Лекція10. Багаторівневі архітектури баз даних

План.

1. Моделі «клієнт-сервер» в технології баз даних.
2. Мережі, багаторівнева архітектура

1. В наш час користувачі мають доступ до бази даних , в основному, через мережу. Паралельний доступ до бази даних кількох користувачів, навіть якщо БД розташована на одному комп'ютері, відповідає режиму розподіленого доступу до централізованої бази даних. Такі системи називаються системами ***розподіленої обробки даних***.

Якщо ж база даних розподілена по кількох комп'ютерах, і до неї мають доступ кілька користувачів через мережу, то ми маємо з паралельним доступом до розподіленої бази даних. Такі системи називаються ***системами розподілених баз даних.***

Технологія «клієнт-сервер» - технологія, що розділяє додаток-СУБД на дві частини: клієнтську (інтерактивний графічний інтерфейс, розташований на комп'ютері користувача) і сервер, власне здійснює управління даними, поділ інформації, адміністрування і безпека, що знаходиться на виділеному комп'ютері. Взаємодія «клієнт-сервер» здійснюється наступним чином: клієнтська частина програми формує запит до сервера баз даних, на якому виконуються всі команди, а результат виконання запиту відправляється клієнту для перегляду і використання. Ця технологія застосовується, коли розміри баз даних великі, коли великі розміри обчислювальної мережі, і продуктивність при обробці даних, що зберігаються не на комп'ютері користувача (у великому установі зазвичай має місце саме така ситуація). Якщо технологія «клієнт-сервер» на застосовується, то для обробки навіть кількох записів весь файл копіюється на комп'ютер користувача, а тільки потім обробляється. При цьому різко зростає навантаження на мережі, і знижується продуктивність праці багатьох співробітників.

Основний принцип технології «клієнт-сервер» у застосуванні до баз даних полягає в розподілі 5-ти основних функції стандартного додатку між «клієнтом» та сервером.:

- Функції введення та відображення даних(Presentation Logic):
- Прикладні функції, що визначають основні алгоритми розв'язування задач додатку(Business Logic):
- Функції обробки даних в середені додатку(Database Logic):
- Функції керування інформаційними ресурсами(Database Maneger System);
- Службові функції, що слугують роль зв'язкових між функціями перших чотирьох груп.

Дворівневі моделі.

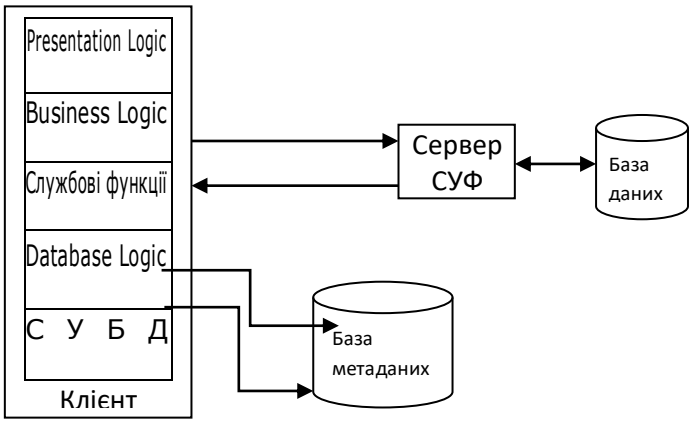
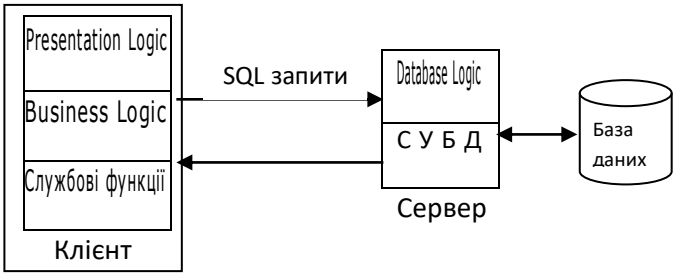


Рис.1. Модель файлового серверу

Переваги: ми маємо розподілення на 2 взаємоповязаних процеси, сервер може обслуговувати багато клієнтів.
Недоліки: високий мережевий трафік, вузький спектр операцій маніпулювання даними(файлові команди), відсутність адекватних засобів безпеки доступу до даних.



Модель віддаленого доступу до даних Рис.2.
Переваги: перенесення частини функцій на сервер розгружує його,, різко зменшується завантаження мережі, процесор виконує властиві йому функції обробки даних.
Недоліки: при інтенсивній роботі клієнтів SQL-запити можуть загрузити мережу, надлишкове дублювання коду додатків для кожного клієнта, сервер відіграє пасивну роль, тому функції керування ресурсами відбуваються на клієнті.

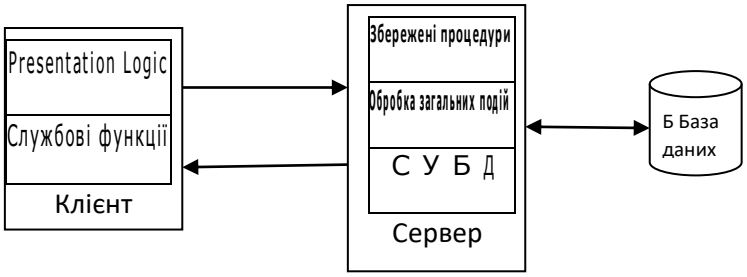


Рис.3.Модель серверу бази даних

Переваги:сервер є активним, збережені процедури та тригери зберігаються у словнику БД, що зменшує дублювання алгоритмів обробки даних для різних клієнтів..
Недолік: перевантаження сервера.

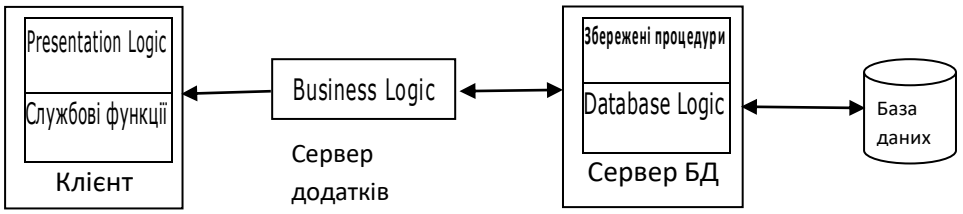


Рис.4. Модель серверу додатків

Ця модель є досить гнучкою, сервер додатків особливо важливий, якщо клієнти виконують велику кількість складних аналітичних розрахунків.

2. Мережі – сукупність комп'ютерів, що спілкуються між собою засобами стандартизованого протоколу. Розрізняють локальні, глобальні, відкриті та закриті мережі. Користувачі баз даних можуть отримати доступ до інформації засобами мобільного Інтернету, корпоративної або відкритої мережі.

В багатьох додатках баз даних, що застосовують Інтернет-технології, використовується трирівнева архітектура, що зображена на малюнку:

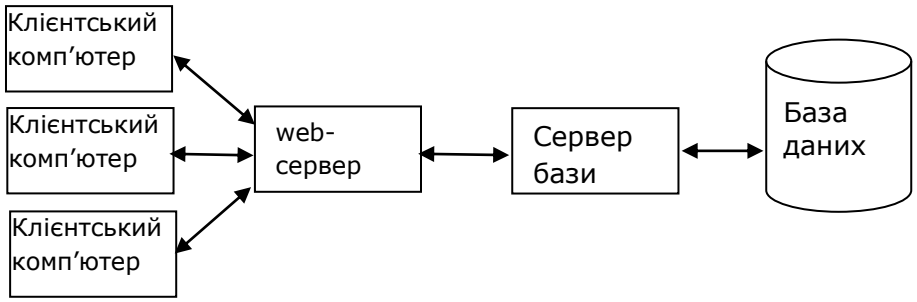


Рис.5. Трирівнева архітектура

Три рівня, або три типи процесорів – сервер бази даних, web-сервер, та клієнтський комп'ютер. Кожний з цих рівнів може працювати під керуванням своєї операційної системи. Наприклад, сервер БД – під керівництвом Unix, web-сервер – Windows NT, клієнтський комп'ютер – Unix. Windows, Macintosh. На сервері баз даних може працювати будь-яка СКБД. Microsoft Access, Microsoft Visual FoxPro, Microsoft Visual Basic забезпечують засоби для створення клієнтських частин у додатках «клієнт-сервер», які поєднують в собі засоби перегляду, графічний інтерфейс і засоби побудови запитів, а Microsoft SQL Server є на сьогоднішній день одним з найпотужніших серверів баз даних.

Найбільш розповсюдженими web-серверами є Netscape Navigator, Apache. Через інтерфейс між web-сервером і сервером БД передаються SQL-оператори та реляційні дані, між web-сервером та браузером – клієнтський код та дані. Функції сервера БД – обробка SQL, керування БД. Функції web-сервера – обробка представлень, функції клієнтів – матеріалізація представлень, генерує запити на сторінки, є HTTP – клієнтом, перетворює HTML чи іншу мову розмітки сторінок у вміст вікна на екрані. Оскільки браузерів багато, а сервер лише один, варто перекинути більше роботи на клієнт. В

деяких додатках трирівнева архітектура покладає на web-сервер величезні навантаження, web-сервер працює не лише як HTTP-сервер, який може відповідати на тисячі запитів, але ще й керувати з'єднаннями з сервером БД, генерує і передає SQL-запити, формує з результатів запитів представлення, забезпечує Дотримання ділового регламенту. При такому навантаженні web-сервер може сповільнювати роботу. Можна використовувати web-сервер тільки для обробки HTTP- запитів, а обробку запитів та представлень використовувати додаткові сервери. В разі розподіленої БД, обробка запитів може вестись кількома серверами БД, один сервер - обробляє розподілені транзакції, другій – обробку даних і повідомлень.

Для відображення даних на Web-серверах використовується мова розмітки HTML та DHTML. Паралельно з DHTML Microsoft була розроблено набір елементів керування ActiveX під назвою Remote Data Services – служба віддаленої обробки даних, за допомогою якої розробник може керувати дані на клієнтській машині, відображати, приймати, модифікувати інформацію, відправляти її назад на сервер. Але RDS придатний лише для представлень, що складаються лише з однієї таблиці. Для більш складних представлень використовується ADO. З допомогою DHTML можна запросити представлення з БД програмним способом, визначити кількість та імена стовпців і динамічно сконфігурувати web-сторінку, щоб відобразити дані.

XML(eXtensible Markup Language) – розширена мова розмітки , зручний тим, що розширює можливості матеріалізації документів на web-сторінках, зручний для передачі представлень з БД, використовується в якості стандарту для віддаленого виклику процедур. XML має значні переваги над HTML та DHTML. Автори XML забезпечили чіткий розподіл між структурою документа, його вмістом та матеріалізацією. Для кожної з цих операції передбачені окремі засоби., і природа їх така, що вони не можуть змішуватись, як це було в HTML. В XML ви не обмежені фіксованим набором елементів типу <title>, <h1>,<p>, ви можете визначати власні елементи Якщо в HTML тег <h2> можна використати як для формування заголовку другого рівня, так і просто для виведення тексти, то в XML структура документа строго визначена, точно відомо що означає будь-який тег і як він співвідноситься з іншими.

Значення XML для додатків баз даних :

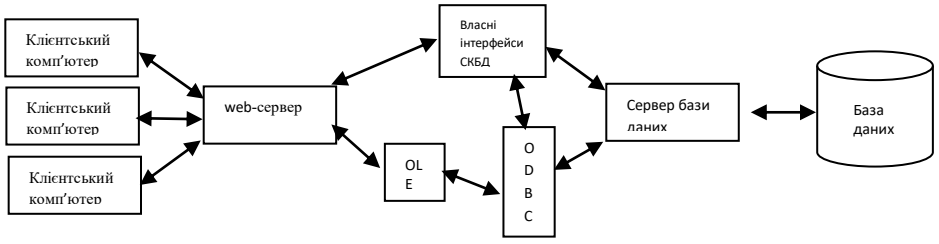
- Стандартизований спосіб представлення доменів;
- Стандартизований спосіб опису представлення даних;
- Чіткий розподіл структури, вмісту і матеріалізації;
- Можливість перевірки допустимості документів;
- Міжнародні стандарти типів документів.

З допомогою XML- технологій можна створювати різноманітні додатки БД, XML підтримуються в середовищі БД Oracle та SQL Server.

