Збережені процедури

Збережені процедури (stored procedures) – фрагменти програмного коду, який зберігається і виконується на сервері БД.

Переваги збережених процедур

Продуктивність

Значна частина операцій з даними виконується на сервері БД, який є зазвичай потужним комп'ютером з великим обсягом оперативної пам'яті. В архітектурі Клієнт-сервер збережені процедури виконуються на сервері і лише результат повертається клієнту, що пришвидшує обробку даних. Тим самим найбільш повно реалізуються переваги архітектури Клієнт-сервер. Крім того, збережені процедури синтаксично вірні та скомпільовані.

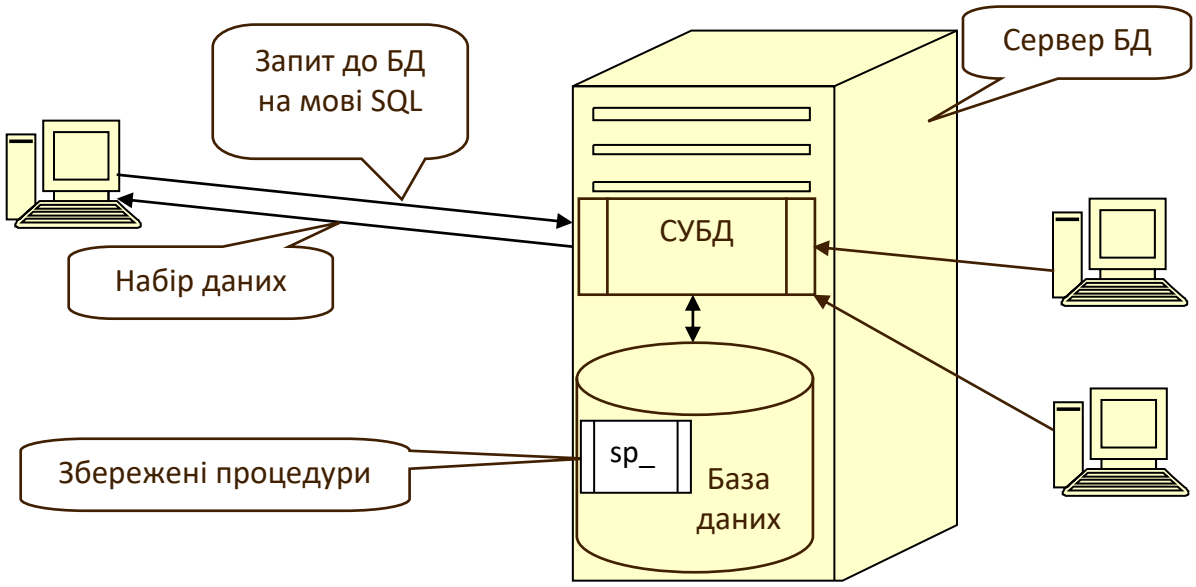


Рисунок 1 – Збережені процедури зберігаються в БД

Централізація бізнес-логіки

Модифікувати збережену процедуру відповідно до нових вимог можна в будь-який момент, в одному місці. Тим самим усі програми, що використовують її, негайно прийдуть у відповідність з новими бізнес-правилами без масового оновлення клієнтів.

Можливості програмування

Можливості програмування розширюються завдяки застосуванню таких поширених засобів програмування, як використання вхідних і вихідних параметрів, а також завдяки повторному використанню процедур.

Якщо розробникам зручно писати складні програми на таких мовах, як C# або Visual Basic.NET, то потім ці програми можна викликати з SQL Server (починаючи з версії 2005) засобами збережених процедур особливого типу, які називаються розширеними збереженими процедурами. CLR-код у цьому разі також зберігається в БД.

Безпека

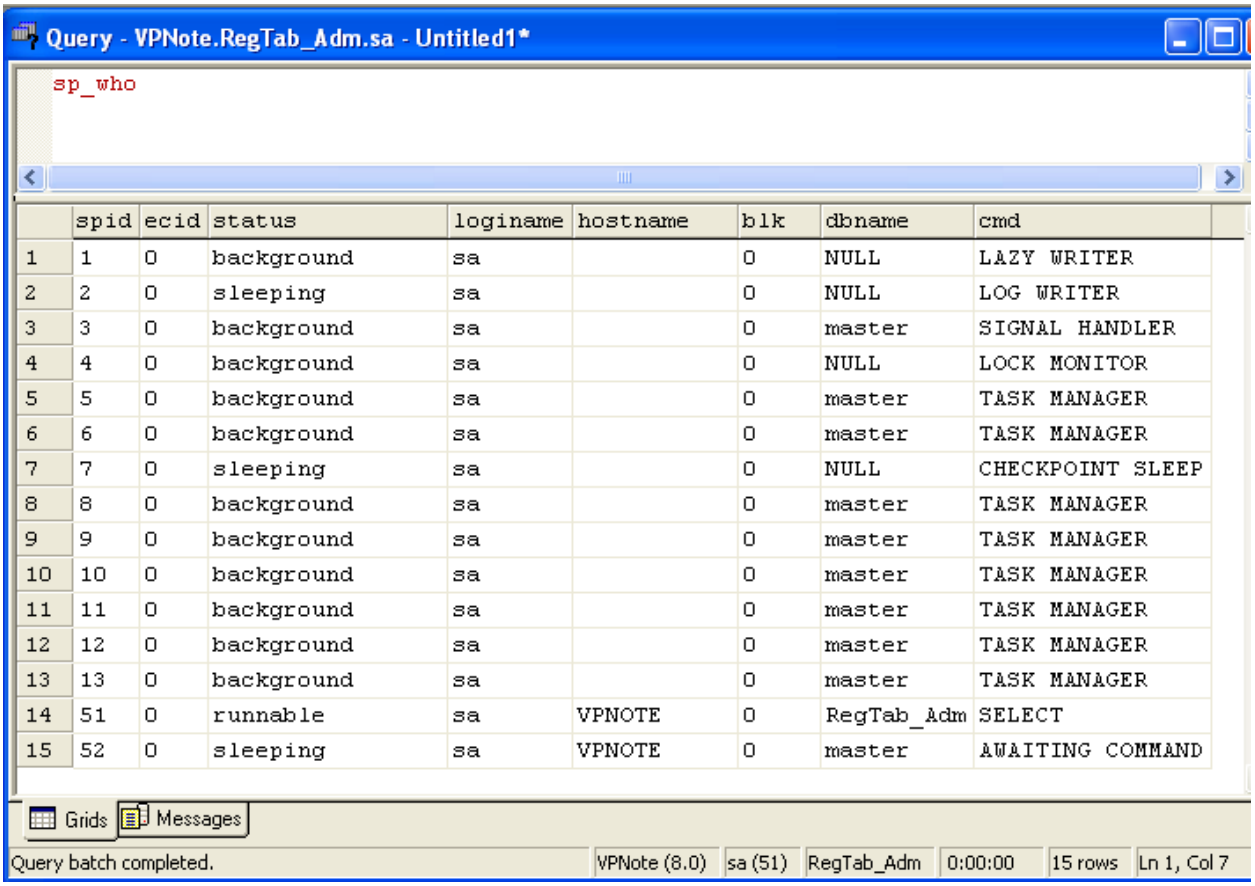
Користувачам можна надати право на виконання збереженої процедури без безпосереднього доступу до об'єктів бази даних, з якими працює збережена процедура, виконавши тим самим інкапсуляцію даних. Крім того, якщо збережену процедуру зашифрувати при створенні або модифікації, користувачам не вдасться прочитати команди Transact-SQL, які містить процедура. Ці функції безпеки дозволяють ізолювати від користувача структуру бази даних, що забезпечує цілісність даних і надійність бази.

