Обмеження та тригери в SQL

Активні елементи

Одна з серйозних проблем розробників застосувань БД: при оновленні інформації (частіше під час „ручного” введення) певні елементи інформації стають невірними або входять у протиріччя з іншими елементами.

Погане (дороге) рішення: покласти піклування про коректність даних на прикладну програму, виходячи з того, що кожній операції зміни БД має передувати відповідна перевірка.

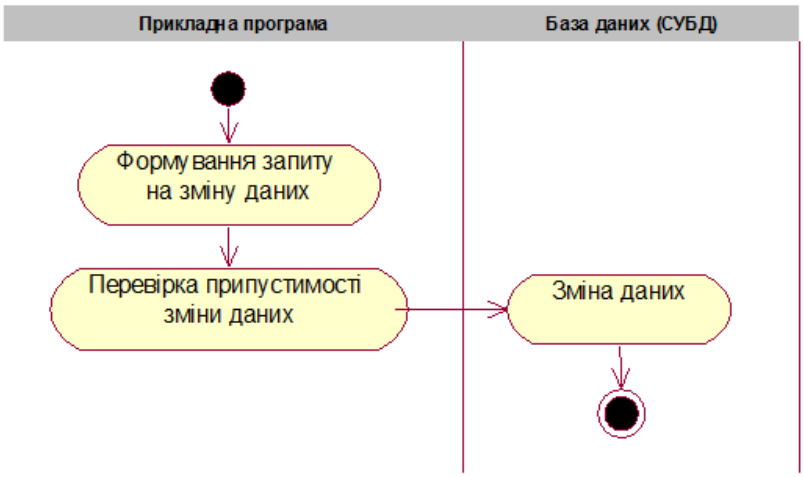

Рисунок 5.40. Контроль припустимості змін БД прикладною програмою

Недоліки такого рішення:

- Візьмемо до уваги велику кількість звернень прикладної програми до БД.
- Головне – умови цілісності БД в більшості випадків випливають зі структури БД. Ті ж умови, що не випливають безпосередньо з відносно простих структур БД, дійсно мають перевірятись прикладною програмою.

Розглянемо так звані „активні” елементи.
Активний елемент (active element) – вираз або команда, які збережені в БД та виконуються по мірі необхідності в певні моменти часу та у зв’язку настанням подій, таких, як вставка кортежу у певне відношення або зміна стану БД, при яких певна умова набуває значення TRUE.

До таких активних елементів відносимо:

- Первинні (власні) ключі,
- Зовнішні ключі,
- Тригери.

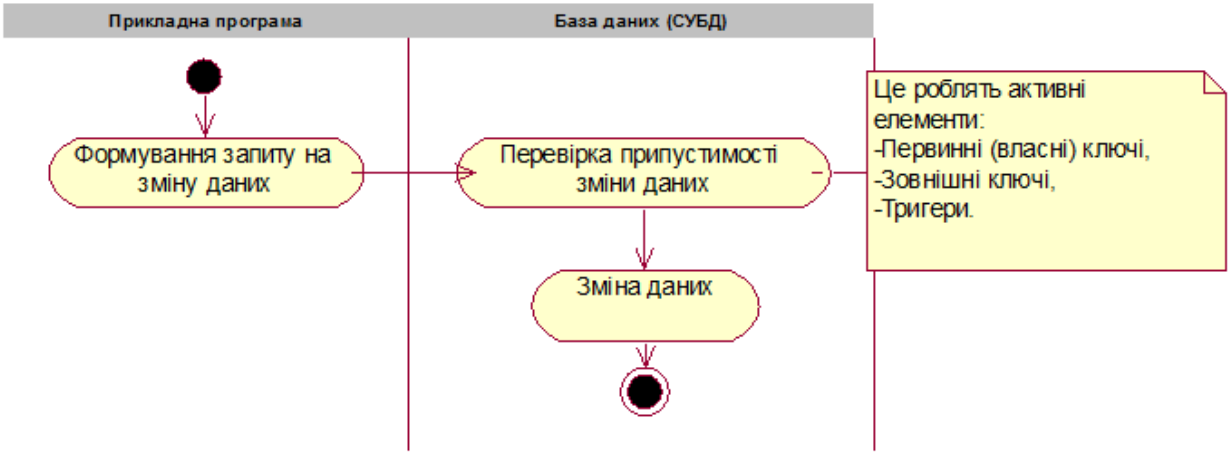

Рисунок 5.41. Контроль припустимості змін БД зі сторони СУБД

Ключі відношень

Види ключів і обмеження

Якщо підмножина атрибутів S є ключем відношення R, будь-які два кортежі R повинні різнитися у значеннях компонентів хоч би одного з атрибутів S. Як наслідок, якщо схема відношення R містить оголошення ключа, R не може містити записи, що дублюються.

Обмеження ключа, як і багато інших обмежень, формулюються в контексті команди CREATE TABLE. Для кожного відношення можуть бути визначені:

- Лише один первинний ключ - парою службових слів PRIMARY KEY,
- Будь-яка кількість „просто унікальних" ключів - службовим словом UNIQUE.

