

Мова SQL. Операції над відношеннями

Видалення кортежів-дублікатів

У SQL відношення розглядаються не як множини, а як мультимножини, де будь-який кортеж може бути включений у відношення кілька разів. Наприклад, однакові кортежі можуть утворитись у результаті проєкції, коли залишається частина атрибутів.

Якщо дублювання кортежів треба уникнути, після слова SELECT вказується службове слово DISTINCT. Ця операція відповідає оператору δ реляційної алгебри.

Приклад 5.12. Повернімося до прикладу 5.10 і знову розглянемо, з якими продюсерами співпрацював актор 'Harrison Ford'.

Реляційна схема БД наступна:

```
Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieExec(name, address, cert#, netWorth),
```

Без підзапитів ця задача вирішується неможливо простіше:

```
SELECT name
FROM MovieExec, Movie, StarsIn
WHERE cert# = producerC# AND
      title = movieTitle AND
      year = movieYear AND
      starName = 'Harrison Ford';
```

Рис.5.11. Запит без підзапитів для рішення задачі прикладу 5.12.

Для видалення кортежів-дублікатів перший рядок має бути таким:

```
SELECT DISTINCT name
```

Операція видалення дублювань досить дорого коштує, оскільки відношення підлягає попередньому сортуванню для визначення, чи треба видаляти кортеж. Зазвичай сортування відношення відбувається у багато разів довше, ніж виконання запиту без сортування. В той же час рішення задачі прикладу 5.12 з використанням підзапитів не призводить до дублювання.

Групування і агрегування в SQL

SQL реалізує ті ж можливості, що й оператор реляційної алгебри γ , надаючи спеціальне речення GROUP BY та дозволяючи задавати оператори агрегування у списку речення SELECT.

Оператори агрегування

Оператор агрегування переводить множину значень в одне. У стандарті SQL є 5 операторів агрегування:

- SUM - сума,
- AVG - середнє,
- MIN – мінімальне значення,
- MAX – максимальне значення,

- COUNT – кількість кортежів.

Оператори задаються у списку речення SELECT. Їх аргументами є скалярні вирази – як правило, назви *агрегованих* атрибутів відношення. Виключення - COUNT;

- якщо його аргументом є „*” (тобто COUNT(*)), він підраховує кількість кортежів.
- якщо його аргументом є назва атрибута x, він підраховує кількість *непустих* значень x.
- Якщо він записаний як COUNT(DISTINCT x), він підраховує кількість *різних* значень x.

Приклади:

SELECT AVG(netWorth) FROM MovieExec	Середнє значення річного доходу керівників кіно
SELECT COUNT(*) FROM StarsIn	Кількість кортежів у відношенні StarsIn
SELECT COUNT(starName) FROM StarsIn	Те саме
SELECT COUNT(DISTINCT starName) FROM StarsIn	Кількість різних акторів у відношенні StarsIn

Групування

Для *групування* (grouping) в SQL використовується речення GROUP BY, яке йде в конструкції „select-from-where” за реченням WHERE. Після службових слів GROUP BY задається список *групуємих атрибутів* (grouping attributes). Алгоритм групування:

- Кортежі відношення, яке задається реченням FROM, ділиться на групи, для яких набір групуємих атрибутів має одне значення.
- Якщо список SELECT має оператори агрегування, вони застосовуються для кожної групи, також повертаючи одне значення для групи.
- У результатуючому відношенні створюється один кортеж для групи, який містить атрибути:
 - Або групуємі,
 - Або результати агрегування.

Приклад 5.13. Обчислимо суми тривалості всіх фільмів кожної студії. Аналізуємо відношення:

Movie(title, year, length, inColor, studioName, producerC#).

Запит:

```
SELECT studioName, SUM(length)
FROM Movie
GROUP BY studioName;
```

Приклад 5.14. Знайдемо студії, які володіють правами на певні фільми:

```
SELECT studioName
FROM Movie
GROUP BY studioName;
```

Приклад 5.15. Знайдемо сумарну тривалість відтворення фільмів, знятих кожним продюсером, використовуючи відношення

Movie(title, year, length, inColor, studioName, producerC#),
MovieExec(name, address, cert#, netWorth).

У запиті групується віртуальне відношення, створене зі з'єднання двох збережених відношень:

```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE producerC# = cert#
GROUP BY name;
```

Рис.5.12. Приклад групування та агрегування даних з кількох відношень

Звернімо увагу, що атрибути, задіяні в реченні WHERE підзапиту, не є ні групуючими, ні агрегуючими. Але їх не повинно бути у списку речення SELECT.

Речення HAVING

Іноколи треба обрати групи, враховуючи властивості груп як таких. Тоді конструкція "SELECT ... GROUP BY" доповнюється реченням HAVING, яке є умовою на включення груп у підсумкове відношення.

Приклад 5.16. Нехай необхідно обчислити сумарні значення тривалості фільмів лише тих продюсерів, які випустили бодай один фільм до 1930 року. Відповідний запит:

```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE producerC# = cert#
GROUP BY name
HAVING MIN(year) < 1930;
```

Рисунок 5.13 - Приклад групування з використанням речення HAVING

Як бачимо, текст запиту прикладу 5.15 доповнено реченням HAVING MIN(year) < 1930 .