Оскільки у трирівневій присутні різноманітні складові і операційні системи, потрібні механізми, що будуть узгоджувати їх взаємодію та інтегрувати коди різних додатків між собою.
Такими компонентами є:

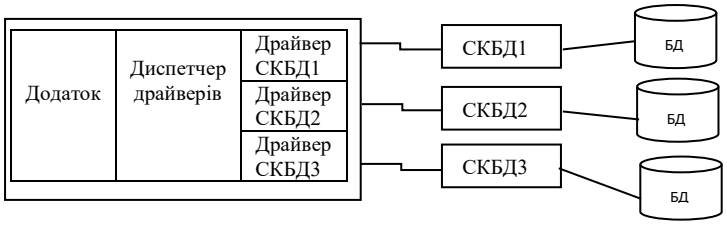
- OLE 2.0 (Object Linking and Embedding - зв'язування і впровадження об'єктів) - стандарт, що описує правила інтеграції прикладних програм. Застосовується для використання можливостей інших додатків. OLE 2.0 використовується для визначення та спільного використання об'єктів кількома додатками, які підтримують дану технологію. Наприклад, використання в середовищі Access таблиць Excel і його могутніх засобів побудови діаграм або використання даних, підготовлених Access, у звітах складених в редакторі текстів Word (зв'язування або включення об'єкта).
- OLE Automation (Автоматизація OLE) - компонент OLE, дозволяє програмним шляхом встановлювати властивості і задавати команди для об'єктів іншої програми. Дозволяє без необхідності виходу або переходу до іншого вікна використовувати можливості потрібного додатку. Додаток, що дозволяє іншим прикладним програмам використовувати свої об'єкти називається OLE сервером. Додаток, який може управляти об'єктами OLE серверів називається OLE контролер або OLE клієнт. З розглянутих програмних засобів як OLE серверів можуть виступати Microsoft Access, а також Microsoft Excel, Word і Graph ... Microsoft Visual FoxPro 3.0 і 5.0 може виступати тільки у вигляді OLE клієнта.
- RAD (Rapid Application Development - швидка розробка додатків) - підхід до розробки додатків, що передбачає широке використання готових компонентів і / або програм і пакетів (в тому числі від різних виробників).
- ODBC (Open Database Connectivity - відкритий доступ до баз даних) - технологія, що дозволяє використовувати бази даних, створені іншою програмою за допомогою SQL.

Розглянемо стандартні інтерфейси для доступу до серверу БД.



ODBC(Open Database Connectivity standard) –відкритий стандарт сумісності БД, був створений у 1990 році з метою надати незалежний від СКБД спосіб обробки інформації з реляційних БД. В середині 1990р. Microsoft представила OLE DB – об'єктна-орієнтований інтерфейс, що мав функціональність сервера БД, він був розроблений не лише для реляційних БД, але й для багатьох типів джерел даних, він безпосередньо доступний з C++, C# та Java. Але не доступний з Visual Basic.

Стандарт ODBC – це інтерфейс, з допомогою якого прикладні програми можуть звертатись до SQL-баз даних та обробляти їх незалежно від СКБД тобто додаток, що використовує інтерфейс ODBC може обробляти будь-яку БД сумісну з ODBC без будь-яких змін в програмному коді. Мета полягає в тому, щоб дати розробнику можливість створити один додаток здатний звертатись до різних СКБД, без необхідності змінювати його.



Прикладна програма, диспетчер драйвера, драйвери СКБД зшаходяться на тому комп'ютері де знаходиться web-сервер. Диспетчер драйвера слугує посередником між додатками та драйверами СКБД, також обробляє певні запти на ініціанізацію, та перевіряє допустимість формату і порядку запитів. Диспетчер драйвера входить до складу Windows. Недолік ODBC – щоб продукт вважався відповідним стандарту ODBC, виробник має створити драйвери майже для всіх функцій СКБД. Коли використовується OLE DB, виробник СКБД може реалізувати іх функції частинами, наприклад, тільки обробник запитів, створивши для нього інтерфейс OLE DB. В результаті продукт буде доступний для користувачів ADO. Об'єктна модель AGO є надбудовою над об'єктною моделлю OLE DB.

Лекція 9. Способи захисту інформації у базі даних

План

1. Безпека даних та привілеї.
2. Робота з системним каталогом.

Однією з основних функції СКБД є безпека даних. В мові SQL використовуються наступні принципи захисту даних:

- Маніпуляції з даними БД відбуваються від імені деякого користувача і СКБД може відмовитись виконати ті чи інші операції, в залежності від того , який користувач їх запрошує;
- Захист об’єктів бази даних можна здійснити засобами SQL одним користувачам дозволяється маніпуляція з об’єктом іншим –ні;
- В SQL існує система привілеїв користувачів.

SQL використовується звичайно в середовищах, де існує потреба розпізнавання користувачів і відмінності між різними користувацькими. Загалом кажучи, адміністратори баз даних, самі створюють користувачів і дають їм привілеї. З іншої сторони користувачі які створюють таблиці, є власниками таблиць і самі мають права на керування ними. Привілеї - це те, що визначає, може чи ні вказаний користувач виконати дану команду. Є кілька типів привілеїв, що відповідають декільком типам операцій. Привілеї даються й відміняються двома SQL-командами GRANT (ДОПУСК) і REVOKE (СКАСУВАННЯ). Нижче ми розглянемо, як ці команди використовуються

Кожен користувач у середовищі SQL, має спеціальне ідентифікаційне ім'я або номер - ідентифікатор (ID) доступу. ID доступу - це ім'я користувача, і SQL може використовувати спеціальне ключове слово USER, яке посилається до ідентифікатора доступу пов'язаного з поточною командою. Команда інтерпретується й дозволяється (або забороняється) на основі інформації пов'язаної з ідентифікатором доступу користувача, що подав команду.

У системах із багатьма користувачами, є певна процедура входу в систему, яку користувач повинен виконати щоб одержати доступ до об’єктів БД. Ця процедура визначає який ID доступу буде пов'язаний з поточним користувачем. Звичайно, кожна людина, що використовує базу даних повинна мати свій власний ID доступу й при реєстрації перетворюється в реального користувачам. Однак, часто користувачі, що мають багато задач можуть реєструватися під різними особистими ID доступу, або навпаки один ID доступу може використовуватися декількома користувачами. З точки зору SQL немає ніякої різниці між цими двома випадками; користувач асоціюється з ID доступу

SQL база даних може використовувати власну процедуру входу в систему, або вона може дозволити іншій програмі, типу операційної системи обробляти файл реєстрації й одержувати ID доступу із цієї програми.

Тим або іншим способом, але SQL буде мати ID доступу щоб зв'язати його з вашими діями, а для вас буде мати значення ключове слово USER.

Кожен користувач в SQL базі даних має набір привілеїв. Це - те що користувачу дозволяється робити.

Ці привілеї можуть змінюватися згодом - нові додаватися, старі віддалятися. Деякі із цих привілеїв визначені в ANSI SQL, але є й додаткові привілеї. Типи привілеїв можуть видозмінюватися в залежності системи яку ви використовуєте. Привілеї, які не є частиною стандарту SQL можуть використовувати подібний синтаксис, що повністю співпадаючий зі стандартом.

SQL привілеї, що визначенні ANSI - це привілеї об'єкта. Це означає що користувач має привілей, щоб виконати дану команду тільки на певному об'єкті в базі даних. Привілеї об'єкта зв'язані одночасно й з користувачами й з таблицями. Тобто, привілей дається певному користувачеві для вказаної базовій таблиці або представлення. Ви повинні пам'ятати, що користувач, що створив таблицю (будь-якого виду), є власником цієї таблиці. Це означає, що користувач має всі привілеї в цій таблиці й може передавати їх іншим користувачам для цієї таблиці.

Привілеї які можна призначити користувачеві наступні:

SELECT користувач, що має такий привілей може виконувати запити до таблиць. .

INSERT користувач, що має такий привілей може виконувати команду INSERT в таблиці.

UPDATE користувач, що має такий привілей може виконувати команду UPDATE в таблиці. Ви можете обмежити цю привілею до окремих стовпців**.**

DELETE користувач, що має такий привілей може виконувати команду DELETE в таблиці

REFERENCES користувач, що має такий привілей може визначати зовнішній ключ, який використовує один чи більше стовпців таблиці, як батьківський ключ, Ви можете обмежити цю привілею до окремих стовпців.

INDEX дає право створювати індекс до таблиці;

SYNONYM користувач, що має такий привілей може створювати синоніми для об’єктів;

ALTER дає право користувачу виконувати команду **ALTER TABLE**

Команда, за допомогою надаються привілеї виглядає наступним чином

GRANT <привілей> **ON** <обєкт> **TO** <користувач>**;**

Напириклад, якщо користувач SA володіє таблицею STUDENTі бажає надати користувачу SHER право виконувати запити до неї

GRANT SELECT ON Student TO Sher;

Тепер Sher може виконати запити до таблиці Student. Без інших привілеїв, він може тільки вибрати значення; але не може виконати будь-які дія, які б впливали на значення в таблиці Student (включаючи використання таблиці Student у якості батьківської таблиці зовнішнього ключа).

Коли SQL одержує команду GRANT, він перевіряє привілеї користувача, що задав цю команду, щоб визначити чи припустима команда GRANT. Sher самостійно не може генерувати цю команду. Він також не може надати право SELECT іншому користувачеві: таблиця ще належить SA (пізніше ми покажемо як SA може надати право Sher надавати SELECT іншим користувачам).

Часто виникає необхідність передавати кілька привілеїв кільком користувачам, тоді в команді їх відділяють комою один від одного

GRANT SELECT, INSERT ON Student TO Sher, Mag;

При цьому весь список привілеїв надається всім переліченим користувачам.

Усі привілеї об'єкта використовують один той же синтаксис, крім команд UPDATE і REGERNCES у яких можна вказувати імена стовпців. Привілей UPDATE можна надавати наприклад наступним чином:

GRANT UPDATE ON Student TO Sher;
GRANT UPDATE(STIP) ON Student TO Sher;
GRANT UPDATE(SFAM,STIP) ON Student TO Sher;

Допускаються списки з кількох полів, що вказуються в довільному порядку. При використанні привілеї REFERENCES правила ті самі, що й при UPDATE.

GRANT REFERENCES (SFAM,STIP) ON Student TO Sher;

Ця команда надає право Sher використовувати поля SFAM,STIP в якості первинних ключів, по відношенню до будь-яких зовнішніх ключів його таблиць. Природно, що привілей буде придатний лише для використання в тих полях, що мають обмеження, необхідні для первинних ключів.

SQL підтримує два аргументи для команди GRANT, які мають спеціальне значення: ALL (всі привілеї) і PUBLIC (на загал). ALL використовується замість імен привілеїв у команді GRANT щоб віддати всі привілеї в таблиці. Наприклад, SA може дати Sher весь набір привілеїв у таблиці Student за допомогою такої команди:

GRANT ALL ON Student TO Sher;

PUBLIC - більше схожий на тип аргументу - захопити всі (catch-all), чому на користувацький привілей.

Коли ви надаєте привілеї PUBLIC, усі користувачі автоматично їх одержують. Найбільше часто, це застосовується для SELECT у певних базових таблицях або представленнях, які ви прагнете зробити доступними для будь-якого користувача.

Щоб дозволити будь-якому користувачеві бачити таблицю Student, ви, наприклад, можете ввести наступне:

```
GRANT SELECT ON Student TO PUBLIC;
```

Звичайно, ви можете надати будь-які або всі привілеї будь-кому, але це очевидно небажане. Усі привілеї за винятком SELECT дозволяють користувачеві змінювати (або, у випадку REFERENCES, обмежити) вміст таблиці. Дозвіл усім користувачам змінювати зміст ваших таблиць викличе проблему.

Будь-який новий користувач, що додається до вашої системи, автоматично одержить усі привілеї призначені раніше всім(PUBLIC), тому якщо ви захочете обмежити доступ до таблиці всім, зараз або в майбутньому, краще
усього надати всі привілеї крім SELECT для певних користувачів.

Іноді виникає необхідність щоб інші користувачі могли надавати та відміняти привілеї до таблиць.. SQL дозволяє це заробити за допомогою команди **WITH GRANT OPTION.**

```
GRANT SELECT  ON Student TO Sher
  WITH GRANT OPTION;
```

Після того Sher одержав право передавати привілей SELECT nhtnsv-особам; він може видати команду

```
GRANT SELECT  ON Student TO Mag;
```

або

```
GRANT SELECT  ON Student TO Mag
 WITH GRANT OPTION;
```

Користувач за допомогою GRANT OPTION може, у свою чергу, надати цей привілей до тієї же таблиці, з або без GRANT OPTION, будь-якому іншому користувачеві. Це не міняє приналежності самої таблиці; як і колись таблиця приналежать тому хто її створив.

Відміна привілеїв. Відміна привілеїв здійснюється командою REVOKE
Синтаксис команди REVOKE - подібний на GRANT, але має зворотній зміст
Щоб відмінити привілей INSERT для Adrian в таблиці **PREDMET**, ви можете ввести

```
REVOKE INSERT ON PREDMET FROM Adrian;
```

Допустиме використання списків привілеїв і користувачів так само, як і в GRANT.

