***Увод в системите бази от данни***

Базите от данни са в основата на всеки бизнес. Освен това, те намират редица приложения при научни изследвания.

Базите от данни се управляват от специален софтуер, който ще наричаме **СУБД** (система за управление на бази от данни).

СУБД представлява мощен инструментариум за създаване и ефективно управление на големи обеми от данни. Данните трябва да се поддържат и съхраняват за толкова време за колкото е необходимо. Освен това, те трябва да се предпазват от неправилен достъп, който може да наруши целостта им (integrity), както и от неправомерен достъп (security).

Поради това, в наши дни СУБД е доста сложен софтуер.

Примери за СУБД са **Oracle**, **DB2**, **Microsoft SQL server**. Първите две системи, за разлика от третата се използват в сериозни организации.

По-нататък под **база от данни** ще разбираме наборът от данни, които се поддържат в даден момент от една СУБД.

От потребителска гледна точка една СУБД трябва да притежава следните свойства:

* **устойчива памет** – данните трябва да се съхраняват независимо от процесите; базите от данни трябва да позволяват ефективен достъп до много големи обеми от данни – времето на достъп до данните трябва да е независимо от техния обем;
* **програмен интерфейс** – СУБД предоставя мощен език за заявки на потребителя или на приложната програма, която използва данните;
* **управление на транзакции** – СУБД поддържа едновременен достъп до данните; той се осъществява чрез множество процеси, наречени **транзакции**; всеки потребител, който работи с данните трябва да инициира такъв процес; СУБД поддържа изолация на транзакциите – всяка транзакция е **атомарна**, т.е. тя или завършва изцяло или бива изцяло отхвърлена; ако по време на изпълнение на една транзакция възникне проблем тя се отменя, заедно с всички изменения, които е причинила върху базата от данни;
* **устойчивост на данните** – при възникване на аварии или грешки данните трябва да могат да бъдат възстановени; естествено, това е свързано с допълнителни операции за четене и писане.

От архитектурна гледна точка една СУБД трябва да предоставя следните функции:

* СУБД позволява на потребителя да създава нови бази от данни;

за целта потребителят задава **схема** на базата от дании; схемата е логическа структура на данните и тя се описва с помощта на специален **език за дефиниция на данните** (**DDL** – data definition language); този език е символен или графичен;

* СУБД трябва да осигури възможност за търсене в данните и за модификация на данните; това се постига с помощта на **език за манипулация с данните** (**DML** – data manipulation language); **езикът за заявки** е подезик на езика за манипулация с данните, който се използва при търсене на данни;
* СУБД трябва да съхранява безопасно и дългосрочно много големи обеми от данни; съхраняването трябва да осигурява ефективно търсене и изменение на данните;
* СУБД трябва да управлява едновременен достъп до данните от много потребители - те не трябва да си влияят един на друг и да развалят целостта на данните.

**Еволюция на СУБД**

Първите СУБД се появяват през 60-те години като резултат от развитието на файловите системи най-вече в посока към безопасно и дългосрочно съхранение. Те намират приложение при резервацията на самолетни билети, при банковите системи, при корпоративните записи и др. При тях визуализацията на данните е така както те се съхраняват.

Естествената еволюция на файловите системи доведе до поява на следните два модела:

* **йерархичен модел** – например IMS на IBM; при него данните са във вид на дърво; в действителност някои данни може да присъстват в повече от едно дърво, ако към тях се използват връзки (reference);
* **мрежови модел** – този модел се появи по-късно; при него навигацията в данните е на значително по-високо ниво.

През 70-те години за пръв път се появиха СУБД, които се подчиняват на така наречения **релационен модел**, който измести напълно другите два модела. Този модел е разработен от Тед Код (Ted Codd).

При релационните СУБД за първи път се появява език за заявки от високо ниво.

Релационният модел почива на сериозни математически основи, което стимулира серия от изследвания в областта на базите от данни.

