа обхват от стойности на хеш-полето в записите или някакъв шаблон;
- условието за търсене е формулирано върху произволно поле, различно от хеш-полето;
- хеш-полето се актуализира често – тогава СУБД изтрива целия ред в таблицата и го премества на нов адрес като този процес оказва влияние на производителността;
- условието за търсене съдържа подниз на хеш-полето.

Затова хеш-файлове се използуват сравнително рядко при базите от данни.

Разгледаните по-горе файлови организации се прилагат когато всички записи в даден файл са от един и същ тип. При повече от приложенията, обаче се установяват връзки между записи от различни файлове с помощта на свързващи полета като чужд ключ при релационния модел или връзките между обектите при обектно-ориентираните модели. Файловите организации при обектните, или при йерархичните и мрежови СУБД, реализират тези връзки чрез физически указатели, като записите се поместват в една област (клъстер). Това повишава ефективното извличане на свързаните записи. Например ако в заявка за търсене се извлича запис с данни за даден отдел и след това записите за всички служители, които работят за същия отдел е добре тези записи да бъдат поместени в една и съща област. Същото важи и за еднотипните обекти при обектните СУБД. При първоначалното зареждане на хранилищата с данни те произлизат от различни източници и се съдържат в различни файлове, които се поместват в един клъстер, за да се подложат на преформатиране и почистване. Възможно е използването и на други структури от данни като B дървета за файлова организация.

С развитието на хардуера се предлагат нови решения за ускоряване извличането на данните. Подобрява се ефективността на дисковата памет чрез разработването на *масиви от независими дискове* (redundant arrays of independent disks – RAID) с подходящи параметри за работа с мултимедийна информация. Съвременните приложения като е-търговия, системи за планиране и управление на ресурсите (Enterprise Resource Planning - ERP), хранилища с данни изискват интегрирането на данните в организацията чрез прилагането на специализирани хардуерни решения. Потребителите на RAID системи не могат да използват техния капацитет ефективно, защото са свързани с един или повече сървъри. Затова в повечето големи организации се изграждат *локални мрежи за съхранение* (storage area networks – SAN), където онлайн периферни устройства за запазване на данни представляват възли във високоскоростна мрежа, която гъвкаво се прикрепя при необходимост към сървърите. С нарастването на дигитализираните данни възниква необходимост от създаването на памет с добра производителност и приемлива ниска цена. Понастоящем се предлагат специализирани устройства, които се включват към мрежата (network-attached storage - NAS). В действителност тези устройства осигуряват памет за споделяне на файлове като позволяват добавянето на дискови ресурси без да се налага изключване или обновяване на сървъри. NAS устройствата могат да се разполагат на произволни места

в локалната мрежа на организацията и да изграждат различни конфигурации. Те съхраняват различни файлове и могат да се разглеждат като заместващи традиционните файлови сървъри.

През последните години развитието на облачните технологии, разпределените архитектури за анализ на данните и уеб приложенията доведоха до основни промени в инфраструктурата за съхранение на данните в организацията. Понастоящем се предстагат обектно-ориентирани хранилища, в които данните се управляват във вид на обекти, а не като файлове, изградени от блокове. Обектите се описват от метаданни и се локализират чрез уникални глобални идентификатори (Mesnier et al., 2003). Хранилищата поддържат дублиране, единно адресиране и разпределение на обектите. Подходящи са за съхраняването на големи обеми от неструктурирани даннн. Прилагат се във Facebook за съхранение на снимки, в Spotify – за песни, в Dropbox като инфраструктура на хранилище и много облачни решения. Като реализация на обектно-ориентираните хранилища могат да се споменат AWS (Amazon Web Service) S3[3] на Amazon, Azure[4] на Microsoft, HCP[5] на Hitachi, Atmos на EMC[6], RING[7] на Scality.

13.4. Методи за проектиране на физическата база

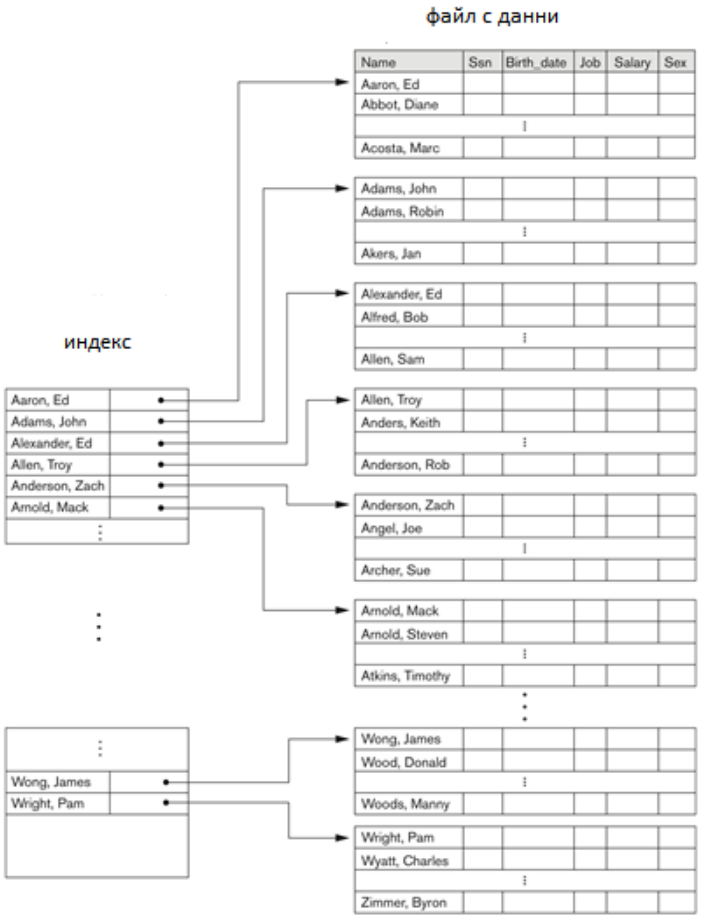
Индексиране

Индексирането е метод за ефективно търсене на записи в база от данни. При него към файла с данните се създават допълнителни структури за достъп, наречени индекси. Индексите спомагат за по-бързото намиране на нужния запис и са ефективно средство за достъп при задаване на критерий за търсене. Те представляват допълнителна структура към основния файл с данни, чиято цел е да ускори извличането на търсената информация. По този начин се избягва последователното търсене за намиране на необходимия запис. Файлът с основните данни се нарича главен или файл с данни, а допълнителните структури – индекси или индексни файлове. Индексната структура може да се сравни с индекса на книга. Тя се дефинира обикновено върху едно поле, което се нарича индексно. Индексът съхранява всяка стойност на избраното индексно поле и указател към блока с данни, в който се съдържа записа със съответната стойност на индексното поле. Достъпът до записите може да бъде: последователен или чрез чрез индекса.