При виконанні запиту рис.5.13 з підсумкового відношення будуть видалені усі групи, у яких значення MIN(year) більше або дорівнює 1930. Очевидно, це можна зробити лише після групування усього запиту! WHERE запобігає обробці зайвих записів, а HAVING ні.

Чи можемо ми вирішити задачу ефективніше, якщо замість речення HAVING доповнимо речення WHERE умовою « AND year < 1930 » ?

Ні, адже нам потрібна сума тривалостей усіх фільмів режисера.

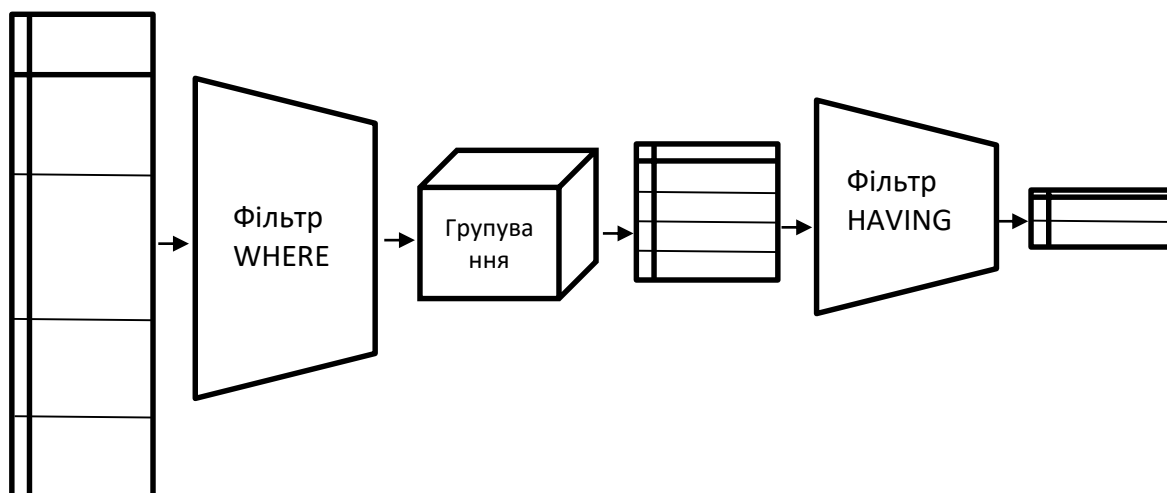


Рисунок 5.14 – Послідовність застосування фільтрів WHERE і HAVING

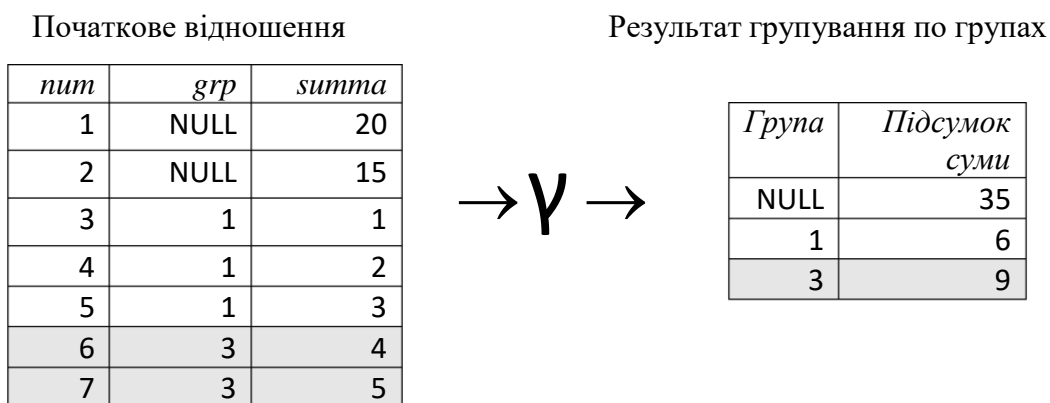
Як і в реченні SELECT, у реченні HAVING можна використовувати 2 види атрибутів:

- Групуєчі атрибути з речення GROUP BY;
- Будь-які інші атрибути відношень, згаданих у реченні FROM, але вони мають бути агреговані.

Групування, агрегування і значення NULL

Якщо ви маєте намір групувати та (або) агрегувати відношення, деякі атрибути яких дорівнюють NULL, треба мати на увазі наступне.

- У процесі *агрегування* значення NULL не враховуються ні в сумі, ні в середньому (тобто суму ділимо на кількість непустих значень), ні в мінімумі або максимумі, ні в лічильнику. Напр., COUNT(*) – число кортежів, а COUNT(A) – число непустих значень A.
- Навпаки, при *групуванні* значення NULL у групуєчому атрибуті трактується як звичайне значення, яке відрізняється від інших, не NULL. Наприклад, якщо хоча б один кортеж відношення R містить у компоненті атрибута *a* значення NULL, при виконанні запиту
SELECT a, AVG(b) FROM R GROUP BY a;
у підсумковому відношенні буде створений окремий кортеж зі значенням NULL для *a* та деяким значенням AVG(b).