Категорії збережених процедур

Системні збережені процедури

У MS SQL системні збережені процедури знаходяться в базі даних Master. Як правило, їх імена починаються з префікса sp_. Вони призначені для підтримки функцій SQL Server (зокрема, процедур для роботи з переліком об’єктів сервера). До цього відноситься вибірка даних з системних таблиць зовнішніми застосуваннями, адміністрування бази даних і керування безпекою.

Наприклад, дуже часто адміністратору потрібно переглянути відомості про користувачів і процеси на сервері. Це передує закриттю бази даних або зупинці сервера. Для цього достатньо набрати на консолі Ms SQL Management Studio
sp_who



	spid	ecid	status	loginame	hostname	blk	dbname	cmd
1	1	0	background	sa		0	NULL	LAZY WRITER
2	2	0	sleeping	sa		0	NULL	LOG WRITER
3	3	0	background	sa		0	master	SIGNAL HANDLER
4	4	0	background	sa		0	NULL	LOCK MONITOR
5	5	0	background	sa		0	master	TASK MANAGER
6	6	0	background	sa		0	master	TASK MANAGER
7	7	0	sleeping	sa		0	NULL	CHECKPOINT SLEEP
8	8	0	background	sa		0	master	TASK MANAGER
9	9	0	background	sa		0	master	TASK MANAGER
10	10	0	background	sa		0	master	TASK MANAGER
11	11	0	background	sa		0	master	TASK MANAGER
12	12	0	background	sa		0	master	TASK MANAGER
13	13	0	background	sa		0	master	TASK MANAGER
14	51	0	runnable	sa	VPNOTE	0	RegTab_Adm	SELECT
15	52	0	sleeping	sa	VPNOTE	0	master	AWAITING COMMAND

Рисунок 2 - Процеси на сервері при відсутності навантаження: життя вирує

У SQL Server включені тисячі системних збережених процедур; у новій користувацькій базі відразу створюється біля 1200 системних збережених процедур.

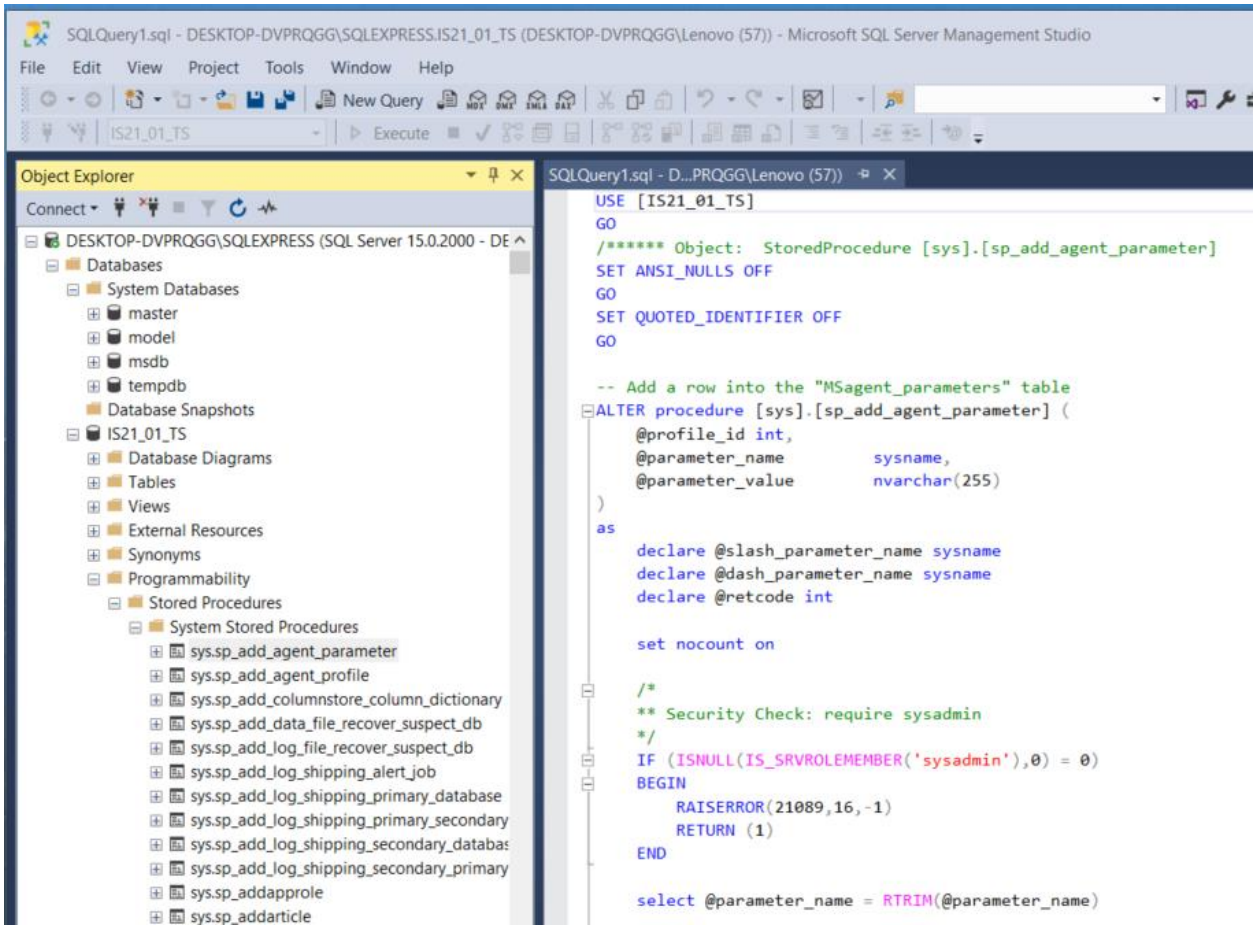


Рисунок 3 - Системні збережені процедури в БД Технічної служби. Тут їх приблизно 1200. У системній БД master їх приблизно 1400.

Локальні збережені процедури

Локальні збережені процедури зазвичай знаходяться в призначеній для користувача базі даних. Як правило, їх створюють прикладні програмісти для вирішення певних завдань, які відповідають бізнес-логіці прикладної системи, в її базі даних (у продуктивній базі).

