5. Език SQL

Езикът SQL (Structured Query Language) притежава елементи на релационното смятане и релационната алгебра. Създаден е в IBM, където през 1974-75г. се разработва една от първите релационни СУБД System/R. Езикът, създаден за тази система, е наречен SEQUEL (Structured English Query Language). След опитна експлоатация, тестване и развитие на езика, той е преименуван в SQL и проекта System/R завършва. Така започва историята на езика SQL, който сега де-факто е стандарт за релационните СУБД. Списъкът на СУБД, реализиращи SQL като език за БД, е много дълъг, само някои от тях са: SQL/DS, DB2, Oracle, Informix, Sybase, Ingres, Postgres, SQL Server, MS Access.

През 1982г. към ANSI е сформиран комитет за стандартизация на SQL, а през 1986г. е приет първия стандарт SQL X3.135, който през 1987г. е приет и от ISO. Тъй като езикът се развива и стандартизацията му продължава. По-късно са приети и следващи стандарти: SQL/89 или SQL1, SQL/92 или SQL2 и SQL3. Въпреки многото стандарти има и много промишлени диалекти, които се различават един от друг. Но все пак повечето съвременни промишлени СУБД реализират езика в съответствие с SQL2, като го разширяват и с нови възможности. Ще разгледаме основните средства на езика, като ще се придържаме към SQL2 .

5.1. Общи елементи

Всеки оператор започва с ключова дума-глагол и включва една или повече фрази. Всяка фраза започва с ключова дума, като някои фрази са задължителни, а други не. Ключовите думи в SQL2 са около 300 и в повечето реализации могат да се пишат с малки или главни букви, без да се прави разлика.

5.1.1. Имена на обекти

Обектите в БД, които се именуват, са таблици, колони, потребители, ограничения за цялостност, индекси, синоними и view. Характерно за SQL е, че се използва нематематическата терминология за обектите в релационния модел, а именно – таблица, колона и ред. Според SQL1 име на обект в БД е символен низ с максимална дължина 18 символа, който започва с буква и съдържа букви, цифри и символа "_". В SQL2 се въвеждат дълги имена до 128 символа.

Някои обекти имат и пълни имена. Пълното име на таблица се състои от име на потребителя, който е собственик и собственото име на таблицата и се записва:

```
име_собственик.име_таблица
```

Аналогично понякога се налага използване на пълно име на колона. Ако в един оператор се използват няколко таблици и в тях има колони с еднакви собствени имена, тогава собственото име е нееднозначно и се уточнява с името на таблицата:

```
име_таблица.име_колона
```

5.1.2. Типове данни

Тип на данните представлява примитивна реализация на понятието домен от релационния модел. Типовете данни според SQL2 са:

- цели числа
- числа с фиксирана точка
- числа с плаваща точка
- символни низове с фиксирана дължина
- символни низове с променлива дължина
- дата, време и временен интервал
- парична величина

В някои СУБД се реализират и допълнителни типове, наречени BLOB:

- поток от байтове (BYTE).

В колона от такъв тип може да се съхраняват графични изображения, изпълним код и всяка друга неструктурирана последователност от байтове.

- дълъг текст (TEXT) – символни низове с променлива дължина, която е максимално 32000 или 65000.

Има различия в типовете данни, реализирани в различните СУБД, и дори в наименованията им, което е съществено препятствие за преносимостта на приложенията. Например в Oracle има тип DATE, в който се съхранява дата и време. В Informix има тип DATE, в който се съхранява само дата, тип DATETIME за дата и време и тип INTERVAL за временен интервал. Различията продължават и по отношение на реализираните операции и вградените функции за тези типове.

5.1.3. Константи

Числови константи се записват като десетично число със или без знак и евентуално с дробна част или като десетично число с плаваща точка (както в повечето езици за програмиране). Например:

21 -123 421.35 1.55E3 (което е $1.55*10^3$)

Низова константа трябва да се ограничава с единични кавички, но в някои СУБД са разрешени и двойни кавички.

```
'низ' или "низ"
```

Константи съдържащи календарна дата, време и временен интервал се записват като символен низ. Форматът на низа е различен за различните СУБД.

Пример в Informix: '3/30/2003', '2003-03-30 12:10:15.20', '20-5'

Пример в Oracle: '30-MAR-2003'

Много често дори в една СУБД се поддържат няколко формата на запис на константите в зависимост от страната.

Съществуват специални символни константи, чиито значения се определят от СУБД. Символна константа може да се използва на всяко място в оператор, където е разрешена константа. Например в Informix:

USER е името на потребителя, който работи с БД

TODAY е текущата дата

CURRENT е текуща дата и време

5.1.4. Изрази

В аритметични изрази могат да се използват операциите: +, -, *, /, ** и скоби. Според стандарта се извършва автоматично преобразуване между различните числови типове, когато това се налага.

В някои СУБД се реализират и символни изрази. Например в Informix:

```
'Mr.' || emp.ename || ', ' || emp.job
addr[1, 20]
```

Изрази над дата и време са включени в SQL2 и се реализират. Например в Informix:

```
education.fdate + 20
```

Ако education.fdate е колона, съдържаща дата на завършване на образование, резултатът е от тип дата, която е 20 дена след датата, съдържаща се в колоната.

```
TODAY – education.fdate
```

Резултатът е цяло число, което е брой дни между двете дати.

Много често се реализират и **вградени функции**, които изпълняват различни преобразования над типове. Такава функция може да се използва на всяко място в оператор, където може да стои име на колона от съответния тип, следователно може

да се използва и в изрази. Списъкът на функциите в различните диалекти на SQL е доста различен. Например в Informix:

`year(fdate), month(fdate), day(fdate), weekday(fdate)`

Връщат цяло число, което е съответно годината, месеца, деня и деня от седмицата на датата в аргумента.

`trunc(sal, 1), round(sal, 1)`

Връщат число, което е значението в аргумента, съответно отрязано или закръглено до един знак след десетичната точка.

`length(addr)`

Връща цяло число, което е дължината на низа в аргумента.

В много случаи значението `NULL` изисква специална обработка. Напр. ако се изчислява израза `sal + 10`, какво да се прави ако `sal` е `NULL`. Отговорите на подобни въпроси дават набор от правила за обработка на значения `NULL` в различни оператори и фрази на езика. Заради необходимостта от такива правила някои теоретици на релационния модел са против това значение. Но независимо от теоретичните спорове значението `NULL` е част от стандартите и се реализира от повечето промишлени СУБД.

При изчисляване на израз, ако някой от операндите е `NULL` и резултатът има значение `NULL`.

5.2. Извличане на данни – оператор SELECT

Ще започнем с операторите за манипулиране на данни (Data Manipulation Language или DML) и най-напред с оператора за извличане на данни от БД. В промишлените СУБД много често се допуска таблицата да съдържа еднакви редове, което означава че тя не е множество, а мултимножество. Ако приемем такова понятие за таблица, то може да кажем, че резултатът от изпълнение на оператор `SELECT` е таблица, чиито редове и колони са изчислени от данните в други таблици. Резултатната таблица е временна, т.е. тя не остава трайно в БД, а само докато е необходима, напр. докато потребителят я разглежда на екрана на терминала или до завършване на приложната програма. Операторът `SELECT` включва следните фрази, от които само първите две са задължителни:

`SELECT` *`целеви-списък`*

Определя колоните на резултата.

`FROM` *`таблици`*

Определя таблиците, от които ще се извличат данни (изходни таблици).

`WHERE` *`условие`*

Определя редовете от изходните таблици, които ще участват в резултата.

`GROUP BY` *`списък-от-колони-при-групиране`*

Задава колоните, по които ще се формират групи от редове.

`HAVING` *`условие`*

Определя групите, които ще участват в резултата.

`ORDER BY` *`списък-от-колони-за-сортиране`*

Определя колоните на резултата, по които се сортират редовете в резултата.

`INTO TEMP` *`име-на-резултатна-таблица`*

Задава име на резултатната таблица.

5.2.1. Прости еднотаблични заявки

Да разгледаме по-подробно синтаксиса на първите две фрази, които задължително присъстват във всеки оператор.

```
SELECT [ALL|DISTINCT|UNIQUE] израз [AS етикет] [, . . .]
FROM [OUTER] име-на-таблица [ псевдоним ] [, . . .]
```

Целевият списък в SELECT е списък от елементи, разделени със запетая. Всеки елемент определя една колона в резултатната таблица, т.е. броят на елементите определя степента на резултата. *Израз* задава начина, по който да се изчисляват значенията в колоната от значенията на колони в изходни таблици и константи. *Етикет* определя името на резултатната колона. Ако не е зададен и *израз* е просто име на колона се използва това име, иначе резултатната колона е безименна. Ключовите думи след SELECT определят дали да се премахват еднаквите редове в резултата (DISTINCT и UNIQUE) или не (ALL). По премълчаване дубликатите не се премахват, защото тази операция е скъпа.

Фразата FROM съдържа списък от имена на таблици, от които ще се изчисляват редовете в резултата според останалите фрази на оператора. В някои случаи се налага определянето на синонимно име на таблица - *псевдоним*, което важи само в съответния оператор. Ключовата дума OUTER указва извършването на външно съединение, което ще разгледаме по-нататък. В този раздел ще предполагаме, че фразата FROM включва една таблица.

♦ Списък на всички служители, включващ име, заплата и длъжност на служителя.

```
SELECT ename, sal, job FROM emp
```

Ако в резултата искаме да включим всички колони на определена таблица, може да използваме символа "*" или "*име_таблица*.*" вместо списък от колони. Това означава всички колони на указаната таблицата или на таблиците от фразата FROM. Това е удобно при интерактивна работа, но използването във вграден SQL е потенциално опасно, тъй като смисълът на "*" с времето може да се измени.

♦ Пълни данни за всички отдели.

```
SELECT * FROM dep;
SELECT dep.* FROM dep
```

♦ Списък на всички служители, включващ име, предшествано от обръщение и годишна заплата на служителя.

```
SELECT 'Mr./Mrs.'||ename AS titename, sal*12 AS ysal FROM emp
```

Фразата WHERE определя редовете, които ще бъдат извлечени от изходните таблици и ще участват при изчисляване на резултата.

```
WHERE условие
```

Условието се конструира от няколко типа прости условия, логическите операции AND, OR, NOT и скоби. Всички редове, за които условието е истина, са избрани и ще участват в резултата. Съществуването на значение NULL влияе при изчисляването на условията, като в същност се прилага тризначна логика. Значението на всеки условен израз може да е истина (true), лъжа (false) или незвестно (unknown). Да разгледаме основните прости условия.

- **Сравнения**

```
израз1  { = | != | <> | < | <= | > | >= } израз2
```

Ако поне един от сравняваните изрази има значение NULL, то резулатът от сравнението е unknown.

- **Проверка за принадлежност на затворен интервал**

израз1 [NOT]BETWEEN *израз2* AND *израз3*

Това условие е еквивалентно на:

[NOT] (*израз1* >= *израз2* AND *израз1* <= *израз3*)

Ако *израз1* е NULL или ако *израз2* и *израз3* са NULL, то условието е unknown.

Ако *израз2* (долната граница) е NULL, то условието е false ако *израз1>израз3* и unknown в противен случай.

Ако *израз3* (горната граница) е NULL, то условието е false ако *израз1<израз2* и unknown в противен случай.

- **Проверка за принадлежност на множество**

израз [NOT]IN (*списък-от-значения*)

Списък-от-значения е списък от константи от един и същи тип, разделени със запетая. Условието е истина ако значението на *израз* е равно на някоя от константите. Ако *израз* е NULL, то условието е unknown.

Проверката за принадлежност на множество също не разширява възможностите на езика, защото условието:

X IN (A, B, C, D)

е еквивалентно на:

(*X* = *A*) OR (*X* = *B*) OR (X = C) OR (X = D)

- **Проверка за съответствие на шаблон**

име-на-колона [NOT]LIKE "*низ*" [ESCAPE *символ*]

Колоната *име-на-колона* трябва да е от символен тип. *Низ* представлява шаблон на значението на колоната. В него освен обикновени символи може да има метасимволи (символи, имащи специално значение в шаблона), които са:

% - разширява се до произволен брой произволни символи;

_ - разширява се до точно един на брой произволен символ;

\ - означава отмяня на специалното значение на символа след него (това е escape символа по премълчанване, но може да се смени чрез фраза ESCAPE).