Обмеження посилальної цілісності вимагає, що якщо певні значення існують у певних компонентах одного відношення, вони повинні існувати і в компонентах первинного ключа іншого відношення. Встановлення такого зв'язку передбачається в команді CREATE TABLE службовими словами REFERENCES або FOREIGN KEY.

Оголошення первинного ключа

Оголошення певних компонентів у складі первинного ключа означає для них:

- Ні у яких двох кортежів цього відношення не можуть співпадати кожні пари наборів ключових компонентів одночасно. Спроба вставки такого кортежу буде відторгнута системою.
- Усі компоненти ключових атрибутів не можуть одночасно містити NULL. (Чи можуть окремі компоненти ключа бути NULL? Це може залежати від реалізації СУБД.)

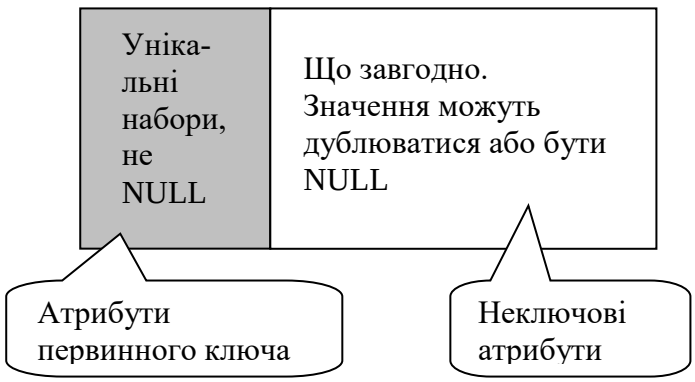

Рисунок 5.42. Зміст атрибутів первинного ключа і неключових

Приклад 5.40. Нехай треба створити відношення MovieStar (актори) з первинним ключем name.

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE
    );
```

Рисунок 5.43. Варіант 1 оголошення первинного ключа шляхом надання атрибуту відповідного статусу

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30),
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE,
6)      PRIMARY KEY (name)
    );
```

Рисунок 5.44. Варіант 2 оголошення первинного ключа у вигляді окремого речення

Якщо ключ складається з двох і більше атрибутів, можливий лише другий варіант оголошення. Напр., для відношення Movie окреме речення має вигляд

```
PRIMARY KEY (title, year)
```

Розглянемо питання:

- кожний компонент первинного ключа одного кортежу не може містити NULL,
- чи всі компоненти первинного ключа одного кортежу одночасно не можуть містити NULL?

Ні Ульман, ні Вікіпедія не відповідають однозначно.
Можна перевірити: кожний компонент первинного ключа одного кортежу не може містити NULL (для Oracle).

Оголошення обмеження унікальності

У синтаксисі CREATE TABLE треба просто замінити PRIMARY KEY на UNIQUE. Нагадаємо про 2 відмінності UNIQUE від PRIMARY KEY:

- UNIQUE індексів може бути кілька для відношення;
- Можливо, що всі компоненти UNIQUE індекса одного кортежу містять NULL. Вимога унікальності в цьому сенсі не порушується, оскільки NULL – не значення.

Ключі та індекси

Індекс – ланцюжок впорядкованих значень певних атрибутів відношення. Він потрібний для швидкого знаходження певних значень атрибутів у відношенні, на кшталт того, що задає речення WHERE. Створення індексів не регламентується жодним стандартом SQL.

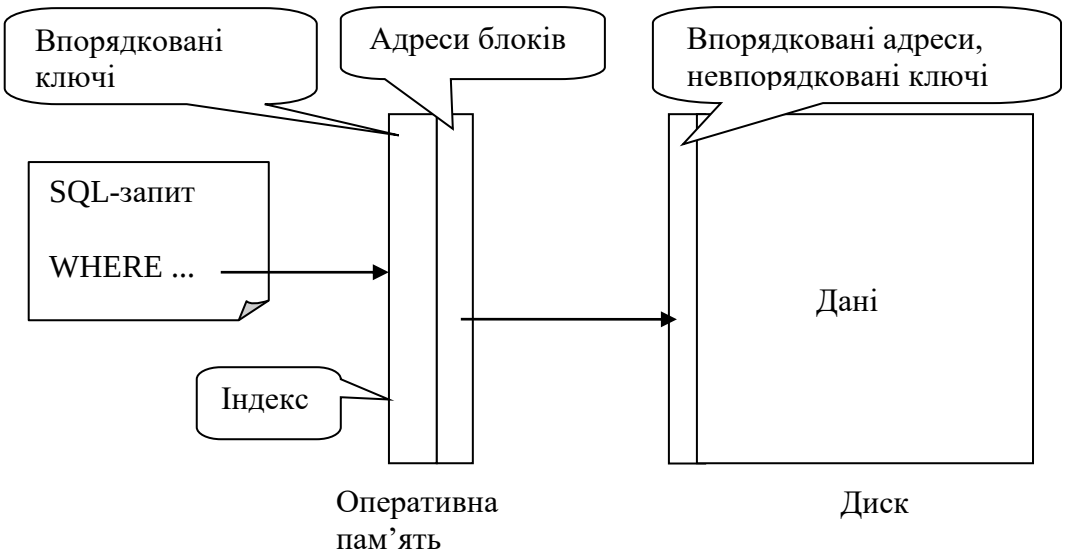

Рисунок 5.44. Пошук кортежів з певним значенням ключа за допомогою індексу

Логічно, що система автоматично створює індекс для первинного, а інколи – для унікального ключа. Імовірно, що запити з пошуком значень ключів будуть найбільш „популярними".