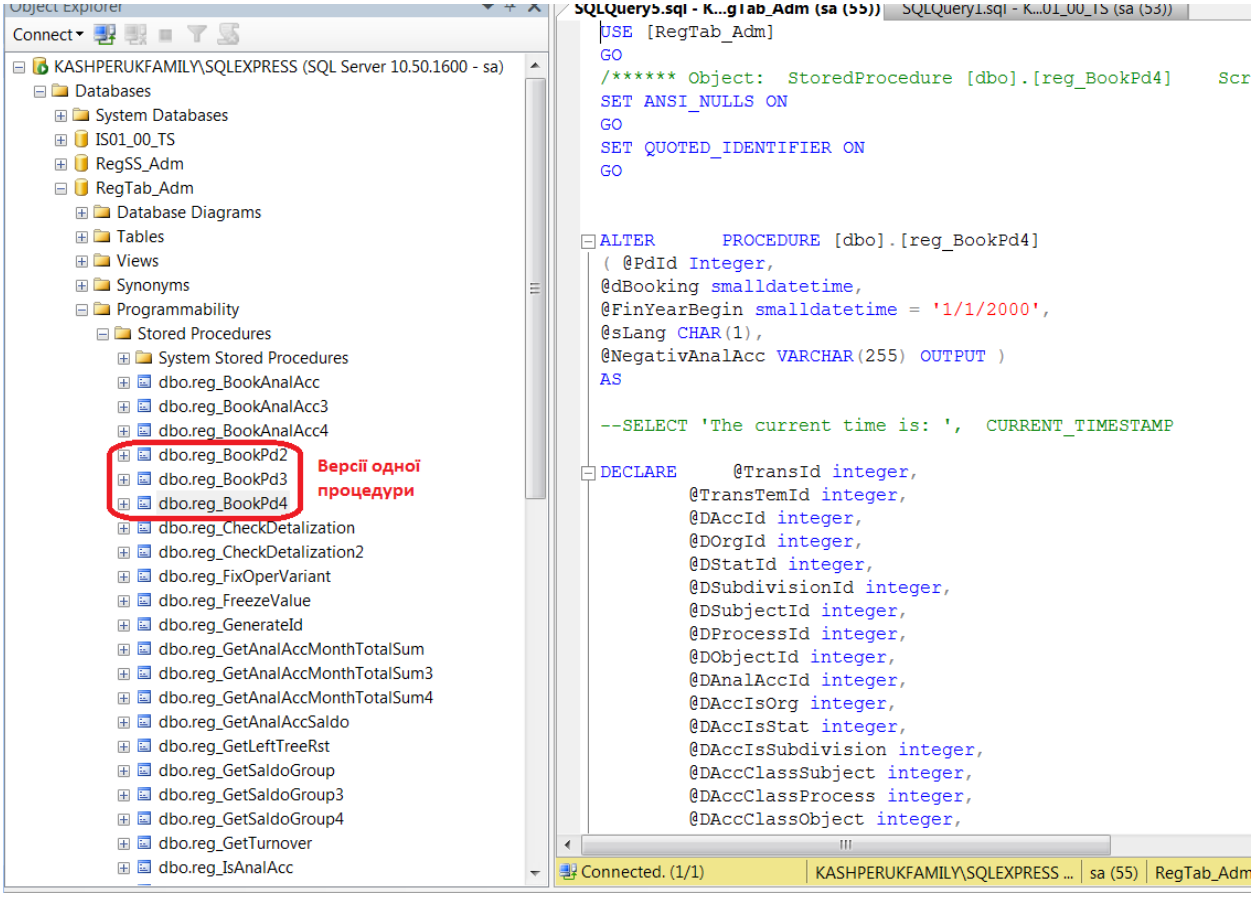


Рисунок 4 - Локальні збережені процедури прикладної системи

Тимчасові збережені процедури

Тимчасова збережена процедура схожа на локальну, однак вона існує лише до закриття з'єднання, в якому створена, або до завершення роботи SQL Server. Залежно від типу така процедура видаляється після завершення роботи сервера або розриву з'єднання. Мінливість обумовлена тим, що тимчасові процедури зберігаються в базі даних TempDB. Ця БД створюється лише в оперативній пам’яті, тому при кожному запуску сервера ця база створюється заново, а після закриття сервера всі об'єкти цієї БД зникають.

У TempDb зручно також створювати тимчасові таблиці (з префіксом ##) зі швидким доступом до них.

Щоб гарантувати унікальність імені тимчасової процедури, SQL Server додає до нього набір символів підкреслення і унікальний номер з'єднання. Тимчасова збережена процедура видаляється з TempDB при закритті з'єднання, в якому вона створена.

Розширені збережені процедури

Розширені процедури звертаються до зовнішніх програм, скомпільованих у вигляді DLL. Деякі системні збережені процедури також є розширеними. Угода про іменування передбачає використання в іменах розширених збережених процедур префікса xp_. Але інколи ця угода порушується.

Збережені процедури і CLR

MS SQL Server 2005 вводить нову можливість використання Common Language Runtime (CLR), що дозволяє програмісту розробляти різні об’єкти БД (збережені процедури,

функції, визначені користувачем, тригери, користувацькі агрегати і користувацькі типи) за допомогою C# та Visual Basic.

Для реалізації, компіляції, збереження та виконання CLR-процедур треба:

- Дозволити базі даних використовувати CLR, як-от:
```
USE sample;
EXEC sp_configure 'clr_enabled', 1
RECONFIGURE
```
- Написати процедуру на C# або Visual Basic і скомпілювати її.
- Використати оператор CREATE ASSEMBLY для створення виконуваного файлу.
- Зберегти процедуру як серверний об'єкт оператором CREATE PROCEDURE.
- Виконати процедуру, використовуючи оператор EXECUTE.

Використання збережених процедур

Виклик збереженої процедури

Для виклику користувацьких і системних збережених процедур використовується оператор EXECUTE. Якщо збережена процедура не вимагає параметрів, або якщо вона не повертає результат, синтаксис її буде дуже простим:

EXECUTE ім'я_процедури
Або
EXEC ім'я_процедури
Або
ім'я_процедури якщо виклик збереженої процедури є першим у пакеті.

Якщо збережена процедура приймає вхідні параметри, можна передати їх, вказавши позицію або ім'я. Щоб задати параметри в позиціях, потрібно просто вказати їх після імені процедури, відокремлюючи комами:

```
EXECUTE ім'я_процедури параметр [ , параметр ...]
```

Параметри також можуть бути передані збереженій процедурі шляхом явного зазначення їх імен. У цьому випадку можна задати параметри в будь-якому порядку. Синтаксис для виклику процедури з *іменованими параметрами*:

```
EXECUTE ім'я_процедури @ім'я _парам = значення [, @ім'я_парам = значення ...]
```

Наприклад:
```
EXECUTE sp_dboption @optname = 'read only', @dbname = 'Aromatherapy'
```

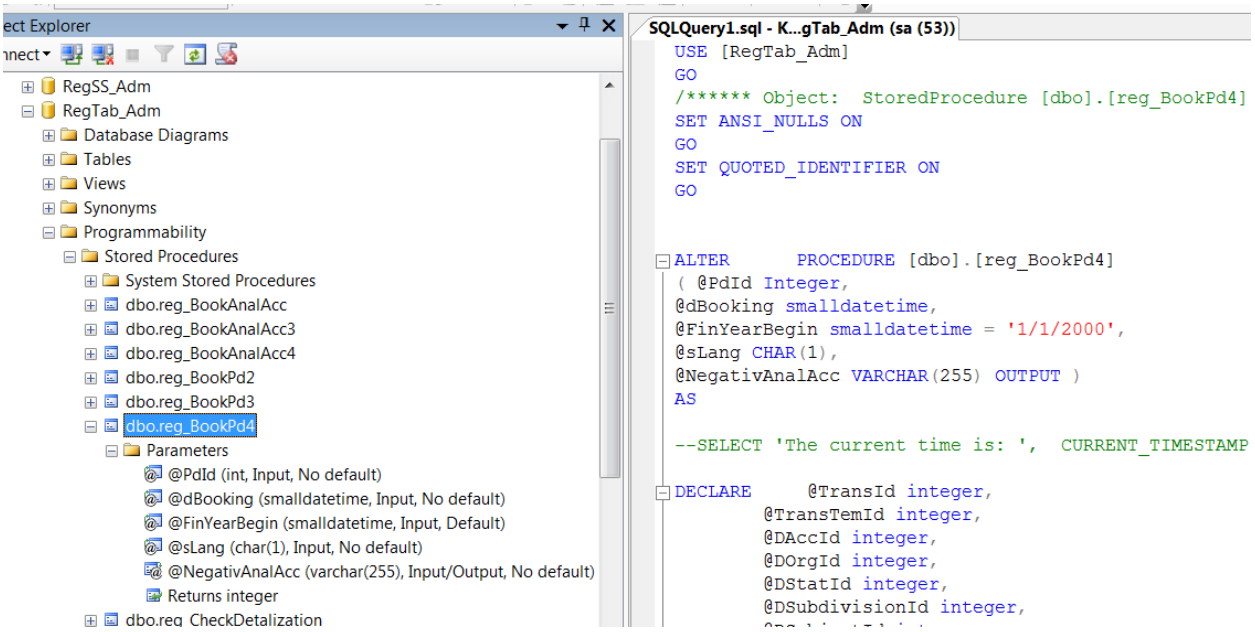


Рисунок 5 - Відображення інтерфейсу збереженої процедури у вікні Object Explorer SQL Management Studio