Този модел е изключително прост от гледна точка на потребителя – използва се само една структура от данни, наречена **таблица** (**релация**). Реализацията на таблицата е скрита за потребителя. Използват се езици за заявки от високо ниво. Примерна таблица:

|  |  |  |
| --- | --- | --- |
| accountNo | balance | type |
| 01234 | 1000.00$ | savings |
| 56789 | 3849.40$ | checking |

В таблицата всяка колона си има име, което се нарича **атрибут** (accountNo, balance, type). Всеки ред в таблицата се нарича **кортеж** (имаме два кортежа). В рамките на една колона стойностите са еднотипни.

Езикът **SQL** (**structured query language**) е разработен във връзка с реализацията на релационния модел. В наши дни по стандарт всички СУБД използват SQL. Направени са изследвания, които показват, че

най-честите заявки са в рамките на една таблица. Поради тази причина, основната конструкция, която се използва в SQL е следната:

*SELECT име\_на\_атрибут*

*FROM име\_на\_таблица*

*WHERE условие*

Условието може да има следния вид *име\_на\_атрибут=стойност*.

В зависимост от типа вместо равенство може да се използва някой друг предикат за сравнение. Няколко условия могат да се комбинират с помощта на логическите съюзи.

Ще опишем как се изпълнява конструкцията. В таблицата, зададена във FROM клаузата се извършва търсене по всички кортежи, като се проверява условието, зададено в WHERE клаузата. SELECT клаузата определя кои атрибути на намерените кортежи да бъдат изведени. Примери: (предполагаме, че горната таблица се казва Accounts)

*SELECT* balance

*FROM* Accounts

*WHERE* accountNo=01234

Тук ще се изведе 1000.00$.

*SELECT* accountNo

*FROM* Accounts

*WHERE* type=’checking’ AND balance > 0

Тук ще се изведе 56789.

СУБД оптимизират заявките за да може те да бъдат ефективно изпълнени.

През средата на 90-те години се появиха обектно-ориентирани СУБД.

Въпреки че не успяха да се наложат, те повлияха на релационния модел и в наши дни СУБД работят с **обектно-релационен** **модел**. Този модел е разширение на обектно-ориентирания модел, но запазва пълна съвместимост с релационния модел. Обектно-ориентираният подход е водещ при проектирането на софтуер и данни в наши дни.

Има тенденция СУБД да са вградени в операционните системи и в бъдеще да изместят файловите системи.

СУБД се развиват в посока на съхранение на данните във връзка с мултимедията, която е много голям по обем (например от порядъка на терабайтове). Концепцията на работа на една СУБД е, че данните се обработват в оперативната памет, а се съхраняват върху дискове. Проблемът при големите бази данни е, че дисковете все още не са достъчно големи. Поради тази причина се промени моделът за съхранение на данните. Той беше на две нива – оперативна памет и вторична памет. Сега вече се появи и **третична памет** – шкафове с носители, които се управляват от роботи (например Juke Box, лентови силози). Недостатъкът е, че времето за достъп до третичната памет е много голямо – приблизително няколко секунди, за разлика от вторичната памет, при която това време е около 10-20 ms.

За да може да се оптимизира управлението на различните памети се използват **индексна структура** на данните или **паралелизъм** в няколко варианта. Паралелизмът може да означава паралелно работещи процесори, паралелно работещи дискове. Възможно е използването на така наречените **разпределени системи** – при изпълнението на една заявка тя се разбива на няколко части, които се обработват от различни разпределени компютърни системи.

**Концепции в развитието на СУБД**

**Client-server** е архитектура, при която един процес (**клиент**, **client**) изпраща заявка за изпълнение към друг процес (**сървър**, **server**). Когато сървърът изпълни заявката, той връща данните на клиента. Естествено, СУБД е сървърът, а клиентите са потребителите или приложните програми. При такава организация се постига полезно натоварване на мрежата.

Проблем при архитектурата client-server е претоварването на сървърите. Поради тази причина се налагат архитектури client-server с множество нива. При тях СУБД отново е сървърът, но негови клиенти са така наречените **приложни сървъри**. Те имат за клиенти **уеб-сървъри**, които вече взаимодействат с потребителите. Приложните сървъри имат за задача да реализират бизнес-логиката на СУБД. Чрез такова разслоение отговорността за изпълнение на заявките се поделя между различни компютърни системи.

Един проблем на съвременните СУБД е търсене в мултимедийни данни. Възможно е мултимедийните обекти да се индексират с ключове, по които да се осъществява търсенето. Такъв подход, обаче, е неприложим в следните ситуации – търсене на кадър от филм, търсене на фрагмент от картинка и т.н.