име-на-колона [NOT]MATCHES "*низ*" [ESCAPE *символ*]

Тази операция е разширение, реализирано в Informix. Метасимволите в низа са по-различни и повече, а именно:

* - еквивалентен на "%" в LIKE;

? - еквивалентен на "_" в LIKE;

[*низ*] - разширява се до точно един на брой символ, от символите в *низ*;

\ - същото както в LIKE.

Условието е истина ако значението на *име-на-колона* съответства на шаблона, т.е. равно е на някой от низовете, генерирани от шаблона. Ако значението на колоната е NULL, то условието е unknown.

- **Проверка за неопределеност**

име-на-колона IS [NOT] NULL

При операция IS NULL условието е истина ако значението на *име-на-колона* е неопределено, а при операцията IS NOT NULL условието е истина ако значението е определено. За разлика от останалите видове прости условия, това условие никога не може да е unknown.

- **Съставни условия**

NOT условие

условие1 AND *условие2*

условие1 OR *условие2*

Както и при простите условия, значението NULL влияе на истинността и на съставните условия.

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN

Наредбата на редовете в резултата е непредсказуема, докато явно не се укаже чрез фразата ORDER BY.

```
ORDER BY име-на-колона [ ASC | DESC ] [, . . . ]
```

Редовете в резултата се сортират по нарастващите (ASC) или намаляващи (DESC) значения на указаните резултатни колони. Това означава, че може да се сортира само по колони или изрази явно или неявно включени в целевия списък. Вместо име на колона може да се използва етикет или пореден номер в целевия списък. Значението NULL се счита за по-малко от всяко друго значение.

Резултатът от оператор SELECT може да се съхрани в БД като временна таблица, която ще съществува до края на сесията. Това може да се укаже чрез фразата INTO TEMP.

```
INTO TEMP име-на-резултатна-таблица
```

Създава се таблица с указаното име и в нея се записва резултата от оператора SELECT. Не трябва да има таблица със същото име в БД. Таблицата се унищожава автоматично при завършване на приложната програма (сесията с БД) или явно с оператор DROP TABLE.

♦ Списък на служителите, включващ името, заплатата и длъжността, за служителите които работят в отдел с номер 1 и получават заплата по-малка от 500. Резултатът да се сортира по азбучен ред на длъжността и по намаляващи значения на заплатата.

```
SELECT ename, sal, job FROM emp
WHERE dno=1 AND sal < 500
ORDER BY job, sal DESC
```

♦ Списък от имената на служителите, които работят в отдел с номер 1 и получават заплата по-голяма от 300 и по-малка от 500.

```
SELECT ename FROM emp
WHERE dno=1 AND sal BETWEEN 300 AND 500
```

♦ Списък от имената на служителите, които работят в отдели с номера 1, 3 или 5.

```
SELECT ename FROM emp
WHERE dno IN (1, 3, 5)
```

♦ Списък от имената на служителите с неизвестен адрес.

```
SELECT ename FROM emp
WHERE addr IS NULL
```

♦ Списък от имената на служителите с адрес в София.

```
SELECT ename FROM emp WHERE addr LIKE 'София%'
```

♦ Списък от номерата на отделите, в които работят служители със заплата по-малка от 500.

```
SELECT DISTINCT dno FROM emp WHERE sal < 500
```

Правила при изпълнение на еднотабличен оператор SELECT.

Ще опишем правилата при изпълнение на еднотабличен оператор SELECT, които по-нататък ще бъдат допълвани и разширявани. Тези правила определят какво означава оператора, а не как СУБД го изпълнява.

1. Взема се таблицата от фразата FROM.
2. Ако има фраза WHERE, то се проверява условието в нея за всеки ред на таблицата и се избират редовете, за които условието е true, останалите редове (за които условието е false или unknown) се игнорират.
3. За всеки от избраните редове се изчисляват последователно елементите в целевия списък и се формира един ред за резултата.
4. Ако има ключова дума DISTINCT, то се изключват повтарящите се редове.
5. Ако има фраза ORDER BY, то се сортират редовете в резултата.
6. Ако има фраза INTO TEMP, то резултатът се съхранява като временна таблица.

5.2.2. Многотаблични заявки – съединение и произведение

На практика повечето заявки извършват търсене и извличане на данни от няколко таблици, т.е. резултатната таблица се формира като се използват редове от няколко таблици. Един от случаите когато това се случва, е ако във фразата FROM има няколко имена на таблици. Тогава ако във фразата WHERE има достатъчно условия, в които се сравняват колони от различните таблици (join условия), това е съединение. Ако няма такова условие, то се прави произведение на таблиците. Какво означава да има достатъчно join условия. Например ако във фразата FROM има две таблици, то в WHERE трябва да има едно join условие:

```
SELECT T1.*, T2.*
FROM T1, T2
WHERE T1.X = T2.Y
```

Колоните, участващи в join условието T1.X = T2.Y:

- Трябва да са сравними, като операцията за сравнение може да е всяка друга.
- Могат да са външен и съответния му първичен ключ, но може и да са произволни сравними колони.
- Могат да участват или не в целевия списък.
- Могат да са съставни колони.

Ако във фразата FROM има три таблици, то в WHERE трябва да има две join условия, които свързват последователно колони от трите таблици. Освен това в WHERE може да има и други условия, които избират редове от всяка една от таблиците.

Ако в таблиците от фразата FROM има колони с еднакви имена, тогава се налага използване на пълни имена на колоните, т.е. името на колоната се уточнява с името на таблицата. Общото правило относно използване на пълни или собствени имена на колони в SQL оператор е, че собствени имена могат да се използват ако това не предизвиква неоднозначност. На някои места името задължително трябва да е собствено, напр., в ORDER BY.

♦ Списък на всички служители, включващ пълните данни за служителя и за отдела, в който работи.

```
SELECT emp.*, dep.*
FROM emp, dep
WHERE emp.dno = dep.dno
```

♦ Списък на всички служители, включващ името, заплатата и името на отдела, в който работи. Резултатът да се сортира по азбучен ред на името на отдела и по намаляващи значения на заплатата.

```
SELECT ename, sal, dname
FROM emp, dep
WHERE emp.dno = dep.dno
ORDER BY dname, sal DESC
```

♦ Списък на началниците, включващ името, заплатата и името на отдела, който ръководи, за началниците със заплата по-голяма от 900.

```
SELECT ename, sal, dname
FROM emp, dep
WHERE eno = mgr AND sal > 900
```

Когато при съединение се използват връзки между редовете на една таблица се казва, че това е съединение на таблица със себе си (selfjoin). В този случай е задължително използването на псевдоними на имената на таблиците, за да се различават различните екземпляри на една таблица. Тогава в пълните имена на колони се използват псевдонимите.

♦ Списък от имената на всички служители, които получават заплата еднаква с (по-голяма от) тази на Иван Иванов от отдел 1.

```
SELECT e.ename
FROM emp e, emp i
WHERE e.sal = i.sal AND i.ename = 'Иван Иванов' AND i.dno = 1
```

♦ Списък на всички служители, включващ името и длъжността, които участват в проект с наименование "ABC".

```
SELECT ename, job
FROM emp e, emp_pro ep, project p
WHERE e.eno = ep.eno AND ep.pno = p.pno AND pname = 'ABC'
```

Използването на псевдоними в този случай не е задължително, но е удобно тъй като пълните имена на колоните са по-кратки.

♦ Списък на всички служители, включващ името, заплатата и името на началника му, за служителите със заплата по-малка от 300.

```
SELECT e.ename, e.sal, m.ename
FROM emp e, emp m, dep
WHERE e.dno = dep.dno AND dep.mgr = m.eno AND e.sal < 300
```

Правила при изпълнение на оператор SELECT над няколко таблици

Ще опишем правилата при изпълнение на този по-сложен вариант на оператора `SELECT`, при който може да има няколко таблици във фразата `FROM`.

1. Формира се произведението на таблиците от фразата `FROM`. Ако във `FROM` има само една таблица, то самата тя ще е произведението.
2. Ако има фраза `WHERE`, то се проверява условието в нея за всеки ред на произведението и се избират редовете, за които условието е `true`, останалите редове (за които условието `false` или `unknown`) се игнорират.
3. За всеки от избраните редове се изчисляват последователно елементите в целевия списък и се формира един ред за резултата.
4. Ако има ключова дума `DISTINCT`, то се изключват повтарящите се редове.
5. Ако има фраза `ORDER BY`, то се сортират редовете в резултата.
6. Ако има фраза `INTO TEMP`, то резултатът се съхранява като временна таблица

Външно съединение

При разгледаната операция съединение, аналог на join от релационната алгебра, се съединява ред от една таблица с ред от друга таблица, за които join

условието е истина. Следователно, ако ред от едната таблица няма нито един съответен ред в другата, то той няма да участва в резултата. В някои случаи това може да доведе до неочаквани резултати. Напр., в задачата:

♦ Списък на всички служители, включващ името на служителя и името на отдела, в който работи.

```
SELECT ename, dname
FROM emp, dep
WHERE emp.dno = dep.dno
```

Резултатът няма да включва служителите, за които колоната `dno` има значение `NULL`, а може би ние очакваме да видим и тях в резултата.

Външното съединение е разширение на обикновеното, наричано още вътрешно. Правилата при изпълнението му за две таблици са следните:

1. Създава се вътрешното съединение на двете таблици.
2. Всеки несвързан ред от първата таблица се включва в резултата, като на всички колони от втората таблица се присвояват `NULL` значения.
3. Всеки несвързан ред от втората таблица се включва в резултата, като на всички колони от първата таблица се присвояват `NULL` значения.

Всъщност съединението, създадено по тези правила се нарича **пълно външно** съединение, има още **ляво външно** съединение (стъпки 1 и 2) и **дясно външно** съединение (стъпки 1 и 3).

Начините за обозначаване на външните съединения са доста различни в различните СУБД, тъй като те се реализират преди да са включени в стандартите. В Informix лявото външно съединение се записва:

```
SELECT *
FROM T1, OUTER T2
WHERE T1.A = T2.B
```

Ключовата дума `OUTER` се поставя пред името на таблицата, която се допълва с празен ред. Първата (лявата) таблица се нарича главната, втората - подчинена в съединението.

Други начини за записване на лявото външно съединение.

Oracle

```
SELECT *
FROM T1, T2
WHERE T1.A = T2.B(+)
```

SQL Server

```
SELECT *
FROM T1, T2
WHERE T1.A *= T2.B
```

В Informix и Oracle няма пълно външно съединение. В SQL Server пълното външно съединение се записва:

```
SELECT *
FROM T1, T2
WHERE T1.A *=* T2.B
```

В SQL2 стандарта е избран метод, който не съответства на никоя от популярните СУБД. Разширена е фразата `FROM` с обозначаването на вида на съединението и условието за съединение, което се пренася от фразата `WHERE`. За вътрешното съединение е запазен и записа според SQL1 стандарта.

```
SELECT *
FROM T1 INNER JOIN T2 ON T1.A = T2.B
```

```
SELECT *
FROM T1, T2
WHERE T1.A = T2.B
```

```
SELECT *
FROM T1 LEFT OUTER JOIN T2 ON T1.A = T2.B
```

```
SELECT *
FROM T1 RIGHT OUTER JOIN T2 ON T1.A = T2.B
```

```
SELECT *
FROM T1 FULL OUTER JOIN T2 ON T1.A = T2.B
```

Външното съединение може да се разшири за три и повече таблици, но тогава резултатът зависи от реда, в който се изпълняват съединенията. Затова в разширената фраза `FROM` се добавят скоби, т.е. резултатът от едно съединение се използва като операнд на друго.

♦ Списък на **всички** служители, включващ името, длъжността и името на отдела, в който работи. Решение със синтаксиса на Informix.

```
SELECT ename, job, dname
FROM emp, OUTER dep
WHERE emp.dno = dep.dno
```

♦ Списък на **всички** служители, включващ името, длъжността, името на проекта и времето, които имат заплата по-голяма от 500.
Резултатът от вътрешното съединение на `emp_pro` и `project` се съединява външно с `emp`.