Панель Object Browser містить папку Stored Procedures для кожної бази даних, включаючи головну. Кожна збережена процедура, що міститься в списку, має папку Parameters. У цій папці в певному порядку розміщуються параметри збереженої процедури. Наприклад, дослідимо параметри системної збереженої процедури sp_helpdb, яка міститься як у системних, так і в користувацьких БД:

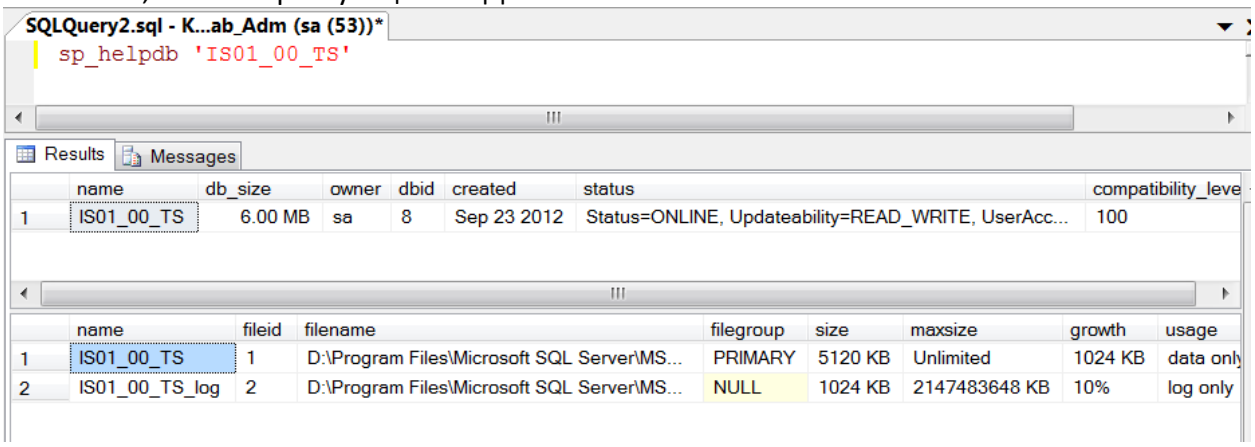


Рисунок 6 - Виклик і результати системної збереженої процедури sp_helpdb

Збережені процедури можуть також повертати дані назад через вихідні параметри. В цьому разі у процедури може бути кілька вихідних параметрів. Вихідні параметри повинні бути локальними змінними. Вони можуть бути задані або шляхом зазначення їх імен, або шляхом зазначення їх позицій, але при цьому після вихідного параметра має бути ключове слово OUTPUT.

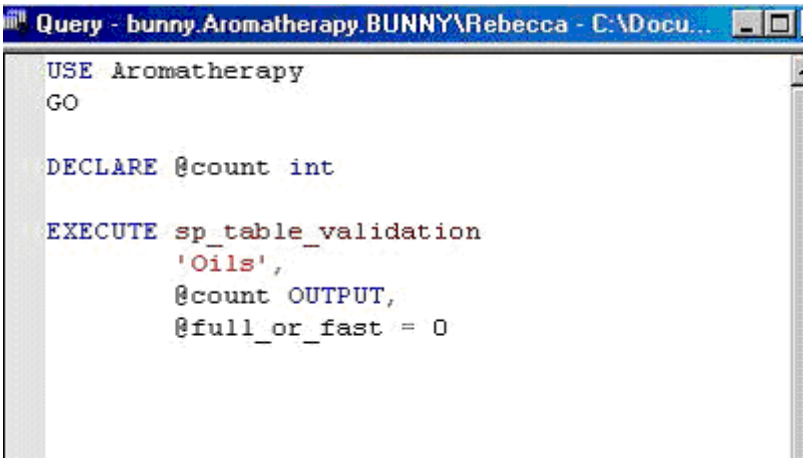


Рисунок 7 - Задання вихідного параметру збереженій процедурі

Якщо не параметр, а сама процедура повертає значення, синтаксис виклику є певним гібридом операторів EXECUTE і SET.

```
EXECUTE @ім'я_змінної = ім'я_процедури [, парам [, парам ...] ]
```

Якщо ви не вказали локальну змінну для прийому результатів, SQL Server просто відкине значення.

Створення збереженої процедури

Збережені процедури створюються з використанням різновиду оператора CREATE – CREATE PROCEDURE.

```
CREATE PROCEDURE ім'я_процедури
[список_параметрів]
AS
<оператори_процедури>
```

Можна створити тимчасову локальну або глобальну збережену процедуру, вказавши перед іменем процедури # або ## відповідно.

Збережені процедури можуть викликати інші збережені процедури. Максимальна глибина вкладеності становить 32.

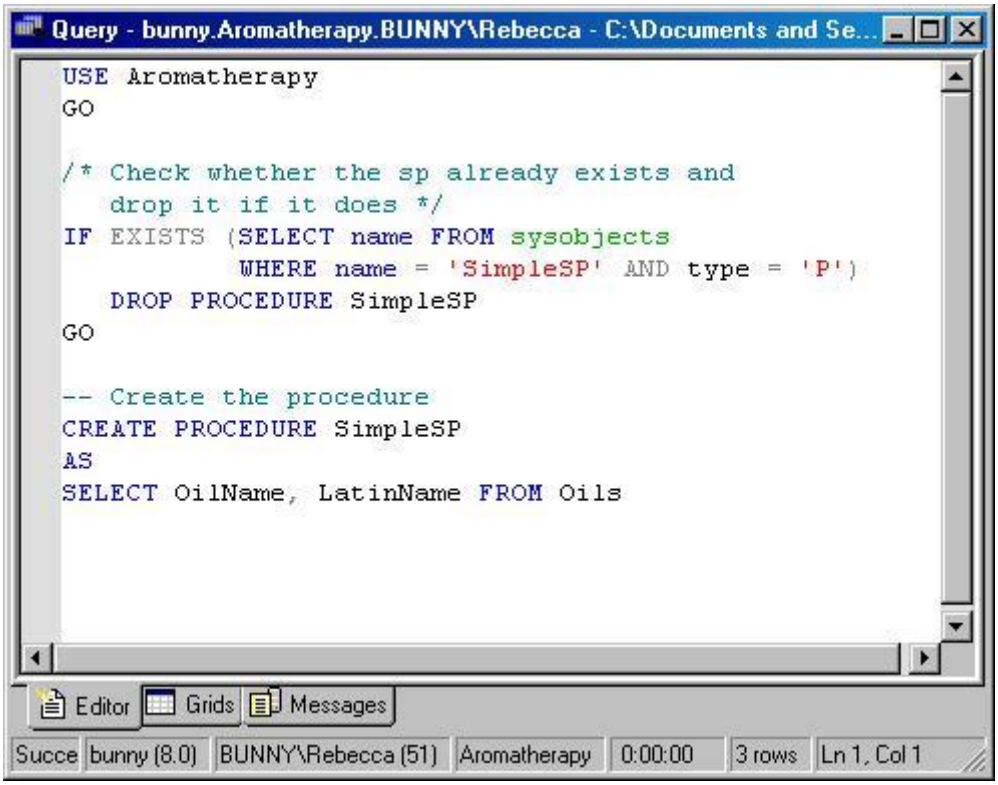


Рисунок 8 - Створення простої збереженої процедури. Якщо ця процедура вже існує (про що дізнаємося з системного розрізу sysobjects), спочатку вона видаляється.