Обектно-релационният модел поддържа операции за работа с мултимедийни обекти. Стремежите са данните да се визуализират пред потребителя във виртуална реалност.

Има тенденция когато една база от данни се създаде, тя да може занапред да съществува самостоятелно. За целта се използват **складове за данни** (**data warehouse**), в които информацията се интегрира и унифицира. Във връзка с постиженията в изкуствения интелект са разработени така наречените **OLAP технологии** (**OLAP technologies**), които позволяват ефективен анализ на данните в складовете. С тези технологии може да се осъществява търсене по образци (**data mining**).

**Структура на СУБД**

DDL команди

потребител/

приложение

компилатор на заявките

заявки

обновяване

изпълнителна машина

план на заявките

управление на индекси, файлове, записи

заявки към индекси, файлове, записи

управление на буферизация

управление на външна памет

външна памет

управление на транзакции

транзакционни команди

администратор

DDL компилатор

журнал

управление на едновременния достъп

журнални страници (log информация)

метаданни статистики

данни метаданни индекси

метаданни

постранични команди

четене/писане по страници

буфери

таблици на блокировката

Потребителят дава заявка към системата. Това става в интерактивен режим. Възможно е това взаимодействие да се осъществява чрез приложение. Самите заявки не променят базата от данни, докато при обновяване тя се променя.

**Компилаторът на заявките** изяснява дали синтаксисът на заявките е правилен и след това проверява тяхната семантика. В крайна сметка компилаторът на заявките изработва **план на заявката**. Обикновено този план е във вид на дървовидна структура. Във възлите на структурата стоят операции от релационната алгебра, а в листата се намират константи, имена на атрибути или цели таблици. Тъй като заявките са от високо ниво се изисква те да бъдат оптимизирани. Това е функция на компилатора на заявките. Всъщност, той се състои от три части:

* синтактичен анализатор;
* семантичен анализатор;
* оптимизатор на заявки.

При оптимизиране на заявките възниква един фундаментален проблем – езиците за заявки са ориентирани за сравнително прости заявки. Например, SQL е разработен върху концепцията, че най-често заявките са в рамките на една таблица, по-рядко върху две таблици и много рядко върху три или повече таблици. При оптимизиране на проста заявка може да се загуби ефективност – времето за оптимизиране да надхвърля многократно времето за изпълнение на заявката.

Обикновено в интерактивен режим заявките не се оптимизират. Когато обаче една заявка се съхранява, т.е. тя ще се изпълнява много пъти, то тя се оптимизира. Освен самият сорс на заявката се съхранява и нейният оптимизиран план (естествено, оптимизацията се извършва само веднъж при първото изпълнение на заявката).

**Изпълнителната машина** има за задача да изпълнява заявките по плана, който е предоставен от компилатора на заявките. Изпълнителната машина изпраща заявки към файлове и заявки към индекси за да извърши своята работа. Всъщност, изпълнителната машина използва различни методи на достъп, познавайки вътрешната структура и организация на физическо ниво на таблиците. Независимо от метода на достъп, той се поддържа във вид на страници.

Външната памет реално е организирана на страници, т.е. четенето и писането се извършва постранично. Размерът на една страница може да е 4K, 16K и др. Обикновено данните се прехвърлят между оперативната памет и външната памет. При СУБД ефективността на една заявка се измерва в това колко постранични операции (прехвърляния) се извършват при нейното изпълнение.

Управлението на външната памет реално извършва четенето и писането върху външната памет.

Управлението на буферизацията се занимава с това дали търсените страници не се намират в буферите на оперативната памет. По този начин се спестяват дискови операции.

Обикновено оптимизациите на заявките се извършват независимо, т.е. съобразно само една заявка. Има СУБД при които в компилатора на заявките постъпват по няколко заявки едновременно, които се компилират в общ план и съответно върху тях се извършва обща оптимизация. Ефективността на такава стратегия силно зависи от управлението на буферизацията.

**Транзакциите** са процесите в СУБД. Изпълнението им трябва да бъде такова, че те да не си влияят една на друга и да изглежда все едно се изпълняват последователно една след друга.