REVOKE INSERT, DELETE ON STUDENT
FROM Adrian, Stephen;

Проте, тут є деяка неясність. Хто має право відміняти привилегии? Коли користувач з правом передавати привелегии іншим, втрачає це право? Користувачі яким він надав ці привілеї, також їх втратять ? Оскільки це не стандартна особливість, немає ніяких авторитетных відповідей на ці питання, але найбільш загальний підхід - це такий: привілею відміняються користувачем який їх надав, і відміна буде каскадуватись, тобто вона буде автоматично розповсюджуватись на усіх користувачам тих, що отримали від нього.

<u>Використання представлень для фільтрації привілеї.</u>

Ви можете зробити дії привілеїв точнішими, використовуючи представления. Кожного разу, коли ви передаєте привілей у базовій таблиці користувачеві, вона автоматично поширюється на усі рядки, а при використанні можливих виключень UPDATE і REFERENCES, на усі стовпці таблиці.

Створюючи представлення, можна обмежити доступ до таблиці завдяки тим обмеженням, що задаються запитом, який міститься в представленні.

Щоб створювати представлення, ви повинні мати привілей SELECT в усій таблиці на які ви посилаєтеся в представленні. Якщо представлення є таким що дозволяє модифікацію, то будь-які привілеї INSERT, UPDATE, і DELETE автоматично передаються і представленню. Якщо ж ваші привелеї обмежені, тоі не дозволяють модифікувати таблиці, то ви не зможете це зробити. Навіть, якщо саме представлення дозволяє модифікацію. Оскільки зовнішні ключи не використовуютьсяв представленні, то й привілей **REFERENCES** на має змісту в цьому випадку.

Припустим, ви бажаете надати користувачеві Claire можливість бачити лище стовпці **sfam, stip** із таблиці **student,** то вам необхідно створити представлення

CREATE VIEW ST
AS SELECT sfam, stip
FROM student;

І надати користувачуClaire привілей SELECT в представленні, а не в таблиці.

GRANT SELECT On ST to Claire;

Для команди DELETE обмеження стовпців з допомогою представлень не має змісту, як не має змісту воно і для REFERENCES **та UPSATE.** Тому що обмеження на стовпці задані вже за змістом самих команд.

Зазвичай представлення створюють, щоб обмежити доступ до рядків таблиці. Наприклад, тільки інформація про певний іспит

```
CREATE VIEW NULLSTIP
  AS SELECT *
    FROM USPISH
    WHERE PNOM=111
    WITH CHECK OPTION;
```

Після чого можна надати привілеї типу UPDATE до цієї табдиці користувачу Adrian:

GRANT UPDATE ON NULLSTIP TO Adrian;

В цьому представленні ми маємо доступ до всіх полів таблиці, але лише з інформацією про заданий предмет. Фраза **WITH CHECK OPTION** забороняє заміну поля PNOM на будь-яке значення крім 111.

Ще одна можливість представлень – доступ надається не до полів а до даних, обчислених з допомогою агрегатних функцій

```
CREATE VIEW SBAL
AS SELECT pnom, Snom, AVG (ocinka)
FROM uspish
GROUP BY date;
```

:

GRANT SELECT ON SBAL TO Diane;

Щоб обмежити введення недопустимих даних в таблицю ви можете створити представлення наступтим чином

```
CREATE VIEW USP
  AS SELECT *
    FROM USPISH
      WHERE OCSNKA IN(2, 3, 4, 5)
     WITH CHECK OPTION;
```

Перевага такого підходу в тому, що для зміни обмежень на введення даних в таблицю легше використати представлення . яке можна легко знащити, на томість створити інше, ніж змінити обмеження **CHECK OPTION**, що накладене на базову таблицю.

Виникають питання: хто може створювати та корегувати таблиці, змінювати обмеження та створювати представлення. Або хто має найвищий статус в системі. Ви знаєте, що є особливий користувач *Администратор Бази Даних , супер користува, DBA* і привілеї, якими він вологіє називаються *привілеями системи.*

Загалом є три базлвих привілеї

- **CONNECT** (*Подключить*),
- **RESOURCE** (*Ресурс*),
- **DBA** **(***Аминистратор Базы Данных***)**

Простіше, можна сказати, що CONNECT складається з права зарегистрироваться і права створювати представлення та синоніми, привилегии объекта. RESOURCE складається з права створювати базові таблиці.

DBA - це привілей суперкористувача, що дає найбільш високі повноваження в базаі даних. Один або кілька користуачів можуть мати ці привілеї.

Деякі системи крім того мають спеціального користувача, іноді званого SYSADM або SYS (Системний Адміністратор Бази Даних), який має найвищі повноваження; це - спеціальне ім'я, а не просто користувач із спеціальним DBA привілеєм. Відмінність дуже тонка і функціонує по різному в різних системах. Для наших цілей, ми посилатимемося на високопривілейованого користувача, який розробляє і управляет базою даних маючи повноваження DBA
:

GRANT RESOURCE TO Rodriguez;

Природно з'являється питання, звідки візьметься користувач з ім'ям Rodriguez? Як визначити його ID допуску? У більшості реалізацій, DBA створює користувача, автоматично надаючи йому привілей CONNECT. В цьому випадку, зазвичай додається речення IDENTIFIED BY, що вказує пароль. (Якщо ж ні, операційна система повинна визначити, чи можете ви зареєструватися у базі даних з цим ID доступу) DBA може, наприклад, ввести

GRANT CONNECT TO Thelonius IDENTIFIED BY Redwagon;

що приведе до створення користувача, з ім'ям Thelonius, дасть йому прав реєструватися, і призначить йому пароль Redwagon, і усе це в одній пропозиції.

REVOKE CONNECT on Thelonius

Якщо ви зробите спробу видалити привілей CONNECT користувача, який має їм созданые таблиці, команда буде відхилена, тому що її дія залишить таблицю без власника, а це не дозволяється. Ви повинні спочатку видалити усі таблиці створені цим користувачем, перш ніж видалити його привілей CONNECT.

Кожного разу, коли ви звертаєтесь в команді до таблиці чи представлення, що не належить вам, ви повинні вказувати префікс імені володаря цього об’єкту, щоб SQL знав де його шукати. Це не зовсім зручно, тому більшість реалізацій SQL дозволяє створювати синоніми (альтернативні імена, прізвиська). Коли ви створюєте синонім, ви є його власником. Якщо вимажте хочаб один привілей в одному чи більше стовпцях таблиці чи представлення, то можете створювати до нього синонім.

CREATE SYNONIM STUD FOR DANA.STUDENT

Якщо ви плануєте мати таблицю, що використовується багатьма користувачами, краще щоб вон посилались на неї з допомогою одного того самого імені. Для цього варто створити загальний синонім
CREATE PUBLIK SYNONIM STUD FOR STUDENT
DROP SYNONIM STUD – знищення синонімів

Використання системного каталогу. Щоб правильно працювала БД, СКБД повинна слідкувати за багатьма різноманітними об'єктами: таблицями, представленнями, синонімами індексами, привілеями, користувачами, тощо. Це може виконуватись різними способами, але найчастіше - шляхом збереження даних в системних таблицях. СКБД має можливість обробляти дані з цих таблиць тими самими засобами що й дані для певної прикладної задачі. Набір таблиць де зберігається службова інформація називають ***системним каталогом***, словником даних або службовими таблицями.

За своєю будовою вони нагадують самі звичайні таблиці з полями та записами, створюються і модифікуються вони як й інші таблиці даних, а ідентифікуються з допомогою спеціальних імен.

Типовий системний каталог може складатись із наступного списку:

SYSTEMCATALOG –інформація про базові таблиці і представлення
SYSTEMCOULUMNS – дані про поля таблиць
SYSTEMTABLE – дані про системні таблиці системного катлогу
SYSTEMINDEX – інформація про індекси
SYSTEMUSERAUTH – дані про користувачів БД
SYSTEMTABAUTH – дані про об'єктні привілеї користувачів
SYSTEMCOLAUTH – дані про привілеї в полях таблиць
SYSTEMSYNONS – синоніми таблиць

Дозволити користувачам бачити тільки свої власні об'єкти. Але для цього потрібно спочатку створити представлення. А потім можна надати привілей SELECT до цього представлення всім користувачам. Таким самим чином можна дозволити користувачам переглядати таблицю SYSTEMCOULUMNS для полів власних таблиць.

В різних СКБД кількість системних таблиць та назві їх полів можуть відрізнятись але сама ідея при цьому зберігається.

Більшість версій SQL дозволяє розташовувати коментарі в спеціальні стовпці таблиць SYSTEMCOULUMNS та SYSTEMCATALOG. Це зручно, бо таблиці не завжди можуть пояснити свій зміст

COMMENT ON TABLE SHER.TEACHER IS 'Це список викладачів '

Цей текст буде розташовано у спеціальну колонку таблиці SYSTEMCATALOG . Сам коментар – в службову таблицю що містить інформацію про таблицю TEACHER.

Технологія фізичного зберігання та доступу до даних

План

1. Основні етапи доступу до даних.
2. Поняття індексу, застосування індексів
3. Процедури індексування та хешування
4. Створення індексів

1.Для збереження даних не існує ідеального способу. Тому СКБД повинна мати кілька структур збереження даних для різних задач та частин системи. Процес пошуку даних полягає в наступному: спочатку в ОП комп'ютера читається частина записів з допомогою диспетчера файлів. Диспетчер файлів визначає сторінку на який розташований потрібний запис,, для читання цієї сторінки використовується диспетчер дисків, він визначає фізичне розташування сторінки і посилає до ОС запит на введення-виведення даних.

Основні функції диспетчера дисків – приховати від диспетчера файлів всі деталі фізичних дискових операцій введення-виведення даних. І замінити їх логічними операціями зі сторінками.

На початковому етапі БД не містить даних, але в ній є набір пустих сторінок. Для розташування записів з даними X ДФ створить набір сторінок та розташує на них дані X, подібним чином будуть створені набори сторінок для даних Y, та Z. Отже буде створено 4 набори сторінок X,Y,Z, та набір порожніх сторінок. При додаванні записів до набору X буде ДФ витягне нову пусту сторінку та додасть її до набору, при знищення, звільнить і перенесе у набір вільних сторінок. Ті ж самі дії будуть виконані для наборів Y,Z. Отже немає гарантії, що логічно зв'язані сторінки будуть фізично розташовані поруч. Тому послідовність сторінок кожного набору вказують з допомогою вказівників. Заголовок кожної сторінки містить дані про фізичну адресу наступної логічної сторінки. Для визначення інформації про розташування різних наборів сторінок диспетчер дисків організує окрему сторінку з номером 0, яку часто називають таблицею розташування або сторінкою 0, де перелічені всі набори сторінок разом із вказівниками на першу сторінку кожного набору.

Якщо при знищенні записів на сторінці утворились пусті місця то ДФ перемістить всі записи на початок сторінки. Таким чином кожний запис ідентифікується за номером запису та місцем розташування його по відношенню до кінця сторінки. При зміщенні запису корегується лише місце розташування на сторінці номер запису залишається сталим, отже доступ по номеру запису є досить швидким. Якщо запис не поміщається на сторінці, то частина його розташовується на так званій сторінці переповнення та ідентифікаційний номер запису буде змінений. Для деякого файлу завжди можна здійснити послідовний доступ до записів, які там зберігаються.

Та крім послідовного доступу існують інші способи : хешування, індексування, використання ланцюжка вказівників технології стискання.

2. Дані, що зберігаються в БД невпорядковані, і СКБД все одно де і як вони

зберігаються. Два записи, що вводились одночасно можуть бути розташовані на різних сторінках БД. За таких умова, якщо пошук по базі даних виконувати простим перебором, то це може займати багато часу і ресурсів ПК. Тому для прискорення пошуку застосовують такий механізм як індексування.

Індекс - це впорядкований вказівник на записи деякої таблиці. Вказівник означає, що індекс містить значення одного або декількох полів таблиці, а також адресу сторінки де ці записи зберігаються. Тобто індекс складається з двох части: „значення поля ” + „фізичне місце розташування ”. Основна функція індексів забезпечити швидкий пошук записів в таблиці.

Індекси використовуються:

- Для прискорення виконання запиту. Індекси створюються для полів, які використовуються в умовах відбору SQL-запитів.
- Забезпечення унікальності значень поля первинного ключа. При додаванні нового запису виконується пошук по індексному файлу, що створений для поля первинного ключа, якщо не знайдено такого значення, то додається нове значення поля первинного ключа.

Часто виникає питання: чому б не створити індекси для всіх полів таблиці раз вини прискорюють виконання запитів. Але надмірне створення індексів запирає дисковий простір. Крім того при модифікації таблиць(вставка/знищення записів) відбувається перебудова індексів, що приводить до збільшення часу на виконання запиту до таблиці з індексами ніж до таблиці без індексів.