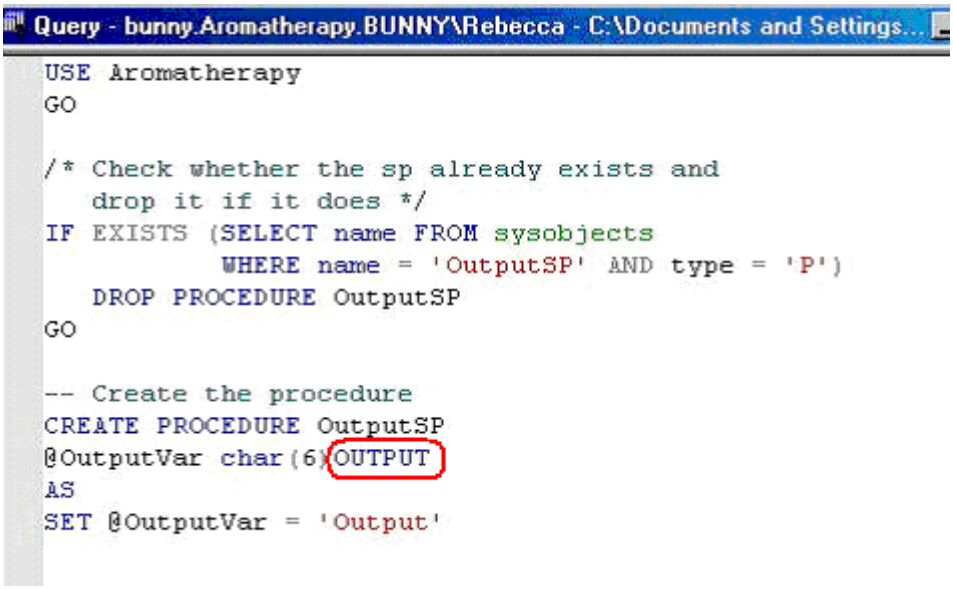


Рисунок 9 - Для задання вихідного параметру процедури треба в його описі поставити слово OUTPUT. В результаті виклику така процедура видає :

Рисунок 10 - Результат виконання збереженої процедури з попереднього прикладу

Використання збережених процедур для виконання об'ємних і тривалих операцій в базі даних

В системі управлінського і фінансового обліку класу ERP оперативна інформація, а саме первинні документи, а також залишки і обороти по бухгалтерських та інших рахунках зберігаються за кілька років (3-5). Приклад тривалої і об'ємної операції в такій системі: відкидання першого з минулих років (як мінімум, видалення первинних документів першого року – це можуть бути десятки або сотні тисяч ПД, мільйони рядків і проводок). Навіть створення заново об'ємної таблиці і копіювання туди даних зі старої таблиці може тривати годинами (півтора мільйона записів – півтори години).

В разі такої операції має бути індикація ходу процесу, як-от ProgressBar. Процедурний SQL не має інтерфейсу, тому не варто повністю покладати процес на збережену процедуру. Зовнішній цикл, що дозволяє відстежувати міру виконання процесу, має бути в прикладній мові програмування, аби виводити індикатор процесу. В циклі можна викликати збережену процедуру.

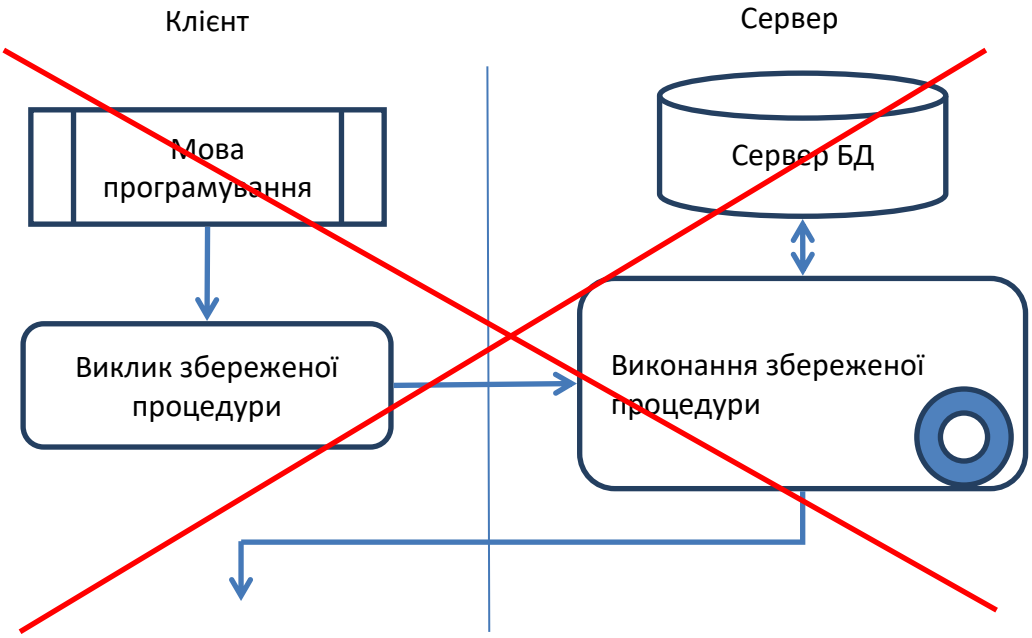


Рисунок 11 – Виконання тривалих операцій в БД, цикл у збереженій процедурі

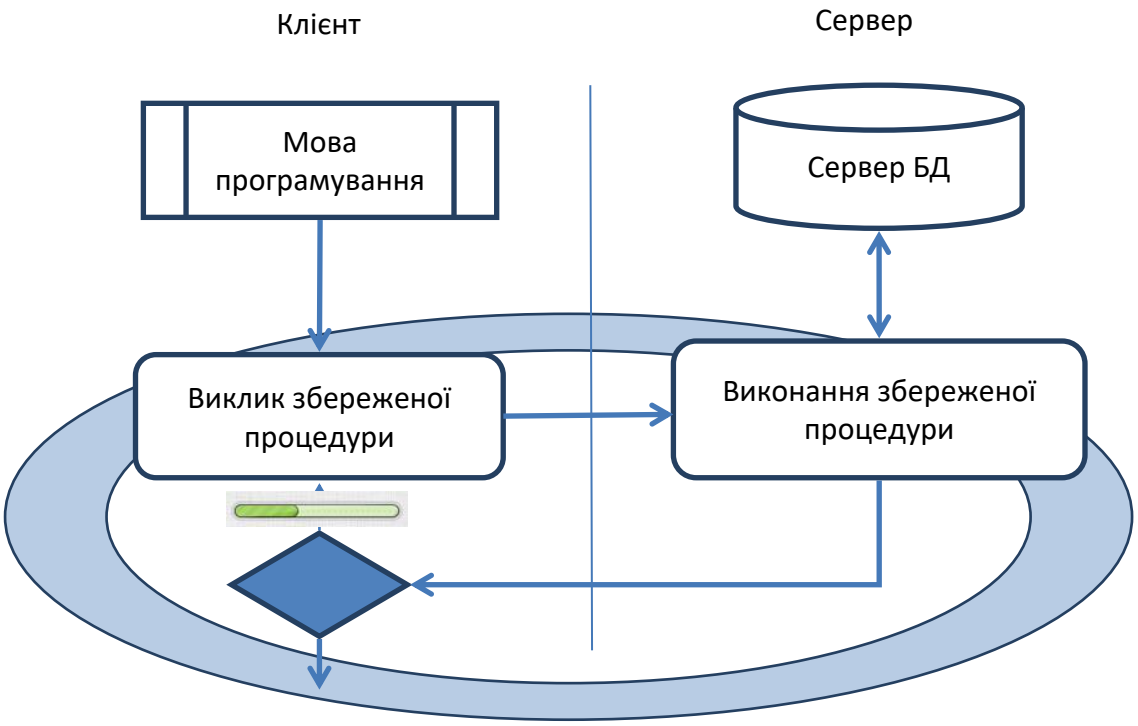


Рисунок 12 – Виконання тривалих операцій в БД, цикл у мові програмування

Динамічні запити та виключні ситуації

Динамічний SQL

У сімействі мов xBase (Dbase, Clipper, FoxPro), орієнтованих на обробку dbf-файлів, в результаті наступних операторів

```
a:=”2+2”
b:=&a
```

b отримує значення 4. Оператор & називають *макропідстановка* . Він означає *runtime-компіляція і виконання.*

У мові VBA (Visual basic for Applications) на відміну від VB – Visual Basic існує функція Eval:

```
Dim a As String
a=”2+2”
b=Eval(a)
```

b також отримує значення 4. Eval може обробляти вирази, що містять як вбудовані, так і користувацькі функції. Runtime-компіляція дозволяє радикально збільшити гнучкість програм.