Първи вариант – със синтаксиса според стандарта.

```
SELECT ename, job, pname, ptime
FROM emp LEFT JOIN
      (emp_pro INNER JOIN project ON emp_pro.pno=project.pno)
      ON emp.eno=emp_pro.eno
WHERE sal > 500
```

Втори вариант – със синтаксиса на Informix.

```
SELECT ename, job, pname, ptime
FROM emp, OUTER (emp_pro, project)
WHERE emp.eno=emp_pro.eno AND emp_pro.pno=project.pno AND sal > 500
```

В Informix външните съединения, в които участват повече от две таблици са следните видове:

```
SELECT *
FROM T1, OUTER (T2, T3)
WHERE T1.A = T2.A AND T2.B = T3.B
```

Резултатът от вътрешното съединение на `T2` и `T3` се съединява външно с `T1` (вложено вътрешно съединение).

```
SELECT *
FROM T1, OUTER (T2, OUTER T3)
WHERE T1.A = T2.A AND T2.B = T3.B
```

Резултатът от външното съединение на `T2` и `T3` се съединява външно с `T1` (вложено външно съединение).

```
SELECT *
FROM T1, OUTER T2, OUTER T3
WHERE T1.A = T2.A AND T1.B = T3.B
```

Всяка от двете таблици `T2` и `T3` се съединява външно с `T1`, като условието е между главната таблица `T1` и подчинените `T2` и `T3`.

5.2.3. Вложени заявки - вложен оператор SELECT

Друг случай, при който резултатната таблица се формира като се използват данни от няколко таблици, е когато се използа вложен оператор `SELECT`. Вложен оператор `SELECT` или подзаявка се нарича оператор `SELECT`, съдържащ се в друг такъв оператор. Местата, на които това се реализира, са условията във фразите `WHERE` или `HAVING`. Условията с вложен `SELECT` са следните.

израз [NOT]IN (*оператор-select*)

Това е същото условие, проверяващо за принадлежност на множество и описано преди, с тази разлика, че списъкът от значения се изчислява посредством вложен оператор `SELECT`.

израз { = | != | <> | < | <= | > | >= } [ANY|ALL](*оператор-select*)

Условието θALL е истина, ако сравнението (означено с θ) е истина за всяко значение, върнато от вложения оператор. Ако вложеният оператор не върне нито едно значение, то условието е истина. Ако сравнението не е false за нито едно върнато значение, но има и значения NULL, то условието е unknown. Условието !=ALL е еквивалентно на NOT IN.

Условието θANY е истина, ако сравнението е истина за поне едно значение, върнато от вложения оператор. Ако вложеният оператор не върне нито едно значение, то условието е лъжа. Ако сравнението не е true за нито едно върнато значение, но има и значения NULL, то условието е unknown. Условието =ANY е еквивалентно на IN. В Informix има и ключова дума SOME, която е синоним на ANY.

Ако сме сигурни, че вложеният оператор ще върне едно значение, то може да не указваме ключовите думи ALL или ANY.

[NOT]EXISTS (*оператор-select*)

Условието EXISTS е истина, ако вложеният оператор върне поне един ред, иначе условието е лъжа. Не е възможно значението на условието да е unknown.

В Informix вложеният оператор SELECT не може да включва фразите ORDER BY, INTO TEMP и UNION.

♦ Списък на имената и заплатите на началниците, които получават заплата по-голяма от 900.

```
SELECT ename, sal
FROM emp
WHERE eno IN (SELECT mgr FROM dep) AND sal > 900
```

♦ Списък от имената на всички служители, които получават заплата еднаква с (по-голяма от) тази на Иванов от отдел 1 (предполагаме, че има само един служител Иванов в отдел 1).

```
SELECT ename
FROM emp
WHERE sal = (SELECT sal FROM emp
             WHERE ename = 'Иван Иванов' AND dno = 1)
```

♦ Списък на всички служители, включващ името и длъжността, които участват в проект с наименование "ABC".

```
SELECT ename, job
FROM emp
WHERE eno IN (SELECT eno FROM emp_pro
              WHERE pno IN (SELECT pno FROM project
                            WHERE pname = 'ABC'))
```

♦ Списък от имената и заплатите на всички служители, които получават заплата по-голяма от максималната заплата в отдел 3.

```
SELECT ename, sal
FROM emp
WHERE sal >ALL (SELECT sal FROM emp WHERE dno = 3)
```

В това решение може да възникне проблем, ако заплатата на някой от служителите в отдел 3 е NULL, т.е. ако вложеният SELECT върне поне едно значение NULL.

Във всеки оператор SELECT имената на колоните са неявно уточнени с името на таблица от съответната фраза FROM. Затова ако външният и вложеният SELECT са върху една и съща таблица, не се налага да използваме псевдоними (освен ако не са корелирани).

5.2.4. Агрегатни функции и групиране

Понякога се изисква извличане на обобщена информация от БД. Например, искаме да получим отговор на въпроси от типа: Каква е средната заплата и броят на служителите в отдел 1? В SQL решението за въпроси от този тип се изразява чрез оператор `SELECT` с агрегатни функции и фрази `GROUP BY` и `HAVING`.

Агрегатни функции

Агрегатната функция приема като аргумент съвкупност от значения на една колона, съдържащи се във всички редове, избрани от условието в `WHERE` (и принадлежащи на една група, ако има групиране) и връща едно значение, което е обобщение на значенията по определен начин. Аргументът може да е колона на изходна таблица или израз (производна колона). Агрегатните функции са:

`SUM([DISTINCT|UNIQUE]` *име-на-колона*`)`	`SUM(`*израз*`)`
`AVG([DISTINCT|UNIQUE]` *име-на-колона*`)`	`AVG(`*израз*`)`
`MIN([DISTINCT|UNIQUE]` *име-на-колона*`)`	`MIN(`*израз*`)`
`MAX([DISTINCT|UNIQUE]` *име-на-колона*`)`	`MAX(`*израз*`)`
`COUNT([DISTINCT|UNIQUE]` *име-на-колона*`)`	`COUNT(*)`

Функцията `SUM` изчислява сумата на всички значения на колоната, която трябва да е от числов тип и връща числов тип.

Функцията `AVG` изчислява средното аритметично на всички значения на колоната, която трябва да е от числов тип и връща числов тип.

С функциите `MIN` и `MAX` се намира най-малкото или най-голямото значение в колоната, която може да е от числов, символен или тип дата/време. Сравненията се извършват по правилата на съответния тип, напр. дати се сравняват хронологически.

Функцията `COUNT` изчислява броя на всички значения на колоната, която може да е от произволен тип и връща цяло число. Вариантът `COUNT(*)` брои редовете.

С ключовата дума `DISTINCT` или `UNIQUE` се указва да се премахнат еднаквите значения на колоната преди да се изчисли агрегатната функция. При функциите `MIN` и `MAX` това е безмислено, но някои СУБД го разрешават. В стандарта SQL1 `DISTINCT` може да се указва само пред име на проста колона, но не и пред израз. В стандарта SQL2 много от ограниченията са снети, но в различните СУБД правилата може да се различават. Синтаксисът, който сме дали по-горе, е от Informix.

Значението `NULL` се отразява и при изчисление на агрегатните функции. Правилата за обработка на `NULL` по стандарта са:

- Значението `NULL` се игнорира при изчисляване на функциите. Изключение прави `COUNT(*)`, тъй като тя брои редове, независимо от `NULL`.
- Ако всички значения в колоната са `NULL`, то `SUM`, `AVG`, `MIN` и `MAX` връщат `NULL`, а `COUNT` връща 0.
- Ако в колоната няма нито едно значение, т.е броят на редовете е 0, то `SUM`, `AVG`, `MIN` и `MAX` и връщат `NULL`, а `COUNT` връща 0.

В някои случаи значението `NULL` може да доведе до проблеми, например:

`SUM(A) + SUM(B)` и `SUM(A+B)`

могат да върнат различни резултати, ако в някоя от колоните в някой ред има `NULL`.

Агрегатни функции могат да стоят непосредствено само във фразите `SELECT` и `HAVING`. Ако агрегатна функция е във фразата `SELECT` и няма групиране, то резултатът е таблица от един ред и целевият списък не може да включва имена на колони. Агрегатни функции не могат да се влагат, тъй като при определяне на аргументите им роля играят фразите `FROM` и `WHERE`.

♦ Средната заплата и брой служители в предприятието (за отдел 1).

```
SELECT AVG(sal) avgsal, COUNT(*) empcount
FROM emp

SELECT AVG(sal) avgsal1, COUNT(*) empcount1
FROM emp WHERE dno = 1
```

♦ Списък от имената и заплатите на всички служители, които получават заплата по-голяма от средната заплата в предприятието.

```
SELECT ename, sal
FROM emp
WHERE sal > (SELECT AVG(sal) FROM emp)
```

За да поставим агрегатна функция в условието на фразата `WHERE`, тя трябва да е във вложен `SELECT`. Тогава той винаги връща едно значение и може да изпуснем `ANY` или `ALL` при сравнението.

♦ Списък от имената и заплатите на всички служители, които получават заплата по-голяма от максималната заплата в отдел 3.

```
SELECT ename, sal
FROM emp
WHERE sal > (SELECT MAX(sal) FROM emp WHERE dno = 3)
```

♦ Списък от имената и заплатите на всички служители, които получават заплата по-голяма от средната заплата за отдела си.

```
SELECT ename, sal
FROM emp x
WHERE sal > (SELECT AVG(sal) FROM emp WHERE dno = x.dno)
```

Това е решение с **корелиран (свързан) вложен `SELECT`**, тъй като в него участва колоната `x.dno` от външния `SELECT`. При проверката на условието във външния `SELECT`, за всеки ред се променя значението на тази колона, следователно вложеният `SELECT` ще връща различни значения за всеки проверяван ред и трябва да се изчислява многократно.

♦ Списък от номерата и имената на отделите, в които работят повече от 5 служители със заплата по-голяма 500.

```
SELECT dno, dname
FROM dep
WHERE 5 < (SELECT COUNT(*) FROM emp
           WHERE dno = dep.dno AND sal >500)
```

При това решение с корелиран вложен `SELECT` в резултата не може да участва броя на служителите за всеки отдел. По-нататък ще разгледаме друго решение, което позволява в резултата да се включи и броя на служителите за всеки отдел.

Групиране - фраза `GROUP BY`

Групата е съвкупност от редове, които са избрани от фразата `WHERE` и имат еднакви значения за определени колони. Групиране в оператор `SELECT` се указва чрез фразата `GROUP BY`, задаваща имената на колоните, по значенията на които ще се формират групите.

`GROUP BY` *име-на-колона* `[, ...]`

Указаните *име-на-колона* са имена на колони от таблиците във фразата `FROM`. В някои системи е позволено да се групира и по значенията на изрази от целевия списък, напр. в Informix, тогава в `GROUP BY` се задава пореден номер на елемента от целевия списък. Групиране може да се прави по повече от една колона (до 8 колони в Informix), но е на едно ниво, т.е. не може групата да се разбие на подгрупи.

Между агрегатните функции в целевия списък и фразата GROUP BY има връзка. Ако има групиране, то за всяка група в резултата се включва един ред. От това следват ограничения за възможните значения на елемент от целевия списък. Разрешените значения са следните:

- константа
- агрегатна функция, която ще връща едно значение за всяка група
- име на колона от фразата GROUP BY
- израз, включващ горните елементи.

Това означава, че с един оператор SELECT не могат да се получат детайлни и сумарни данни едновременно. Това е така защото резултатът трябва да е таблица, а не комбинация от редове от различен тип.

Ако има фраза GROUP BY, но в целевия списък няма агрегатни функции, то операторът може да се изрази без GROUP BY, а чрез DISTINCT.

Когато в колона, по която се групира се съдържат значения NULL, това отново изисква допълнително правило. Правилото за фразата GROUP BY в стандарта и в Informix е: ако два реда имат значение NULL в една колона за групиране и еднакви значения в другите колони за групиране, то те попадат в една група. Това правило не е много логично, тъй като то на практика приема, че две значения NULL са еднакви, но е по-удобно.