Наявність індексів для ключів життєво важлива для дотримання обмежень (унікальності) ключів. Зазвичай перевірки унікальності виконуються для команд INSERT та UPDATE.

Якщо індекс для ключа не створено, дотримання його обмежень можливе, якщо постійно підтримувати сортування відношення по ключових атрибутах. У разі відсутності будь-якої відправної точки системі для пошуку значення треба переглянути все відношення. Тоді задача модифікації великого відношення стає нерозв'язною.

Оголошення зовнішнього ключа

Оголошення зовнішнього ключа тягне дві вимоги:

- Адресуються або первинний ключ, або унікальний ключ. Ці ключі інколи називаються *власний ключ*.
- У відношенні, яке адресується (напр., адресується відношення *Деталі* з відношення *Залишки*) має існувати відповідний кортеж.

Приклад 5.41. Нехай треба оголосити відношення

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

яке має первинний ключ name та зовнішній ключ presC#, який посилається на атрибут cert# відношення MovieExec.

```
CREATE TABLE Studio (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  presC# INT REFERENCES MovieExec(cert#)
);
```

Рисунок 5.45. Варіант 1 оголошення зовнішнього ключа шляхом надання атрибуту відповідного статусу

```
CREATE TABLE Studio (
  name CHAR(30) PRIMARY KEY,
  address VARCHAR(255),
  presC# INT,
  FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)
);
```

Рисунок 5.46. Варіант 2 оголошення зовнішнього ключа у вигляді окремого речення

Стратегії забезпечення посилальної цілісності

„ Висячі" кортежі, тобто такі, зовнішній ключ яких посилається на неіснуючий власний ключ, потенційно можуть утворюватись при:

- Додаванні кортежів у відношення зовнішнього ключа;
- Зміни значень як власного, так і зовнішнього ключів;
- Видалення кортежів власного ключа.

Існують 3 стратегії забезпечення посилальної цілісності, тобто запобігання утворенню висячих посилань зовнішнього ключа.

- По замовчанню – заборона таких операцій, які порушують цілісність.
- Стратегія каскадної модифікації передбачає видалення або оновлення висячих кортежів відношення, яке містить зовнішній ключ, у відповідності зі змінами у відношенні власного ключа.
  - Видалення кортежів власного ключа тягне видалення відповідних кортежів зовнішнього ключа. В команді CREATE TABLE позначається
    ON DELETE CASCADE
  - Зміна значень власного ключа тягне таку ж зміну значень зовнішнього ключа. В команді CREATE TABLE позначається
    ON UPDATE CASCADE
- Стратегія *set-null* передбачає присвоєння NULL компонентам зовнішнього ключа кожного з висячих кортежів. Відповідні речення в команді CREATE TABLE:
  ON DELETE SET NULL
  ON UPDATE SET NULL

Ці стратегії задаються в контексті оголошення зовнішнього, а не власного ключа.

Приклад 5.42. Розглянемо оголошення відношення студій, яке містить зовнішній ключ, що посилається на відношення президентів (містить власний ключ).

```
1)  CREATE TABLE Studio (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      presC# INT REFERENCES MovieExec(cert#)
5)          ON DELETE SET NULL
6)          ON UPDATE CASCADE
    );
```

Рисунок 5.47. Приклад оголошення стратегій забезпечення посилальної цілісності

Відповідно до рис.5.47 передбачаються дії:

- Якщо президента звільнено, студія тимчасово працює без президента (рядок 5).
- Якщо у відношенні MovieExec у президента змінився номер, він автоматично змінюється у відношенні Studio (рядок 6).

Обмеження рівня атрибутів та кортежів

Розглянемо різновид обмежень, у відповідності з якими компоненти певних атрибутів відношення можуть мати значення лише з зарані обумовленої підмножині припустимих значень.

Обмеження рівня атрибуту	Перевіряється, якщо змінюється атрибут
Обмеження рівня кортежу	Перевіряється, якщо додається кортеж або змінюється будь-який атрибут кортежу

Обмеження NOT NULL

Одне з найпростіших обмежень рівня окремого атрибуту дозволяє заборонити присвоєння компонентам атрибутів значення NULL.

Приклад 5.43. У відношенні Studio (приклад 5.42) для заборони присвоєння компонентам presC# значень NULL треба доповнити рядок 4 так:

```
4)  presC# INT REFERENCES MovieExec(cert#) NOT NULL
```

Наслідком цього оголошення, крім неможливості додавання кортежів без серт.№ президента, є неможливість застосування стратегії set-null (рядок 5 рис. 5.47).

Обмеження CHECK рівня атрибуту

Більш складне обмеження рівня атрибуту задається службовим словом CHECK, за яким у круглих дужках іде умова, така ж, як у реченні WHERE. Можливе використання підзапитів. Зазвичай так задаються перелічувані множини значень або їхні межі.