Існує ще так зване привило 20%:якщо запит на вибірку повертає більш ніж 20% даних з таблиці, то використання індексів може сповільнити вибірку.

Ще один момент використання індексів пов'язаний з оптимізатором запитів. Оптимізатор – це сукупність механізмів, які будують план виконання запиту. Коли користувач формує запит до БД, він вказує, що повинно бути виконано але не вказує як це зробити. На основі переданого запиту оптимізатор будує план виконання. Коли аналізуються умови на вибірку типу WHERE, ORDER BY оптимізатор намагається використовувати індекси. Нажаль це не завжди ті індекси, що є найбільш ефективними для даного запиту. Що також сповільнює час виконання запиту.

Третім випадком, коли не потрібно використовувати індекси є поля з обмеженим набором даних. Наприклад, якщо поле приймає значення «ч» і «ж» немає ніякого змісти використовувати індекс..

Три основних випадки, коли використання індексу прискорює виконання запиту:

- Коли це поле використовується в умовах пошуку в запиті;
- Коли з’єднання таблиць(join) використовує це поле;
- Коли це поле використовує фразуORDER BY.

3. Забезпечення цілісності посилань – при додаванні записів у пов’язані таблиці виконується перевірка чи існує відповідний запис у батьківській чи дочірній таблиці.

Часто виникає питання:чому б не створити індекси для всіх полів таблиці? Відповідь очевидна – це приведе до надмірного завантаження пом’яті ком’ютера та сповільненню виконання запитів. Так, наприклад, при додаванні/віднімані

записів в таблиці, серверу приходиться перебудовувати індекси. Це дві основні причини, що перешкоджають загальній індексації.

Та є ще 2 зауваження. Перше – якщо запит відбирає 20% всіх записів таблиці, то існуючі індекси затримують виконання запит у.

3. Спробуємо розібратись з індексами. Припустимо нам потрібно вибрати інформацію про студентів, зо здали іспит з певного предмету. Запит заснований на інформації, що зберігається в 2-х файлах, які можуть зберігатись на різних сторінках. Припустимо, що у файлі предметів використовується список предметів, впорядкований за алфавітом.

PN(індекс)
Алгебра
Мат.аналіз
Програмування
ЧММФ

SNOM	PN	NAME	OCINKA
121	Мат.аналіз	Поляков	5
122	Алгебра	Козак	4
123	ЧММФ	Вус	4
124	Програмування	Гриценко	3
125	ЧММФ	Хомяк	3

Можливі два способи пошуку інформації про студентів, що здали іспит з ЧММФ.:

- Знайти файл записів про успішність і вибрати з нього всі, в якому є рядок з ЧММФ;
- Знайти файл з назвами предметів, в ньому ЧММФ, та за вказівниками знайти всі записи з файлу про успішність. Якщо доля всіх студентів що здали ЧММФ по відношенню до всіх інших записів невелика , то другу стратегія буде кращою, тому що пошук у файлі предметів буде швидшою, а потім і пошук по вказівникам пройде швидко.

Файл предметів називають ***індексом*** по відношенню до успішності, а файл успішності індексованим по предметам. Індексний файл є файлом особливого типу,: кожен запис складається з 2-х частин даних та вказівника на номер запису з такими даними. Якщо індекс заснований на полі первинного ключа він називається первинним, якщо на іншому полі – вторинним(в нашому прикладі - вторинний)..Індекс заснований на ключовому полі називається ***унікальним***. Недоліком використання індексів є сповільнення процесу додавання нових записів, адже при додаванні нового запису потрібно оновлювати ще й файл індекса, додаючи до нього запис новий вказівник.

Кожен файл може мати кілька індексів. Наприклад : успішність проіндексована за предметами та оцінками. Для пошуку записів потрібно буде здійснити 2 пошуки. Часто індекси будують на основі кількох полів. В такому випадку пошук записів можна здійснити за одним переглядом.

Отже, основною метою використання індексів є прискорення процесу пошуку зва рахунок зменшення операцій введення-виведення. Якщо індекс створений на основі первинного ключа, то не маж необхідності в індексному файлі зберігати вказівники на всі вказані записи, можна лише вказати максимальне значення ключа на сторінці та номер сторінки. Індекс з описаною структурою називається ***нещільним.*** Перевага нещільного індексу в тому, що розмір його невеликий, тож і пошук ведеться швидше. Недолік – не модна здійснити перевірку деякого значення.

Іншим способом впорядкування даних є хешування. Хешування називається процедура швидкого прямого доступу до записів на основі заданого значення деякого поля, при чому не обов'язково, що це поле було ключовим. Особливості цієї технології наступні:

- Кожен запис в БД зберігається за адресом (хеш-адресом), що розрахована на основі спеціальної хеш-функції, що заснована на значенні деякого поля(хеш-поля) даного запису;
- Спочатку розраховується адреса, а тоді ДФ розташовує за заданою адресою вказаний запис;
- Для пошуку запишу теж спочатку обчислюється хеш-адреса, а тоді ДФ посилає запит на читання потрібного запису.

Найпростіший приклад хеш-функції:

Хеш-адреса\ = залишок від ділення значення хеш-поля на просте натуральне число.

Теоретично для визначення хеш-адреси можна використовувати значення самого ключового поля, але практично діапазон значень ключових полів, може бути значно ширшим за діапазон можливих адрес. Отже, потрібно знайти таку хеш –функцію, що звузити діапазон до оптимальної величини з врахуванням можливості резервування додаткового простору.

Недоліками хешування є

1. фізична послідовність записів в середині збереженого файлу майже завжди відрізняється від послідовності ключового поля, або будь-якої іншої логічної послідовності, і між послідовно розміщеними записами можуть бути проміжки невизначеного розміру.
2. Часто може виникнути ситуація що кільком записам відповідає однакове значення хеш-функції, тоді хеш-функцію потрібно виправляти. Можна скористатись методом прямого перебору. Припустимо на деякий пустій сторінці можна розмістити записів, що мають однакову хеш-адресу. Пошук в межах сторінки можна вести простим перебором. Але n+1 запис потрібно вже буде розташовувати на додатковій сторінці переповнення. При збільшенні розмірів файлу, збільшується і час пошуку та ймовірність спів падіння значень хеш-функції. Цю проблему можна вирішити шляхом перебудови хеш-функції та завантаження файлу з розширеною хеш-функцією. Щоб всі значення хеш-функції були унікальними можна використати в якості хеш-поля ключове поле..

Основні принципи методу розширеного хешування наступні:

- Якщо в якості хеш-функції використовується функція φ, а значення деякого ключового поля z дорівнює p, то значення псевдоключа буде φ(p). Псевдоключ - вказівник на місце збереження запису.:
- Файл,що зберігається буде містити пов'язаний з ним каталог. Що складається із заголовка який містить значення g –глибина каталогу,

та 2^g на сторінки з кількома записами на них.

Тут був описаний лише один спосіб побудови хеш-функцій, їх є безліч.

Для виконання запитів можна використовувати і не менш ефективний спосіб ланцюжків вказівників. Обидва файли(файл вказівників та файл даних) при цьому можуть знаходитись в одному наборі. Файл вказівників називають батьківським, а записів –дочірнім.

Розглянемо вищезгаданий приклад

PN(індекс)
Алгебра
Мат.аналіз
Програмування
ЧММФ

SNOM	NAME	OCINKA
121	Поляков	5
122	Козак	4
123	Вус	4
124	Гриценко	3
125	Хомяк	3

В дочірньому файлі відсутнє поле з назвою предмета, але є вказівники на пов’язані записи, тобто на тих хто отримав оцінки по однакових предметах.

Переваги такої структури: простіше виконання операції вставки та знищення та менший розмір файлів на диску, ніж займає відповідна індексна структура. Але якщо провести запит на пошук предмету ,що здали студенти, то така структура не буде зручною, тут більше підійде індексна або хеш структура. Та якщо батьківська структура має велику кількість записів, то для неї теж прийдеться застосовувати індексну, або хеш структуру.. Вдосконалення батьківської структури можна було б провести за рахунок додавання нового вказівника в дочірній частині на пов'язаний запис в батьківський структурі. тоді можна уникнути перегляду всіх записів при пошуку.

3. Процедура стискання економить не лише місце на диску але й кількість операцій введення-виведення, тому що доступ до даних меншого розміру потребує меншої кількості операцій введення-виведення. Найбільш розповсюдженою технологією є технологія заснована на різницях, при який деяке значення заміняється на даними про його відмінності від попереднього. При такий технології дані потрібно зберігати послідовно і поруч, тому що при їх розпакуванні потрібно мати значення попередньої величини. Такий вид стискання ефективний для списків(вони зберігаються послідовно), в таких випадках вдається стиснути навіть вказівники. Один із способів стискання на основі різниці – видалення символів, що повторюються на початку запису, із виказанням їх кількості, тобто переднє стискання

Студент	0 - Студент
Студентка	7- кА
Студентський	7-ський

Подібним чином виконується заднє стискання(видалення пробілів в кінці слова).

Інший спосіб = ієрархічне стискання: припустимо в деякому файлі. Завдяки

кластеризації виконаній по полю PNOM з назвами предмету, окремо містяться всі записи . про студентів, що здали цей предмет, назва предмету записана один раз. Тобто всі дані будуть стиснуті в один ієрархічний запис.: назва предмета, та всі дані про студентів, що здали цей предмет. Такий запис складається з 2-х частин: постійної з назвами предметів, і змінної(або групи повторень) – про студентів.

USPISH			
	SNOM	SNOM	OCINKA
Алгебра	3415	Котенко	5
Фізика	3412	Поляков	5
	3414	Грищенко	3
Хімія	3413	Старова	4

Подібне стискання можна використовувати для індексів, в яких кілька значень, що розташовані послідовно містять одинакові дані, але різні значення вказівника, або при між файловій кластеризації, коли записи про студента об'єднуються із записами про його оцінки по всіх предметах.

Створення індексів в SQL.

Індексом називають впорядкований список полів чи груп полів в таблиці, це корисний інструмент, що використовується у всіх сучасних СКБД. Коли створюється індекс БД запам'ятовує відповідний порядок всіх значень цього поля. Перевагою використання індексів є прискорення пошуку відповідних записів. Але вони мають і недоліки - сповільнення виконання операцій модифікації таблиць, займають місце на диску. Тобто при створенні індексу потрібно приймати зважене рішення створювати його чи ні. Індекси можуть складатись з кількох полів, тоді головним є перше поле, друге впорядковується в середині першого, третє – другого і т.д.

CREATE [UNIQUE] [ASC[ENDING]| [DESC[ENDING]]

INDEX <ім'я індексу> **ON** <ім'я таблиці> **(**<ім'я поля>**[,**< ім'я поля >**]…)**

Зрозуміло, що таблиця вже має бути в пам'яті комп'ютера та мати відповідні поля. Наприклад, в таблиці студент найбільш ймовірним полем індексу може бути поле прізвище

CREATE INDEX SFAMIND ON STUDENT(SFAM)

Для створення унікальних індексів (без повторень значень поля) використовується ключове слово UNIQUE. Практично такий індекс є первинним ключем таблиці, для таблиці студент таким індексом може бути SNOM

CREATE UNIQUE INDEX SNAMB ON STUDENT(SNOM)

Такий індекс не буде створений, якщо поле SNOM значення, що повторюються. Тому краще створювати індекси відразу після створення таблиці до введення в неї значень. При створенні первинного ключа на таблицю автоматично створюється унікальний індекс. Але не всякий унікальний індекс є

первинним ключем. Первинні та унікальні індекси використовуються для забезпечення цілісності посилань. При створенні зовнішнього ключа теж автоматично створюється індекс, який використовується для перевірки чи інує відповідне значення у пов'язаній таблиці.

Індекси реалізовані у вигляді двійкового дерева і коли в таблицю додаються нові записи в дерево додаються нові гілочки. Причому додавання відбувається не в «хвіст», а на кінцях інших гілочок і з часом дерево стає розбалансованим і пошук по ньому сповільнюється. Тому індекси потребують періодичної перебудови. Перебудову індексу послідовно виконують дві команди

ALTER INDEX <name> INACTIVE;

ALTER INDEX <name> ACTIVE;

INACTIVE переводить індекс в неактивний стан і дерево можна перебудувати.

ALTER INDEX – має кілька обмежень, з його допомогою не можна перебудувати індекси. що використовуються в первинних зовнішніх та унікальних ключах, якщо вини в даний момент використовуються в запитах. Також для перебудови індексу потрібно мати права SYSDBA.

Перебудова індексу командами

DROP INDEX <ім'я індексу>

CREATE INDEX<ім'я індексу>

Призводить до створення індексу з «чистої сторінки». Але знову ж таки дійсні обмеження вказані вище для ALTER INDEX.