♦ Списък на отделите, включващ номер на отдела, средна заплата и брой служители в отдела.

```
SELECT dno, AVG(sal) avgsal, COUNT(*) empcount
FROM emp
GROUP BY dno
```

♦ Списък на отделите, включващ номер на отдела и брой служители в отдела със заплата по-малка от 500.

```
SELECT dno, COUNT(*) empcount
FROM emp
WHERE sal < 500
GROUP BY dno
```

♦ Списък на отделите, включващ номер, име на отдела и брой служители в отдела със заплата по-малка от 500.

```
SELECT dep.dno, dep.dname, COUNT(*) empcount
FROM emp, dep
WHERE dep.dno = emp.dno AND sal < 500
GROUP BY dep.dno, dep.dname
```

При анализ на коректността на оператор SELECT с групиране се игнорира информацията за първични и външни ключове. Затова в горното решение трябва да включим и двете колони в GROUP BY, ако искаме да участват в резултата. Това решение може да върне един ред по-малко в сравнение с предходната задача, ако колоната emp.dno има значение NULL в някои редове.

Условия за групи - фраза HAVING

Фразата HAVING избира някои от групите, формирани от GROUP BY, които да участват при изчисляване на резултата, т.е. има същата роля за групите както WHERE за редовете.

HAVING *условие*

За всяка група се проверява *условие* и ако то е истина, групата се избира и от нея се формира един ред за резултата. Условието в тази фраза се конструира както и условието в WHERE, но има някои особености. В простите условия могат да участват имена на колони, по които се групира, агрегатни функции и константи, т.е. в условието се проверяват свойства на групата като цяло. Ако в HAVING има

агрегатна функция, то тя се изчислява за всяка група при проверка на условието за нея. В HAVING може да има и условие с вложен оператор SELECT, който може да е корелиран по колони на групиране или не.

Ако в HAVING няма агрегатна функция, то условието може да бъде пренесено във фразата WHERE.

Правилата за обработка на значения NULL в условието на HAVING са същите както при условието във фразата WHERE.

Фразата HAVING почти винаги се използва в съчетание с GROUP BY, но синтаксисът на SELECT не изисква това. Ако HAVING е без GROUP BY, то всички редове, избрани от WHERE, образуват една група.

♦ Списък от номер, име на отдела и брой служители в отдела със заплата по-голяма от 500, който включва отделите със повече от 5 служители със заплата по-голяма 500.

```
SELECT d.dno, d.dname, COUNT(*) cnthighsal
FROM emp e, dep d
WHERE d.dno = e.dno AND e.sal > 500
GROUP BY d.dno, d.dname
HAVING COUNT(*) > 5
ORDER BY cnthighsal DESC
```

♦ Списък от номер, име на отдела и средна заплата за отдела, който включва отделите със средна заплата по-голяма от средната заплата за предприятието.

```
SELECT d.dno, d.dname, AVG(sal) avgsal
FROM emp e, dep d
WHERE d.dno = e.dno
GROUP BY d.dno, d.dname
HAVING AVG(sal) > (SELECT AVG(sal) FROM emp)
ORDER BY 3 DESC
```

Правила при изпълнение на многотабличен оператор SELECT с групиране и агрегатни функции

Ще опишем правилата при изпълнение на този по-сложен вариант на оператора SELECT, при който може да има няколко таблици във фразата FROM, вложена заявка, групиране и агрегатни функции.

1. Формира се произведението на таблиците от фразата FROM. Ако във FROM има само една таблица, то самата тя ще е произведението.
2. Ако има фраза WHERE, то се проверява условието в нея за всеки ред на произведението и се избират редовете, за които условието е true, останалите се игнорират. Ако има вложен SELECT, той се изпълнява за всеки проверяван ред.
3. Ако има фраза GROUP BY, то от избраните редове се формират групи, така че в една група да са редове с еднакви значения във всички колони за групиране.
4. Ако има фраза HAVING, то за всяка група се проверява условието в нея и се избират групите, за които е условието е true. Ако има SELECT, вложен в HAVING той се изпълнява за всяка проверявана група.
5. За всеки избран ред или за всяка избрана група се изчисляват последователно елементите в целевия списък и се формира един ред за резултата. Ако има фраза GROUP BY и агрегатни функции, то като аргумент се използват значенията на колоната от всички редове в групата. Ако има агрегатни функции без GROUP BY, като агрумент се използват значенията на колоната във всички редове.
6. Ако има ключова дума DISTINCT, то се изключват повтарящите се редове.
7. Ако има фраза ORDER BY, то се сортират редовете в резултата.
8. Ако има фраза INTO TEMP, то резултатът се съхранява като временна таблица.

5.2.5. Обединение - UNION

Операцията обединение, позната ни от релационната алгебра, позволява да се обединят редовете на две таблици в една. Тази операция се поддържа в SQL и чрез нея могат да се обединят резултатите от два оператора SELECT в една таблица.

SELECT-*оператор* UNION [ALL] SELECT-*оператор* [...]

За да могат да се обединят резултатите от операторите SELECT е необходимо резултатите от тях да са сравними:

- Целевите им списъци трябва да включват еднакъв брой елементи.
- Съответните елементи да са от един тип.
- Не се изискват еднакви имена на съответните елементи. Прието е резултатната колона да взема името от първия оператор SELECT.
- Никой от резултатите на оператор SELECT не може да е сортиран. Сортиране може да се прилага към резултата от обединението, т.е. ако е необходимо фразата ORDER BY се указва след последния оператор SELECT.
- Същото се отнася и до фразата INTO TEMP. Ако искаме резултатът да се съхрани като временна таблица, задаваме това след последния оператор SELECT.

Какви са правилата за обработка на повтарящите се редове? При операцията UNION се премахват повтарящите се редове в резултата след обединението. Ако е указана думата ALL, повтарящите се редове не се премахват след обединението. Ако искаме да се изключат дубликатите преди обединението, е необходимо в съответния оператор SELECT да зададем ключова дума DISTINCT.

♦ Списък от имената и адресите на всички студенти и преподаватели, сортиран по азбучен ред на името. Това е пример от БД-студенти.

```
SELECT ime, adres, 's' AS who
FROM student
UNION ALL
SELECT ime, adres, 't'
FROM prepod
ORDER BY 1
INTO TEMP names
```

Операциите сечение и разлика от релационната алгебра не се поддържат директно в SQL, но може да се изразят чрез оператор SELECT.

Ако T1 и T2 са сравними таблици с първични ключове T1.X и T2.X, които също са сравними по тип, то операцията на релационната алгебра T1 INTERSECT T2 може да се изрази чрез:

```
SELECT *
FROM T1
WHERE X IN (SELECT X FROM T2)
```

или

```
SELECT *
FROM T1
WHERE EXISTS (SELECT * FROM T2 WHERE T1.X = T2.X)
```

Операцията на релационната алгебра T1 MINUS T2 може да се изрази чрез:

```
SELECT *
FROM T1
WHERE X NOT IN (SELECT X FROM T2)
```

или

```
SELECT *
FROM T1
WHERE NOT EXISTS (SELECT * FROM T2 WHERE T1.X = T2.X)
```

5.3. Обновяване на данни

5.3.1. Оператор INSERT

Добавяне на нови редове към таблица се извършва чрез оператор INSERT, който има два варианта. Първият вариант добавя винаги един ред към таблица.

```
INSERT INTO име-на-таблица [( списък-от-колони )]
VALUES ( списък-от-константи )
```

Фразата INTO задава името на таблицата, към която ще се добави новия ред (целевата таблица). Фразата VALUES съдържа значенията на колоните в новия ред. Значенията може да са константи, включително символни константи (USER, TODAY, CURRENT) и NULL. Списъкът от имена на колони определя коя константа на коя колона е значение. Следователно, трябва да има позиционно съответствие между двата списъка. Ако името на някоя колона на целевата таблица отсъства от списъка, тя ще има значение по премълчаване в новия ред, най-често NULL. За удобство се разрешава *списък-от-колони* да не се задава, тогава се подразбира списъка от всички колони на целевата таблица (в същия ред както при SELECT *).

Редовете във всяка таблица са ненаредени (както е и според теорията на релационния модел) и затова в оператора не се указва къде да се добави новия ред по отношение на съществуващите редове.

♦ Да се добавят данни за нов служител.

```
INSERT INTO emp (eno, ename, job, sal, addr, dno)
VALUES (301, 'Иван Петров', 'ac.', 300, 'София, ул.Латинка 6', NULL)
```

или

```
INSERT INTO emp (eno, ename, job, sal, addr)
VALUES (301, 'Иван Петров', 'ac.', 300, 'София, ул.Латинка 6')
```

Вторият вариант на оператор INSERT добавя няколко реда към целевата таблица, които се изчисляват от данни в други таблици на БД чрез вложен оператор SELECT.

```
INSERT INTO име-на-таблица [( списък-от-колони )]
оператор-select
```

Вложеният *оператор-select* може да е произволно сложен, но се налагат някои ограничения:

- Целевият списък на вложения *оператор-select* и *списък-от-колони* трябва да са съвместими по брой и по тип на данните за съответните елементи.
- Вложеният оператор не трябва да съдържа фразата ORDER BY, тъй като няма смисъл да се сортират редовете преди да се добавят към таблица, където редовете са ненаредени.
- Вложеният оператор не трябва да съдържа фразата INTO TEMP.
- Вложеният оператор не трябва да е с обединение UNION.
- Вложеният оператор трябва да е върху таблици различни от целевата.

В новите стандарти последните две ограничения са отменени, но е възможно някои СУБД все още да ги изискват.

♦ Към таблица losalemp (eno, name, job, dreamsal) да се добавят данни за служители със заплати по-малки от 300.

```
INSERT INTO losalemp (eno, name, job, dreamsal)
      SELECT eno, ename, job, sal*2
      FROM emp
      WHERE sal < 300
```

5.3.2. Оператор UPDATE

Оператор UPDATE изменя значенията на една или повече колони в избрани редове на една таблица.

```
UPDATE име-на-таблица
SET    { име-на-колона  =  израз [, ... ]
       | (списък-от-колони ) =  ( списък-от-изрази )
       }
[ WHERE условие ]
```

Първата фразата задава името на целевата таблица. Фразата WHERE избира редовете, които ще бъдат изменени. Условието се конструира по същия начин, както във фразата WHERE на оператор SELECT. Това означава, че може да има условие с вложен SELECT, но върху таблици, различни от целевата. Това е полезно когато изборът на редовете за изменение се основава на данни от други таблици. Вложеният SELECT може да е независим, но може и да е корелиран с целевата таблица. Ако няма фраза WHERE, то измененията се извършват във всички редове на целевата таблица.

Фразата SET определя колоните, които ще бъдат изменяни и начина, по който това ще стане, т.е представлява списък от присвоявания. Значението на *израз* се изчислява като се използват значенията в текущия ред преди изменението. Типът на израза трябва да съответства на типа на съответната му колона. В някои СУБД, напр. Informix, е възможно *израз* да е вложен SELECT, връщащ едно значение. Там е реализирано и допълнението към синтаксиса, където *списък-от-изрази* може да е вложен SELECT, връщащ един ред с необходимия брой значения. И в двата случая вложеният SELECT трябва да е върху таблици, различни от целевата, но може да е колелиран с нея.