Обмеження CHECK перевіряється, коли компонент, для якого установлене обмеження, отримує нове значення (шляхом INSERT або UPDATE). Якщо нове значення атрибуту порушує обмеження, операція відміняється.

Приклад 5.44. Нехай у відношенні

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

(приклад 5.42) виникла вимога: номери президентів мають містити не менше 6 цифр. Замінимо рядок 4 рядком

```
4)  presC# INT REFERENCES MovieExec(cert#)
        CHECK (presC# >= 100000)
```

Приклад 5.45. Нехай ми передбачаємо, що атрибут gender схеми відношення

```
MovieStar(name, address, gender, birthdate)
```

не лише відноситься до типу CHAR(1), але й приймає одне з двох значень: ‘F’ – жінка, або ‘M’ – чоловік. В обмеженні виразимо це так:

```
gender CHAR(1) CHECK (gender IN ('F', 'M'))
```

Можливості порушення обмеження CHECK рівня атрибуту.

В умові CHECK можна використати інші компоненти кортежу і навіть підзапити. Оскільки перевірка виконується лише для атрибуту під CHECK, зміни значень інших атрибутів та зміни в інших залучених відношеннях не контролюються цим обмеженням, що може привести до його порушення. Обмеження CHECK не може замінити механізму посилальної цілісності RDI (Relational Data Integrity), вбудованого в реляційну СУБД.

Обмеження CHECK рівня кортежу

Для визначення обмеження, що регламентує склад кортежів відношення R, в оголошенні CREATE TABLE після списку атрибутів, ключів та зовнішніх ключів слід написати CHECK та умову по правилах WHERE у круглих дужках.

Оскільки обмеження рівня кортежу, умова перевіряється до кожного додавання кортежу в відношення або до оновлення існуючого кортежу. Але, як і в обмеженні CHECK рівня атрибуту, обмеження CHECK рівня кортежу не реагує на зміни, які вносяться в інші відношення. Таким чином, гарантується дотримання обмеження, якщо в ньому задіяні лише компоненти кортежу, що перевіряється, та це обмеження не містить підзапитів.

Незважаючи на „діру", обмеження CHECK потрібні, оскільки, виконуючи перевірку лише при зміні контрольованого компонента або кортежу, вони реалізуються більш ефективно, ніж обмеження загального виду (assertions, у Transact SQL, на жаль, відсутні) або тригери.

Приклад 5.46. Розглянемо оголошення відношення MovieStar, в якому передбачається, що в рядку імені актора не повинен бути префікс "Ms.", якщо він є чоловіком.

```
1)  CREATE TABLE MovieStar (
2)      name CHAR(30) PRIMARY KEY,
3)      address VARCHAR(255),
4)      gender CHAR(1),
5)      birthdate DATE,
6)      CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
    );
```

Рисунок 5.48. Простий приклад обмеження CHECK рівня кортежу

Модифікація обмежень

Для модифікації обмеження воно мусить мати ім'я. Для надання імені треба перед оголошенням обмеження поставити слово CONSTRAINT і після нього – ім'я.
Приклад обмеження рівня атрибуту:

```
name CHAR(30) CONSTRAINT NameIsKey PRIMARY KEY
```

Приклад обмеження рівня кортежу:

```
CONSTRAINT RightTitle
    CHECK (gender = 'F' OR name NOT LIKE 'Ms.%')
```

Щоби змінити обмеження, треба його видалити і додати заново. Приклади видалення обмежень:

```
ALTER TABLE MovieStar DROP CONSTRAINT NameIsKey;
ALTER TABLE MovieStar DROP CONSTRAINT RightTitle;
```

Приклади додавання (відновлення) обмежень:

```
ALTER TABLE MovieStar ADD CONSTRAINT NameIsKey
  PRIMARY KEY (name);

ALTER TABLE MovieStar ADD CONSTRAINT RightTitle
  CHECK (gender = 'F' OR name NOT LIKE 'Ms.%');
```

Можуть бути створені обмеження лише рівня кортежу, але не атрибуту. Для останньої дії треба перестворювати атрибут.

Тригери в SQL

Тригери як правила „подія-умова-дія”

Тригери ще називають *правилами „подія-умова-дія” (event-condition-action rules, ECA rules).* Особливості тригерів як обмежень:

1. Тригер „спрацьовує” лише при настанні певної *події* (event), яка описана в тексті тригера. Зазвичай це такі події, як операції додавання, видалення і оновлення відношення. Інколи це факт завершення транзакції.
2. До виконання дій тригер перевіряє умову. Для виконання дій умова має повернути TRUE, в разі FALSE виконання тригера завершується.
3. В разі істинності умови виконується дія, яка є послідовністю операцій з БД, у відповідності до логіки розробника.

Приклад 5.47 (простий). Нехай є відношення

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

яке містить перелік керівних осіб кіноіндустрії, netWorth – річний дохід кожного. Треба створити тригер, який запобігає будь-якій спробі його зниження.

```
 1)  CREATE TRIGGER NetWorthTrigger
 2)  AFTER UPDATE OF netWorth ON MovieExec
 3)  REFERENCING
 4)      OLD ROW AS OldTuple,
 5)      NEW ROW AS NewTuple
 6)  FOR EACH ROW
 7)  WHEN (OldTuple.netWorth > NewTuple.netWorth)
 8)      UPDATE MovieExec
 9)      SET netWorth = OldTuple.netWorth
10)      WHERE cert# = NewTuple.cert#;
```

Рисунок 5.51. Приклад тригера рівня кортежу.