SELECT grp, SUM(summa) FROM cList GROUP BY grp

Рисунок 5.15 – Обробка NULL як групуєчого атрибуту

Послідовність речень у SQL-запитах

Має бути наступна послідовність речень у виразі (операторі) SELECT:

SELECT	Обов'язкове речення
FROM	Обов'язкове речення
WHERE	
GROUP BY	
HAVING	Можна задати лише в разі наявності речення GROUP BY
ORDER BY	

Мова SQL. Модифікація бази даних. Визначення схем відношень. Віртуальні таблиці.

До модифікації БД відносяться:

1. Вставка (insertion) нових кортежів у відношення;
2. Видалення (deletion) певних кортежів з відношення;
3. Оновлення (updating) значень деяких компонентів у певних існуючих кортежах відношень.

Вставка кортежів

Основна синтаксична форма виразу вставки кортежу в відношення має вигляд:

```
INSERT INTO R(A1, A2, ..., An) VALUES (v1, v2, ..., vn);
```

де A_1, A_2, \dots, A_n - список атрибутів відношення R,

v_1, v_2, \dots, v_n - значення компонентів A, які вставляються.

Якщо список атрибутів не охоплює всі атрибути R, то компонентам кортежу, відсутнім у списку, присвоюється значення по замовчанню (зокрема, це може бути NULL).

Приклад 5.17. Поповнимо відношення StarsIn новим фактом зйомки актора у фільмі.

- 1) INSERT INTO StarsIn(movieTitle, movieYear, starName)
- 2) VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');

Значення рядка 2 ставляться у відповідність атрибутам відношення StarsIn у порядку, вказаному у першому рядку.

Якщо вставляються всі атрибути, список атрибутів можна опустити:

```
INSERT INTO StarsIn
VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');
```

В цьому разі компонентам кортежу, що вставляється, значення присвоюються в порядку, визначеному при створенні відношення.

Якщо треба вставити не один кортеж, а багато, замість речення VALUES можна використати підзапит, який повертає цю множину кортежів.

Приклад 5.18. Нехай треба доповнити відношення

```
Studio(name, address, presC#)
```

назвами студій, які містяться у відношенні

```
Movie(title, year, length, inColor, studioName, producerC#),
```

але які відсутні у Studio. Оскільки адреси студій та сертифікаційні номери президентів студій (не плутати з продюсерами) у Movie не вказані, компонентам Address та presC# буде присвоєне значення по замовчанню NULL. Запит має вигляд:

```

1)  INSERT INTO Studio(name)
2)      SELECT DISTINCT studioName
3)      FROM Movie
4)      WHERE studioName NOT IN
5)          (SELECT name
6)          FROM Studio);

```

Рисунок 5.16 - Запит INSERT з підзапитом

Розглянемо запит „з середини назовні”.

- Рядки 5, 6 генерують набір кортежів з назвами студій, які містяться у Studio.
- Рядок 4 перевіряє, чи не співпадає значення компонента StudioName з одним з цих значень.
- Запит у рядках 2-6 повертає множину назв студій, які є у Movie, але відсутні у Studio.
- Службове слово DISTINCT у рядку 2 забезпечує включення назви студії у Studio лише один раз, незалежно від кількості фільмів цієї студії у Movie.
- Рядок 1 виконує вставку назв з підзапиту у відношення Studio, компоненти Address та presC# по замовчанню заповнюються NULL.

Звернімо увагу:

Підзапит у рядках 2-6 не є корельованим, він ніяк не залежить від „основного” запиту, тому, імовірно, він виконується один раз і подальша вставка у Studio не впливає на його зміст. Однак це залежить від реалізації. Без DISTINCT запит поверне різні результати:

- Якщо вставка кортежів відкладається до завершення виконання підзапиту (рядки 2-6), запит поверне по кілька дублів студій, по числу фільмів.
- Якщо вставка кортежів виконується по мірі їх знаходження в Movie, тобто в процесі виконання підзапиту, жодний вставлений рядок не буде повторюватись.

Видалення кортежів

Вираз видалення можна представити як

```
DELETE FROM R WHERE C;
```

Буде видалений кожний кортеж відношення R, який задовольняє умові C.

Відмінність від виразу вставки: ми не вказуємо атрибути. Видаляються не значення атрибутів, а кортежі цілком.

Подібність до виразу вставки: у реченні FROM указується лише один персистентний (збережений) об’єкт – таблиця або віртуальна таблиця (View).

Приклад 5.20. Звільнимо відношення

```
MovieExec(name, address, cert#, netWorth)
```

від “олігархів, які п’ють кров американського народу” – тих керівників кіноіндустрії США, чий річний дохід, на нашу думку, переходить межі пристойності.

```
DELETE FROM MovieExec
```

```
WHERE netWorth>=10000000;
```

Оновлення даних

У SQL терміну *оновлення* надається суворо визначений зміст: заміна значень заданих компонентів у групі існуючих кортежів. Вираз оновлення можна записати так:

```
UPDATE R SET  $A_1 = v_1, A_2 = v_2, \dots, A_n = v_n$  WHERE C;
```

При виконанні команди система відшукує у відношенні R усі кортежі, які задовольняють умові C, обчислює вирази v_i та присвоює їх значення компонентам атрибутів A_i знайдених кортежів.