♦ Да се измени длъжността и заплатата на служител с номер 120.

```
UPDATE emp SET sal = 350, job = 'ac.'
WHERE eno = 120
```

или записано с другия синтаксис, реализиран допълнително в Informix:

```
UPDATE emp SET (sal, job) = (350, 'ac.')
WHERE eno = 120
```

♦ Да се увеличи заплатата на всички служители с 10%.

```
UPDATE emp SET sal = sal*1.10
```

♦ Да се увеличи с 10% заплатата на всички служители от отдел 1 и те да се преместят в отдел 10.

```
UPDATE emp SET sal = sal * 1.10, dno = 10
WHERE dno = 1
```

♦ Да се увеличи с 10лв. заплатата на всички служители, работещи в отдел с началник служител с номер 100.

```
UPDATE emp SET sal = sal + 10
WHERE dno IN (SELECT dno FROM dep WHERE mgr = 100)
```

♦ Да се увеличи с 20лв. заплатата на всички служители, които участват в повече от 3 проекта.

```
UPDATE emp SET sal = sal + 20
WHERE eno IN (SELECT eno FROM emp_pro
                  GROUP BY eno
                  HAVING COUNT(*) > 3)
```

или друг вариант с корелиран вложен SELECT

```
UPDATE emp SET sal = sal + 20
WHERE 3 < (SELECT COUNT(*) FROM emp_pro
               WHERE emp_pro.eno = emp.eno)
```

♦ Да се увеличи с 50лв. заплатата на всички служители, които са работели по проекти повече от 100 дена общо.

```
UPDATE emp SET sal = sal + 50
WHERE eno IN (SELECT eno FROM emp_pro
                  GROUP BY eno
                  HAVING SUM(ptime) > 100)
```

♦ Да се измени длъжността и заплатата на служителите, за които има редове в `losalemp`, като новите данни се вземат от там.

```
UPDATE emp
SET (sal, job) = ((SELECT dreamsal, job FROM losalemp
                       WHERE  losalemp.eno = emp.eno))
WHERE eno IN (SELECT eno FROM losalemp)
```

5.3.3. Оператор DELETE

Операторът `DELETE` изтрива избрани редове от една таблица.

```
DELETE FROM име-на-таблица
[ WHERE условие ]
```

Фразата `FROM` задава името на целевата таблица. Фразата `WHERE` избира редовете, които ще бъдат изтрити. Условието се конструира по същите правила, както условието във фразата `WHERE` на оператор `UPDATE` (може да има условие с вложен `SELECT`, но върху таблици, различни от целевата; вложеният `SELECT` може да е независим или да е корелиран с целевата таблица). Ако няма фраза `WHERE`, то се изтриват всички редове от целевата таблица.

Операторите `UPDATE` и `DELETE` имат и друг вариант, който е реализиран във вградения SQL и действа винаги на един ред (позиционен).

♦ Да се изтрият всички данни от таблица `losalemp`.

```
DELETE FROM losalemp
```

♦ Да се изтрият всички данни за проект с номер 20.

```
DELETE FROM emp_pro WHERE pno = 20;
DELETE FROM project WHERE pno = 20
```

♦ Да се изтрият всички данни за проект с наименование ABC.

```
DELETE FROM emp_pro
WHERE pno IN (SELECT pno FROM project WHERE pname = 'ABC');
DELETE FROM project WHERE pname = 'ABC'
```

♦ Да се изтрият всички данни за отдел с номер 10.

```
UPDATE emp SET dno = NULL
WHERE dno = 10;
DELETE FROM dep WHERE dno = 10;
```

♦ Да се изтрият всички данни за служител с номер 120.

```
DELETE FROM emp_pro WHERE eno = 120;
UPDATE emp SET mgr = NULL
WHERE mgr = 120;
DELETE FROM emp WHERE eno = 120
```

♦ Да се изтрият всички данни за служител с име Иван Иванов.

```
DELETE FROM emp_pro
WHERE eno IN (SELECT eno FROM emp WHERE ename = 'Иван Иванов');
UPDATE emp SET mgr = NULL
WHERE mgr IN (SELECT eno FROM emp WHERE ename = 'Иван Иванов');
DELETE FROM emp WHERE ename = 'Иван Иванов'
```

♦ Да се изтрият всички данни за отдел с номер 10, ако в него няма служители.

```
DELETE FROM dep
WHERE dno = 10 AND NOT EXISTS (SELECT * FROM emp WHERE dno = 10)
```

5.4. Описание на данните

Ще разгледаме операторите на езика SQL за описание и създаване на БД. Наборът от оператори, които определят структурата на БД, се нарича Data Definition Language (DDL). Макар, че DDL и DML са две отделни части на езика SQL, в повечето СУБД такова разделяне е само на абстрактно ниво. Те са напълно равноправни и могат да се използват смесено, както при интерактивна работа, така и във вграден SQL. Това е едно от преимуществата на релационния модел в сравнение с по-ранните модели на данните. Структурата на БД е динамична, т.е. могат да се създават, унищожават и изменят обекти едновременно с достъпа на потребителите до БД. Ядрото на DDL се състои от три глагола, с които започва почти всеки оператор:

- `CREATE` – Описва и създава обект в БД.
- `ALTER` – Изменя описанието на съществуващ обект в БД.
- `DROP` – Унищожава обект в БД.

След всяка от тези думи в оператора следва ключова дума, определяща вида на обекта, за който се отнася оператора – таблица, синоним, виртуална таблица (view), индекс и др.

5.4.1. Базови таблици

5.4.1.1. Оператор CREATE TABLE

Чрез този оператор се определя и създава най-важният обект в БД - таблица. Така създадена таблица ще наричаме **базова таблица**, защото е постоянна и реална и за да я различаваме от таблиците, създадени чрез оператор `SELECT` с фраза `INTO TEMP` или чрез оператор `CREATE VIEW`. Операторът работи с абстрактните понятия на релационния модел, но не може да не се засегне и физическото представяне на таблиците. Има няколко фрази, които определят параметри на физическото преставяне. Те са различни в различните СУБД (тук е даден синтаксисът в Informix) и ще ги разгледаме в раздела за вътрешно представяне на БД. Потребителят, изпълнил оператора `CREATE`, става собственик на таблицата.

```
CREATE TABLE име-на-таблица
( име-на-колона тип-данни [DEFAULT значение] [NOT NULL]
                    [ ограничение-за-цялостност-за-колоната ]
  [, ...]
[, ограничение-за-цялостност-за-таблицата ] [, ...]
)
[ WITH NO LOG ]
[ IN име-на-област ] [ EXTENT SIZE първи ] [ NEXT SIZE следващ ]
[ LOCK MODE { PAGE | ROW } ]
```

Името на таблицата трябва да е уникално в БД. Според стандарта се изисква уникалност на пълното име на таблица, т.е. "*име-на-собственик.име-на-таблица*", но в някои СУБД се изисква уникалност за собственото име на таблица. След името на таблицата следва описанието на колоните и ограничения за цялостност, което е заградено в кръгли скоби. Накрая са параметрите, управляващи физическото представяне.

Описанието на всяка колона включва следната информация:

- Име на колоната. Името трябва да е уникално в таблицата.
- Тип на данните
- Значение по премълчаване - `DEFAULT` *значение*

Това е значението, което се използва когато в оператор `INSERT` не е зададено значение за колоната в добавяните редове. Значението може да е константа, `NULL`

или символна константа – USER, TODAY, CURRENT. По премълчаване се използва NULL.

- Ограничение, забраняващо неопределено значение – NOT NULL.
- Ограничения за цялостност, отнасящи се само до описваната колона.

Всъщност това е "съкратена" форма, която може да се използва когато ограничението за цялостност се отнася само до една колона на таблицата. Тогава то може да се добави в края на описанието на колоната.

След описание на колоните на таблицата могат да се определят и ограничения за цялостност, които се отнасят до всички описани вече колони. Това е "пълната" форма при задаване на ограничения за цялостност.

Типове данни в Informix

INTEGER, INT	цяло число, съхранявано в двоично представяне в 4 байта
SMALLINT	цяло число, съхранявано в двоично представяне в 2 байта
SERIAL[(*s*)]	цяло число, представяно като INT, чиито значения се генерират автоматично от СУБД като последователни цели числа, започвайки от *s* или 1 по премълчаване
DECIMAL[(*p*[,*q*])]	десетично число с *p* цифри (16 по премълчаване), от които *q* са след десетичната точка. Ако *q* не е зададено, се счита, че точката може да плава. ($1 \le q \le p$, $1 \le p \le 32$)
MONEY[(*p* [,*q*])]	десетично число с *p* цифри (16 по премълчаване), от които *q* са след десетичната точка (2 по премълчаване)
FLOAT,DOUBLE PRECISION	число, съхранявано в представяне с плаваща точка в 8 байта
REAL, SMALLFLOAT	число, съхранявано в представяне с плаваща точка в 4 байта
CHAR(*n*)	символен низ с фиксирана дължина *n* байта ($1 \le n \le 32767$)
VARCHAR(*n*)	символен низ с променлива дължина, най-много *n* байта ($1 \le n \le 255$)
DATE	календарна дата, съхранявана като цяло число, което е брой дни след 31/12/1899
DATETIME *точност*	календарна дата и време с определена точност
INTERVAL *точност*	времеви интервал, измерван с определена точност и мерни единици

При типа DATETIME, *точност* е от вида: *first* TO *last*, където *first* и *last* са от списъка:

```
YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRACTION(n)
```

При типа INTERVAL, *точност* е от вида: *first*[(*n*)] TO *last*[(*m*)], където *first* и *last* са и двете от един от двата списъка:

```
YEAR, MONTH
DAY, HOUR, MINUTE, SECOND, FRACTION
```

а *n* и *m* са максималният брой десетични цифри, използвани при съхраняването.

Примери за типове дата и време:

```
DATETIME YEAR TO MINUTE
DATETIME HOUR TO MINUTE
INTERVAL YEAR TO YEAR
INTERVAL DAY(3) TO HOUR
```

Ограничения за цялостност

Типовете ограничения, които могат да бъдат зададени в описанието на таблица са:

- **Първичен ключ - `PRIMARY KEY`**

Една таблица може да има най-много един първичен ключ. Всяка СУБД налага ограничения за максималната дължина на първичния ключ (напр., в Informix е до 16 колони с максимална обща дължина 255 байта).

- **Ограничение за уникалност – `UNIQUE` или `DISTINCT`**

Това е ограничение чрез което може да се поддържа понятието възможен ключ от релационния модел. Една колона не може да е определена едновременно като първичен ключ и като `UNIQUE`. Ограниченията за максималната дължина са същите, както за първичен ключ.

- **Външен ключ - `FOREIGN KEY`**

Определя колоните, които са външен ключ в описваната таблица и връзката, която той създава с друга таблица *име-на-таблица-pk* и съответния му първичен ключ *име-на-колона-pk* (може да е и `UNIQUE`). Когато се изпълнява операторът СУБД сравнява определяния външен ключ със съответния му първичен ключ по брой колони и тип на данните. Следователно таблицата, към която сочи външния ключ, трябва вече да съществува освен ако не е създаваната в изпълнявания оператор.

- **Условие за проверка - `CHECK`**

Това условие налага ограничения на съдържанието на всеки ред от таблицата, разглеждайки го независимо от останалите. Следователно условието е логически израз, в който се сравняват изрази върху колони на описваната таблица и константи. Условието се проверява при всеки опит да се добави или измени ред и ако не е истина операцията не се изпълнява.

В синтаксиса има два варианта за определяне на ограничения за цялостност:

ограничение-за-цялостност-за-колоната

Може да се използва, когато ограничението се отнася до една колона и се добавя в края на описанието на колоната.

```
{ PRIMARY KEY | DISTINCT | UNIQUE
| REFERENCES име-на-таблица-pk ( име-на-колона-pk )
| CHECK ( условие ) } [ CONSTRAINT име-на-ограничение ]
```

ограничение-за-цялостност-за-таблицата

Ограничението е зададено след описанието на колоните.

```
{ { PRIMARY KEY | DISTINCT | UNIQUE }(име-на-колона [, ...])
| FOREIGN KEY(име-на-колона-fk [,...])
             REFERENCES име-на-таблица-pk (име-на-колона-pk [,...])
| CHECK ( условие ) }
[ CONSTRAINT име-на-ограничение ]
```

На всяко ограничение за цялостност може да се даде име чрез фразата `CONSTRAINT`, което е удобно когато впоследствие се наложи да се отменят ограничения. Името на ограничението е идентификатор. Ако не е зададено име на ограничението, то СУБД генерира по свои правила уникално име.