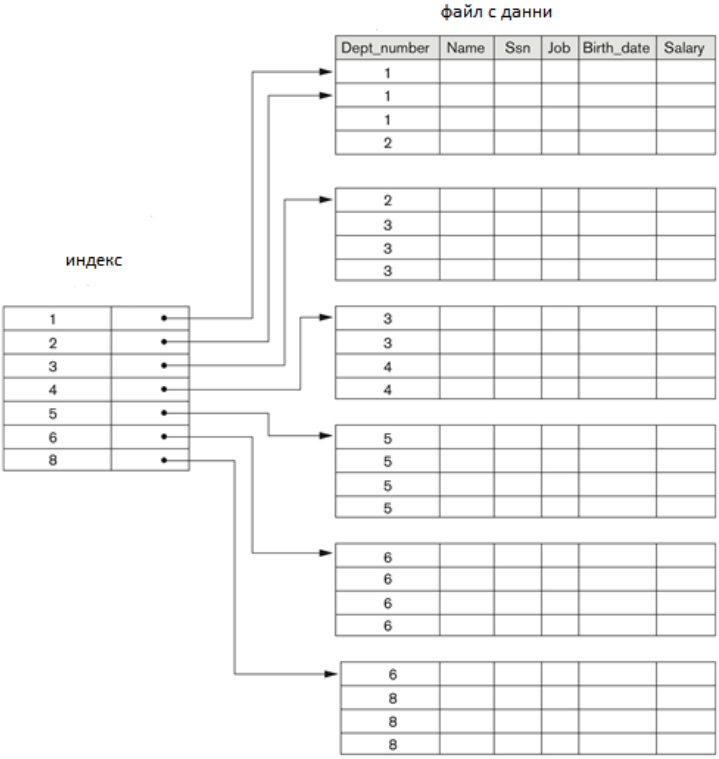
Съществуват следните видове индекси:

1. Главен индекс – дефинира се върху поле с уникални стойности (нарича се главен ключ), по което физически е подреден даден файл. Главният индекс представлява сортиран файл, чиито записи са с постоянна дължина и съдържат две полета: стойността на избраното поле с уникални стойности и адрес на блока, който съдържа търсения запис (Фиг.13.2).
2. Клъстер индекс – дефинира се върху поле (нарича се клъстер-поле) с повтарящи се стойности, по което физически е подреден даден файл. Клъстер индексът е сортиран файл, съдържащ две полета: различните стойности на избраното поле и адрес на блока, който съдържа търсения запис (Фиг.13.3).

[3] https://aws.amazon.com/s3/ ,достъпен на 27.07.2017 г.
[4] https://azure.microsoft.com/,
[5] https://www.hds.com/en-us/products-solutions/storage/content-platform.html, достъпен на 27.07.2017 г.
[6] https://www.emc.com/storage/atmos/atmos.htm, достъпен на 27.07.2017 г.
[7] http://www.scality.com/, достъпен на 27.07.2017 г.

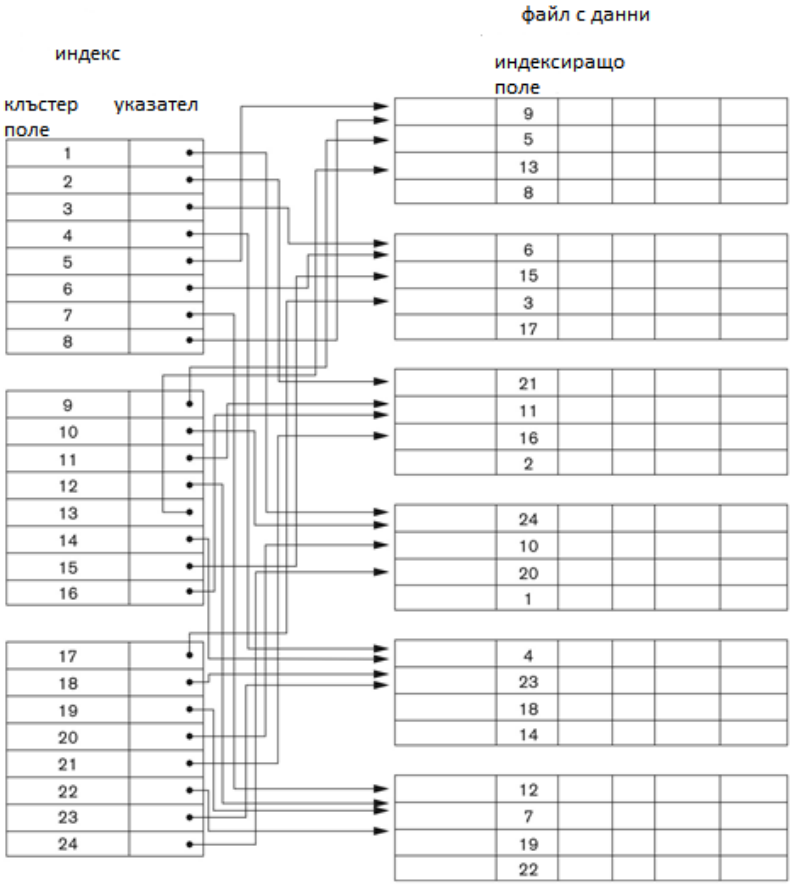


Фиг.13.2 Главен индекс по полето Име



Фиг.13.3 Клъстер индекс по полето „№ на отдел“ от файл с данни за служителите

3. Вторичен индекс - дефинира се върху произволно поле на файла (наречено индексиращо поле), по което се търси често. Вторичният индекс е средство за бърз достъп до съдържанието на файл, чиито записи са сортирани, хеширани или неструктурирани. Обикновено се изгражда върху кандидат-ключ или друго поле с повтарящи се стойности и се състои от две полета както разгледаните по-горе индекси (Фиг.13.4). Върху даден файл могат да се изградят произволен брой вторични индекси, но трябва да се има предвид, че тези структури заемат място и се обновяват при всяка актуализация на данните. Затова идеята да се използва голям брой вторични индекси не винаги ускорява достатъчно извличането на търсените записи, предвид служебните обработки по поддържането им, изпълнявани от СУБД. Да отбележим, че намирането на оптимално множество от вторични индекси за дадена база е NP - пълна задача.