Приклад 5.21. Хай необхідно змінити зміст відношення

```
MovieExec(name, address, cert#, netWorth)
```

так, щоби імені кожного керівника, який є президентом кіностудії, передував префікс "Pres." Цей почесний префікс отримують лише ті керівники, чий сертифікаційний номер (cert#) був присутній у компоненті presC# одного з кортежів відношення Studio. Відповідний запит (команда?):

- 1) UPDATE MovieExec
- 2) SET name = 'Pres. ' || name
- 3) WHERE cert# IN (SELECT presC# FROM Studio);

Рис.5.17. Запит на оновлення значень атрибута відношення

Рядок 2 містить інструкцію привласнення; при її виконанні у компонент *name* кожного кортежу заноситься почесний префікс та рядок, який там був раніше.

Символами „||” позначений оператор зчеплення рядків. Зчеплення в різних діалектах SQL:

ANSI SQL	Jet SQL	Transact-SQL	PL/SQL
	&	+	 Concat(str1, str2)

Визначення схем відношень у SQL

Засоби SQL, розглянуті вище, тобто запити на вибірку і модифікацію змісту БД, відносяться до категорії *управління даними* (data manipulation). У цьому розділі розглянемо засоби *визначення даних* (data definition), тобто опису структури зберігання інформації засобами SQL.

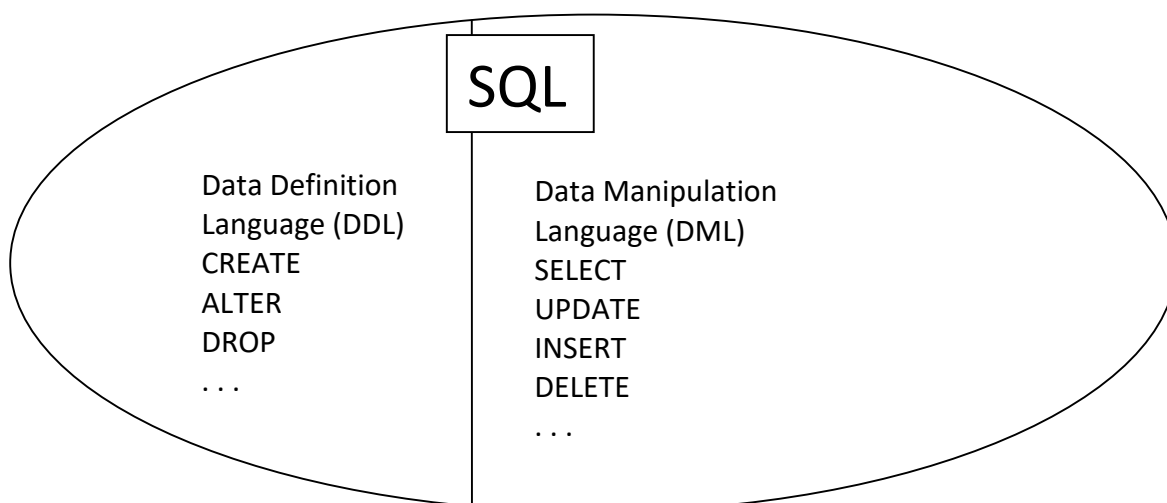


Рисунок 5.17 – Структура SQL

Прості оголошення схем відношень

Конструкція оголошення таблиці:

CREATE TABLE <назва нового відношення>

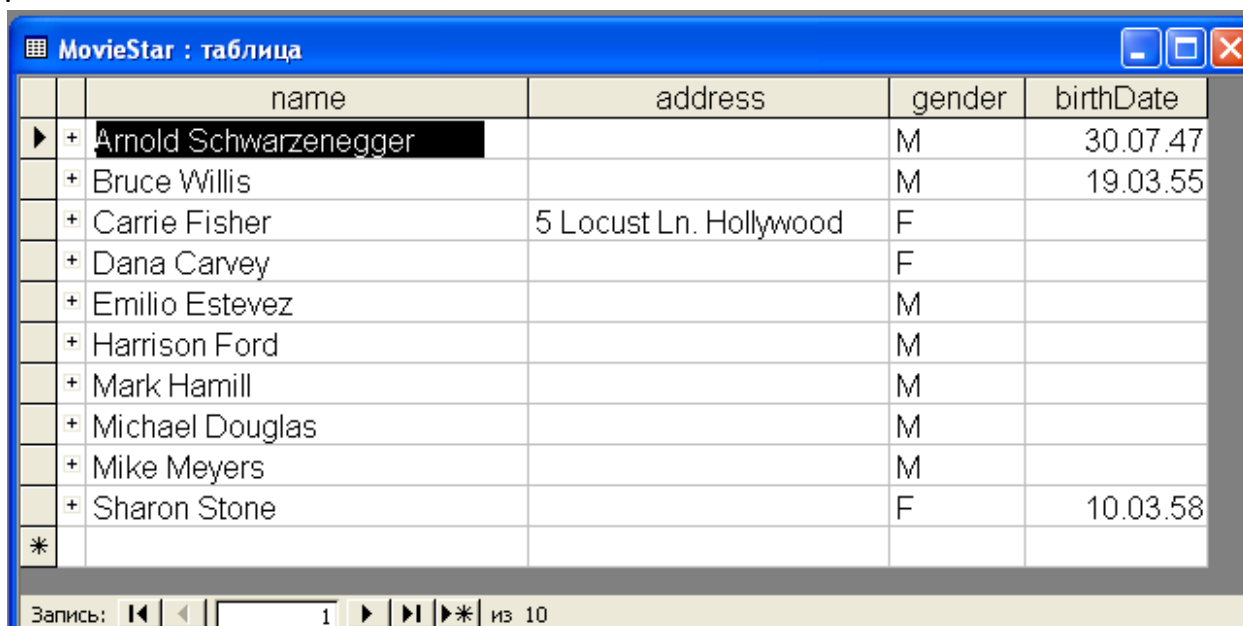
(<атрибут тип>

<атрибут тип>

...)

Перелік типів наведено в табл.5.1.

Приклад 5.22. Створимо відношення (таблицю) MovieStar(name, address, gender, birthdate)



	name	address	gender	birthDate
▶ +	Arnold Schwarzenegger		M	30.07.47
+	Bruce Willis		M	19.03.55
+	Carrie Fisher	5 Locust Ln. Hollywood	F	
+	Dana Carvey		F	
+	Emilio Estevez		M	
+	Harrison Ford		M	
+	Mark Hamill		M	
+	Michael Douglas		M	
+	Mike Meyers		M	
+	Sharon Stone		F	10.03.58
*				

Записи: 1 из 10

Рисунок 5.18 - Бажаний зміст таблиці MovieStar

Прийmemo такі рішення щодо структури:

- Ім'я актора – рядок фіксованої довжини 30 символів CHAR(30), доповнюється зліва пробілами або усікається до 30 символів.
- Адреса – рядок змінної довжини VARCHAR(255).
- Стать (gender) – рядок одиничної довжини CHAR(1), зберігає символи 'M' (male) або 'F' (female).
- Дата народження birthdate має тип DATE. Якщо його в реалізації SQL нема, доводиться зберігати дату як рядок (або як число – кількість днів з початку нашої ери).

SQL-оголошення має вигляд:


```
CREATE TABLE MovieStar (
    name CHAR(30),
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE
);
```

Рисунок 5.19 - SQL-оголошення реляційної схеми відношення MovieStar

Модифікація реляційних схем

Для видалення відношення з БД треба виконати команду

```
DROP TABLE R;
```

Фізично у файлах БД з'являються вільні блоки, які в подальшому можуть бути використані.

Перейменування таблиці або колонки в SQL не передбачене. Для цього, наприклад, в Transact SQL є збережена процедура `sp_rename`.

Частіше виникає необхідність *зміни* (altering) схеми існуючого відношення. Це робить команда `ALTER TABLE`, яка містить

- Назву відношення, яке підлягає зміні;
- Одне з додаткових речень, найважливішими серед яких є:
 - `ADD`, з іменем *A* атрибуту та позначенням його типу ("додати атрибут *A*");
 - `DROP`, з іменем *A* атрибуту ("видалити атрибут *A*").

Приклад 5.23. Поповнимо схему відношення MovieStar атрибутом `phone` („номер телефону”) символного типу постійної довжини 16.

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

Кожний існуючий кортеж поповниться компонентом `phone`, але оскільки реальні номери телефонів невідомі, система присвоїть їм значення `NULL`, передбачене по замовчанню.

Для видалення зі схеми відношення MovieStar атрибута `birthdate` треба виконати команду

```
ALTER TABLE MovieStar DROP birthdate;
```

Значення за замовчанням у реляційній схемі

Будь-яку інструкцію, де визначається назва атрибута і його тип, можна доповнити службовим словом `DEFAULT` та певним значенням. Цим значенням може бути:

- `NULL`;
- Константа;
- Інше значення, як-от значення поточного моменту часу.

Приклад 5.24. Доповнимо приклад 5.22. Хай треба визначити 2 атрибути зі значеннями по замовчанню.

- Для компонентів атрибуту *gender* значення по замовчанню – ‘?’.
- Для компонентів атрибуту *birthDate* значення по замовчанню – `DATE '0000-00-00'`, тобто „нульова” дата. Рядки 4, 5 рис.5.26 будуть такими:

- 4) gender CHAR(1) DEFAULT '?',
 5) birthdate DATE DEFAULT DATE '0000-00-00'

Приклад 5.25. Нехай у прикладі 5.22 компоненту phone має присвоюватись значення по замовчанню „Номер невідомий”.