5.4.1.2. Оператор `DROP TABLE`

Чрез оператор `DROP TABLE` се унищожава таблица. Това означава, че се унищожават данните в нея, ограниченията за цялостност върху нея и други обекти в БД, свързани с таблицата – синоними, views, индекси, ограничения за достъп.

```
DROP TABLE име-на-таблица
```

Този оператор може да промени и описанието на други таблици. Напр., ако с унищожаваната таблица има свързан външен ключ в друга таблица, то се унищожава и ограничението за съответния външен ключ.

5.4.1.3. Оператор ALTER TABLE

Чрез оператор ALTER TABLE се изменя описанието на таблица. Някои изменения могат да предизвикат и изменения на съдържанието й, ако не е празна или са невъзможни, ако не е празна. Най-общият вид на оператора е следния:

```
ALTER TABLE име-на-таблица
{ ADD фраза |
  MODIFY фраза |
  DROP фраза |
  ADD CONSTRAINT фраза |
  DROP CONSTRAINT фраза |
  MODIFY NEXT SIZE фраза |
} [ ,... ]
```

Всяка от фразите определя различен вид изменение на описанието на таблицата с име *име-на-таблица*. В един оператор може да има няколко различни фрази.

Добавяне на нова колона

```
ADD (име-на-нова-колона тип-данни [DEFAULT значение] [NOT NULL]
                  [ ограничение-за-цялостност-за-колоната ][,...])
                  [BEFORE име-на-стара-колона ]
```

Добавят се една или повече нови колони към съществуваща таблица. Описанието на всяка нова колона включва същите елементи както в оператора CREATE TABLE. Ако таблицата не е празна, във всичките съществуващи редове новата колона ще има значение по премълчаване от DEFAULT или NULL. Това определя някои ограничения в случай, че таблицата не е празна. Ограничение UNIQUE може, ако таблицата има най-много един ред. Ограничение PRIMARY KEY може само за празна таблица. За непразна таблица NOT NULL може заедно с DEFAULT.

Изменение на описанието на колона

```
MODIFY (име-на-стара-колона тип-данни [DEFAULT значение] [NOT NULL]
                  [ ограничение-за-цялостност-за-колоната ][,...])
```

Чрез тази фраза може да се измени описанието на съществуваща колона, а именно:

- типа на данните;
- да се добави или измени значение по премълчаване;
- да се забрани или разреши неопределено значение;
- да се добави или отмени ограничение за цялостност.

При смяна на типа на данните е позволено почти всичко, стига съшествуващите данни в таблицата да могат да бъдат преобразувани към новия тип и след преобразуването да не бъде нарушено ограничение UNIQUE. Следователно, ако таблицата не е празна, в някои случаи изменението може да не се извърши.

Когато се променя описанието на колона, всички характеристики от старото определение, които искаме да запазим, трябва отново да се зададат в оператора.

Унищожаване на колона

```
DROP (име-на-стара-колона [, ...])
```

Унищожават се колоните на таблицата и всички свързани с тях ограничения за цялостност и други обекти в БД. Следователно това изменение може да промени

и описанието на други таблици. Ако унищожаваната колона е първичен ключ или UNIQUE, към който сочи външен ключ от друга таблица, то се унищожава и ограничението за съответния външен ключ.

Добавяне на ограничение за цялостност

```
ADD CONSTRAINT ( ограничение-за-цялостност-за-таблица [, ...])
```

Добавят се нови ограничения за цялостност. Използва се пълният формат на синтаксиса, както в оператора CREATE TABLE. Ако таблицата не е празна, данните в нея трябва да не нарушават добавяните ограничения за цялостност.

Унищожаване на ограничение за цялостност

```
DROP CONSTRAINT ( име-на-ограничение [, ...])
```

Унищожават се ограниченията за цялостност, идентифицирани чрез имената си. Следователно, както при унищожаване на колона, това изменение може да промени и описанието на други таблици, напр. ако се унищожи ограничение за първичен ключ, то се унищожават и ограниченията за съответните му външни ключове, ако съществуват.

♦ Да се опишат таблиците на БД-служители.

```
CREATE TABLE dep (
   dno SMALLINT PRIMARY KEY,
   dname CHAR(30) NOT NULL,
   budget MONEY DEFAULT 0 NOT NULL,
   mgr INTEGER,
   CHECK (budget >= 0)
);
CREATE TABLE emp (
   eno INTEGER,
   ename CHAR(50) NOT NULL,
   sal MONEY DEFAULT 0 NOT NULL,
   job CHAR(20),
   addr CHAR(50),
   dno SMALLINT,
   PRIMARY KEY(eno),
   FOREIGN KEY (dno) REFERENCES dep(dno),
   CHECK (sal >= 0)
);
CREATE TABLE project (
   pno SMALLINT PRIMARY KEY,
   pname CHAR(30) NOT NULL,
   description CHAR(100),
   finished DATE
);
CREATE TABLE emp_pro (
   eno INTEGER,
   pno SMALLINT,
   ptime INTERVAL DAY(3) TO DAY,
   PRIMARY KEY(eno, pno),
   FOREIGN KEY (eno) REFERENCES emp(eno),
   FOREIGN KEY (pno) REFERENCES project(pno)
);
ALTER TABLE dep
   ADD CONSTAINT ( FOREIGN KEY (mgr) REFERENCES emp(eno));
```

5.4.2. Синоними

Много СУБД изискват използване на пълни имена на таблици, когато потребителят работи с чужди таблици както е и по стандартите. Това води до

неудобството, че имената стават дълги и трябва да се знае и помни собственика. За решаване на тези проблеми в някои СУБД се въвежда понятието синоним. Синоним е алтернативно име на таблица (базова или view), което се създава с оператора `CREATE SYNONYM` и съществува докато явно не се унищожи с оператор `DROP SYNONYM`. Синоним може да се използва в SQL оператори като име на таблица.

`CREATE [PUBLIC | PRIVATE] SYNONYM` *`синоним`* `FOR` *`име-на-таблица`*

Името на синонима трябва да е уникално. Пространството от имена на базови таблици, виртуални таблици, временни таблици и синоними е едно. Потребителят, изпълнил `CREATE SYNONYM` става собственик на синонима. В някои системи има два вида синоними - `PUBLIC` и `PRIVATE`. Синоним `PUBLIC` може да се използва от всички потребители, имащи достъп до БД. Синоним `PRIVATE` може да се използва само от собственика си, другите потребители трябва да използват пълно име на синонима. Синоним се унищожава с оператора `DROP SYNONYM`:

`DROP SYNONYM` *`синоним`*

5.4.3. Създаване и унищожаване на база данни

Преди да започне създаването на таблици, синоними и други обекти е необходимо да се създаде самата база данни, която първоначално ще е празна. В стандарта SQL1 не е определен начинът, по който това се извършва, затова в СУБД има различни подходи. Няколко примера:

В Oracle БД се създава в процеса на инсталиране на системата, тъй като се предполага, че на всеки компютър има една база данни и създаването е най-често еднократно действие. БД може да се създаде и впоследствие, но процедурата е доста сложна.

В Ingres и Postgres има специални сервизни програми: `createdb` - за създаване на нова БД, `destroydb` - за унищожаване на БД. Това не са оператори на езика SQL.

В Informix, SQL Server, Sybase има оператори на езика SQL, част от DDL, чрез които се създава нова БД, унищожава се БД или се извършват други действия с БД като цяло.

В същност в СУБД се реализират два основни подхода за организация на базите данни в една компютърна система.

Еднобазова архитектура

СУБД поддържа работата с една обща БД, която може да съдържа таблици за различни информационни системи. Преимуществото на този подход е, че БД не трябва да се идентифицира и достъпът до нея е по-прост. Недостатък тук е опасността БД да стане прекалено голяма и трудно управляема. Освен това много често се налага за едно приложение и на един компютър да се поддържат две БД – една за промишлено използване и друга за развитие и тестване на нови приложения.

Многобазова архитектура

СУБД поддържа работа с няколко БД, всяка предназначена за независимо приложение. Например БД за заплати, счетоводна БД, БД за обработка на поръчки. Всяка БД, съхранявана в компютърната система, има уникално име. Преимуществото при тази организация се състои в разделянето на задачата за управление на данните на по-малки и леки задачи. Всяка БД може да има свой АБД. Проблемът е, че трябва да има средства за свързване с БД.

От казаното до тук става ясно, че съществува разнообразие по отношение на това как са организирани базите данни, как се създават и как се осъществява достъп към тях. Тази част от езика SQL е една от най-нестандартизираните. По нататък ще

разгледаме подхода, реализиран в Informix и DDL операторите за БД.

БД се създава с оператора CREATE DATABASE.

```
CREATE DATABASE име-на-БД [IN име-на-област ]
[WITH {[BUFFERED] LOG | LOG MODE ANSI } ]
```

Името на БД *име-на-БД* е идентификатор и трябва да е уникално на сървера. Фразата IN определя областта, в която се създава БД. Област е понятие, което не е част от релационния модел и е свързано с физическото съхранение. По премълчаване се използва така наречената основна област. Фразата WITH определя модела на обработка на транзакции за БД. Потребителят, изпълнил този оператор, става неин АБД.

```
DROP DATABASE име-на-БД
```

Унищожава БД заедно с всички обекти в нея и освобождава дисковото пространство, заемано от БД.

```
DATABASE име-на-БД [EXCLUSIVE]
```

Създава връзка между приложната програма и указаната БД (отваря БД). БД става текуща и всички SQL оператори се изпълняват в нея. Ключовата дума EXCLUSIVE изисква монополен режим на отваряне, който ще разгледаме в раздела за управление на конкурентния достъп.

```
CLOSE DATABASE
```

Прекратява връзката на приложната програма с БД (затваря БД).

5.4.4. Виртуални таблици (view)

View е таблица, която за разлика от базовата таблица е:

- Производна – Редовете в нея се изчисляват от данните в базови таблици.
- Виртуална – Данните не се съхраняват отделно от данните на базовите таблици, от които е производна.

По друг начин погледнато view е съхранен SELECT оператор, който има име и чрез който се виждат данните в изходните базови таблици. От тук и името му view (изглед), тъй като е прозорец, през който се гледат данните в БД. Предимствата, които дава използването му, са няколко:

- Дава възможност различните потребители да виждат едни и същи данни по различен начин. Така за всеки потребител може да се изгради най-подходящата за него структура на БД.
- Използва се при осигуряване на сигурност на данните за ограничаване достъпа на потребителите само до производни данни.
- Използва се и за осигуряване на цялостност при view с режим на контрол.

Създаване и унищожаване

Нова виртуална таблица се създава с оператор CREATE VIEW. Ядрото на този оператор е вложен SELECT оператор, който определя начина на изчисляване на редовете и колоните на виртуалната таблица. Вложеният SELECT оператор може да е произволно сложен, но без фразите ORDER BY, INTO TEMP и обединение - UNION. Потребителят, изпълнил оператора CREATE VIEW, става собственик на новата виртуална таблица.

```
CREATE VIEW име-на-view [( списък-от-имена-на-колони )]
AS оператор-select
[WITH CHECK OPTION]
```

Името *име-на-view* трябва да е уникално в БД. Правилата са същите както при имена на базови таблици. След него в скоби се задават имената на колоните му. Ако този списък е пропуснат, тогава имената се наследяват от имената на колоните

от SELECT оператора. Следователно, ако в целевия списък на SELECT оператора има израз или колони с еднакви собствени имена, е задължително да се дадат имена на колоните на view. Между *списък-от-имена-на-колони* и целевия списък на оператора SELECT има позиционно съответствие. Това означава, че ако искаме да дадем име на една виртуална колона, трябва да дадем на всички.

Виртуална таблица може да се създава от базови и други виртуални таблици.

Фразата WITH CHECK OPTION определя създаване на виртуална таблица с режим на контрол, което е свързано с използването й. Този режим не позволява да се изпълни оператор INSERT или UPDATE чрез виртуалната таблица, който нарушава условието във фразата WHERE на CREATE VIEW.

Това, което се случва при изпълнение на оператора CREATE VIEW, е че се съхранява определението на виртуалната таблица. Не се изпълнява вложения оператор SELECT и следователно не се материализира виртуалната таблица.

Виртуална таблица се унищожава чрез оператор DROP VIEW.