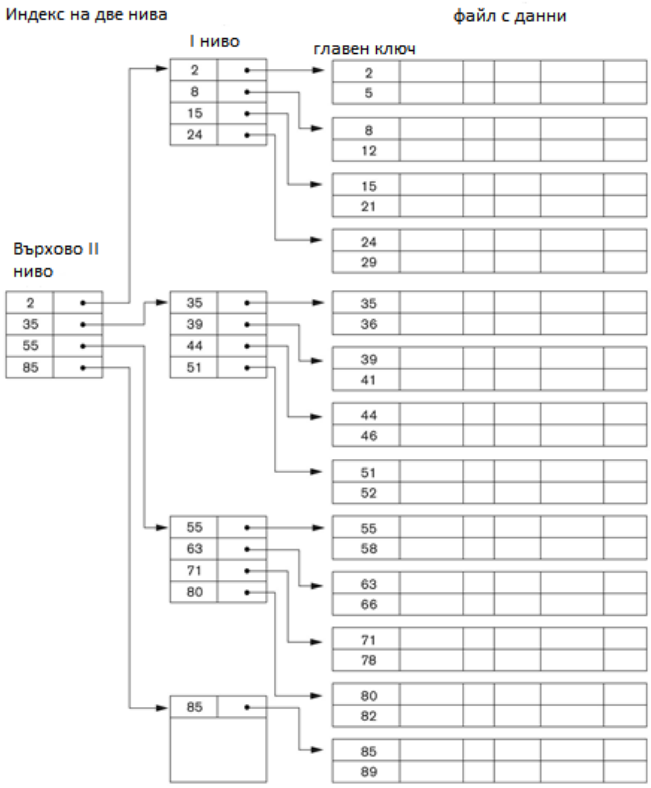
Фиг.13.4 Вторичен индекс по несортиращо ключово поле на файл

Вторичният индекс предоставя логическо сортиране на записите по избраното индексиращо поле. Освен това индексът може да бъде компактен (dense) или разпилян (sparse). При компактния индекс за всяка стойност на избраното за индексно поле се създава запис. При разпиляния индекс запис в индексния файл се създава запис само за определени стойности.

Един файл може да има най-много един главен или клъстер индекс, защото физически може да бъде подреден само по стойностите на едно поле. Сортиран файл с данни с дефиниран главен индекс се нарича индексно-последователен файл (Larson 1981). Такъв файл е структура, която съчетава характеристики от последователната и директната файлови организации. Неговите записи могат да се обработват

последователно или поотделно. Достъпът до определен запис става с помощта на индекс, дефиниран върху стойностите на избрано поле за търсене. Индексно-последователният файл е гъвкава структура, която се състои от главна област за съхранение, отделен индекс на индексите и област за препълване. За първи път е реализирана като метод достъп от IBM. Първоначалният й вариант е силно зависим от хардуера, но в последствие е предложена независима от хардуера версия. Обикновено голяма част от главния индекс се съхранява в оперативната памет на компютъра и затова се обработва достатъчно бързо. За ускоряване на достъпа може да се приложи двоично търсене. Основен недостатък на метода е съобразяването с наложения от сортирането ред на записите при актуализация на данните. Индексно-последователната организация е по-гъвкава от директната, защото позволява извличане на данните когато условието за избор съдържа обхват от стойности, шаблон или подниз на ключовото поле. За съжаление тази организация е статична и се създава заедно с файла, съдържащ данните. Всяка актуализация на таблицата води до забавянето на обработките, защото може да се загуби подреждането на ключа за достъп.

В допълнение върху даден файл могат да се дефинират произволен брой вторични индекси. В много системи индексът не е част от файла с данни, а може да се създава и променя динамично. Затова се нарича структура за достъп. Обикновено индексни файлове се създават върху полета, които се използуват често в критериите за търсене. Вторичните индекси се създават, за да се избегне физическото сортиране на файла. Те имат преимущество пред останалите индекси, защото могат да бъдат създавани успоредно. За разлика от главния индекс, при вторичния не се изисква индексиращото поле да съдържа уникални стойности. Динамичното създаване на главен или клъстер индекс е по-скъпо, защото първо файла трябва да се сортира физически. При нарастване размера на индексния файл се прибягва до създаването на индекси на много нива, като по този начин се намалява обхвата на търсенето (Фиг.13.5).



Фиг.13.5 Индекс на две нива

В края на 70-години на миналия век са разработени редица методи за индексиране - инвертирани файлове, различни видове дървовидни структури, многосписъчни файлове и др. (Comer, 1979). На програмистите са се предоставяли различни възможности за избор на метода за индексиране на дадена база, което е преполагало значителни умения и е било трудно за вземане решение. Голяма част от релационните СУБД използуват различни видове дървета за реализация на индексирането. Например, *B* дърво се прилага при СУБД Oracle, DB2, Ingress, Sybase, Microsoft SQL Server и др. При него всеки възел в дървото се състои от *p-1* записа и *p* указателя към следващото ниво в дървото, което от своя страна се състои от *p* възела. Стойността *p* се нарича степен на дървото. Най-честата реализация на B дървото е чрез специализирана структура, наречена *B+* дърво, при които се елиминират голяма част от указателите, като се използува индексен файл. Така обхождането на дървото е ефективно. B+ дървото е по-гъвкава организация от индексно-последователната и от директната, защото позволява извличане на данните когато условието за избор съдържа обхват от стойности, шаблон или подниз на ключовото поле. Освен това актуализацията на данните не води до забавянето на обработките. При СУБД от трето поколение се прилагат *R* дървета и се разработват нови методи за индексиране.