Когато един потребител работи интерактивно със системата, всяка негова заявка се счита за една транзакция. Той, обаче, може да изпълнява няколко заявки в рамките на една транзакция с помощта на специални команди. Има и други команди за управление на транзакциите.

Модулът управление на транзакциите използва друг модул – **журнал**,

в който се регистрират транзакциите. По време на изпълнението на една транзакция, изпълнителната машина записва в журнала всякакви промени, които извършва върху транзакцията. Грубо казано, журналът се състои от записи. Всеки запис има нова част и стара част. Изпълнителната машина извършва промените върху новата част. Ако по време на изпълнение на транзакцията възникне проблем, т.е. тя не успее да завърши, чрез старата част на записа се възстановява старото състояние на базата от данни.

**Администраторът** на базата от данни отговаря за нейната структура. Освен това, той може да дава различни права на достъп (например read, write, read/write) на други потребители. Администраторът дефинира схемата на базата от данни като подава DDL-команди на

DDL-компилатора. Самите DDL-команди след компилация се изпълняват от изпълнителната машина.

Управлението на едновременния достъп е модул, който има за задача да гарантира атомарността на транзакциите. Това става с помощта на таблици на блокировката, които се намират в оперативната памет.

Когато една транзакция се обработва от модула управление на транзакциите, то е известно кои таблици ще се използват за четене/писане в рамките на транзакцията. Тогава този модул се обръща към управлението на едновременния достъп, при което се блокират съответните таблици (релации). В таблицата на блокировката се съдържа информация кои от релациите са блокирани само за четене или за четене и за обновяване. Ако се прави опит за повторно блокиране за четене, няма проблем, т.е. от една релация едновременно могат да четат много транзакции. От друга страна, само една транзакция може да блокира релация за обновяване, т.е. ако една транзакция пише в релация, то други транзакции нямат достъп до тази релация (нито за четене, нито за обновяване). Най-често блокировката се извършва на отделни релации. Възможно е блокировката да се извършва на ниво кортежи, т.е. върху една и съща релация може да работят едновременно няколко транзакции, но върху различни кортежи. Ако блокировката е на по-едро ниво, то се губи ефективност. Ако блокировката е на по-ситно ниво, то се усложнява значително управлението на едновременния достъп.

Транзакциите са процеси – възможно е няколко транзакции да попаднат в дедлок. Механизмът, който се използва за преодоляване на този проблем е откриване и възстановяване от дедлок – всички транзакции, които участват в дедлока се елиминират.

**Буферите** се намират в оперативната памет и с тях взаимодейства модула управление на буферизацията. Ако една страница се прочита от външната памет, то тя първо се записва в буферите.

Изпълнителната машина взаимодейства с модула за управление на едновременния достъп – например, тя изпълнява командите на този модул.

Журналът се управлява самостоятелно и работи пряко с управлението на буферизацията. Всички останали модули се изпълняват на изпълнителната машина. Журналът, обаче, слиза на по-ниско ниво, тъй като той се нуждае от гаранция, че неговите изменения са направени върху диска преди да се обработва базата от данни.

**Метаданните** са описание на схемата на базата от данни и те се разполагат в буфери в оперативната памет. DDL-компилаторът се нуждае от метаданните – например, ако администраторът изменя

базата от данни.

Журналните страници също се разполагат в буфери в оперативната памет. Компилаторът на заявките също има свои буфери – метаданни, които се използват при проверка на семантиката и статистики, които се натрупват от СУБД и се използват при оптимизиране на заявките.

В хода на своята работа изпълнителната машина също използва

буфери – метаданни, данни, индекси.

Често при описание на свойствата на транзакциите се използва съкращението **ACID** – атомарност (atomicity), цялостност (consistency), изолирано изпълнение (isolation), надеждност и устойчивост (durability).

По-отношение на СУБД се изучават следните компоненти:

* проектиране на базите от данни – каква информация се съхранява в базата от данни;
* структура на базата от данни – какви структури от данни, типове от данни се използват и какви са връзките между типовете данни;
* програмиране на базата от данни – програмиране на заявки и на конвенционални програми;
* реализация на СУБД – този въпрос няма да го разглеждаме подробно.