DROP VIEW *име-на-view* [CASCADE | RESTRICT]

Унищожава се определението на виртуалната таблица *име-на-view*. Ако на базата на *име-на-view* са създадени други виртуални таблици, тогава при CASCADE се унищожават и всички други виртуални таблици, определени чрез него. В противен случай (RESTRICT) операторът не се изпълнява, ако има други виртуални таблици, определени чрез унищожаваната.

Използване на виртуални таблици

Виртуална таблица може да се използва само в операторите за манипулиране на данни – SELECT, INSERT, UPDATE и DELETE. Когато СУБД срещне името на виртуална таблица в оператор, тя преобразува потребителския оператор в еквивалентен оператор върху изходните базови таблици и го изпълнява. По такъв начин системата поддържа илюзията, че виртуалната таблица съществува, като я материализира когато има обръщение към нея.

Когато потребител се опита да изпълни обновяващ оператор (INSERT, UPDATE и DELETE) върху виртуална таблица, системата трябва да преобразува тази операция в операция по обновяване на изходните базови таблици. Това обаче не винаги е възможно и затова се налагат някои ограничения на възможните начини за използване на виртуална таблица. Виртуална таблица е напълно обновяема, т.е. разрешени са всички обновяващи оператори, ако операторът SELECT в определението й отговаря на следните изисквания:

- Не съдържа ключовата дума DISTINCT.
- Във фразата FROM има само едно име на таблица.
- Всеки елемент от целевия списък е име на колона, т.е. няма изрази или агрегатни функции.
- Условието в WHERE не съдържа подзаявка.
- Няма групиране – фрази GROUP BY и HAVING.

Някои виртуални таблици са частично обновяеми, т.е. някои обновяващи оператори са възможни, а други не. Правилата за обновяване чрез виртуални таблици в някои СУБД може да не са толкова строги.

♦ Да се създаде виртуална таблица, съдържаща пълните данни за служителите от отдел с номер 1.

```
CREATE VIEW vemp1
AS SELECT * FROM emp WHERE dno = 1
WITH CHECK OPTION;
```

Таблица като vemp1 представлява хоризонтално подмножество на една базова

таблица и може да се използва, за да се ограничи достъпа на потребители само до част от редовете на базова таблица. Тя е напълно обновяема. Това, че е създадена с режим на контрол, означава невъзможност да се изпълни оператор:

```
UPDATE vemp1 SET dno = 2 WHERE eno = 100;
```

Чрез този оператор би се изменил ред във виртуалната таблица, който след това няма да е видим чрез нея. Това ще е допустимо, ако е създадена без `WITH CHECK OPTION`.

♦ Да се създаде виртуална таблица, съдържаща за всеки служител номер, име, длъжност и адрес на служителя.

```
CREATE VIEW vemp
AS SELECT eno, ename, job, addr FROM emp;
```

Таблица като `vemp` представлява вертикално подмножество на една базова таблица и може да се използва, за да се ограничи достъпа на потребители само до част от колоните на базова таблица. Такъв вид виртуална таблица може да е частично обновяема. Операторът `INSERT` ще е разрешен само ако не се нарушава ограничение `NOT NULL` за колона, не участваща във виртуалната таблица. В този случай `INSERT` е разрешен (виж описанието на таблица `emp`). Операторите `DELETE` и `UPDATE` са разрешени.

♦ Да се създаде виртуална таблица, съдържаща за всеки служител номер, име, месечна и годишна заплата на служителя.

```
CREATE VIEW vpay (nom, name, msal, ysal)
AS SELECT eno, ename, sal, sal*12 FROM emp;
```

Тук има виртуална колона - `ysal`, която е израз. Затова `vpay` е частично обновяема. Операторът `DELETE` е разрешен, но операторите `INSERT`, `UPDATE` не са. Възможно е в някои СУБД да е разрешен `UPDATE`, ако не изменя колоната `ysal`.

♦ Да се създаде виртуална таблица, съдържаща номер, име, месечна заплата за служителя и номер на началника му, за служителите със заплата по-малка от 500.

```
CREATE VIEW vlopay (nom, name, msal, mgr)
AS SELECT eno, ename, sal, mgr FROM emp, dep
WHERE emp.dno = dep.dno AND sal < 500;
```

Това е напълно необновяема виртуална таблица, тъй като се изчислява от две базови таблици. Какво значи, например "да се измени значението на колоната `mgr` за служител с номер 100"? Дали това означава, че служителят преминава към друг отдел или се сменя началникът на отдела му. Невъзможно е такъв оператор `UPDATE` да бъде преобразуван еднозначно в оператори върху базовите таблици `emp` и `dep`.

♦ Да се създаде виртуална таблица, съдържаща за всеки отдел номер на отдела, минимална, максимална и средна заплата за отдела.

```
CREATE VIEW vdep (dno, losal, hisal, avgsal)
AS SELECT dno, MIN(sal), MAX(sal), AVG(sal)
FROM emp GROUP BY dno;
```

Това също е напълно необновяема виртуална таблица, тъй като съдържа групиране и агрегатни функции. Причината е очевидна. Какво значи, например "да се измени средната заплата за отдел 1"? Невъзможно е такъв оператор `UPDATE` да бъде преобразуван в оператор върху базовата таблица `emp`. Освен това тук съществуват и ограничения за операторите `SELECT` върху `vdep`. Тъй като е недопустимо влагането на агрегатни функции е недопустим и следният оператор:

```
SELECT MIN(avgsal) FROM vemp;
```
Преобразуваният оператор ще изглежда така:
```
SELECT MIN(AVG(sal)) FROM emp GROUP BY dno;
```

Предимства и недостатъци на виртуалните таблици

Използването на виртуални таблици е полезно в най-различни ситуации и по различни причини. Основните преимущества, които дава използването им са:

Сигурност на данните. Да се ограничи достъпът на потребител до част от БД чрез виртуални таблици, до които той има достъп, а няма такъв за базовите.

Простота на структурата. За всеки потребител може да се създаде собствена "структура" на БД.

Цялостност на данните. Когато обновяването на данни се извършва чрез обновяема виртуална таблица с режим на контрол.

Независимост на данните. При реструктуриране на БД, за потребителя "структурата" може да остане непроменена.

Но наред с преимуществата използването на виртуални таблици има два основни недостатъка:

Производителност. Преобразуването на оператора и материализирането на виртуалната таблица при всяка заявка към нея изисква време, което зависи от сложността на определението на виртуалната таблица.

Ограничения при обновяване. Сложните виртуални таблици са необновяеми, което ограничава тяхното използване.

5.5. Вътрешно представяне на БД

Как обектите на релационния модел – базови таблици, виртуални таблици, синоними и други се съхраняват в паметта? Различните СУБД реализират различно физическо представяне на БД, което определя сложността на системата и за какви цели може да се използва. СУБД, които работят на персонални компютри, са предназначени за малки БД и реализират по-проста физическа организация. Например, в MS Access цялата БД се съхранява като един файл на операционната система. Всъщност в по-новите версии организацията малко е усъвършенствана и БД може да се разположи в няколко файла. В системи, работещи на сървери, като Oracle, Infomix, Ingres и др. физическата организация е доста по-сложна. Това позволява на АБД да управлява разполагането на данните, разпределяйки ги на различни дискове и така да повиши производителността при големи бази данни.

Въпреки съществуващите различия, СУБД имат една обща характеристика: каквото и да е физическото представяне на таблиците, и данните и метаданните се съхраняват като таблици. Освен таблиците създавани с оператор `CREATE TABLE` съществуват и така наречените **системни таблици**. Те се създават автоматично когато се създава БД и съдържат метаданни, т.е. данни за обектите в БД. Обикновено тази група от таблици се нарича с общото име **системен каталог**. Какво е предимството при такъв подход? Едно важно предимство е, че информация за БД може да се получава чрез оператор `SELECT`. Но обновяването на данните в системните таблици обикновенно се извършва само от СУБД при изпълнение на различните DDL оператори.

5.5.1. Вътрешно представяне в Informix

Ще разгледаме физическото представяне на БД в Infomix, понятията свързани с него и елементите на SQL операторите, където се задават параметри на вътрешното представяне.

Физически понятия

Това са понятия, използвани при управлението на дисковата памет и свързани с организацията на дисковата памет.

- **Файл (chunk)**

Това е най-голямата физическа единица дискова памет, която се разпределя от АБД за съхраняване на данни от БД. Може да е файл на операционната система (cooked disk space) или цял неформатиран дял на диск (row disk space). При втория вариант, целият дял на диска е отделен за БД, на него не се изгражда файлова система и не съдържа други данни, което осигурява по-висока надеждност и по-добра производителност.

- **Страница (page)**

Това е най-малката единица дискова памет, разпределяна за таблица. Всички страници на сървера са с фиксирана дължина - 2KB или 4KB в зависимост от платформата, на която работи Infomix - операционната система и хардуера.

- **Екстент (extent)**

Екстент е непрекъсната последователност от страници в един файл, които са разпределени за една таблица. Разпределянето на екстент за таблица се извършва един път при създаването ѝ и динамично при нарастване на таблицата. Данните на всяка таблица се съхраняват в един или повече екстента. С всяка таблица са свързани два параметра – размер на началния екстент и размер на следващите екстенти (8 страници по премълчаване и за двете).

Логически понятия

Едно от основните задължения на АБД е да управлява разполагането на данните на БД по дисковете на сървера, да следи за наличната свободна памет и да се грижи за ефективната работа на системата. Понятието, което дава тази възможност на АБД е **област** (**Dbspace**). То осигурява връзка между понятията на релационния модел и физическите единици.

Всяка област има име и включва един или повече файла. На всеки сървер има една основна област (root dbspase), която се създава след инсталиране, при инициализиране на СУБД. Тогава се разпределя и първият файл за нея. След това при необходимост могат да се разпределят допълнителни файлове за основната област.

Създаването на други области, добавянето на файл към област, унищожаването на области или на някои техни файлове е задължение на АБД и се извършва чрез специална сервизна програма (не с оператори на SQL).

Понятие със същото предназначение се реализира и в други СУБД, напр. в Oracle се нарича tablespace, но там за създаване, изменение и други операции над област се използват SQL оператори.

В кои DDL оператори се задават параметри на физическото представяне и са свързани с управлението на разполагането на данните?

```
CREATE DATABASE име-на-бд IN име-на-област
```

Фразата `IN` определя областта, в която ще бъдат създадени системните таблици на създаваната БД и областта по премълчаване, където ще се създават впоследствие базовите таблици. Ако тази фраза не е зададена, БД се създава в основната област.

```
CREATE TABLE име-на-таблица (  ...  )
[IN име-на-област ]
[EXTENT SIZE първи ] [NEXT SIZE следващ ]
```

Фразата `IN` определя областта, в която се разполага създаваната таблица. Ако не е указана тук, се подразбира областта от оператора `CREATE DATABASE`. Това

означава, че една таблица се свързва с една област, но различните екстенти на таблицата могат да са в различни файлове на областта.