ALTER TABLE MovieStar ADD phone CHAR(16) DEFAULT 'Номер невідомий'

Для зміни типа існуючого поля є інструкція ALTER COLUMN. В ній треба вказати ім'я поля, його тип і (для текстових і двійкових полів) необов'язковий розмір. Наприклад, наступна інструкція дозволяє в таблиці Employee (Співробітники) змінити тип поля PostIndex (Поштовий індекс), спочатку визначений як INTEGER, перевизначивши це поле як текстове завдовжки 10 знаків:

ALTER TABLE Employee ALTER COLUMN PostIndex TEXT(10)

Чи збережуться дані у зміненій колонці? Це залежить від старого і нового типів, від реалізації. Гарантії нема. Тому в умовах експлуатації системи класу ERP на базі MsSQL використовувався такий алгоритм оновлення структури таблиці А в БД:

№пп	Дія	Інструкція SQL або збережена процедура
1	Видалити зв'язки таблиці А	ALTER TABLE A DROP CONSTRAINT
2	Перейменувати таблицю (напр. у A_Old)	sp_rename
3	Створити таблицю А нової структури	CREATE TABLE A
4	Імпортувати по можливості всі дані з таблиці A_Old в А	INSERT INTO
5	Відновити зв'язки таблиці А, перевіряється цілісність	ALTER TABLE A ADD CONSTRAINT
6	Видалити A_Old	DROP TABLE A_Old

Додатково цей процес корисний тим, що зменшує фрагментованість таблиці, групуючи разом її блоки, що дозволяє ефективно стиснути БД (SHRINK DATABASE).

Віртуальні таблиці (подання)

Визначення подання

Відношення, створене командою CREATE TABLE, є *персистентним*, тобто таким, що зберігає зміст, допоки не буде явно змінено командами INSERT, DELETE, UPDATE, DROP.

Інша категорія відношень (*неперсистентні*) не створюється на фізичному рівні. Вони не зберігають кортежі фізично, а відбирають дані з інших відношень кожного разу під час звертання до них. Вони зберігають фізично лише текст відповідного запиту. Такі відношення називають:

- Віртуальні таблиці (virtual tables);
- Представлення (views);
- Подання;
- Розрізи.

Відношення є загальною назвою для персистентних відношень і для неперсистентних (розрізів). Персистентні відношення програмісти схильні називати *таблиці* (базові таблиці, базові відношення).

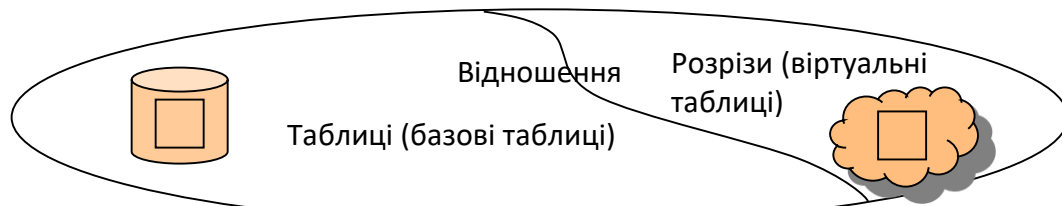


Рис.5.19. Таблиці та подання

Вираз створення розрізу:

```
CREATE VIEW R AS Q;
```

де R – назва розрізу,

Q - запит, який є визначенням розрізу; при зверненні до розрізу з певним запитом система виконає запит Q, після чого використає результат для задоволення свого запиту.