Разработени са методи за индексиране и съответни структури за реализация когато индексиращото поле се състои от няколко елемента (съставен индекс). Предлагат се структури за достъп, които са подобни на индексирането, но прилагат хеширане или използват някои от функциите, заложени в СУБД. Последният начин е реализиран в СУБД Oracle и в други търговски продукти. Индексите винаги съхраняват указател към местоположението на стойностите на ключа в базовата таблица. Тази препратка се нарича индентификатор на записа (record identifier – RID) и е необходима, за да се извлекат конкретните данни, участвуващи в отговора на дадена заявка.

При хранилищата с данни се изполват присъединителни индекси (join index) и растерни индекси (bitmap index). При първите се създава индекс върху атрибути, принадлежащи на две таблици, но имащи един и същ домейн от стойности. Вторият тип индекси се прилага при домейни, чиито възможни стойности са малко. При тях вместо да се съхранява реалната стойност на полето се създава растерен вектор, който указва в кои редове се среща съответната стойност. Ако в определен бит е записана единица, това означава, че има ред със съответен идентификатор в таблицата. Ако броят на различните стойности е малък, растерните индекси са ефективни и пестят доста място в паметта. Повече подробности могат да се намерят в специализираната по тази тематика литература (Lightstone et.al., 2007).

Създаването на индекси в повечето релационни СУБД се извършва с команда от вида:

CREATE [UNIQUE] INDEX <index name>
ON <table name> (<column name> [<order>] { , <column name> [<order>] })

Тази команда не е част от стандарта на езика SQL. В повечето системи индексът се реализира като отделна структура от файл с данни.

Клъстериране

При някои СУБД, например Oracle, таблици които имат общи колони се съхраняват физически заедно и образуват клъстер или група. Предполага се, че тези таблици често се използват съвместно и по този начин се подобрява времето за достъп до диска. Общите колони образуват *клъстер-ключ* (cluster key), който се съхранява само на едно място и подобрява едновременно достъпа и до двете таблици. Например

таблиците с данни за проекти и отдели могат да образуват клъстер по полето „№ на отдел“ (фиг.13.6).

Име на отдел	ЕГН на мениджъра	Дата на назначение	№ на отдел	Име на проект	№ на проект
Административен	.....	.....	1	Планиране на средства	1
Конструкторски	.....	.....	4	Изделие А	2
				Изделие Б	6
				Изделие В	8
Автоматизация	.....	.....	2	Маркетингова ИС	13
Научни изследвания	.....	.....	3	Нови технологии и производства	22

Фиг.13.6 Клъстер на таблиците Проект и Отдел

В Oracle са реализирани *индексирани клъстери*[8] (indexed clusters) и *хеш-клъстери*[9] (hash clusters). При индексирания клъстер записите с еднаква стойност на клъстер-ключа се съхраняват на едно място. При хеш-клъстерите се прилага хешираща функция върху клъстер-ключа на записа и всички записи с една и същи хеш ключ се съхраняват заедно на диска. Използването на индексирани клъстери се препоръчва при възможност за неконтролируемо нарастване на размера на клъстерираните таблици или при достатъчен брой на заявки с условие за търсене върху обхвата на стойностите на клъстер-ключа. В клъстер се групират таблици, чиито колони участват често в условия за съединение и тези колони не се актуализират често. Таблици, чиито редове трябва да се преглеждат последователно или размера им не превишава повече от един или два блока в Oracle, също не се включват в клъстери. Формирането на индексиран клъстер е подходящо при обработването на връзки от вида 1:N.

При статични клъстерирани таблици и при възможност да се определи предварително максималния брой записи, и обем памет, необходима при създаването на клъстера, или при условия за търсене по съвпадение на стойностите върху всички колони, изграждащи клъстер-ключа, се използват хеш-клъстери. Не се препоръчва тяхното използване в следните случаи:

- размерът на таблицата нараства непрекъснато и се налага честа реорганизация на хеш-клъстера;
- при често последователно търсене в таблицата;
- при чести промени в стойностите на клъстер-ключа.

За създаването на различните видове клъстери в SQL е предвидена специална команда CREATE CLUSTER.

Съществен недостатък на клъстерирането е, данните могат да се групират върху диска по единствен начин. Ако проектанът на базата желае да приложи и друг начин на клъстериране, се налага дублиране на данните за всеки от моделите на групиране. Преодоляване на този недостатък е възможно при *многомерното клъстериране* (multidimensional clustering - MDC) (Liou & Yao, 1977) където едновременно се изграждат клъстери по наколко размерности без дублиране на данните. MDC е особено

[8] https://docs.oracle.com/cd/B19306_01/server.102/b14211/data_acc.htm#i2781, достъпен на 27.07.2017 г.

[9] https://docs.oracle.com/database/121/ADMIN/hash.htm#ADMIN019, достъпен на 27.07.2017 г.

полезно при средствата за аналитична интерактивна обработка и извличане на данни, където се изпълняват сложни заявки и се управляват големи обеми от данни. Изборът на грануларността и размерностите при клъстериране не е тривиална задача. Прилагането на техники за многомерно клъстериране е особено полезно при обработването на транзакции от тип „работен поток“. Тези техники са реализирани от IBM в ограничен брой търговски продукти като DB2 UDB за Linux, UNIX и Windows. Преимуществата на многомерното клъстериране като средство за проектиране на физическата база са дискутирани в (Kennedy, 2005). Прилагането му предоставя значително подобрение при обработването на заявки, поради намаляване на входно-изходните операции.

Материализиране на потребителски представи

Резултатът от изпълнението на заявки, при които се извличат данни от няколко таблици, може да се запази под формата нова таблица в базата от данни, наречена *материализирана представа* (materialized view) (Lightstone et.al., 2007). Подобни представи са особено полезни за ускоряване изпълнението на често повтарящи се заявки или заявки, при които се изчисляват агрегиращи функции. Запазването на подходящо множество от материализирани представи позволява значително намаление на времето за получаване на отговор.