Има и друг вариант при свързване на таблица с област, когато редовете на една таблица могат да бъдат разпределени в няколко области по определени правила. Това е така нареченото фрагментиране на таблица и е полезно при много големи таблици, тъй като дава възможност за паралелен достъп до данните в таблицата. Вместо фразата `IN` е зададена фраза `FRAGMENT`, например:

```
CREATE TABLE emp (  ...  )
FRAGMENT BY EXPRESION
      eno<1000 IN dbspace1,
      eno>=1000 AND eno <2000 IN dbspace2,
      eno>=2000 IN dbspace3
```

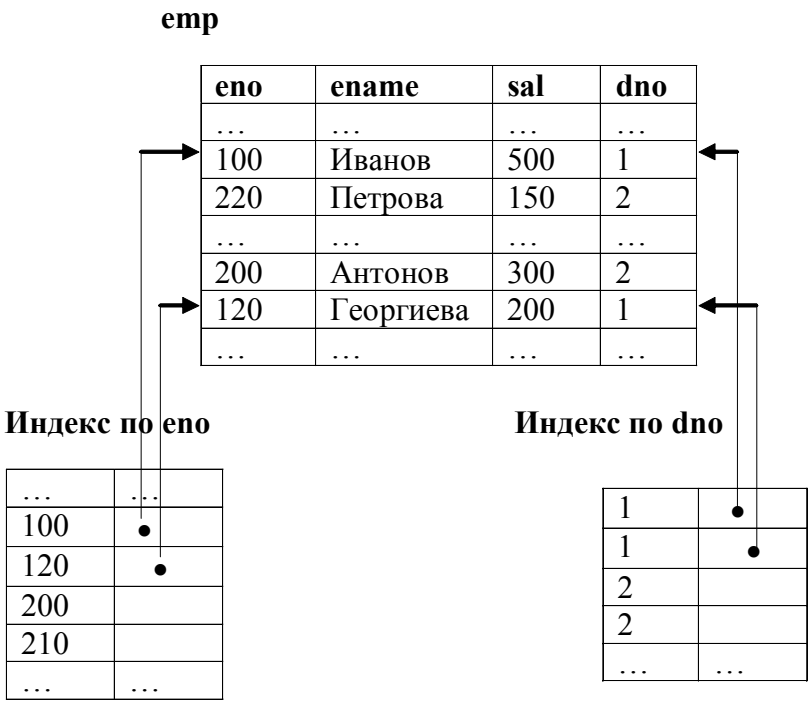
Фразата `EXTENT SIZE` определя размера на началния екстент, който се разпределя по време на изпълнение на оператора `CREATE TABLE`. Фразата `NEXT SIZE` определя размера на следващите екстенти, които се разпределят по необходимост. Размерите се задават в брой КВ, но трябва да са кратно на размера на страница.

Размерът на следващите екстенти може да се измени след като таблицата е създадена чрез оператора:

`ALTER TABLE` *име-на-таблица* `MODIFY NEXT SIZE` *следващ*

5.5.2. Индекси

Индекс е обект в БД, осигуряващ бърз достъп до редовете на базова таблица по значенията на определена колона(и) и е свързан с физическото представяне на таблица. На следната рисунка е изобразена таблицата `emp` и два създадени за нея индекса. Единият осигурява достъп по значенията на първичния ключ `eno`, а другият по значенията на колоната `dno`. Всеки ред в индекса съдържа значение на колоната, по която е създаден и адрес на ред в базовата таблица, съдържащ това значение. Колоната (колоните) на базовата таблица, по която е създаден индекса ще наричаме ключ на индекса. Редовете в индекса са разположени по нарастващи или намаляващи значения на ключа.



Наличието или отсъствието на индекс е незабележимо за потребителя и приложната програма. В SQL операторите за манипулиране на данни не се указва използването на индекси. При изпълнение на оператор СУБД (оптимизаторът на заявките) решава дали да използва или не индекс когато той съществува. Напр., при изпълнение на `SELECT` оператора:

```
SELECT  eno, ename, sal FROM  emp
WHERE dno = 1 and sal < 300
```

Ако не съществува индекс по `dno`, то последователно ще се обходят всички редове на таблицата, за да се извлече необходимата информация. Ако съществува индекс по `dno`, то системата може първо да търси в индекса значението на номер на отдел 1, и след това чрез адресите в намерените там записи да прочете директно редовете в таблицата `emp`.

Търсенето в индекса е по-бързо, тъй като записите в него са сортирани по значението на ключа и са по-къси от редовете в базовата таблица. Много СУБД, включително и Informix, при физическото представяне на индекса използват структурата B+ tree, което още повече ускорява търсенето в индекса. Преходът от запис в индекса към ред на базовата таблица също е бърз, тъй като се осъществява по адрес на ред в някаква форма (напр. адрес на страница и отместване в страницата). Ускоряването на изпълнението на `SELECT` оператори при наличието на индекси има своята цена. Всеки индекс заема допълнителна дискова памет. Обновяването на данните в базовата таблица изисква изпълнение на съответна операция и върху индекса, тъй като съдържанието на индекса трябва да съответства на съдържанието на таблицата.

За всяка таблица динамично могат да се създават и унищожават индекси чрез оператори на SQL без това да влияе на операторите за манипулиране на данни, като изключим времето за изпълнение. Това е пример за физическа независимост на данните.

```
CREATE [UNIQUE|DISTINCT] [CLUSTER] INDEX име-на-индекс
ON име-на-таблица ( име-на-колона [ASC|DESC] [, …] )
[ FILLFACTOR процент ] [ IN име-на-област ]
```

Чрез оператора `CREATE INDEX` се създава нов индекс с име *име-на-индекс* за базовата таблица *име-на-таблица*. След името на таблицата се задава ключа на индекса, т.е. списък от колони на базовата таблица и указание за наредбата на значенията в индекса (нарастващи или намаляващи). Всяка СУБД налага ограничения за максималната дължина на ключа на индекса (напр. в Informix е до 16 колони с максимална обща дължина 255 байта).

Ако е зададена ключовата дума `UNIQUE` или `DISTINCT`, то се създава уникален индекс, т.е. индекс, забраняващ еднакви значения на ключа на индекса. В този случай данните в таблицата трябва да не нарушават това изискване и в момента на създаване на индекса, в противен случай операторът завършва с грешка.

Ако е зададена ключовата дума `CLUSTER`, това води до физическо пренареждане на редовете на таблицата в последователността на индексните записи. Това ще доведе до допълнително повишаване на ефективността при търсене и сортиране по ключа на индекса. За всяка таблица може да има само един такъв индекс. Наредбата в базовата таблица обаче не се поддържа при последващи обновявания в таблицата.

Повечето СУБД автоматично създават индекси, когато се задават някои ограничения за цялостност в оператора `CREATE TABLE`:

- уникален индекс за всеки първичен ключ и колони обявени за `UNIQUE`;
- неуникален индекс за всеки външен ключ.

Индексите могат динамично да се унищожават чрез оператор `DROP INDEX` без това да повлияе на базовата таблица.

```
DROP INDEX име-на-индекс
```

`CLUSTER` свойството на индекс може да се отмени и друг индекс да се избере за `CLUSTER` индекс или да се пренареди базовата таблица в последователността на `CLUSTER` индекс, чрез оператор `ALTER INDEX`.

```
ALTER INDEX име-на-индекс TO [NOT] CLUSTER
```

Препоръки при използване на индекси

Системата не ограничава броя на индексите, но все пак те трябва да се създават само когато ще е полезно, а именно:

- По колони, по които често се търси и сортира.
- По колони, които се използват при съединение.
- Да не се създава индекс по колони, които имат само няколко различни значения.
- За определен набор от колони на таблица и в определена наредба (`ASC` или `DESC`) може да има само един индекс.
- Неуникалните индекси е по-добре да се създадат след въвеждане на данните в таблицата, тъй като:
  - При обновяване на таблицата ще се поддържат по-малко индекси.
  - Новосъзданеният индекс осигурява по-ефективен достъп.
- Ако обновяванията се правят периодично (напр. един път месечно) по-ефективно ще е да се следва процедурата:
  - `DROP INDEX` за неуникалните индекси.
  - Обновяване на базовите таблици.
  - Отново `CREATE INDEX` за унищожените в началото индекси.

5.5.3. Системни таблици

Системните таблици са специални таблици, които се създават и поддържат автоматично от СУБД. В тях се съхраняват не потребителски данни, а метаданни – данни за всички обекти в БД. При изпълнение на SQL операторите СУБД непрекъснато осъществява достъп до системните таблици. Макар, че те са за вътрешна употреба, потребителят може да получи достъп до тях чрез `SELECT` оператор и така да получи информация за структурата на БД. В различните СУБД се реализират системни таблици, които имат различна структура, наименования и дори броят им е различен.

Всяка системна таблица съхранява информация за определен вид обекти в БД, като основните видове, поддържани от всички системи са:

- **Таблици**. За всяка таблица се съхранява: име, собственик, тип на таблицата, дата на създаване, брой колони и др.
- **Колони**. Всяка колона се описва с: име, към коя таблица принадлежи, тип на данните, размер, разрешено ли е неопределено значение и др.
- **Виртуални таблици**. Всяка виртуална таблица може да е описана като таблица, но освен това за нея трябва да се съхранява `SELECT` оператора, чрез който се изчислява тя.
- **Потребители**. Всеки потребител регистриран в СУБД се описва с име, парола за достъп и др.
- **Права**. Всяко право дадено на потребител се описва с: кой е дал привилегията, на кого е дадена, самата привилегия и обекта, за когото се отнася. (Разглеждат се в раздела сигурност на данните.)

Основни системни таблици в Informix

Броят на системните таблици в Informix е над 20 и зависи от версията. Ще разгледаме структурата на част от тях, които са основни и присъстват във всички версии.

systables

tabname	Char(18)	Име на таблицата
owner	Char(8)	Име на потребителя, който е собственик
partnum	integer	Идентификатор на tblspase
tabid	serial	Системен идентификатор на таблицата
rowsize	smallint	Размер на ред
ncols	smallint	Брой колони
nindexes	smallint	Брой индекси
nrows	integer	Брой редове
created	Date	Дата на създаване
version	integer	Номер, който се променя при alter table
tabtype	Char(1)	Тип: T-таблица, V-view, P-synonym, S-synonym
locklevel	Char(1)	Режим на заключване
npused	integer	Брой използвани страници
fextsize	integer	Размер на началния екстент
nextsize	integer	Размер на следващ екстент
flags	smallint	Резервирано
site	Char(18)	Резервирано
dbname	Char(18)	Резервирано

syscolumns

colname	char(18)	Име на колоната
tabid	integer	Идентификатор на таблицата
colno	smallint	Пореден номер на колоната в таблицата
coltype	smallint	Код за тип на данните
collength	smallint	Размер на колоната в брой байтове
clmin	smallint	Второто min значение в колоната
colmax	smallint	Второто max значение в колоната

sysindexes

idxname	char(18)	Име на индекса
owner	char(8)	Собственик на индекса
tabid	integer	Идентификатор на таблицата
idxtype	char(1)	Тип: U-уникален, D-неуникален
clustered	char(1)	C-Clustered, или не
part1	smallint	Номер на колона (colno), първа в индекса
…		
part16	smallint	Номер на колона (colno), 16-та в индекса
levels	smallint	Брой нива в B+ tree
leaves	integer	Брой листа в B+ tree
nunique	integer	Брой уникални значения
clust	integer	Степен на клъстеризация

sysviews

tabid	integer	Идентификатор на таблицата
seqno	smallint	Пореден номер на ред в SELECT оператора
viewtext	Char(64)	Текст на SELECT оператора

sysusers

username	char(8)	Име на потребител
usertype	char(1)	Привилегии: D=DBA, R=Resource, C=Connect
priority	smallint	Резервирано
passwd	char(8)	Резервирано

systabauth

grantor	char(8)	Потребител, даващ правото
grantee	char(8)	Потребител, на когото се дава правото
tabid	integer	Идентификатор на таблицата
tabauth	char(8)	Правото, което е низ: su*idxar

syscolauth

grantor	char(8)	Потребител, даващ правото
grantee	char(8)	Потребител, на когото се дава правото
tabid	integer	Идентификатор на таблицата
colno	smallint	Номер на колона
colauth	char(3)	Правото, което е низ: sur

sysconstraints

constrid	serial	Системен идентификатор на ограничението
constrname	char(18)	Име на ограничението
owner	char(8)	Собственик на ограничението
tabid	integer	Идентификатор на таблицата
constrtype	char(1)	Тип: P, U, R, C
idxname	char(18)	Име на индекса

syschecks

constrid	integer	Системен идентификатор на ограничението
type	char(1)	Формат на съхранение: B=binary, T=ASCII text
seqno	smallint	Пореден номер на ред
checktext	char(32)	Текст на ограничението

sysreferences

constrid	integer	Системен идентификатор на ограничението
primary	integer	Идентификатор на съответното ограничение за първичен ключ
ptabid	integer	Идентификатор на таблицата с първичния ключ
updrule	char(1)	R
delrule	char(1)	R=restrict, C=cascaded
matchtype	char(1)	Резервирано
pendant	char(1)	Резервирано

sysdefaults

tabid	integer	Идентификатор на таблица
colno	smallint	Идентификатор на колона
type	char(1)	Тип: L, U, C, N, T, S
default	char(256)	Константа, ако типът е L