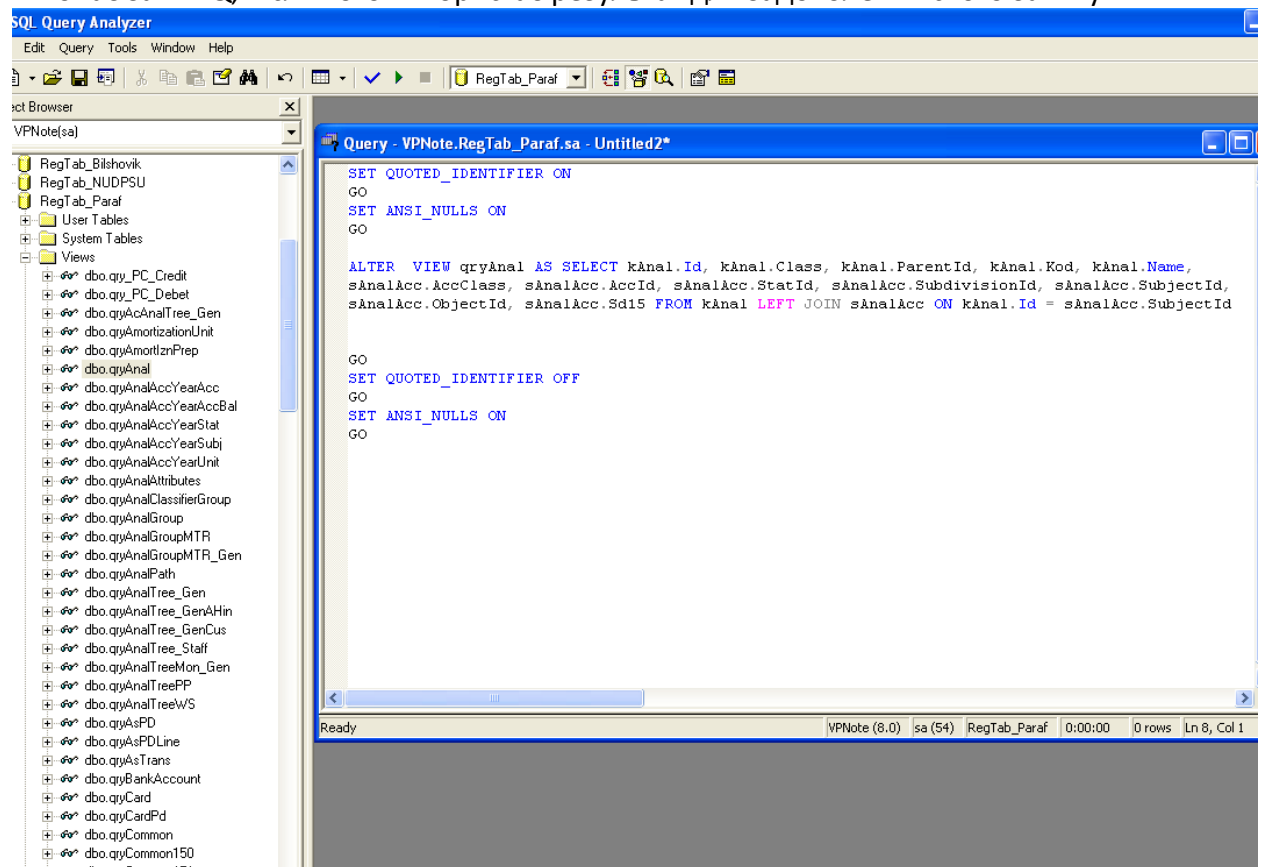


Рисунок 5.20 – Розрізи в БД системи класу ERP „Регістри”

Приклад 5.26. Нехай треба створити розріз, який є частиною відношення

`Movie(title, year, length, inColor, studioName, producerC#),`

а саме значення компонентів title і year тих кортежів, які містять інформацію про фільми, зняті на студії *Paramount*.

```

1) CREATE VIEW ParamountMovie AS
2)     SELECT title, year
3)     FROM Movie
4)     WHERE studioName = 'Paramount';

```

Приклад 5.27. Нехай нас цікавлять дані про всіх акторів, які знімалися у фільмах студії Paramount.

```

SELECT DISTINCT starName
FROM ParamountMovie, StarsIn
WHERE title = movieTitle AND year = movieYear;

```

Припустимо в контексті одного запиту об'єднувати таблиці та розрізи.

Матеріалізований розріз

Матеріалізований розріз - об'єкт бази даних, який містить результати запиту. Речення FROM запиту може посилатись на таблиці, розрізи і інші матеріалізовані розрізи. Загалом ці об'єкти називаються master tables (термін реплікації) або detail tables (термін сховищ даних).

У MySQL якщо для розрізу створюється унікальний кластеризований індекс, розріз стає індексованим, тобто його результуючий набір фізично зберігається точно так же, як і базові таблиці.

Перейменування атрибутів розрізу

Нехай ми маємо намір надати атрибутам розрізу імена, відмінні від успадкованих від схем базових відношень (як-от для обчислюваних атрибутів). Для цього в інструкції оголошення подання після його назви в круглих дужках треба перелічити нові імена. (Те ж можна виконати в тексті запиту службовим словом AS або навіть без нього).

Приклад 5.28. Нехай нас цікавлять назви фільмів та їхні продюсери.

```

CREATE VIEW MovieProd (movieTitle, prodName) AS
  SELECT title, name
  FROM Movie, MovieExec
  WHERE producerC# = cert#;

```

Напр., аби знайти ім'я (або імена) продюсера фільму *Gone With the Wind*, треба виконати запит

```
SELECT prodName FROM MovieProd WHERE movieTitle='Gone With the Wind'
```

Модифікація змісту розрізів

В разі виконання певних досить жорстких умов припустимо звертатись до розрізу з командами вставки (INSERT), видалення (DELETE), та оновлення (UPDATE) кортежів. Для деяких порівняно простих розрізів (розрізів, що оновлюються – updatable) система в змозі перетворити команду модифікації такого розрізу в еквівалентну інструкцію зміни змісту базової таблиці. Стандарт SQL містить формальне визначення умов, які сприяють можливості модифікації розрізів. У грубому наближенні ці правила можна викласти так:

Розріз припускає модифікацію, якщо він визначений на основі запиту, який передбачає вибір (SELECT – не SELECT DISTINCT) значень певних атрибутів одного відношення R, яке в свою чергу також може бути розрізом, що відповідає цим вимогам).

```
CREATE VIEW S AS
```

```
SELECT DISTINCT L FROM R, S WHERE C;
```

Додаткові технічні нюанси:

- Речення WHERE C не повинно містити відношення R у складі жодного підзапиту.
- Список речення SELECT L має охоплювати достатньо атрибутів, аби при спробі вставки кортежу в розріз система змогла присвоїти іншим атрибутам значення по замовчанню, передбачені в базовій таблиці. Тобто обов'язкові поля таблиці, у яких нема значень по замовчанню, мають бути в розрізі.

Приклад 5.29. Повернімося до прикладу 5.26.

```
1) CREATE VIEW ParamountMovie AS
2)     SELECT title, year
3)     FROM Movie
4)     WHERE studioName = 'Paramount';
```

Спробуємо вставити новий кортеж у розріз.

```
INSERT INTO ParamountMovie
VALUES('Star Trek', 1979);
```

Розріз ParamountMovie задовольняє формальним вимогам стандарту SQL для модифікації, оскільки побудований на виборі з одної таблиці

Movie(title, year, length, inColor, studioName, producerC#).