Наприклад, особливістю процесу нарахування зарплати є наявність складних і час від часу змінюваних нарахувань та утримань, як-от податок (ПДФО), лікарняні, відпускні. У разі компіляції цих формул у єдиний програмний продукт цей продукт ніколи не досягне стабільності і завершеності.

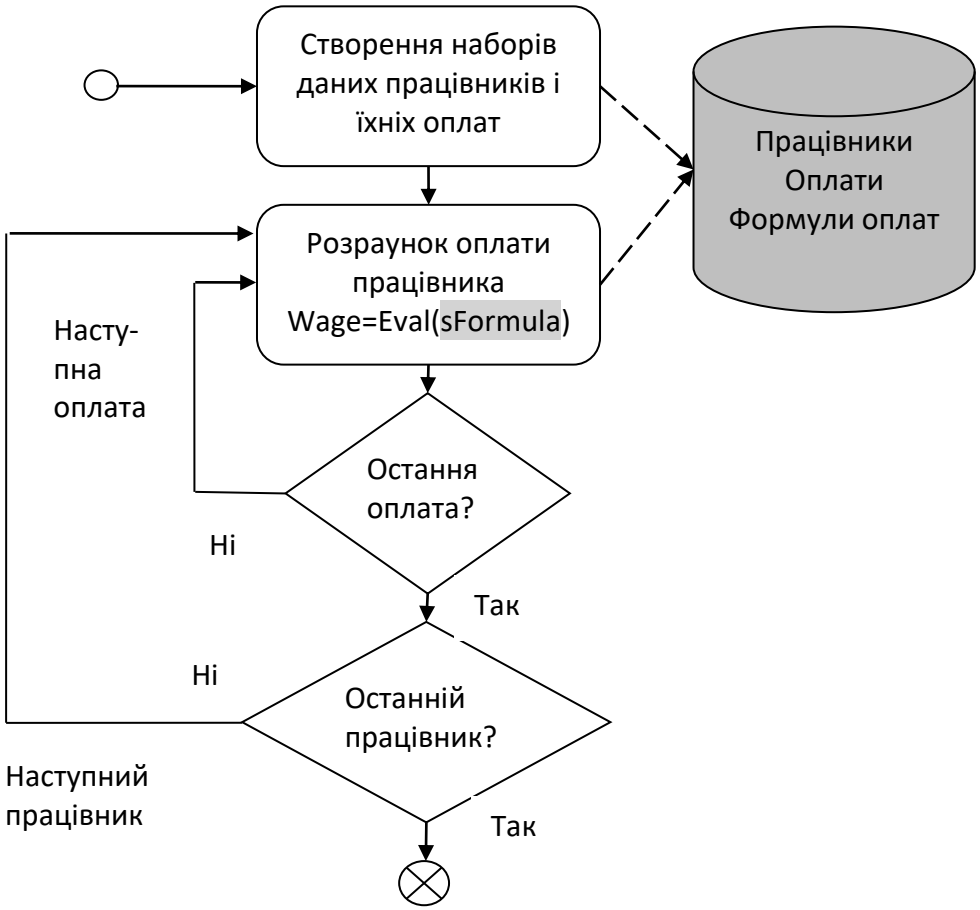


Рисунок 13 - Розрахунок зарплати з використанням Runtime-компіляції

Але Runtime-компіляція дуже витратна під час виконання: доводиться при компіляції зберігати у скомпільованому файлі не лише адреси змінних (функцій), а й їхні символічні імена, тобто компілювати в π-код.

У JavaScript також існує функція Eval(). Вважається, що вона створює проблеми для безпеки, дозволяючи безконтрольно запустити довільний код на робочій станції. Тому рекомендують запускати цю функцію через додатковий засіб контролю, як-от функцію SendBox(), яка інкапсулює виклик Eval(), контролюючи відсутність небезпечних викликів.

У SQL аналогічна можливість має назву *динамічний SQL.* У Transact SQL її забезпечує команда EXECUTE, а також системна збережена процедура sp_executesql.

EXEC [UTE] **(** { *@string_variable* | [N] '*tsql_string*' } [**+** ...*n*] **)**

sp_executesql [**@stmt =**] *stmt*
[
  {**,** [**@params =**] **N'@***parameter_name data_type* [**,**...*n*]**'** }
  {**,** [**@***param1* **=**] '*value1*' [**,**...*n*] }
]

Ця процедура виконує вираз або цілий пакет на Transact-SQL. На жаль, у пакеті динамічного SQL невідомі користувацькі змінні викликаючого пакету. Натомість цей вираз або пакет може мати включені параметри. Кожен параметр, що входить в stmt, повинен

мати відповідний вхід як у списку визначення параметрів @params, так і у списку значень параметрів. Компіляція рядка, що містить SQL-вираз, відбувається під час виконання команди EXEC або процедури sp_executesql.

Приклад 1. Найпростіший приклад:

```
DECLARE @proc_name varchar(30)
SET @proc_name = 'sp_who'
EXEC @proc_name
```

Приклад 2. Складніший приклад: наступний код Transact-SQL створить курсор з іменем crsrProducts:

```
DECLARE @Color nvarchar(15)
SET @Color = 'Black'
DECLARE @sql nvarchar(255)
SELECT @sql = 'DECLARE crsrProducts CURSOR FAST_FORWARD FOR ' +
        'SELECT ProdNumber, ListPrice, StandardCost ' +
        'FROM Production.Product ' +
        'WHERE Color = ''' + @Color + ''';' +
        'OPEN  crsrProducts ' +
        'WHILE (@@FETCH_STATUS = 0) ' +
        'BEGIN ' +
        '   FETCH NEXT FROM crsrProducts ' +
        'END; ' +
        'CLOSE crsrProducts ' +
        'DEALLOCATE crsrProducts '
EXEC sp_executesql @sql
```

Аналогічно, у PL/SQL динамічний SQL реалізується за допомогою команди EXECUTE IMMEDIATE.

Приклад 3. В ньому у PL/SQL спочатку створюється таблиця, а потім у неї вставляються дані. Без динамічного SQL цей фрагмент навіть не скомпілювався би: адже на момент компіляції таблиця temp_table була би відсутня. Рятує runtime-компіляція.