Третім способом перебудови індексу є резервне копіювання бази даних. При цьому, дані що входять ів індекс не зберігаються в резервній копії. Зберігаються лише визначення індексів. При відновленні з резервної копії індекс перестроюється заново.

Четвертий спосіб покращити ефективність індексу – зібрати статистику по індексам командою

SET STATISTICS INDEX <name>:

Статистика це величина в межах від 0 до 1, значення якої залежить від числа неоднакових записів в таблиці. Перерахунок статистик не перебудовує індексів, тому вільний від більшості обмежень, згаданих вище, і дає оптимізатору можливість оптимізатору прийняти вірне рішенні про використання індексу. Перерахувати статистику може системний адміністратор, або той, хто створив індекс.

Якщо індекс є комбінацією кількох полів, то значення кожного з них може повторюватись, але комбінація значень має бути унікальною.

Тема: Багатокористувацький режим роботи з БД.

План.

1. Поняття транзакції.
2. Проблеми паралельної обробки даних.
3. Серіалізація транзакцій.

Рівні ізольованості транзакцій.

1. Щоб зрозуміти що собою представляє транзакція, розглянемо приклад. Нехай існує база даних, в який зберігається інформація про деяку торгову фірму, що постачає продукти в супермаркети. Потрібно змінити номер продукту з 13 на 20. Для цього потрібно провести зміни в 4-х взаємозв'язаних таблицях.

```
UPDATE    Продукти   UPDATE    Склад
SET    ПР = 20       SET   ПР = 20
WHERE     ПР = 13;   WHERE     ПР = 13;

UPDATE    Поставки   UPDATE    Наявність
SET    ПР = 20       SET   ПР = 20
WHERE     ПР = 13;   WHERE     ПР = 13;
```

Цей приклад показує, що єдина(суцільна), з точки зору користувача, операція може потребувати кількох операцій над БД. Крім того між виконанням цих операцій може навіть порушуватись цілісність даних, наприклад, в ній можуть бути записи про поставки, для яких немає записів про відповідні продукти. Протиріччя зникне лише після виконання всіх операцій на зміну номера.

Отже, транзакція - це послідовність операцій, які переводять базу даних з одного непротирічного стану в інший, але не гарантує збереження цілісності БД в проміжних операціях.

Ніхто крім користувача, що генерує транзакцію не може знати, про те коли може виникнути протирічний стан в БД, і після якої операції він зникне, тому в мові SQL є спеціальні команди, з допомогою яких можна визначити, що транзакція завершена, або виконати так званий відкат транзакції.

Властивості транзакцій. Способи завершення транзакцій.

На даний час розрізняють: плоскі чи класичні транзакції, ланцюгові транзакції, вкладені транзакції.

Плоскі транзакції [характеризуються 4-ма класичними властивостями:

- Атомарністю – виражається в тому, що транзакція повинна бути виконана повністю або не виконана зовсім;
- Узгодженість – гарантує, що транзакція не порушує взаємної узгодженості даних;
- Ізольованість – означає, що конкуруючі за доступ до БД транзакції обробляються послідовно, ізольовано один від одного, але для користувачів це виглядає ніби вони виконуються паралельно.
- Довговічність – якщо транзакція завершена успішно, то ті зміни, які були виконані, не можуть бути втрачені ні за яких обставин (навіть у випадку наступних операцій).

Можливі два варіанти завершення транзакцій: якщо всі операції виконані успішно і не відбулося ні яких збоїв транзакція фіксується (запис на диск).

До тих пір поки транзакція не зафіксована БД може перевести (повернути) до попереднього стану (відкат транзакції).

Для успішного завершення транзакції потрібне завершення всіх її операторів, якщо сталось щось таке, що робить неможливим завершення транзакції, потрібно перевести БД до попереднього стану. В стандарті SQL для цього використовуються оператори COMMIT та ROLLBACK. Стандарт визначає, що транзакція починається з першого SQL-оператора, що задається користувачем, або програмно, всі наступні оператори складають тіло транзакції. Транзакція завершується одним з 4-х можливих шляхів:

1) Оператор COMMIT означає успішне завершення, і відображення зміни в БД;
2) ROLLBACK перериває транзакцію, відміняє зміни в рамках цієї транзакції; нова транзакція починається безпосередньо після використання ROLLBACK;
3) Успішне завершення програми, в якій була ініціалізована поточна транзакція, означає успішне завершення ніби був виконаний оператор COMMIT;
4) Помилкове завершення програми перериває транзакції ніби був виконаний оператор ROLLBACK.

В цій моделі кожний оператор, що змінює стан БД, розглядається як транзакція.

В подальших версіях в СКБД була реалізована розширена модель транзакцій в ній використовуються наступні 4 оператора:

- BEGIN TRANSACTION – повідомляє про початок транзакції;
- COMMIT TRANSACTION – повідомляє про успішне завершення та фіксує всі зміни;
- SAVE TRANSACTION – створює всередині транзакції точку збереження, яка відповідає проміжному стану БД, збереженому на момент виконання цього оператора. В цьому операторі може стояти ім’я точки збереження, в ході виконання транзакцій може бути кілька точок збереження, що відповідає кільком проміжним станам.
- ROLLBACK має дві модифікації. Якщо він використовується без додаткового параметра, то він інтерпретується як оператор відкату всієї транзакції. Якщо він має параметр і зустрічається у вигляді ROLLBACK B, то він інтерпретується як оператор часткового відкату в точку збереження B.

Доцільно використовувати точки збереження в довгих і складних транзакціях.

Реалізація в СКБД принципу збереження проміжних станів забезпечується журналом транзакцій. Він призначений для забезпечення надійного збереження даних в БД, можливість відновлення узгодженого стану бази після будь-яких збоїв. При цьому потрібно дотримуватись наступних правил:

- результати зафіксованих транзакцій повинні бути збережені у відновленому стані БД;
- результати незафіксованих транзакцій повинні бути відсутніми у відновленому стані БД.

.

Відновлення БД потрібно проводити в таких випадках:

- індивідуальний відкат транзакцій;
- відновлення після втрати даних з ОП (м'який збій);
- відновлення після поломки зовнішнього носія (жорсткий збій).

Для відновлення узгодженого стану БД при індивідуальні й відкатці транзакцій потрібно знищити наслідки операторів модифікації БД, які виконували ці транзакції. При відновленні при м'якому збої необхідно відновити вміст журналу транзакцій, що зберігаються на дисках. При жорсткому збої – відновити вміст БД по архівних копіях та журналах транзакцій з неушкоджених дисків.

У всіх трьох випадках основою відновлення є надлишкове збереження даних. Ці надлишкові дані зберігаються в журналі, що містить послідовність записів про зміни в БД. Можливі два варіанта ведення журналу транзакцій:

1.Для кожної транзакції підтримується окремий локальний журнал зміни в БД (локальний журнал, для індивідуальних відкатів і можуть підтримуватись в ОП). Крім того підтримується загальний журнал змін БД, який використовується. для відновлення стану БД після м'яких та жорстких збоїв. Цей підхід дозволяє швидко виконувати індивідуальні відкати, але приводить до дублювання інформації в загальному та локальному журналах. Тому частіше використовують другий варіант

2. Ведення лише загального журналу змін БД, який використовують також при виконанні індивідуальних відкатів.

Розглянемо саме цей варіант: загальна структура журналу може бути представлена у вигляді деякого послідовного файлу в якому фіксуються всі зміни в БД, які відбуваються в ході виконання транзакцій. Всі транзакції мають свої внутрішні номери, тому в журналі фіксуються всі зміни, що виконуються всіма транзакціями. Кожний запис в журналі транзакцій помічається № транзакції, до якої він відноситься та значенням атрибутів, які він міняє. Крім того для кожної транзакції фіксується команда початку та кінця транзакції. Для більшої надійності журнал транзакцій часто дублюється системними засобами СКБД.

Журналізація змін тісно пов'язана з буферизацією сторінок в ОП. Якби запис при будь яких змінах в БД реально записувалась би в зовнішню пам'ять, це привело б до істотного сповільнення роботи. Тому запис в журналі також буферизується і чергове оновлення переноситься в зовнішню пам'ять при повному заповненні буфера. Якщо відбувається м'який збій то вміст буферів втрачений.

Для проведення відновлення БД необхідно мати деякий узгоджений стан журналу і БД у зовнішній пам'яті. Тобто запис про зміни об'єкта БД повинна попадати в зовнішню пам'ять раніше ніж змінений об'єкт виявиться у зовнішній пам'яті БД. Відповідний протокол називається Wribe Abead Log (WAL) – „пиши спочатку в журнал ".

Розглянемо тепер, як може виконати операції відновлення СКБД, якщо в системі підтримується журнал у відповідності з протоколом WAL.

Для того, щоб індивідуальний відкат транзакцій був можливий, всі записи в журналі по даній транзакції зв'язуються в зворотній список. Відкат можливий лише для незакінчених транзакцій. Він виконується наступним чином:

- вибирається остання операція;
- виконується протилежна їй за змістом операція;

- будь які з протилежних операцій теж заносяться в журнал, щоб у випадку м'якого збою можна було виконати відновлення;
- при успішному завершенні відката в журнал заноситься запис про це.

На момент м'якого збою можливі наступні стани транзакцій:

- транзакція успішно завершена (COMMIT) і для всіх операцій отримано підтвердження в зовнішній пам'яті;
- транзакція успішно завершена (COMMIT), але для деяких операцій не отримано підтвердження в зовнішній пам'яті;
- транзакція отримала і виконала команду відката ROLLBACK;
- транзакція незавершена.

Є два підходи: використання тіньового підходу або журналізація посторінкових змін в БД.

1) При відкритті файлу таблиця відображає номер логічних блоків у фізичні адреси блоків зчитується в ОП. При модифікації блоку у зовнішній пам'яті виділяється новий блок. При цьому поточна таблиця в ОП міняється, а тіньова в зовнішній пам'яті не змінюється. При збої в ОП в зовнішній пам'яті зберігається стан БД до відкриття.
2) Періодично виконуються операції встановлення точки фізичної узгодженості БД і тіньова таблиця в зовнішній пам'яті змінюється. При відновленні просто зчитується тіньова таблиця. Недолік – велике завантаження зовнішньої пам'яті.

Відновлення після жорсткого збою: основою для відновлення є журнал та архівна копія.

Відновлення починається з копіювання архіву по журналу в прямому напрямку виконуються всі операції; для незакінчених транзакцій виконується відкат. Якщо журнал втрачений – лише архівна копія. Архівують БД при переповненні журналу. В журналі вводиться так звана „жовта зона", коли всі транзакції тимчасово припиняються, незавершені закінчуються, база приводиться в узгоджений стан. Можна проводити архівацію.

2. Якщо з БД працює одночасно кілька користувачів, то кожна транзакція повинна виконуватися так, ніби вона ізольована. Таке виконання називається паралельним. Є кілька шляхів розпаралелювання запитів.

Горизонтальний паралелізм – виникає тоді, коли БД розподілена на кількох фізичних пристроях збереження (кількох дисках). При цьому інформація з одного відношення розбивається на частини по горизонталі. Цей вид паралелізму називають розпаралелюванням або сегментацією даних. Паралелізм досягається шляхом виконання однакових операцій (напр. фільтрації) над різними даними. Ці операції можуть виконуватись паралельно різним процесам, вони не залежні. Результат виконання цілого запиту складається з результатів виконання окремих операцій. Час виконання такого запиту при відповідному сегментуванні даних істотно менше, чим час такого ж запиту традиційним способом одним процесом.

Вертикальний паралелізм – досягається конвеєрним виконанням операцій, що відповідають запиту користувача. Цей підхід вимагає серйозного вдосконалення в

моделі виконання реляційних операцій ядром СКБД. Він передбачає, що ядро СКБД може провести декомпозицію запита, базуючись на його функціональних компонентах при цьому ряд підзапитів може виконуватись паралельно, з мінімальним зв'язком між окремими кроками виконання запита.

Дійсно, якщо ми розглянемо послідовність операцій RA

R5=R1[A,C] – проекція
R6=R2[A,B,D] – проекція
R7=R5[A>128] – фільтрація
R8=R5[A]R6 – умовне з'єднання

то 1 і 3 операції можуть об'єднати і виконати паралельно з 2, а потім виконати 4. Загальний час такого виконання буде менший ніж при звичайному виконанні.

Третій вид паралелізму є гібридом 2-х перших. Ці види паралелізму використовують в додатках, це може істотно скоротити час виконання складних запитів над великими об'ємами пам'яті.