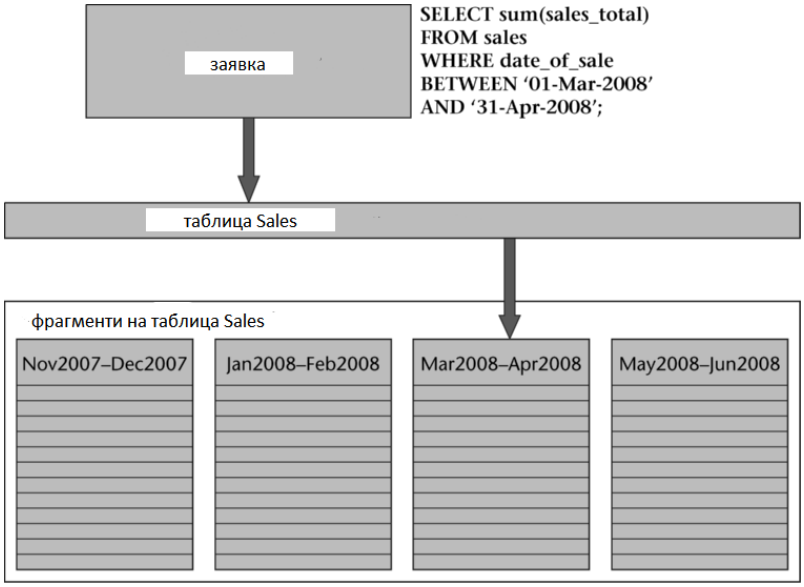
Практически е невъзможно да се съхраняват всички представи поради ограничения обем на паметта. Задача на проектанта е да определи кои да от тях да се материализират. Обикновено заявките се анализират и от тях се избират онези, при които се наблюдава изпълнение на голям брой входно-изходни дискови операции. Целта е повторно използване на вече получени резултати, което ускорява обработките. Следва да се отбележи проблемът с необходимостта от каскадно обновяване на представите при актуализиране на данните в базовите таблици. Каскадното обновяване на данните в добавените нови таблици към базата намалява ефективността от използването на материализирани представи.

Материализирането на потребителски представи се прилага при хранилищата с данни и средствата за интерактивна аналитична обработка.

Разделяне (partitioning)

При проектирането на физическата база разделянето е метод за намаляване на натоварването в произволен хардуерен компонент на системата (Ceri et al., 1982). Например данните от един диск могат да се разпределят върху няколко диска с цел уравновесяване на натоварването на устройството и предотвратяване на забавянето на входно-изходните операции. При *разделянето по обхват* (range partitioning) данните в дадена таблица се сортират и се разделят на сегменти (таблици с по-малък брой редове) с приблизително еднакъв брой от записи (фиг.13.7). Всяка група или фрагмент се разполага на отделен диск и може да се обработва независимо от останалите.

Разделянето по обхват почти винаги се използва за фрагментиране на данните върху единствен сървър. Не се предвижда разполагане на отделните дялове върху различни машини. Разделянето на таблицата по обхват остава прозрачно за приложенията. Поместването на отделните сегменти на различни устройства обаче преодолява ограниченията относно размера на таблицата, породени от адресирането на записите, където за целта се използва индентификатор на записа. Поради разделянето по обхват на дадена таблица практически става възможно записването на повече записи във всяка една от групите.



Фиг.13.7 Разделяне по обхват на данните

Преимущества на метода:

- съхраняват се таблици с по-големи размери поради повишените възможности за адресиране на техните записи;
- елиминира се обработването на част от сегментите, което позволява на компилатора да избира ефективен план за изпълнението на заявките;
- подобрено администриране на базата, защото АБД изпълнява специализираните команди (BACKUP, LOAD и др.) върху отделни сегменти, а не върху цялата таблица;
- неявно клъстериране на данните при входно-изходните операции.

Разделянето по обхват с известни модификации е реализирано в СУБД DB2, Informix и Oracle.

Други техники

Съществуват много други начини за осигуряване на ефективен достъп до данните в базата:

- компресиране - позволява разполагането на по-големи обеми от данни в определено пространство на диска като се осигурява по-бърз достъп при необходимост от преглеждане на голяма част от записите; следва да се отбележи допълнителното натоварване на системата поради компресиране на данните и връщането им в първоначалния вид;
- образуване на групи (data striping) – данните, до които е необходим едновременен достъп, се разпределят върху няколко диска с цел уравновесяване на натоварването и паралелно изпълнение на входно-изходните операции; при това се постига намаляване на времето за обработка на заявките и се увеличана производителността на системата; техниката е особено подходяща при наличието на съответен хардуер например масиви от независими дискове;
- дублиране на данните върху няколко диска – необходимостта от обновяване на всички копия едновременно, както и заемането на допълнителна памет, е съществен недостатък на този метод; докато цената на паметта намалява непрекъснато, същото не може да се твърди

за времето; ако данните само се четат или се обновяват рядко тогава е добре да се съхраняват с излишество;

- денормализиране на данните – с оглед ефективното изпълнение на заявки и транзакции към базата се допуска нарушаване на изискването за трета нормална форма, като част от отношенията се съхраняват в по-ниски нормални форми; решението се взима след анализ на честотата на изпълнение, обем и приоритет на обработките, като се отчитат страничните ефекти от евентуална загуба на състоятелността на базата; целта е да се намали изпълнението на операцията „съединение“.

13.5. Добри практики при проектиране на физическата база

Проектирането на физическата база е процес, при който не само се избират подходящи структури за съхранение на данните в паметта. Този процес трябва да гарантира също така и добра производителност. За една концептуална схема съществуват различни алтернативни проекти на физическата база, с използване на конкретна СУБД. Затова решения относно проекта на физическата база се вземат от администратора или проектанта на базата след изследване на заявките, транзакциите и приложенията, които се очаква да работят върху нея. Анализът на производителността обаче е свързан с трудоемки изчисления както и със сравнение на различни конфигурации на структурите за достъп и натоварване на базата. За да се облекчи работата по проектиране на физическата база се предлагат средства, които автоматизират тази дейност – например DB2 Design Advisor на IBM (Zilio D. et al., 2004), SQL Access Advisor и SQL Tuning Advisor на Oracle (Burleson, 2014) и Database Tuning Advisor[10] на Microsoft. Те подпомагат настройването на базата и анализа на производителността на системата като автоматизират трудоемките изчисления. Продуктите са сравнени в (Lightstone et.al., 2007).