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE temp_table (
num_value NUMBER,
char_value CHAR(10))';
EXECUTE IMMEDIATE 'INSERT INTO temp_table (num_value, char_value)
VALUE(10, "Hello")';
END;
```

Виключні ситуації

Під *виключною ситуацією* розуміємо подію часу виконання (runtime), яка потребує переривання виконання передбаченої послідовності операцій і негайної реакції. Це може бути помилка, яка генерується при невдалому виконанні команд SQL, як-от у разі неможливості вставити запис у таблицю з-за неунікального значення ключа нового запису.

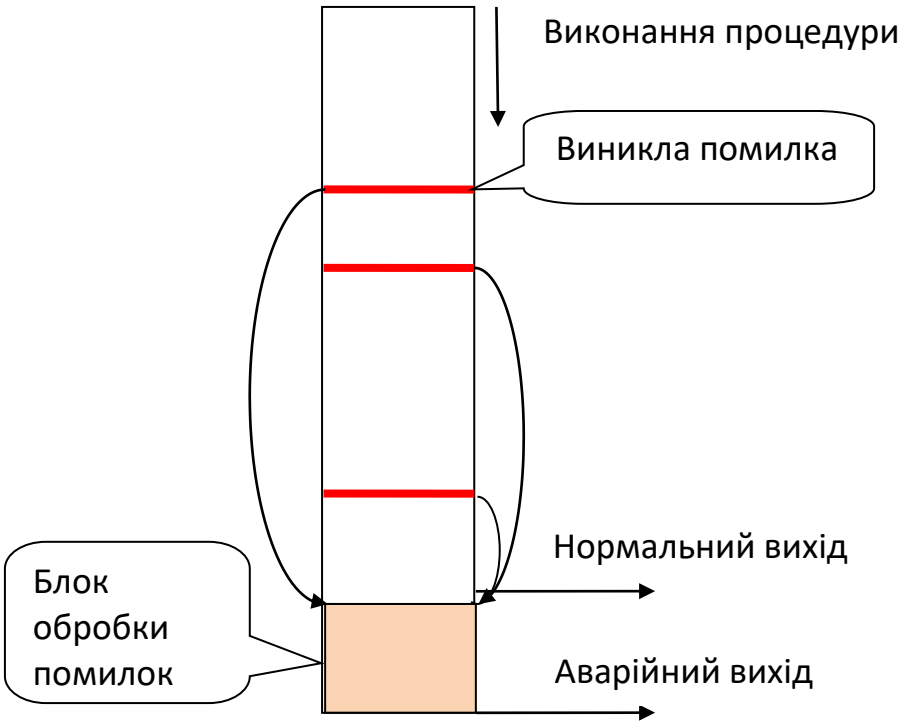


Рисунок 14 - Типова обробка помилок у процедурі

Використання системної змінної @@ERROR для виявлення помилки

У TransactSQL системна змінна @@ERROR повертає номер помилки, яка спричинена попередньо виконаною інструкцією. Якщо помилки не було, змінна повертає 0. Треба перевіряти її значення після *кожної* інструкції, яка може викликати помилку.

Приклад 60.

```
BEGIN TRAN;
        INSERT T1 (Col) VALUES (1);
IF (@@ERROR <> 0) GOTO ErrorHandler;
        INSERT T1 (Col) VALUES (2);
IF (@@ERROR <> 0) GOTO ErrorHandler;
        INSERT T1 (Col) VALUES (3);
IF (@@ERROR <> 0) GOTO ErrorHandler;
COMMIT TRAN;
RETURN;
ErrorHandler: ROLLBACK TRAN; RETURN;
```

Як бачимо, код виходить громіздкий, в ньому багато операторів GOTO.

Системний розріз з користувацької БД sys.messages містить описи кодів помилок.

Використання сеансового параметру XACT_ABORT для обробки помилок

Цей параметр визначає, чи відміняти всю транзакцію при виникненні помилки в одному з її операторів (ON), або ігнорувати помилковий оператор, переходячи на наступний (OFF).

Приклад 61. Розглянемо таблицю *CREATE TABLE Test.Atable (ID INT PRIMARY KEY);*
Нехай параметр XACT_ABORT установлений у стандартнее значення OFF.

```
SET XACT_ABORT OFF; BEGIN TRAN;
        INSERT Test.ATable (ID) VALUES (1);
        INSERT Test.ATable (ID) VALUES (1);        --Невдача!
        INSERT Test.ATable (ID) VALUES (2);        --Виконання продовжується
COMMIT TRAN;
```

В результаті в таблицю вставлено 2 записи і транзакція зафіксована.

```
SET XACT_ABORT ON; BEGIN TRAN;
        INSERT Test.ATable (ID) VALUES (1);
        INSERT Test.ATable (ID) VALUES (1);        --Невдача!
        INSERT Test.ATable (ID) VALUES (2);        --Не виконується
COMMIT TRAN;
```

В результаті транзакція відмінена і в таблицю не вставлено жодного запису.

Параметр XACT_ABORT особливо доцільно використовувати в розподілених транзакціях, які розповсюджуються на кілька віддалених серверів (див.тему 10). Для цих транзакцій XACT_ABORT має бути ON, оскільки зазвичай в цьому разі ризик аварійного завершення є значним з-за проблем у мережі.

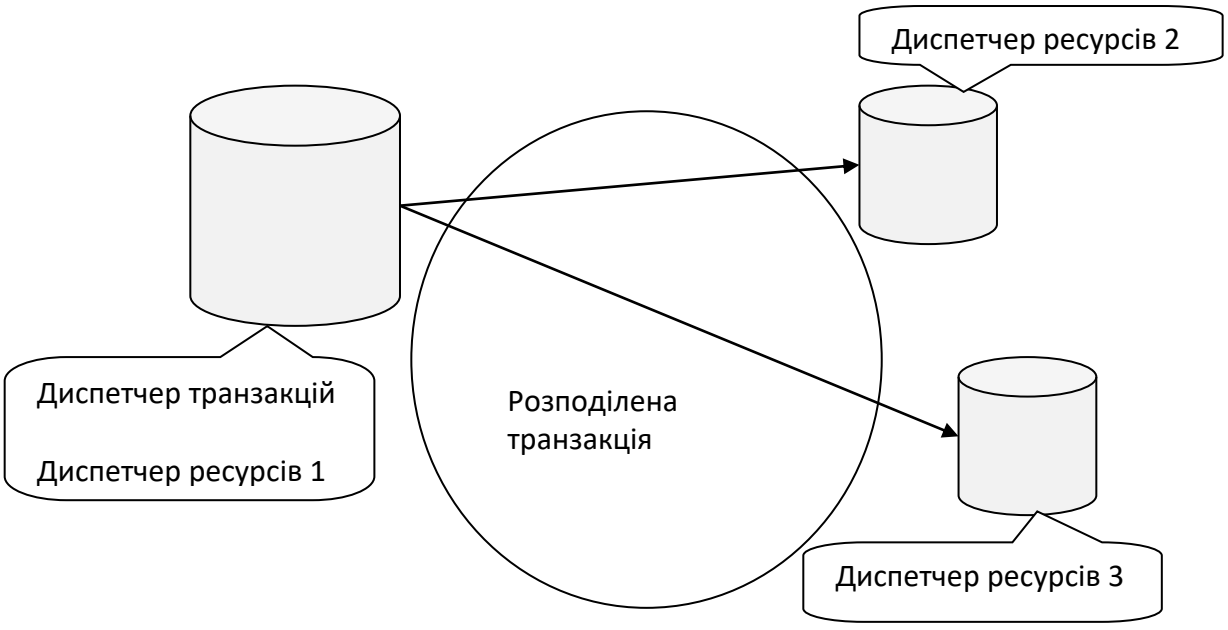


Рисунок 15 - Виконання розподіленої транзакції. Якщо виникає відмова в одній з гілок мережі, відкочується вся транзакція.

Структурована обробка помилок за допомогою блоку try/catch

Починаючи з версії MsSQL2005 існує можливість обробки помилок за допомогою блоку try/catch. Принципи його роботи:

- Помилка, що виникла в блоці *try* (спробувати), або виконання інструкції RAISEERROR (штучне створення помилки), тягне передачу управління асоційованому з ним блоку *catch* (зловити).
- Асоційований блок *catch* має йти безпосередньо після свого блоку *try.*
- У блоці *catch* транзакція може бути або зафіксована, або відмінена (або залишена без змін).
- Помилка, яка виникла у блоці *catch,* тягне передачу помилки застосуванню, що викликало процедуру, далі пакет продовжить виконання. Якщо блок *try/catch* вкладено в зовнішній блок *try*, управління передається найближчому зовнішньому блоку *catch.* Ця поведінка є типовою для вкладених збережених процедур.
- Помилки з рівнем серйозності 20 і більше є фатальними, вони тягнуть відкати всіх відчинених транзакцій, зупинку виконання пакету і розрив з'єднання.

Приклад 62. Застосуємо блок *try/catch* до тої ж таблиці Test.ATable.