При паралельній обробці транзакцій виникають деякі проблеми. Вони умовно дільться на 4 типи:

1. **втрачені зміни.** Ця ситуація може виникнути, коли дві транзакції змінюють один і той же запис. Приклад: перший оператор прийняґв замовлення на продаж 30 моніторів (на складі є 40); 2-й – 20 моніторів; 2-й update 20 шт; 1-й update – 10 шт. БД в неузгодженому стані, та ще 10 моніторів числяться но складі.
2. **проблема проміжних даних.** Той самий приклад. Припустим 1-й оператор оформляє замовлення, в БД внесені зміни залишок 10 моніторів, Але замовник ще обговорює деталі з оператором. В цей час 2-й оператор намагається прийняти замовлення на 20 моніторів, але бачить, що на складі залишилось всього 10, оператор віджмовляє клієнту. В цей час 1-й клієнт передумав купляти монітори, 1-й оператор виконує відкат транзакцій, на складі знову 40 моніторів. Замовник втрачений, але було б гірше коли оператор продав 10 моніторів заніс 0 в БД; після цього1-й оператор заніс 40 на склад, хоча 10 з них продано. Така ситуація стала можливою, тому що 2-й оператор мав доступ до проміжних даних.
3. **проблеми неузгоджених даних.** Припустимо обидва оператори починають роботу в один час і отримують початковий стан БД 40. 1-й оператор змінив транзакції, 2-й отримує нове значення – 10. 2-й оператор буде вважати, що порушена цілісність його транзакції.
4. **проблема рядків привидів.** Припустим оператор готує два звіти детальний та розширений. Детальний вже підготовлений, а коли готується розширений, оператор проводить транзакцію, другий зовсім на співпадає з першим, хоча бази знаходяться в узгодженому стані.

3. Для того, щоб уникнути подібних проблем, потрібно виконати деяку процедуру узгодженого виконання паралельних транзакцій. Ця процедура повинна задовольняти наступним правилам:

- в ході виконання транзакцій користувач бачить лише узгоджені дані, користувач не повинен бачити проміжних неузгоджених даних.
- Коли дві транзакції виконуються паралельно, то СКБД гарантовано підтримує принцип незалежного виконання тразакцій, який проголошує, що результати

виконання такими ж, якби спочатку виконувалась транзакція 1 а потім транзакція 2.

Така процедура називається серіалізацією транзакцій. Фактично, вона гарантує, що кожен користувач (програми), звертаючись до БД, працює з нею так, ніби не існує інших користувачів, що одночасно з нею звертаються до даних.

Для підтримки паралельної роботи транзакцій будується спеціальний план. План (спосіб) виконання транзакцій називається серіальним, якщо результат одночасного виконання транзакцій еквівалентний результату деякого послідовного виконання цих транзакцій.

Найбільше розповсюдження механізмом виконання серіальних транзакцій є механізм блокування. Самий простий варіант блокування об`єкта на весь час виконання транзакції. Якщо дві транзакції працюють з 3-ма таблицями T1, T2, T3. в момент звертання транзакцій, таблиця блокується об`єктом, що до неї звернувся, 2 інші транзакції знаходяться в стані очікування.

Недолік: затримка виконання транзакцій.

Між двома паралельними транзакціями існують наступні типи конфліктів:

- W-W – транзакція 2 намагається змінити об`єкт, змінений незакінченою транзакцією 1;
- R-W – транзакція 2 намагається змінити об`єкт, прочитаний назакінченою тоанзакцією 1;
- W-R – транзакція 2 намагається читати об`єкт, змінений незакінченою транзакцією 1.

Практичні методи серіалізації транзакцій засновані на врахуванні цих конфліктів.

Блокування (сінхронічніі захоплення) можуть бути застосовані до різного типу об`єктів. Найбільшим об`єктом може бути вся БД, але цей вид блокуавння робить БД недоступною для всіх додатків. Інший вид блокування таблиць, цей вид блокування кращий, тому що дозволяє паралельно працювати з іншими таблицями. В деяких СКБД блокування виконується на рівні сторінок. Цей вид ще більш гнучкий, тому, що дозволяє працювати з однією таблицею кілька транзакцій, з різнимии сторінками.

Для підвищення паралельності виконання транзакцій використовують комбінування різних типів синхронізованих захватів.

- сумісний режим блокування – нежорстоке, розділене блокування, позначають S (Shared). Цей режим означає розділений захват обєкта і вимагається для виконання операції читання об`єкта. Об`єкти заблоковані даним типом блокування, не змінюються у ході виконання транзакцій і дотупні іншим лише в режимі читання.
- Монопольний режим блокування – жорстке, або ексклюзивне блокування, X (exelusive). Передбачає монопольне захоплення об`єкта і вимагає для здійснення операції занесення, знищення, модифікації. Об`єкти, що заблоковані даним типом блокування, фактично в монопольному режимі обробки та недоступний для інших.

Захоплення об`єктів кількома транзакціями по читанню сумісні. (кілька транзакцій може читати один об`єкт). Захоплення одного об`єкта 1 транзакцією по читанню, другою по запису несумісні.

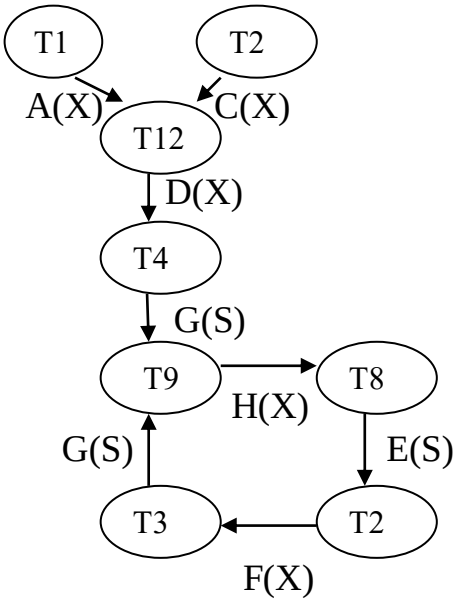
На жаль, застосування різних типів блокувань приводить до проблеми **тупиків**. Приклад: нехай транзакція А спочатку жорстко блокує Т1, а потім жорстко блокує Т2; а транзакція В навпаки. Якщо дві транзакції почали роботу одночасно, то А буде безкінеченно очікувати розблокування Т2, а В навпаки. Ситуації можуть бути більш складними.

Основою виявлення тупікової ситуації є побудова графа очікування транзакцій. Це направлений граф в вершинах якого розташовані імена транзакцій. Якщо транзакція А очікує закінчення транзакції В, то з вершини в вершину іде стрілочка.

Нехай граф побудований для транзгакцій Т1, Т2, ... , Т12, які працюють з об`єктами БД А, В, …, Н.

Н – жорстке блокування

S – нежортстке



.З діаграми видно, що Т9, Т8, Т2, Т3 утворюють цикл. Саме наявність циклу є признаком виненнення тупіка. Розпочинається починається з вибору транзакції-жертви. Критерієм вибору є вартість транзакцій. Жертвою вибирається сама дешева транзакція. Вартість транзакції визначається на основі багатофакторної оцінки: час виконання, число накопичених захоплень, пріорітет.

Після вибору транзакції-жертви виконується відкат цієї транзакції., звільняються захаплення і може бути продовжене виконання транзакції.

Рівні ізольованості користувачів пов`язані з проблемами, які виникають при паралельному виконанні транзакцій. Всього введено 4 рівні ізольованості користувачів.

Самий високий відповідає протоколу серіалізацій, це рівень SERIALIZABLE. Цей рівень забезпечує повну ізоляцію транзакцій і повну коректну обробку паралельних транзакцій.

Наступний рівень – рівень підтивердженого читача – REPEATABLE READ. На цьому рівні транзакція на має доступ до проміжних чи кінцевих результатів інших транзакцій. Тому такі проблеми, як пропавші очікування, проміжні чи неузгоджені дані виникнути не можуть. Але можуть бути побачити наступний записв, добавлений іншим користувачем, залишиться проблема рядків-призраків. Цю проблему краще розвязувати алгорітмічно, виключаючи повторне виконяння запиту в одній транзакції.

Наступний рівень ізольованості пов`язаний з підтвердженням читанням він називається READCOMITED. На цьому рівні транзакція не має доступу до проміжних результатів інших транзакцій;транзакції не можуть обновити рядок вже поновлену іншиою транзакцією.

Самий низький рівень ізольованості називається рівень непідтвердженого (брудного) читання READUNCOMMITED. Транзакція бачить проміжні і неузгоджені дані, та рядки-призраки.

В стандарті SQL2 існують операції задання рівня ізольованості виконання транзакцій.

SET TRANSACTION IZEOLATION LEVEL [{SERIALIZABLE
REPEATABLE READ/
READ COMMITED/
READ UNCOMITED}] [{READ WRITE/
READ ONLY}]

Гранульовапні сінхронізаційні захоплення. При виконанні захопленькрупних об`єктів, ймовірність конфліктів зростає. В більшості сучасних СКБД – покортежне блокування. Але було б нерозумно застосовувати покортежне блокування при знищені всіх рядків відношення.

Це привело до поняття гранульованого сінхронного захоплення і розробці відповідного механізма.

Синхронічні захоплення можуть бути застосовані по відношенню до різних об`єктів; файлів, відношеннь і кортежів. Рівень об`єкта визначається тим, яка операція виконується (при знищені відношення - захоплення відношення, при знищені кортежа – кортеж). Об`єкти будь-якого рівня може бути захоплений в режимі S (розподільному) або X (монопольному). Вводиться спеціальний протоколгранульованих захоплень і визначаються нові типи захоплень. Об`єкт більш високого рівня повинен бути захоплений в режимі IS або IX або SIX.

IS (INTENED FOR SHARED LOCK) – запобігаєрозподільному блокуванню. По відношенню до деякого складного об`єкта. О означає бажання захопити даний об`єкт, що входить в нього в сумісному режимі.

IX (interned fox exclusive lock) – запобігає жорстокому блокуванню відношення до деякого складного об`єкту (при занченні IX)

SIX (shared, Internded for exclusive lock) розподіляє блокування, запобігає жорстокому блокуванню складових. Наприклад при переключанні відношення з можливим знищенням будь-яких кортежей, економічне захоплення відношення відносно в SIX, а файл в IS)

Представлення та їх організація

План.

1. Поняття представлення, призначення представлення.
2. Приклади використання представлень.

Типы таблиц, с которыми вы имели дело до сих пор, назывались – *базовыми таблицами*. Это - таблицы, которые содержат данные. Однако имеется другой вид таблиц: - представления. *Представления* - это таблицы чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления - подобны окнам, через которые вы просматриваете информацию(как она есть, или в другой форме, как вы потом увидите), которая фактически хранится в базовой таблице. *Представление* - это фактически запрос, который выполняется всякий раз, когда представление становится темой команды. Вывод запроса при этом в каждый момент становится содержанием представления.

```
CREATE VIEW Londonstaff
  AS SELECT *
  FROM Salespeople
        WHERE city = 'London'
```

Теперь Вы имеете представление, называемое **Londonstaff**. Вы можете использовать это представление точно так же как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена и соединена с, другими таблицами и представлениями.

Представлення завжди відображає самі останні дані вибрані з таблиці.

Представления значительно расширяют управление вашими данными. Это - превосходный способ дать публичный доступ к некоторой, но не всей информации в таблице.

Представление может теперь изменяться командами модификации DML, но модификация не будет воздействовать на само представление. Команды будут на самом деле перенаправлены к базовой таблице:

```
UPDATE Salesown
  SET city = 'Palo Alto'
  WHERE snum = 1004
```

Имеется возможность комбинации из двух полностью допустимых предикатов и получения предиката, который не будет работать. Например, предположим что мы *создаем* (**CREATE**) следующее представление:

```
CREATE VIEW Ratingcount (rating, number)
  AS SELECT rating, COUNT (*)
  FROM Customers
  GROUP BY rating;
```

Это дает нам число заказчиков, которые мы имеем для каждого уровня оценки(rating). Вы можете затем сделать запрос этого представления чтобы

выяснить, имеется ли какая-нибудь оценка, в настоящее время назначенная для трех заказчиков:

Чем конструировать каждый раз сложный запрос, вы можете просто создать следующее представление:

```
CREATE VIEW Totalforday
    AS SELECT odate, COUNT (DISTINCT cnum), COUNT
        (DISTINCT snum), COUNT (onum), AVG
        (amt), SUM (amt)
        FROM Orders
    GROUP BY odate;
```

Теперь вы сможете увидеть всю эту информацию с помощью простогозапроса:

```
SELECT *
  FROM Totalforday;
```

Как мы видели, SQL запросы могут дать вам полный комплекс возможностей, так что представления обеспечивают вас чрезвычайно гибким и мощным инструментом чтобы определить точно, как ваши данные могут быть использованы. Они могут также делать вашу работу более простой, переформатируя данные удобным для вас способом и исключив двойную работу.