За всяка база от данни теоретично може да се намери оптимален проект, но в действителност никой не си поставя подобна задача заради цената на самата оптимизация. Практически се прибягва до евристични решения и се търси ефективен, не непременно оптимален физически проект на базата от данни. Да отбележим, че „ефективен проект“ е доста размито понятие, но обикновено се има предвид потребителската удовлетвореност от работата на системата. В повечето от случаите перфектния физически проект на базата е непостижим и скъпо струващ. Прилага се метода „проба-грешка“ и се изследват различни конфигурации. Затова използването на различни автоматизиращи следства за облекчаване на работата е препоръчително. Освен това следва да се изследват различните методи за проектиране на физическата база, предоставени от използваната СУБД.

Фактори, които влияят върху физическия проект на базата от данни

Преди да започне проектирането на физическата база е необходимо да се установи как тя ще се използва от приложенията и потребителите. За целта се анализират:

1. Изпълнявани заявки и транзакции.

За всяка заявка, с която се извличат данни от базата се събират данни за:

- таблици (файлове), до които се осъществява достъп;

[10] https://docs.microsoft.com/en-us/sql/tools/dta/tutorial-database-engine-tuning-advisor, достъпен на 27.07.2017 г.

- колони, които участвуват в условието за избор;
- тип на сравненията в условието за търсене;
- колони, които участвуват в операция „съединение“;
- колони, чиито стойности се извеждат в резултата от заявката.

Обикновено структури за достъп, например индекси, се дефинират върху колоните, които участвуват в условие за избор или операция „съединение“.

За всяка заявка или транзакция, които обновяват базата се събират данни за:

- файлове, които се обновяват;
- тип на операцията;
- участуващи в операции „изтриване“ или „модифициране“ стойностите на колони.

Структури за достъп се дефинират върху колоните, които участвуват операции „изтриване“ или „модифициране“, защото чрез тях намирането на необходимия за обработване запис става по-бързо.

2. Честота на изпълнение на транзакции и заявки.

Целта е да се оцени участието на всяка колона в условия за търсене или съединение. Практиката показва, че 80% от обработките се извършват от около 20% от заявките или транзакциите. Поради тези причини статистика се води само за тези 20%, които са най-често използвани.

3. Изисквания към време за отговор на транзакции и заявки.

Изиксвания към времето за отговор указват влияние върху определяне на структурите за достъп. Например колоните, по които се осъществява извличане на данни от заявка с ограничения във времето за отговор, е добре да се индексират.

4. Честота на обновяване записите на даден файл.

При файлове, чиито записи често се обновяват трябва да се дефинират минимален брой структури за достъп, защото самите те подлежат на променяне всеки път когато се въведе нов елемент. Допълнителните системни операции, изпълнявани от СУБД, могат да забавят обработките значително.

5. Ключови колони.

Обикновено структури за достъп се задават върху колони, които са главни или кандидат ключове в таблицата. По този начин се спазва изиксването за уникалност на стойностите на ключа като идентификатор на отделните редове.

След анализ на посочените по-горе данни се определят файловата организация и пътищата за достъп до файловете от базата.

Как да се индексира базата?

Изпълнението на заявките до голяма степен зависи от наличието на индекси или схеми за хеширане, които да ускорят изпълнението на операциите „селекция“ и „съединение“. При голям брой на операциите за обновяване обаче трябва да се отчита, че актуализирането на индексите от СУБД допълнително забавя обработката поради допълнителния брой входно-изходни операции и използваното процесорно време. Освен това индексирането има цена, която се измерва с необходимото дисково пространство за съхраняване на самия индекс. При изграждане на голям брой индексни структури заетото от тях място може да се окаже съизмеримо с това на файла, съдържащ данните!

Определяне на множеството от индекси

Индекс върху дадена колона се създава когато тя е ключ или се използва в условия за търсене или съединение. Съставен индекс се използва при наличие на повече заявки, чиито условия за търсене са формулирани върху няколко колони от дадена таблица.

Множеството от индекси се определя след оценяване производителността на базата с помощта на модел, който пресмята времето за входно-изходни операции и закъсненията по мрежата (Lightstone et.al., 2007). По този начин могат да се сравнят различни проекти на физическата база и да се сравни времето за обработване на множество от заявки.

Определяне типа на индекса

Индексите могат да бъдат реализирани с различни структури и да имат специфично приложение, което определя техния вид. Различават се:

- индекси, реализирани чрез B+ дървета;
- индекси, реализирани чрез хеш таблици;
- съставни индекси;
- клъстер индекси;
- присъединителни индекси;
- растерни индекси;
- компактни/разпиляни.

За даден индекс са възможни различни комбинации между използването и реализацията с определена структура. Например с помощта на предоставените от съответната СУБД възможности потребителят може да създаде съставен индекс, реализиран чрез B+ дърво или хеш таблица.

Вградените в СУБД методи за достъп се използват за разглеждане на таблиците и индексите. Обикновено за реализиране на индексите повечето СУБД използват B+ дървета, които ускоряват изпълнението на заявки с условие за търсене, формулирано върху индексното поле. Някои системи позволяват използване на хеш индекси или индексно последователен достъп до записите. Хеш индексите работят добре при заявки, налагащи изпълнението на съединение по еквивалентност или чийто резултат е единствен ред от дадена таблица, но за съжаление не се поддържат от повечето СУБД. Когато се налага извличане на група от записи, удовлетворяващи някакво условие, по-добро решение са клъстер или реализираните чрез B+ дървета индекси, а не хеш индексите. Клъстер индекси се поддържат от DB2, Microsoft SQL Server, Oracle и Sybase. Растерни индекси се използват при големи обеми от данни, защото в тези случаи се получават особено обширни B+ дърветата.

Практически правила при индексиране на базата

Правило 1: Индекси се създават върху главния и чуждите ключове в базата. Тези структури изграждат началния проект на физическата база, независимо от нейното бъдещо натоварване. Причина за това решение е участието на ключовете в операцията „съединение“. От своя страна натоварването е трудно предвидимо предварително и често се променя при работата на дадено приложение с базата. Противно на общоприети схващания се препоръчва използването на индекси и при таблици с малък брой редове. Доказано е, че липсата им при определени обстоятелства води до особено лоша производителност.