Але є проблема:

Атрибут таблиці studioName не входить у набір атрибутів розрізу. Тому при додаванні кортежу компоненту studioName таблиці Movie буде присвоєне значення NULL, а не 'Paramount', як хотілось би.

Рішення. Треба внести у визначення розрізу атрибут studioName, незважаючи на те, що в відношенні-результаті виконання розрізу він буде містити одне для всіх кортежів значення 'Paramount'. Оновлена інструкція створення розрізу:

```
CREATE VIEW ParamountMovie AS
  SELECT studioName, title, year
  FROM Movie
  WHERE studioName = 'Paramount';
```

Команда додавання кортежу в ParamountMovie тоді така:

```
INSERT INTO ParamountMovie
VALUES('Paramount', 'Star Trek', 1979);
```

Атрибути таблиці Movie такі, як length, inColor, producerC# невідомі під час додавання кортежу. Якщо для атрибуту length у схемі Movie передбачене значення по замовчанню 0, у відношення Movie було б додано кортеж:

<i>title</i>	<i>year</i>	<i>length</i>	<i>inColor</i>	<i>studioName</i>	<i>producerC#</i>
Star Trek	1979	0	NULL	Paramount	NULL

Розрізи, що оновлюються, припускають також **видалення** кортежів.

Приклад 5.30. Нехай треба видалити з розрізу ParamountMovie фільми, в назві яких присутнє слово "Trek". Для цього застосовуємо команду

```
DELETE FROM ParamountMovie
WHERE title LIKE '%Trek%';
```

Ця команда перетворюється в еквівалентну інструкцію видалення кортежів з таблиці Movie, де в реченні WHERE буде додаткова умова з визначення подання:

```
DELETE FROM Movie
WHERE title LIKE '%Trek%' AND studioName = 'Paramount';
```

Аналогічно, команди **оновлення** розрізу (якщо вони припустимі) транслюються в інструкції зміни змісту відповідних кортежів базової таблиці.

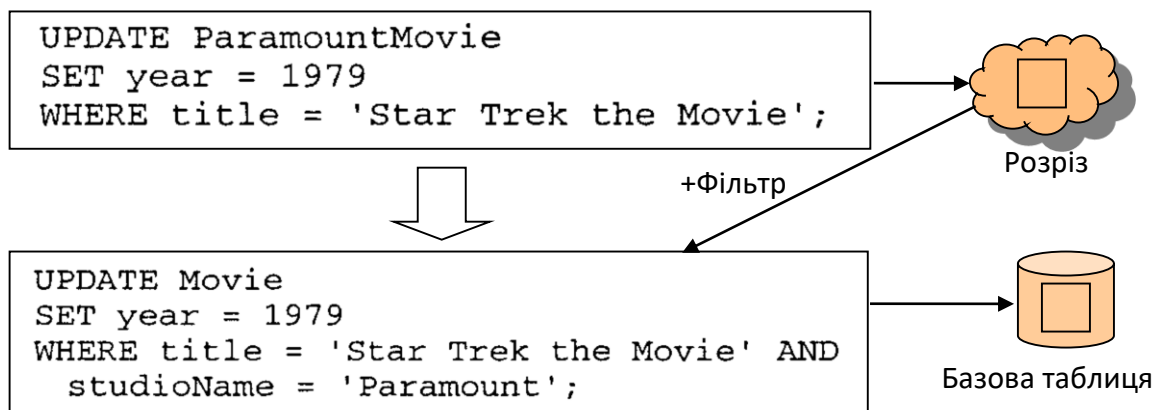


Рисунок 5.21. Перетворення команди оновлення розрізу на команду оновлення базової таблиці.

Нарешті, ще одна команда модифікації – **видалення** розрізу. Її можна виконати незалежно від того, чи оновлюваний розріз. Приклад:

```
DROP VIEW ParamountMovie;
```

Після її виконання розріз зникає, базова таблиця – ні. Якщо виконати команду `DROP TABLE Movie;` то зникне і базова таблиця, і розріз.

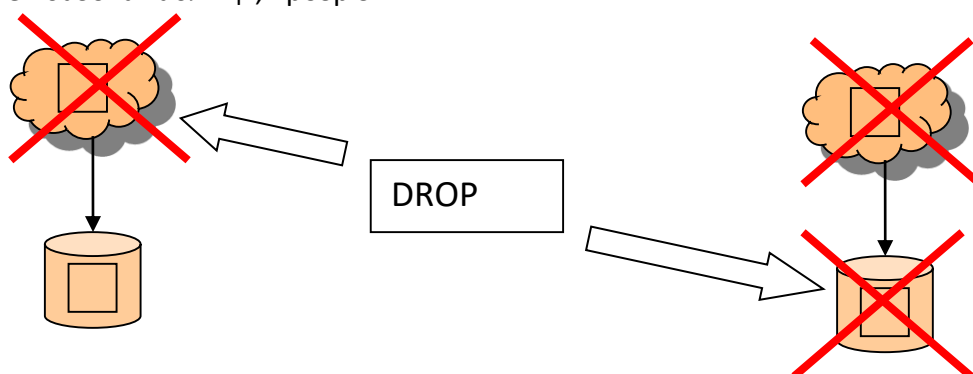


Рисунок 5.22 - Наслідки видалення розрізу та базової таблиці.