Представления не требуют, чтобы их вывод осуществлялся из одной базовой таблицы. Так как почти любой допустимый запрос SQL может быть использован в представлении, он может выводить информацию из любого числа базовых таблиц, или из других представлений. Мы можем, например, создать представление которое показывало бы, порядки продавца и заказчика по имени:

```
CREATE VIEW Nameorders
  AS SELECT onum, amt, a.snum, sname, cname
    FROM Orders a, Customers b, Salespeople c
    WHERE a.cnum = b.cnum
      AND a.snum = c.snum;
```

Теперь вы можете выбрать (**SELECT**) все порядки заказчика или про-давца (*****), или можете увидеть эту информацию для любого порядкаНапример, чтобы увидеть все порядки продавца Rifkin, вы должны ввести следующий запрос (вывод показан в 20.3 Рисунке):

```
SELECT *
  FROM Nameorders
  WHERE sname = 'Rifkin';
```

Вы можете также объединять представления с другими таблицами, илибазовыми таблицами или представлениями, поэтому вы можете увидетьвсе порядки Axelroda и значения его комиссиионных в каждом порядке:

```
SELECT a.sname, cname, amt  comm
  FROM Nameorders a, Salespeople b
```

```
WHERE a.sname = 'Axelrod'
  AND b.snum = a.snum;
```

Представления могут также использовать и подзапросы, включая соотнесенныеподзапросы. Предположим ваша компания предусматривает премию для тех продавцов которые имеют заказчика с самым высоким порядком для любой ука-занной даты. Вы можете проследить эту информацию с помощью представления:

```
CREATE VIEW Elitesalesforce
  AS SELECT b.odate, a.snum, a.sname,
    FROM Salespeople a, Orders b
    WHERE a.snum = b.snum
      AND b.amt =
       (SELECT MAX (amt)
          FROM Orders c
          WHERE c.odate = b.odate);
```

Если, с другой стороны, премия будет назначаться только продавцу которыйимел самый высокий порядок за последние десять лет, вам необходимоИмеются также некоторые виды запросов, которые не допустимы в опреде-лениях представлений. Одиночное представление должно основываться на одиночном запросе; *ОБЪЕДИНЕНИЕ* (**UNION)** и *ОБЪЕДИНЕНИЕ ВСЕГО* (**UNIOM ALL)** не разрешаются.УПОРЯДОЧЕНИЕ ПО(**ORDER BY)** никогда не используется в опреде-лении представлений. Вывод запроса формирует содержание представления, которое напоминает базовую таблицу и является - по определению – неупорядоченным

Синтаксис удаления представления из базы данных подобен синтаксисуудаления базовых таблиц:

```
DROP VIEW < view name >
```

Один из наиболее трудных и неоднозначных аспектов представлений - непосредственное их использование с командами модификации DML.Как упомянуто в предыдущей главе, эти команды фактически воздействуют на значения в базовой таблице представления. Это является некоторым противоречием. Представление состоит из результатов запроса, и когда вы модифицируете представление, вы модифицируете набор результатов запроса. Но модификация не должна воздействовать на запрос ; она должна воздействовать на значения в таблице к которой был сделан запрос, и таким образом изменять вывод запроса. Это не простой вопрос. Если команды модификации могут выполняться в представлении, представление как сообщалось будет *модифицируемым*; в противном случае оно предназначено только для чтения при запросе. Непротивореча этой терминологии, мы будем использовать выражение "*модифицируемое представление*"(updating a view), чтобы означает возможность выполнения в представление любой из трех команд модификации DML (*Вставить, Изменить* и *Удалить*), которые могут изменять значения.Как вы определите, является ли представление модифицируемым? В теориибазы данных, это - пока обсуждаемая тема. Основной ее принцип такой:*модифицирумое представление* - это представление в котором команда моди-фикации может выполниться, чтобы изменить одну и только одну строку ос-новной таблицы в каждый момент времени, не воздействуя на любые другие строки любой таблицы. Использование этого принципа на практике, однако, затруднено. Кроме того, некоторые представления, которые являются модифи-цируемыми в теории,

на самом деле не являются модифицируемыми в SQL. Критерии по которые определяют, является ли представление модифицируемым или нет, в SQL, следующие:

* Оно должно выводиться в одну и только в одну базовую таблицу.

* Оно должно содержать первичный ключ этой таблицы (это технически не предписывается стандартом ANSI, но было бы неплохо придерживаться этому).

* Оно не должно иметь никаких полей, которые бы являлись агрегатными функциями.

* Оно не должно содержать DISTINCT в своем определении.

* Оно не должно использовать GROUP BY или HAVING в своем определении.

* Оно не должно использовать подзапросы (это - ANSI_ограничение которое не предписано для некоторых реализаций)

* Оно может быть использовано в другом представлении, но это представление должно также быть модифицируемыми.

* Оно не должно использовать константы, строки, или выражения значений (например: comm * 100) среди выбранных полей вывода.

* Для INSERT, оно должно содержать любые поля основной таблицы которые имеют ограничение NOT NULL, если другое ограничение по умолчанию, не определено

Одно из этих ограничений то, что модифицируемые представления, фактически, подобны окнам в базовых таблицах. Они показывают кое-что, но не обязательно все, из содержимого таблицы. Они могут ограничивать определенные строки (использованием предикатов), или специально именованные столбцы (с исключениями), но они представляют значения непосредственно и не выводит их информацию, с использованием составных функций и выражений.
Они также не сравнивают строки таблиц друг с другом (как в обьединениях и подзапросах, или как с DISTINCT). Различия между модифицируемыми представлениями и представлениями только_чтение неслучайны. Цели для которых вы их используете, часто различны. Модифицируемыепредставления, в основном, используются точно так же как и базовые таб-лицы. Фактически, пользователи не могут даже осознать, является ли объ-ект который они запрашивают, базовой таблицей или

представлением.Это превосходный механизм защиты для сокрытия частей таблицы, которые являются конфиденциальными или не относятся к потребностям данного пользователя. (В Главе 22, мы покажем вам, как позволить пользователям обращаться к представлению, а не к базовой таблице).
Представления только_чтение, с другой стороны, позволяют вам получатьи переформатировать данные более рационально. Они дают вам библиотекусложных запросов, которые вы можете выполнить и повторить снова, сохраняя полученную вами информацию до последней минуты.
Кроме того, результаты этих запросов в таблицах, которые могут затем использоваться в запросах самостоятельно (например, в обьединениях) имеют преимущество над просто выполнением запросов.
Представления только_чтение могут также иметь прикладные программызащиты. Например, вы можете захотеть, чтобы некоторые пользователи виде-ли агрегатные данные, такие как усредненное значение комиссионных про-давца, не видя индивидуальных значений комиссионных Имеются некоторые примеры модифицируемых представлений и пред-ставлений только_чтение:

```
CREATE VIEW Dateorders (odate, ocount)
  AS SELECT odate, COUNT (*)
    FROM Orders
    GROUP BY odate;
```

Это - представление только_чтение из-за присутствия в нем агрегатной функции и GROUP BY.

```
CREATE VIEW Londoncust
  AS SELECT *
    FROM Customers
    WHERE city = 'London';
```

А это - представление модифицируемое.

```
CREATE VIEW Salesonthird
  AS SELECT *
    FROM Salespeople
    WHERE snum IN
     (SELECT snum
       FROM Orders
       WHERE odate = 10/03/1990);
```

Это - представление только_чтение в ANSI из-за присутствия в немподзапроса. В некоторых программах, это может быть приемлемо.

ИЗМЕНЕНИЕ ЗНАЧЕНИЙ С ПОМОЩЬЮ ПРЕДСТАВЛЕНИЯ

```
CREATE VIEW Someorders
  AS SELECT snum, onum, cnum
    FROM Orders
    WHERE odate IN (10/03/1990,10/05/1990);
```

Это - модифицируемое представление Другой вывод о модифицируемости представления тот, что вы можете вводить значения которые "

проглатываются " (swallowed) в базовой таблице. рассмотрим такое представление:

```
CREATE VIEW Highratings
  AS SELECT cnum, rating
    FROM Customers
    WHERE rating = 300;
```

Это - представление модифицируемое. Оно просто ограничивает ваш дос-туп к определенным строкам и столбцам в таблице. Предположим, что вы *вставляете* (**INSERT**) следующую строку:

```
INSERT INTO Highratings
  VALUES (2018, 200);
```

Это - допустимая команда INSERT в этом представлении. Строка будетвставлена, с помощью представления Highratings, в таблицу Заказчиков.Однако когда она появится там, она исчезнет из представления, посколькузначение оценки не равно 300. Это - обычная проблема. Значение 200 может быть просто напечатано, но теперь строка находится уже в таблице Заказчиков где вы не можете даже увидеть ее. Пользователь не сможет понять, почему введя строку он не может ее увидеть, и будет не-способен при этом удалить ее.

Лекція 7. Створення та використання stored-процедур

План

1. Збережені на сервері процедури.
2. Генератори та тригери.

1. Створення та використання процедур в SQL

Клієнтські додатки можуть звертатись до потужності SQL-сервера для розв'язування складних задач Збережені процедури, це підпрограми, що виконуються на сервері(в середині серверного процесу), створюються та зберігаються в БД. Таки процедури можуть виконуватись в клієнтських додатках в будь-який кількості, можуть маніпулювати даними в базі даних а також повертати клієнту результати свого виконання. Пишуться таки процедури на мові SQL. Кожна така процедура компілюється при першому виконанні, в процесі компіляції будується оптимальний план її виконання. Створюються процедури командою

CREATE PROCEDURE *ім'я_процедури*
[(*вхідний_параметр тип* [, *вхідний_параметр тип* ...])]
[**RETURNS** (*вихідний_параметр тип* [, *вихідний_параметр тип*...])]
AS
тіло_процедури;
[*термінатор*]

Синтаксис збереженої процедури складається з двох частин: заголовка і тіла. Заголовок включає команду **CREATE PROCEDURE** ім'я процедури, список вхідних параметрів і список параметрів, які повертаються з процедури. Деякі з цих складових можуть бути відсутні.
Тіло процедури може включати DML-SQL-конструкції, а також, програмні конструкції FOR SELECT…DO, IF-THEN, WHILE...DO та інші.

Конструкції мови SQL

1. Коментар

/*коментар***/**

2. Конкатенація

'симв_рядок1'||'симв_рядок2'

3. Опис локальних змінних

DECLARE VARIABLE <*ім'я змінної*> <*тип змінної*>

4. Оператор умови IF ... THEN ... ELSE ...

IF (*умова***)**
THEN *оператор_1*
[ELSE *оператор_2***];**

умова – логічний вираз, який має значення TRUE або FALSE;
оператор_1 – виконується, якщо умова приймає істинне значення. Замість *оператор_1* може бути блок операторів, обмежений конструкцією **BEGIN … END**;

оператор_2 – виконується, якщо умова хибна. Замість *оператор_2* може бути блок операторів, обмежений конструкцією **BEGIN … END**.

5. Оператор циклу WHILE … DO …

WHILE (*умова***) DO** *оператор;*

умова – логічний вираз, значення якого перевіряється перед кожним виконанням циклу;
оператор – оператор чи блок операторів **BEGIN … END**, який виконується, якщо умова приймає істинне значення.

6. Оператор SELECT

SELECT_команда INTO *список_змінних;*

Синтаксис даного оператора подібний до синтаксису команди **SELECT.** На відміну від звичайної команди **SELECT** результатом даної команди є один рядок, отримані дані записуються у змінні, вказані після ключового слова **INTO.**

7. Оператор циклу FOR … DO …

FOR SELECT_команда INTO *список_змінних* **DO** *оператор;*

SELECT_команда отримує дані з бази даних. Дана команда отримує за один раз тільки один рядок, і результат записується у відповідні змінні, вказані після ключового слова **INTO.** Перед кожною змінною після **INTO** ставиться двокрапка (**:**) для того, щоб відрізнити ім'я стовпця від імені змінної;
оператор – оператор чи блок операторів **BEGIN … END**, який виконується для кожного рядка, отриманого командою **SELECT.**

В тілі процедури може використовуватись оператор EXIT. Він здійснює вихід з циклу і перехід на останній END в процедурі. Команда SUSPEND икористовується тільки в процедурах, що повертають результати в точку виклику.

