МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Волинський національний університет імені Лесі Українки
Кафедра комп'ютерних наук та кібербезпеки

Л. В. Булатецька, В. В. Булатецький

ТРАНЗАКЦІЇ В SQL

Тестові завдання з нормативних навчальних дисциплін "Бази даних та розподілені інформаційно-аналітичні системи" та "Організація баз даних та знань"

Луцьк 2021

УДК 004.655

Рекомендовано до видання науково-методичною радою
Волинського національного університету імені Лесі Українки
(протокол №8 від 22 квітня 2021 р.)

Рецензенти:
Собчук О. М. – кандидат пед. наук, доцент кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки

Костючко С. М. – кандидат технічних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету

Транзакції в SQL [Електронний ресурс] : тестові завдання з нормативних навчальних дисциплін "Бази даних та розподілені інформаційно-аналітичні системи", "Організація баз даних та знань" / Л. В. Булатецька, В. В. Булатецький; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 632 КБ). – Луцьк : ВНУ ім. Лесі Українки, 2020. – 41 с.

Викладено теоретичні відомості, які відносяться до теми "Транзакції в SQL", що входить до лекційного курсу нормативних навчальних дисциплін "Бази даних та розподілені інформаційно-аналітичні системи" та "Організація баз даних та знань". Наведено приклади розв'язків типових задач, запропоновано тестові завдання для самостійного розв'язку та для самостійної підготовки до контрольної роботи по темі "Транзакції в SQL". Рекомендовано студентам 2, 3 курсів, спеціальності 122 Комп'ютерні науки, 014 Середня освіта (Інформатика), 113 Прикладна математика та 125 Кібербезпека, освітньо-професійних програм Комп'ютерні науки та інформаційні технології, Середня освіта. Інформатика, Прикладна математика та Інформаційна безпека відповідно.

УДК 004.655

ВСТУП

“Бази даних та розподілені інформаційно-аналітичні системи” є нормативною навчальною дисципліною підготовки бакалавра напрямів 122 Комп’ютерні науки та інформаційні технології, 014 Середня освіта (Інформатика), 113 Прикладна математика. “Організація баз даних та знань” є нормативною навчальною дисципліною підготовки бакалавра напряму 125 Кібербезпека.

В програмі навчальних дисциплін “Бази даних та розподілені інформаційно-аналітичні системи” та “Організація баз даних та знань” велика увага приділяється теорії реляційних баз даних. При вивчені теорії реляційних баз даних неможливо обійтися без матеріалу, який відноситься до мови запитів SQL. Однією з тем, які розглядаються при вивченні мови запитів SQL є вивчення транзакцій. У даній методичній розробці, розглянуто поняття транзакцій в мові SQL, розглянуто, яким чином можна організувати паралельний доступ до даних та основні аномалії які можуть виникати при паралельному доступі до даних. Для кращого розуміння студентами теоретичного матеріалу по даній темі, виклад супроводжується прикладами ситуацій, які виникають при виконанні транзакцій та поясненнями їх розв’язання. Для контролю якості знань студентів, запропоновано приклади тестових запитань.

1. ПОНЯТТЯ ТРАНЗАКЦІЇ В SQL

Транзакція – це дія або серія дій, які здійснюють доступ або зміну вмісту бази даних і розглядаються СУБД як єдине ціле. Транзакція є логічною одиницею роботи, виконуваної в базі даних. Вона може бути представлена окремою програмою, бути частиною алгоритму програми або навіть окремою командою (наприклад, командою INSERT або UPDATE) і включати довільну кількість операцій, які виконуються в базі даних. Будь-яка транзакція завжди повинна переводити базу даних з одного узгодженого стану в інший, хоча допускається, що узгодженість стану бази буде порушуватися в ході виконання транзакції. Транзакція складається з будь-яких операторів SQL, які можуть змінити базу даних. У SQL: 2003 не передбачений спеціальний оператор початку транзакції. У системах, сумісних з SQL: 2003, якщо ніяка транзакція ще не розпочата, а на виконання відправляється оператор SQL, то автоматично створюється нова транзакція. Наприклад, оператори

CREATE TABLE (створити таблицю), UPDATE (оновити) I SELECT (Вибрати) виконуються тільки в транзакції.