Правило 2: Индекси се изграждат върху атрибути, които се срещат след оператора WHERE в заявка, формулирана на SQL когато оператора за сравнение е „=“. При оператор за сравнение „<>“ индексите се използват рядко, поради високата селективност на предиката, т.е. резултатът съдържа голям брой редове, които удовлетворяват условието.

Правило 3: Излишните или дублирани индекси трябва да се премахват. Те усложняват ефективното изпълнение на заявките, но от друга страна трябва да се следи дали отстраняването им не забавя някои от обработките.

Правило 4: Нови индекси се добавят само при доказана необходимост – например, ако дадена таблица се преглежда последователно достатъчно често, а резултатът от заявката съдържа малко на брой редове (Hubel, 2001).

Правило 4: Върху колони, чиито стойности се актуализират често, индекси се създават при необходимост. За целта се отчитат входно-изходните операции, необходими за обновяването на структурата и таблицата и се сравняват със съответните операции при изпълнението на заявки с използване на индекса. Индекс не се създава, ако не допринася за ефективно изпълнение на заявката.

Правило 5: Индексите трябва да се поддържат регулярно. При индикации, че даден индекс забавя обработките, особено при командите за обновяване на базата, е добре той да се изтрие или реорганизира. При унищожаването на структури обаче трябва да се внимава с ограниченията за цялостност върху чуждите ключове.

Правило 6: Индексите подпомагат изпълнението на заявките, с които се извличат данни и забавят операциите за обновяване на базата. Затова при интензивно актуализиране на данните се дефинират по-малко на брой индекси.

Правило 7: Размерът и броя на индексите се променя при обработката на данните. Индексите обикновено трябва да заемат между 10 и 20% от дисковото пространство на базата. Заемането на над 25% от него е знак за извършване на сериозен преглед и ревизия на създадените структури.

Избор на потребителски представи за материализиране

Материализираните представи представляват предварително изчислени резултати на някои заявки, които се съхраняват на диска. Обикновено това са заявки, които се използват често и данните се извличат след изпълнение на операцията „съединение“ върху големи по размер таблици, което води до интензиви входно-изходни операции. Да отбележим, че базата от данни е в трета нормална форма. Целта на материализирането е изведат бързо резултатите без да се изпълняват едни и същи обработки многократно.

Практически правила при избор на потребителски представи за материализиране

Правило 1: Запазват се резултати от заявки, които се изпълняват често и натоварват системата с много входно-изходни операции.

Правило 2: При хранилища с данни могат да се материализират доста представи. Схемата от тип “звезда”, използвана за реализация на многомерния модел (вж.Глава 12), позволява избор на голямо множество от заявки, чиито резултати да се съхраняват и използват многократно.

Правило 3: Определя се стратегия за актуализиране на всяка материализирана представа. Тя зависи от обема на променените данни и необходимостта от обновянето им в реално време. Възможно е постепенно обновяване във фиксиран времеви интервал извън часовете на работа на системата или непосредствена актуализация при приключване.

Правило 4: Определя се разумен брой на материализирани представи. Съхраняването на повече от тях води до забавяне изпълнението на заявката и до заемането на място върху диска.

Правило 5: Определя се размера на дисково пространство, предназначено за съхраняване на материализирани представи. Повечето хранилища на данни използват за тази цел от 10 до 20% от общата памет.

Клъстериране на физическата база

Многомерното клъстериране подобрява производителността основно чрез намаляване на броя на входно-изходните операции. Разработва се поради нарастването размера на релационните бази и необходимостта да се управлява изпълнението на сложни заявки. Системите, подпомагащи вземането на решения, както и средствата за интерактивна аналитична обработка изиксват изпълнението не само на транзакции, но и на многомерни операции върху данните. Доказано е, че производителността на заявките се подобрява значително при прилагане на клъстериране.

Практически правила при клъстериране на базата

Правило 1: В клъстер-ключ участвуват колоните, използвани при началното индексиране на базата, т.е. тези които се използват в предикати за селекция или сортиране на данните.

Правило 2: Клъстер-таблицата не трябва да съдържа прекалено много колони. Обемът на заеманата от нея памет не бива да е по-голям от този на участвуващите таблици, увеличен с 5 до 10% за всяка от тях и се изчислява определя приблизително със следната формула.

$$W = \eta_{cells} \cdot \mathrm{P}_{\%} \cdot \beta$$

където η_{cells} е броя на неповтарящите се колони в клъстер-таблицата, $P_{\%}$ - празно място, оставено във всеки блок за всяка колона, β – размер на блока.

Правило 3: Не се препоръчва изграждане на клъстери по повече от три размерности, защото практическото им приложение не оправдава обема на заетата памет.

Правило 4: Препоръчва се изграждане на клъстери по една размерност, защото експерименти доказват че те ускоряват обработките повече от едномерните традиционни клъстер индекси.

Правило 5: Препоръчва се провеждане на експерименти с различни проекти за многомерно клъстериране с цел намиране на най-добре работещия от тях за дадено приложение.

Разделяне на физическата база

Разделянето по обхват на физическата база се използва преобладаващо за подобряване администрирането на системата и не влияе толкова сериозно върху ефективното изпълнение на заявките. При него става възможно изпълнението на сервизни програми като дефрагментиране, клъстериране и архивиране само върху

отделните сегменти, а не върху целите таблици. Експерименти доказват значително подобряване на производителността понеже сервизните програми, които обикновено се изпълняват самостоятелно, работят върху по-малки части от базата. Освен това се обработва единствен сегмент, което междувременно позволява използването на останалите и намалява риска от загуби на всички данни при настъпване на отказ в системата.

Зареждането и архивирането на данни от хранилища обикновено се осъществава от оперативните бази веднъж седмично или месечно. При разделянето по обхват е възможно обработване на отделните сегменти.

Практически правила при разделяне на физическата база

Правило 1: Разделянето по обхват се използва при големи таблици, за да се улесни адресирането на записите им.

Правило 2: Разделянето по обхват е подходящо при хранилищата където се зареждат или архивират данни, свързани с определени времеви интервали.

Правило 3: При разделянето по обхват най-често се използва колоната, съдържаща времеви данни – например дата. Могат да се изследват и други колони, по чиито стойности таблицата да се сегментира, за да се получи ефентивно изпълнение на определени групи от заявки.