Існує два види збережених процедур: **SELECT** і **EXECUTE**.
SELECT-процедури обов'язково повертають одне або декілька значень(рядків) і можуть використовуватись при створенні запитів разом з таблицями і представленнями.
EXECUTE-процедури можуть повертати 1 або не повертати жодного значення у точку виклику.
SELECT-процедури і EXECUTE-процедури описуються однаково, але викликаються по різному.
SELECT-процедури викликаються за допомогою конструкції:

SELECT * FROM *ім'я_процедури* [(*фактичний_параметр* [, *фактичний_параметр*…])];

EXECUTE - процедури викликаються за допомогою конструкції:

EXECUTE PROCEDURE *ім'я_процедури* [(*фактичний_параметр* [, *фактичний_параметр*…])]

Приклади:

1. Збільшити значення поля kurs на 1(при переведення на новий курс)
SET TERM ^ ;

```
CREATE PROCEDURE PR_1
as
begin
 update student set kurs=kuer+1 ;
 end^
```

```
SET TERM ; ^
```

Зверніть увагу, що в середині процедури використовується «;» - розділювач команд. Як відомо «;» є стандартним розділовим знаком для команд і сигналом для інтерпретатора SQL, що команда ведена повністю і можна приступати до її виконання. Тому, якщо створювати процедури з допомогою SQL- скриптів, необхідно перед створення процедури змінювати розділювач команд SET TERM ^ ;. В кінці процедури записується протилежна команда SET TERM ; ^

2. Проіндексувати поле стипендія на задане число для певних студентів

```
SET TERM ^ ;

CREATE PROCEDURE PR2 ( ssum decimal(15,2), kf integer)
as
begin
 update student set stip=stip*:kf
 where stip=:ssum ;
end^

SET TERM ; ^
```

В наведеному прикладі використовується змінна :ssum, яка є вхідним параметром процедури, тут символ «:» використовується для тог щоб відрізнити змінні від назв полів, Двокрапка пере іменем змінної використовується лише в SQL-операторах, у всіх інших операторах програми ці змінні використовуються без «:».

3. Проіндексувати поле стипендія для студентів, що здали іспит з певного предмету, знищити інформацію про деякого студента

```
CREATE PROCEDURE NEW_PROCEDURE
(    kf integer,    pnom_pm integer,     snom_pm integer)
as
begin
 if (exists(select snom,pnom from uspish
 where snom=:snom_pm and pnom=:pnom_pm))
 then
  begin update student set stip=:kf;
delete from uspish
where snom=:snom_pm and pnom=:pnom_pm ;
end
```

4. Вивести на екран інформацію про предмет, на вивчення якого відводиться задана кількість годин

```
CREATE PROCEDURE PR3
returns (  pn varchar(20),  g integer)
as
begin
for select pnam,god from predmet where god=162
into
:pn,:g
do
  suspend;
end^
```

```
SET TERM ^ ;
```

5. Процедура нарахування стипендії

```
CREATE PROCEDURE NEW_STIP
returns (  sn integer,  st decimal(15,2), avg_bal decimal(5,1))
as
begin
for select snom, avg(ocinka) from uspish group by snom
into :sn,:avg_bal
do
begin
if avg_bal >=4  then
 update student set stip=720
 where student.snom =:sn ;
else
if avg_bal <4  then
update student set stip=null
suspend;
end;
end^
```

Обробка виключень, повідомлення про помилки

Однією з особливостей мови збережених на сервері процедур і тригерів Interbase є можливість використовувати так звані виключення. Виключення Interbase багато в чому схожі на виключення інших мов програмування високого рівня, проте мають свої особливості. Фактично виключення Interbase - це повідомлення про помилку, яке має власне ім'я і текст повідомлення, що задається програмістом. Створюється виключення наступним чином.

CREATE EXCEPTION *<ім'я виключення>* *<текст виключення>*;

Виключення легко видалити або змінити: видалення здійснюється командою

DROP EXCEPTION < имя_исключения >

ALTER EXCEPTION <имя_исключения> < текст виключення >.

Щоб використовувати виключення в процедурі, що зберігається, або тригері, необхідно скористатися командою наступного вигляду.

EXCEPTION < ім'я виключення >;

Давайте розглянемо використання виключень на простому прикладі процедури, що зберігається. Нехай процедура виконує ділення одного числа на інше і повертає результат. Нам необхідно відстежити випадок ділення на нуль і активувати виключення, якщо дільник дорівнює нулю.

Створимо виключення

CREATE EXCEPTION zero_divide ' Cannot divide by zero!'

яке буде генеруватись процедурою

```
CREATE PROCEDURE SP_DIVIDE ( DELIMOE DOUBLE PRECISION, DELITEL DOUBLE PRECISION)
RETURNS (
RESULT DOUBLE PRECISION)
AS BEGIN
if (Delitel<0.0000001) then BEGIN
EXCEPTION zero_divide; Result=0; END ELSE BEGIN
Result=Delimoe/Delitel; END
SUSPEND; END
```

У виключеннях було б мало користі, якби не було можливості обробляти їх на рівні бази даних. Для цього використовується конструкція

```
WHEN EXCEPTION <ім'я_виключення> DO BEGIN
 /*обробка виключення*/
END
```

Використання цієї конструкції допомагає уникнути стандартного повідомлення про помилки(того, що генерується СКБД) і виконати власні дії по обробці помилки.

Наступний приклад демонструє використання виключень. Припустимо, в нас є процедура sp_test_except, що викликає нашу процедуру SP_DIVIDE. Звичайно, приклад досить надуманий, але він демонструє зміст використання виключень в SQL.

```
CREATE PROCEDURE sp_test_except (Delitel DOUBLE PRECISION)
RETURNS (rslt DOUBLE PRECISION, status VARCHAR(50)) AS
BEGIN
  Status='Everything is Ok';
  SELECT result FROM sp_divide(12,:Delitel) INTO :rslt; SUSPEND;
  WHEN EXCEPTION Zero_divide DO BEGIN
Status='zero value found!'; rslt=-l; SUSPEND;
END
END
```

2. Генератори та тригери

Генератор – це механізм, який створює послідовний унікальний номер, що автоматично вставляється в стовпець під час таких операцій, як **INSERT** або **UPDATE**. Генератори зазвичай використовуються для створення унікальних значень, які можуть бути вставлені в стовпець, який використовується як первинний ключ.

Створення генератора

CREATE GENERATOR *ім'я_генератора*;
При створенні генератора за замовчуваннчм його початкове значення дорівнює нулю.

Ініціалізація генератора

SET GENERATOR *ім'я_генератора* **TO** *ціле_число***;**

Функція GEN_ID

GEN_ID (*ім'я_генератора* **,** *ціле_число***);**
Дана функція збільшує поточне значення генератора на ціле число.

Приклад:

Створити генератор, який використовується для створення унікальних значень поля SNOM, який є первинним ключем таблиці, починаючи з 101.

```
CREATE GENERATOR GEN_STUDENTS;
SET GENERATOR GEN_STUDENTS TO 100;
INSERT INTO STUDENTS (SNOM, SFAM)
VALUES(GEN_ID(GEN_STUDENTS,1), 'IBAHOB');
```

Тригери в InterBase - це особливий вид процедури, що зберігається, яка виконується автоматично при вставці, видаленні або модифікації запису таблиці чи представлення (view). Тригери можуть "спрацьовувати" безпосередньо до або відразу ж після вказаної події.

Як відомо, SQL дає можливість нам вставляти, видаляти і модифікувати дані в таблицях бази даних за допомогою відповідних команд - INSERT, DELETE і UPDATE. Погодьтеся, що було б непогано мати можливість перехопити передавану команду і що-небудь зробити з даними, які додаються, віддаляються або змінюються. Наприклад, записати ці дані в спеціальну табличку, а заразом записати, хто і коли зробив операцію над цією таблицею. Чи відразу ж перевірити дані, що вставляються, на яку-небудь хитру умову, яку неможливо реалізувати за допомогою опції CHECK, і залежно від результатів перевірки прийняти зміни, що проводяться, або відхилити їх; змінити дані на основі якого-небудь запиту в одній або кількох пов'язаних таблицях. Ось для того і існують тригери.

На відміну від процедур, що зберігаються, тригер ніколи нічого не повертає (та і нікому повертати, адже тригер явно не викликається). З тієї ж причини він не має вхідних параметрів, але замість них має контекстні змінні NEW і OLD. Ці змінні дозволяють отримати доступ до полів таблиці, до якої приєднаний триггер.

Тригер завжди прив'язаний до якоїсь певної таблиці або представлення і може "перехоплювати" дані тільки з цього об’єкта. Він виконує, так би мовити, роль віртуального «цензора», що дозволяє або відхиляє зміни, аналізує похибки, або може в разі необхідності «доповісти куди треба».

1. Створення тригера

CREATE TRIGGER *ім’я_тригера* **FOR** *ім’я_таблиці*
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
[POSITION *номер***]**
AS *тіло_тригера*
термінатор

[ACTIVE | INACTIVE] – необов’язковий параметр, який визначає дію тригера після завершення транзакції. **ACTIVE –** тригер використовується (по замовчуванню). **INACTIVE –** тригер не використовується.
{BEFORE | AFTER} – обов’язковий параметр, який визначає коли відбудеться активізація тригера до події (**BEFORE**) чи після (**AFTER**).
{DELETE | INSERT | UPDATE} – визначає операцію над таблицею, яка викликає тригер для виконання.
[POSITION *номер***] –** визначає порядок виклику тригера для обробки однієї події, наприклад при додаванні запису в таблицю. Номер має бути цілим від 0 до 32 767, по замовчуванню дорівнює нулю. Тригер, який має менший номер, виконується раніше.
тіло_тригера – складається з двох частин:
1) блок опису локальних змінних, які використовуються в тригері. Кожна змінна описується конструкцією
DECLARE VARIABLE *ім’я_змінної тип_змінної* ;
і закінчується крапкою з комою (;);
2) блок програмного коду, який починається оператором **BEGIN** і завершується оператором **END**
BEGIN
команди_InterBase
END
Кожна команда завершується крапкою з комою (;).
У блоці програмного коду тригера використовуються дві контекстні змінні: **OLD** і **NEW**. Змінна **OLD.***ім’я_стовпця* відповідає за старі значення стовпця, змінна **NEW**.*ім’я_стовпця* – за нові значення.
До виконання команди, дані з таблиці заносяться в буфер пам’яті, де тригер має до них доступ через змінні **OLD** і **NEW та** може дозволити чи заборонити дії над ними.
Розглянемо існуючи види тригерів -**Active (inactive) before(after)**

Дані, що прислані на вставку поміщаються в буфер та можуть бути переглянуті або змінені з допомогою контекстної змінної NEW, до полів цього запису можна звертатись, як до полів запису.

```
CREATE TRIGGER Table_ex FOP Table_example
ACTIVE BEFORE  INSERT POSITION 0
AS BEGIN
```

```
IF (NEW.ID IS NULL) THEN
NEW.ID=GEN_ID (GEN_TABLE_EXAMPLE_ID,1)
END
```

Ми бачимо, що в даному тригері використовується змінна NEW. Загалом існують деякі умови використання цих двох змінних. Так, контекстна змінні OLD не може бути використана в тригерах before(after) insert, а змінна NEW в тригерах before(after) delete. Але обидві ці змінні можуть використовуватись в тригерах before(after) update

Приклад. Створити генератор **GEN_STUDENTS** і створити тригер **STUDENTS_BI**, який використовує даний генератор як лічильник для поля **SNOM** при додаванні записів у таблицю **STUDENTS**.

```
CREATE GENERATOR GEN_STUDENTS;

SET TERM ^ ;

CREATE TRIGGER STUDENTS_BI FOR STUDENTS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.SNOM IS NULL) THEN
    NEW.SNOM = GEN_ID(GEN_STUDENTS,1);
END ^

SET TERM ; ^
```

Модифікація тригера

ALTER TRIGGER *ім'я_тригера*
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION *номер***]**
[AS *тіло_тригера* **]**
[*термінатор***]**

При модифікації можна змінити статус тригера, порядок і спосіб його виконання, внести зміни у тіло тригера. За замовчуванням тригер створюється активним, якщо ж зробити його неактивним він не буде використовуватись при виконання операції. Це буває корисним при виконанні деяких позапланових операцій над даними або при виправлені даних вручну. Не варто через тригер змінювати дані в таблиці до якої він прив'язаний не через контекстні змінні, а через оператори SQL, бо це може привести до за циклювання і аварійній зупинці сервера InterBase.

Приклад:

Змінити статус активного тригера *TR1* на пасивний.

ALTER TRIGGER TR1 **INACTIVE;**

2. Видалення тригера

DROP TRIGGER *ім'я_тригера***;**

Якщо база даних досить складна, є досить велика ймовірність виникнення помилок на будь-якому етапі коригування даних, коли, наприклад, при вставці даних в головну таблицю запускаються ЗП, що активізують тригери і вставку даних у пов'язані таблиці, тощо. При появі помилок InterBase повідомить про це і виконає

відкат змін у таблицях Але клієнтський додаток може відслідкувати помилку і підтвердити транзакцію, якщо це допустимо. Можна виконати обробку помилки безпосередньо в тілі тригера з допомогою конструкції WREN…DO. Так само як і ЗП тригер може генерувати обробку виключень, тому що він є різновидом ЗП.