Рядки	*Що вони містять*
1	Заголовок, назва тригера
2	Подія тригера – після оновлення компонента netWorth
3-5	Посилання на кортежі зі старими та новими значеннями. OLD ROW та NEW ROW надаються програмісту для використання в SQL-коді тригера, а OldTuple та NewTuple використовуються ним надалі як псевдоніми цих відношень.
6	Код тригера повинен спрацьовувати для кожного оновленого кортежу. Альтернатива (по замовчанню) – FOR EACH STATEMENT, тоді тригер мав би спрацьовувати для SQL-виразу оновлення відношення в цілому.
7	Умова спрацювання тригера: старий дохід більше нового
8-10	Відновлення старого рівня доходу

Доповнення:

- Крім AFTER, на його місці можливе слово BEFORE.
- Крім UPDATE, можливі події-операції INSERT і DELETE.

- Речення OF (рядок 2) необов'язкове. Без нього тригер спрацьовує при оновленні будь-якого компонента кортежу. OF не застосовується в командах INSERT та DELETE.
- Умова WHEN не обов'язкова. Без неї тригер спрацьовує завжди при оновленні кортежу.
- Наступні команди не можна використовувати в секції дій:
  - Команди управління транзакціями, як-от COMMIT та ROLLBACK.
  - Управління з'єднанням клієнта з сервером (CONNECT, DISCONNECT).
  - Створення (CREATE), зміна (UPDATE) та видалення (DROP) елементів схеми.
  - Визначення параметрів сеансу (SET).
- В команді видалення недоступна таблиця нових записів, лише старих. Симетрично, в команді додавання недоступна таблиця старих даних.

Тригери INSTEAD OF

Цієї можливості нема у стандарті SQL-99, але вона є в багатьох комерційних СУБД, зокрема в Ms SQL. Замість слів BEFORE або AFTER ставиться вираз INSTEAD OF, тобто *замість*, що означає: код дії тригера повинен виконуватись замість тої операції, яка викликала спрацювання тригера.

Цей засіб вдало застосовується до подань, оскільки останні рідко дозволяють безпосередню модифікацію. Тригер INSTEAD OF замість зміни змісту подання виконує іншу операцію, доцільну в конкретному випадку.

Приклад 5.48. Знову звернімося до прикладу 5.26.

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

Це подання містить інформацію про всі фільми, випущені студією Paramount. При додаванні кортежів у відношення Movie туди не заносилось значення студії Paramount. Більш досконале рішення полягає у використанні тригера INSTEAD OF.

```
1)  CREATE TRIGGER ParamountInsert
2)  INSTEAD OF INSERT ON ParamountMovie
3)  REFERENCING NEW ROW AS NewRow
4)  FOR EACH ROW
5)  INSERT INTO Movie(title, year, studioName)
6)  VALUES(NewRow.title, NewRow.year, 'Paramount');
```

Рисунок 5.52. Приклад тригера INSTEAD OF.

У рядках 5, 6 у новий кортеж вставляються не 2, а 3 значення, в тому числі атрибуту studioName завжди присвоюється значення 'Paramount'.

Тригери у Transact SQL

Наведемо визначення синтаксису команди CREATE TRIGGER у Transact SQL.

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
   { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
      [ WITH APPEND ]
      [ NOT FOR REPLICATION ]
      AS
      [ { IF UPDATE ( column )
         [ { AND | OR } UPDATE ( column ) ]
            [ ...n ]
      | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
            { comparison_operator } column_bitmask [ ...n ]
      } ]
      sql_statement [ ...n ]
   }
}
```

Рисунок 5.54. Transact-SQL синтаксис команди CREATE TRIGGER у BOOKS ONLINE

Рядки	*Що вони містять*
CREATE TRIGGER *trigger_name*	Заголовок, назва тригера
ON { *table* | *view* }	Ім'я таблиці або розрізу
[WITH ENCRYPTION]	Тригер зачиняється від інших користувачів і не реплікується
{ { FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] }	FOR еквівалентно AFTER, залишено для сумісності. Помилково пропущено слово [DELETE]. **deleted** and **inserted** – таблиці попередніх і нових даних.
[WITH APPEND]	Для сумісності з версією 6.5.
[NOT FOR REPLICATION]	Тригер не виконується при реплікації
[{ IF UPDATE (*column*) [{ AND | OR } UPDATE (*column*)] [...*n*]	Перевірка, чи оновлювались колонки *column*
| IF (COLUMNS_UPDATED ( ) { *bitwise_operator* } *updated_bitmask*) { *comparison_operator* } *column_bitmask* [...*n*]	Визначення, чи оновлювався набір колонок, за допомогою бітової маски
sql_statement [...*n*]	Умови і дії тригера як послідовність команд Transact-SQL **deleted** and **inserted** – таблиці попередніх і нових даних.

Усього опис з прикладами займає12 великих екранних сторінок.

Приклад 5.49. Використання тригера для нагадування за допомогою Email