```
TRUNCATE TABLE Test.ATable;    --Видалити всі кортежі ATable
BEGIN TRY
        BEGIN TRAN;
                INSERT Test.ATable (ID) VALUES (1);
                INSERT Test.ATable (ID) VALUES (1); --Невдача!
                INSERT Test.ATable (ID) VALUES (2);
        COMMIT TRAN;
END TRY
BEGIN CATCH
        ROLLBACK TRAN;
        RAISEERROR('An error occurred! ', 16, 1);
END CATCH
```

Фрагмент виконується так:

- Перша вставка виконується.
- Друга вставка: збій з помилкою 2627, інструкцію відмінено, управління передається блоку *catch.*
- Інструкція RAISEERROR відправляє повідомлення застосуванню, що викликало процедуру.
- Далі виконуються інструкції, що йдуть після блоку CATCH, якщо помилки не трапилось.

Функціональність блоку *try/catch* подібна до типових реалізацій *try/catch* у мовах C++, C#, J#, Java, Jscript, Visual Basic.NET та ін.

Ініціювання виключних ситуацій в Oracle

Коли виникає помилка, пов'язана з деякою виключною ситуацією, ініціюється (встановлюється) ця виключна ситуація. Виключні ситуації, що визначаються користувачем, встановлюються явно за допомогою оператора RAISE, тоді як стандартні виключні ситуації ініціюються неявно при виникненні відповідних помилок Oracle.

Приклад 63. Якщо кількість студентів у групі більше припустимої кількості, створимо користувацьку виключну ситуацію.

```
DECLARE
e_TooManyStudents EXCEPTION;
- Поточне число студентів, зареєстрованих в HIS-101
v_CurrentStudents NUMBER(3);
- Максимальне число студентів, припустиме в HIS-101
v_MaxStudents NUMBER(3);
BEGIN
      /* Читаємо дані одного кортежу */
      SELECT current_students, max_students
            INTO v_CurrentStudents, v_MaxStudents
            FROM classes
            WHERE department = 'HIS' AND course = 101;
      /* Порівняємо отримані значення. */
      IF v_CurrentStudents > v_MaxStudents THEN
            /* Зареєстровано забагато студентів – встановимо виключну ситуацію
            */
            RAISE e_TooManyStudents;
      ENF IF;
END;
```

При виявленні виключної ситуації управління відразу ж передається розділу виключних ситуацій блоку. Якщо такого розділу немає, виключна ситуація передається блоку (поширюється на блок), в який входить даний блок.

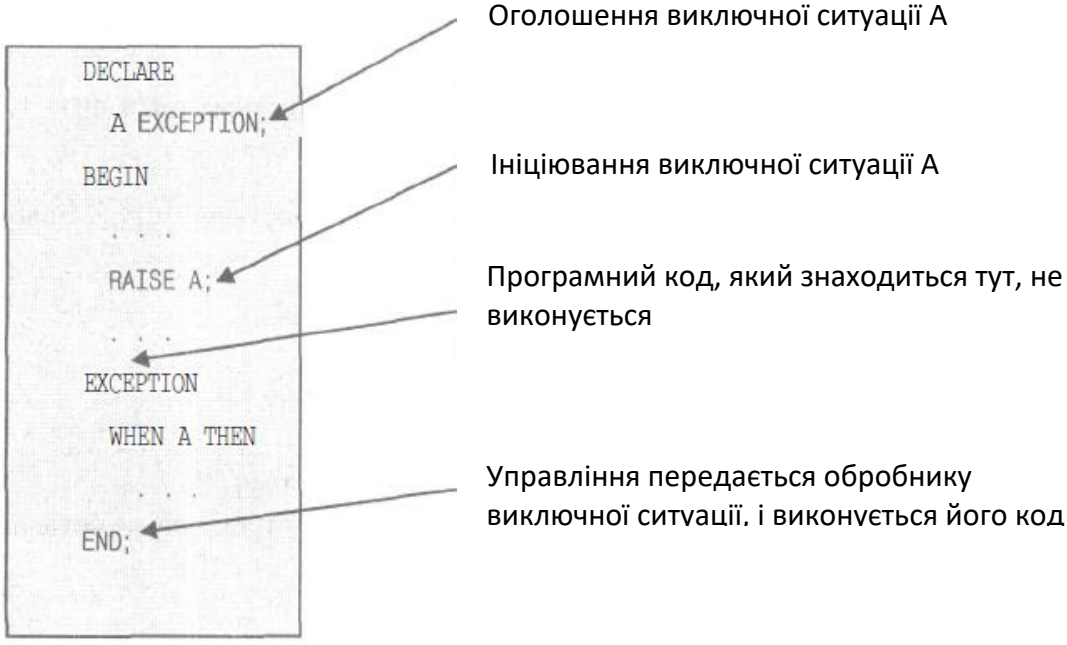


Рисунок 16 - Передача управління обробнику виключної ситуації в Oracle PL/SQL

Синтаксис обробника виключних ситуацій:

EXCEPTION
WHEN *ім'я виключної_ситуації1* **THEN**
послідовність_операторів1;
WHEN *ім'я_виключної_ситуації 2* **THEN**
послідовність_операторів2;
[**WHEN** OTHERS **THEN**
послідовність_операторів3;]
END;

Кожний обробник складається з умови WHEN (коли) і операторів, що виконуються при встановленні виключної ситуації. У WHEN вказується, для якої виключної ситуації призначений даний обробник.

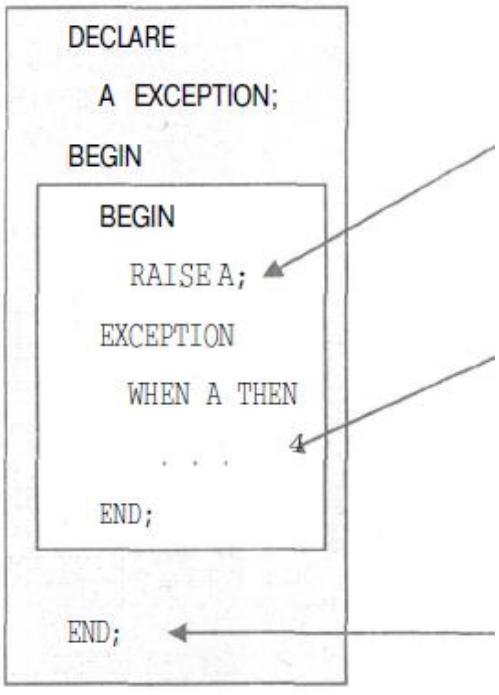
WHEN OTHERS перехоплюватиме всі виключення, системні та визначені користувачем. Рекомендується вказувати обробник OTHERS на найвищому рівні програми (у самому зовнішньому блоці) для забезпечення розпізнавання всіх можливих помилок. Інакше помилка поширюватиметься в середовище, з якого викликана процедура. Це може привести до небажаних наслідків, як-от відкат поточної транзакції.

Після передачі управління обробнику виключної ситуації неможливо повернутись у розділ блоку, де ця ситуація була створена.

Розповсюдження виключних ситуацій

Якщо виключна ситуація виникає у виконуваному розділі блоку PL/SQL, то:

1) Якщо в поточному блоці є обробник даної виключної ситуації, він виконується, блок успішно завершується і управління програмою передається вищестоящому блоку.
2) Якщо обробник відсутній, виключна ситуація передається у вищестоящий блок і ініціюється там. Після цього у вишестоящему блоці виконується крок 1). Якщо вищестоящого блоку не існує, виключна ситуація буде передана середовищу, яке викликало программу PL/SQL, як-от SQL*Plus.



Виключна ситуація породжується у внутрішньому блоці

Обробляється також у внутрішньому блоці

Сюди передається управління, якщо виключна ситуація не виникла, або вона успішно оброблена в блоці EXCEPTION

Рисунок 17 - Виключна ситуація A породжується і обробляється у внутрішньому блоці. Після цього управління програмою передається зовнішньому блоку.

Рисунок 18 – Дії в разі відсутності обробника виключної ситуації C