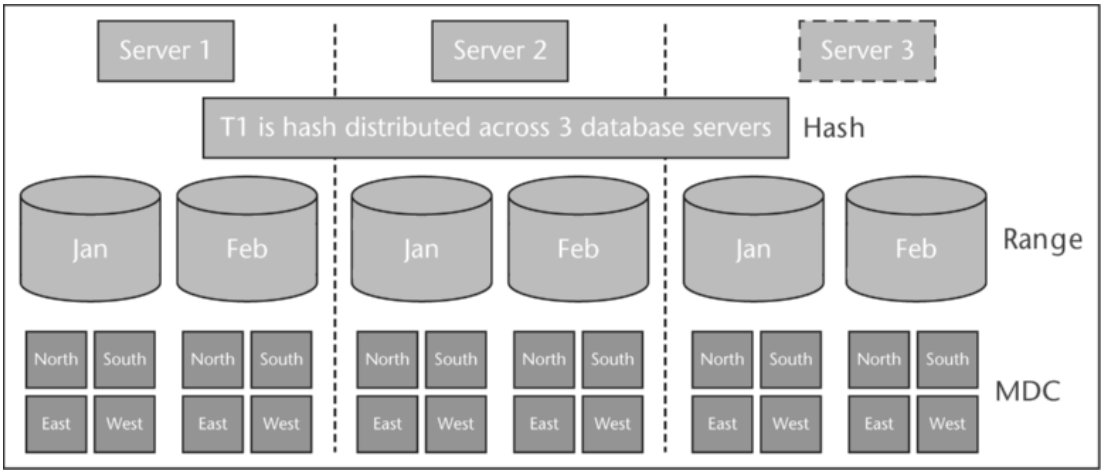
Правило 4: При зареждане или архивиране на данни от хранилище сегментирането използва времевия период – например месец, тримесечие или година.

Правило 5: Разделянето по обхват се използва за да се подобри изпълнението на сервизните. програми.

Правило 6: Разделянето по обхват се използва за бързо архивиране на данни за определен времеви период – например при добавянето на данни в хранилището за нов месец, сегменът, съдържащ исторически данни за най - стария месец се архивира.

Правило 6: Разделянето по обхват обикновено се реализира чрез множество от таблици. Затова не се препоръчва сегментиране, което съдържа като резултат хиляди фрагменти. Допустимият обем на сегмента е най-малко 50 MB данни като няколко гигабайта за фрагмент е желателен размер. Броят на дяловете не трябва да надвърля повече от 500 сегмента за таблица.

Правило 7: Разделянето по обхват, многомерното клъстериране и други подобни техники за проектиране на физическата база, които оформят групи от данни по различни начини, трябва да се използват внимателно съвместно върху една и съща таблица. Не се препоръчва комбинирането на повече от два метода, защото администрирането на системата силно се усложнява, а резултатът върху производителността остава неясен. На фиг.13.8 е представено използването на няколко техники. Първоначално базата е фрагментирана върху три отделни сървъра с използване на хеширане. В рамките на всеки сървър данните са сегментирани по месеци и е приложено многомерно клъстериране по географски региони.



Фиг.13.8 Сегментиране и многомерно клъстериране (източник IBM)

Резюме

В настоящата глава се разглеждат основни понятия, свързани с физическото съхранение на базата върху външен носител. Понеже данните са устойчиви и с голям обем те не могат да се пазят в основната памет на компютъра, която е енергозависима и със сравнително по-висока цена. Независимо, че СУБД предлага вградени възможности за организация на данните, необходими са познания относно реализацията на базата във вид на файлове и пътища за достъп.

Проектът на физическата база е особено важен при бази с голям обем, съдържащи различни видове данни: структурирани (релационни кортежи), полуструктурирани (например XML) и неструктурирани (например аудио, видео или уеб страници). Под "проектиране на физическата база" се разбира избор на най-подходящите за дадено приложение методи за организация на данните.

Физическата база се изгражда от отделни елементи: файлове със записи и структури, ускоряващи извличането на определени данни. Операциите върху базата са ефективни благодарение на реализирани в конкретната СУБД методи за търсене. Съществена е разликата между физически и логически аспект на данните. Логическият запис се състои от данните, с които потребителят или приложната програма работят. Реално това е съвкупността от полетата на записа. Физически запис или блок е единицата информация, която реално се чете от едно устройство или се записва на него. Той отразява начина на съхранение и представяне на данните върху определен носител.

От гледна точка на базите операциите върху файлове биват: „търсене“ и „обновяване“. И при двата вида характерното е, че се избира запис или група от записи, които удовлетворяват някакво условие. Действителните операции върху файлове зависят от конкретната операционна система. Всяка СУБД осигурява достъп до записите в базата чрез различни видове команди.

Начинът на съхранение и достъпът до записите в базата зависят от файловата организация. Тя е особено важен фактор за проектирането на бази от данни с добра ефективност при различните обработки. Затова в главата се провежда анализ на основните файлови организации:

- неструктуриран файл;
- подреден файл;
- директен файл.

Тези организации се прилагат когато всички записи в даден файл са от един и същ тип. При някои приложения, обаче се установяват връзки между записи от различни файлове с помощта на свързващи полета. Подобни свързани записи обикновено се поместват от СУБД в една област и връзките се реализират чрез физически указатели. Накратко са разгледани и някои хардуерни решения (масиви от независими дискове, локални мрежи за съхранение, специализирани устройства), които ускоряват извличането на данните.

В главата се дискутират основни методи за проектиране на физическата база. Особено внимание се отделя на индексирането като главно средство за подобряване производителността при извличане на данните. Разгледани са и други техники като клъстериране, разделяне, материализиране на потребителските представи, които осигуряват ефективен достъп до данните в базата. Систематично са представени добри практики, свързани с проектирането на физическата база.

Допълнителна литература

1. Connolly T., Begg C. (2015). Database Systems a Practical Approach to Design, Implementation, and Management, Глава 18. Boston: Pearson.
2. Elmasri R., Navathe S. (2016). Fundamentals of Database Systems, Глави 16 и 17. Boston: Pearson.
3. Ramakrishnan R., Gehrke J. (2003). Database Management Systems, Глави 8, 9, 10 и 11. McGraw-Hill.