```
USE pubs
1)      IF EXISTS (SELECT name FROM sysobjects
2)            WHERE name = 'reminder' AND type = 'TR')
3)         DROP TRIGGER reminder
```

```
4)       GO
5)       CREATE TRIGGER reminder
6)       ON titles
7)       FOR INSERT, UPDATE, DELETE
8)       AS
9)          EXEC master..xp_sendmail 'MaryM',
10)            'Don''t forget to print a report for the distributors.'
11)      GO
```

Рисунок 5.55. Використання тригера для нагадування за допомогою Email

У цьому прикладі рядки 1-4 забезпечують видалення попередньої версії тригера. Будь-які зміни відношення titles (рядки 6,7) тягнуть відсилання Email адресату `'MaryM'` з нагадуванням про необхідність роздрукувати звіт.

`xp_sendmail` відсилає вказаному адресату листа та приєднує результати виконання запиту. Опис `xp_sendmail` займає 4 великих екранних сторінки.

Команда GO (рядки 4, 11) формує пакет (від GO до GO). Пакет виконується по одному плану виконання, хоча не є транзакцією.

Приклад 5.50[1]. Нехай відношення Books містить перелік книг, а відношення BookOrders фіксує факти продажу книг. Якщо книга продана, у відношення BookOrders додається кортеж, і тоді у відношення Books в атрибуті sold треба поставити 1.

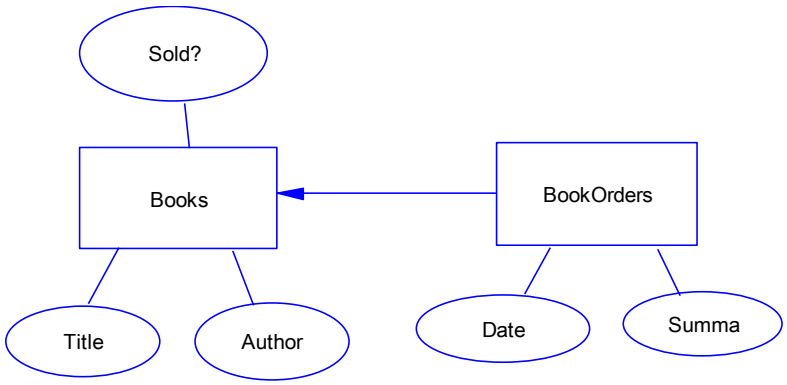

Рисунок 5.56. ER-діаграма бази продажу книг для прикладу 5.50

```
CREATE TRIGGER dbo.update_book_status
ON dbo.bookOrders
AFTER INSERT
AS
UPDATE Books
SET sold=1
WHERE titleId=
        (SELECT bo.titleId
                FROM bookOrders bo INNER JOIN inserted i
                ON bo.orderId=i.orderId)
```

Рисунок 5.57. Приклад тригера для оновлення іншого відношення в разі вставки.

[1] Проектирование и реализация баз данных Microsoft SQL Server 2000. Учебный курс Microsoft / Пер. с англ. – 3-е изд. – СПб.: Питер, 2006. – 512стр. – стр. 293

В передостанньому рядку використане відношення inserted, по якому ми визначаємо Id щойно проданої книги, та оновлюємо відповідний кортеж у Books. Напряму звернутись до inserted не можна, треба його з’єднати з іншим відношенням.

Створення запитів мовами реляційної алгебри і SQL

Методика створення запитів мовою РА

1. Зазвичай запит спочатку **формулюється природною мовою**. Це робить кінцевий користувач.
2. Перш за все треба **проаналізувати запит на кванторіальність (тобто наявності умов типу «для всіх» або «існує»)**.
   Про кванторіальність запиту свідчать слова «існує», «тільки», «лише» та подібні їм за змістом. Кванторіальні запити в РА реалізуються за допомогою теоретико-множинної операції віднімання.

   Наприклад: Нехай у предметній області складального виробництва відомо, з яких деталей складається кожний виріб. Зрозуміло, що однакові деталі можуть входити в різні вироби. Треба скласти запит: *Які деталі не застосовуються в виробі X?*
   Інше формулювання: *Лише ті деталі, які не існують у специфікації виробу X.*
   = ~~Деталі, які застосовуються в інших виробах, крім X~~ – невірно.
   =(Всі деталі) – (Деталі, які застосовуються в виробі X) – вірно.
3. Сформулювати кожний текстовий вираз у вигляді реляційного оператору:
   – Якщо запит є кванторіальним, то окремо побудувати лівий та правий **операнди віднімання**. Ці операнди звичайно виявляються некванторіальними. В нашому прикладі треба побудувати вирази для (*Всі деталі*) і (*Деталі, які застосовуються в виробі X*).
   – Якщо запит не є кванторіальним, то треба вибрати таблиці, в яких містяться реквізити структури відповіді та проаналізувати **характер сполучення** (природне, по умові, відкрите).
   – Якщо в запиті є **умова відбору над полями**, що не містяться в таблицях, вибраних в попередньому пункті, то додати в запит відповідні таблиці.
   – Якщо **в умові селекції міститься зв'язка «та»**, в якій певне поле має більше одного значення, то окремо формуємо праве та ліве відношення перетинання. Наприклад: «Які деталі виробляються в цеху «X» та в цеху «У»?»
   – Якщо вибрані таблиці прямо не сполучаються (не мають спільних полів у своїх схемах), то **додати таблиці,** через які це сполучення можливе.
   – Якщо запит містить агреговані значення, то **застосувати оператори групування та агрегування**.
   – Якщо запит містить **приписи щодо сортування** отриманих кортежів, то застосувати оператор сортування.

Розглянемо застосування даної методики на прикладах.

Аби зосередитись на побудові запитів, у наведених прикладах будемо мати справу із готовими БД різних предметних областей.

Приклад 1.

Предметна область техніко-економічного планування для обробного виробництва

	НОР-МА	ПЛАН	ЗАПАС	?ПОТ-РЕБА	?ДЕФІ-ЦИТ	?СОБІ-ВАРТ
Виріб	*к	*к				*
Матеріал	*к		*к	*	*	
НормаВитрат	1					
Ціна		1	1			
ПланВипуску		1				
Запас			1			
ПотребаМатеріалів				1		
ДефіцитМатеріалів					1	
СобівартістьПоМатеріалах						1

Тут і далі предметна область буде представлена в скороченому табличному вигляді, як в таблиці вище. В колонках наведені відношення (іншими словами - основні показники, які розглядаються), в рядках – можливі атрибути відношень. Для кожного відношення перелік атрибутів визначається його колонкою з такими позначеннями:

*к – ключовий атрибут, вимір

1 – неключовий атрибут, міра, частіше це число.

Якщо в таблиці не визначено ключові та неключові атрибути, то у відповідних атрибутах буде проставлено зірочки *.

Запити:

1.Потреба в матеріалах на план випуску:

$\pi_{Мат, \rightarrow ПотребаМатріалів}(\gamma_{Мат,SUM(НормаВитрат*ПланВипуску)}(НОРМА \blacktriangleright\blacktriangleleft ПЛАН))$;

Цей запит будується з кінця.

Спершу бачимо, що дані про потребу в матеріалах вираховуються із даних про план випуску (таблиця ПЛАН) та про норму витрат матеріалів на одиницю виробу (таблиця НОРМА).

Сполучаємо ці таблиці природним сполученням, оскільки відсутність кортежу з нормою тут вважається ознакою відсутності витрат.

Групуємо отримані кортежі по матеріалах.

Агрегуємо витрати матеріалів та перейменовуємо результат.

Відповідь отримано.

Запит на SQL:

```
SELECT Матеріал, SUM(НОРМА.НормаВитрат * ПЛАН.ПланВипуску)
FROM НОРМА, ПЛАН
WHERE НОРМА.Виріб=ПЛАН.Виріб
GROUP BY Матеріал
```

2. Дефіцит матеріалів на план випуску

Дефіцит матеріалів розуміємо як потреба мінус запас.

$\pi_{Мат, (ПотребаМатеріалів- NVL(Запас,0))\rightarrow ДефіцитМатеріалів}(ЗАПАС \blacktriangleright\blacktriangleleft R)$

$R = \pi_{Мат,SUM(Норма витрат*ПланВипуску)\rightarrow ПотребаМатріалів}(\gamma_{Мат}(НОРМА \blacktriangleright\blacktriangleleft ПЛАН))$;

В основі цього запиту лежить попередній, який сполучається природно із даними про запас. Якщо дані про запас матеріалу відсутні, то відкрите сполучення повертає значення «NULL», яке за допомогою функції NVL буде замінено на значення нуль.

Запит на SQL:

```
SELECT P.ПотребаМат - NVL(Z.Запас, 0)
FROM ЗАПАС Z RIGHT JOIN (
SELECT Матеріал, SUM(НОРМА.НормаВитрат * ПЛАН.ПланВипуску) AS
ПотребаМат
FROM НОРМА, ПЛАН
WHERE НОРМА.Виріб=ПЛАН.Виріб
GROUP BY Матеріал) P
ON Z.Матеріал=P.Матеріал
```

3.Собівартість виробів по статті «Витрати матеріалів».

$\pi_{\text{Виріб,SUM(Норма витрат*Ціна)}\rightarrow\text{СобівартістьПоМатріалах}}$
$(\gamma_{\text{Мат}}(\text{НОРМА} \blacktriangleright\blacktriangleleft \text{ЗАПАС}));$

Запит на SQL:

```
SELECT НОРМА.Виріб, SUM(НОРМА.НормаВитрат * ЗАПАС.Ціна)
FROM НОРМА, ЗАПАС
WHERE НОРМА.Матеріал = ЗАПАС.Матеріал
GROUP BY НОРМА.Виріб
```

Оскільки залишки і номенклатура матеріалів тут суміщені, ми виходимо з того, що в разі відсутності залишку в наявності він присутній у відношенні залишків з кількістю 0.

4. Собівартість виробів, що не використовують матеріал «мідь» по статті «витрати матеріалів».

$\pi_{\text{Виріб,SUM(Норма витрат*Ціна)}\rightarrow\text{СобівартістьПоМатріалах}}$
$(\gamma_{\text{Вироб}}(\sigma_{\text{Матеріал= «Мідь»}}(\text{НОРМА} \blacktriangleright\blacktriangleleft \text{ЗАПАС})));$

Цей запит має приховано кванторіальний характер, оскільки його можна переформулювати «по виробах, для яких не існує кортежу з міддю». Цей запит є не релевантним, оскільки рахує собівартість всіх виробів по всіх матеріалах, крім міді. Тобто собівартість без врахування міді.

$\pi_{\text{Вироб,SUM(Норма витрат*Ціна)}\rightarrow\text{СобівартістьПоМатріалах}}$

$(\gamma_{\text{Виріб}}($

$(\pi_{\text{Виріб}}(\text{НОРМА})\backslash\pi_{\text{Виріб}}(\sigma_{\text{Матеріал= «Мідь»}}(\text{НОРМА}))\ ) \blacktriangleright\blacktriangleleft \text{НОРМА} \blacktriangleright\blacktriangleleft \text{ЗАПАС})$

$));$

Цей запит є релевантним, оскільки перший аргумент відкритого сполучення містить список всіх виробів, які не використовують мідь.

Запит на SQL:

```
SELECT НОРМА.Виріб, SUM(НОРМА.НормаВитрат * ЗАПАС.Ціна)
FROM НОРМА, ЗАПАС
WHERE НОРМА.Матеріал= ЗАПАС.Матеріал
        AND НОРМА.Виріб NOT IN (
                SELECT Виріб FROM НОРМА WHERE Матеріал='Мідь')
```

GROUP BY НОРМА.Виріб

Приклад 2.

Предметна область техніко-економічного планування збирального виробництва.

	НОРМА	ПЛАН	ЗАПАС	?	ПОТРЕБА і ДЕФІЦИТ
Виріб («ціле»)	*к	*к	*к	*	
КомплектВиріб («частина», деталь)	*к				*
НормаВитрат	1				
Ціна			1		
ПланВипуску		1			
Запас			1		
ПотребаДеталей					1
ДефіцитДеталей					1
СобівартістьПоМатеріалах				1	

Коментар: *Відношення НОРМА побудовано на абстрактному відношенні «частина-ціле», яке, як відомо, має властивість транзитивності. Транзитивне замикання є рекурсивною операцією, що в реляційній алгебрі не забезпечена, тому спростимо ситуацію, обмеживши відношення двома рівнями «Кінцеві вироби – Деталі». Тобто комплектуючі вироби – це деталі, або матеріали.*

1.Потреба в деталях на план випуску:

$\pi_{\text{Деталь,SUM(ПланВипуску * НормаВитратДеталей)} \rightarrow \text{ПотребаМатріалів}}$ (
$\gamma_{\text{Деталь}}$ (ПЛАН► ◄$\pi_{\text{КомплектВироб}\rightarrow\text{Деталь, НормаВитрат}\rightarrow\text{НормаВитратДеталей}}$ НОРМА));

Запит на SQL:
SELECT НДЕТ.КомплектВиріб AS Деталь,
SUM(НОРМА.НормаВитрат*ПЛАН.ПланВипуску) AS Потреба
FROM ПЛАН, НОРМА НДЕТ
WHERE ПЛАН.Виріб=НДЕТ.Виріб
GROUP BY НДЕТ.КомплектВиріб

3.Вартість дефіциту деталей:

Коментар: Доцільно додати операцію селекції за знаком дефіциту, щоб уникнути компенсації дефіциту профіцитом, якщо це правильно з точки зору бізнесу.

Із врахуванням зазначеного запит матиме вигляд:

$\pi_{\text{SUM((ПотребаДеталей – NVL(Запас,0)*Ціна)}}$(
$\sigma_{\text{NVL(Запас,0)< Потреба деталей}}$(
ЗАПАС ►°◄ $\pi_{\text{Деталь,SUM(ПланВипуску * НормаВитратДеталей)} \rightarrow \text{ПотребаДеталей}}$ (
$\gamma_{\text{Деталь}}$ (
ПЛАН► ◄$\pi_{\text{КомплектВироб}\rightarrow\text{Деталь, НормаВитрат}\rightarrow\text{НормаВитратДеталей}}$ НОРМА))
));

Запит на SQL:
SELECT SUM((ПОТР.Потреба – NVL(ЗАП.Запас))* ЗАП.Ціна)
FROM ЗАПАС ЗАП,

```
RIGHT JOIN (
SELECT НДЕТ.КомлектВиріб AS Деталь,
SUM(НОРМА.НормаВитрат*ПЛАН.ПланВипуску) AS Потреба
FROM ПЛАН, НОРМА НДЕТ
WHERE ПЛАН.Виріб=НДЕТ.Виріб) ПОТР
ON ЗАП.Виріб=ПОТР.Деталь AND NVL(ЗАП.Запас)< ПОТР.Потреба
```

RIGHT JOIN визначає, що деталь вважається дефіцитною, якщо вона є в нормах, але її нема в запасах; якщо деталь є в запасах, але її нема в нормах, вона не дефіцитна.