Отже, оператора початку транзакції може і не бути. Однак для її завершення передбачені два оператора:

- COMMIT – завершити виконання, коли виконання всіх операторів транзакції відбувається послідовно в єдиному блоці;
- ROLLBACK – відкат, коли відбувається скасування дії всіх операторів транзакції, і база даних повертається в початковий стан, в якому вона перебувала до початку транзакції.

Оператор COMMIT застосовується тоді, коли всі SQL-оператори транзакції імовірно виконані (сприйняті СКБД і не викликали помилок) і потрібно підтвердити зміни, внесені в базу даних. Якщо при виконанні оператора COMMIT станеться системний збій або помилка, можна виконати відкат транзакції, а потім спробувати виконати її знову.

Оператор ROLLBACK застосовується тоді, коли необхідно відмінити зміни, внесені транзакцією, і відновити базу даних в попередній стан. Якщо при виконанні оператора ROLLBACK станеться системний збій, то після перезавантаження оператор ROLLBACK можна виконати знову, і він повинен відновити базу даних.

Додаток може складатися з декількох транзакцій. Якщо не зазначений явний оператор початку транзакції, то перша транзакція починається на самому початку додатка. Остання транзакція закінчується в кінці додатку. Для вказівки кінця транзакції використовується оператор COMMIT або ROLLBACK.

Існують деякі властивості, які має будь-яка з транзакцій.

Атомарність. Це властивість типу “все або нічого”. Будь-яка транзакція являє собою неподільну одиницю роботи, яка може бути або виконана вся цілком, або не виконана зовсім.

Узгодженість. Кожна транзакція повинна переводити базу даних з одного узгодженого стану в інший узгоджений стан.

Ізольованість. Всі транзакції виконуються незалежно одна від іншої. Інакше кажучи, проміжні результати незавершеної транзакції не повинні бути доступні іншим транзакціях.

Довговічність. Результати успішно завершеної (зафіксованої) транзакції повинні зберігатися в базі даних постійно і не повинні бути загублені в результаті подальших збоїв.

2. СУБТРАНЗАКЦІІ

Починаючи з SQL: 1999 транзакції можуть складатися з декількох субтранзакцій. Субтранзакції відокремлюються одна від одної так званими точками відкату, які задаються за допомогою оператора SAVEPOINT (точка збереження, відкату). Він може використовуватися разом з оператором ROLLBACK. ЯКЩО раніше ROOLLBACK застосовувався для скасування всієї транзакції, то тепер його можна використовувати для скасування тільки тієї частини транзакції, яка розташована між ROLLBACK і точки відкату.

Субтранзакції використовуються не для того, щоб частково відновити базу даних у випадку виникнення помилки (в цьому випадку необхідно виконати відкат всієї транзакції). Бувають ситуації, коли виконується складна послідовність дій з безліччю варіантів вибору. На якомусь етапі за отриманими результатами ви можете вирішити, що слід піти по іншому шляху, а для цього доведеться повернутися на декілька кроків назад. Якби не субтранзакції, то довелося б відміняти всю транзакцію повністю.

Точка відкату позначається за допомогою наступного оператора:

SAVEPOINT ім’яТочкиВідкату;

Відкат транзакції до цієї точки виконується за допомогою оператора:

ROLLBACK TO SAVEPOINT ім’яТочкиВідкату;

Розглянемо приклади.

Приклад 1.

Дано таблицю EXAMPLE в базі даних:

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай транзакція T виконується, в порядку поданому в таблиці:

Крок	Транзакція T
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE example SET dat=100 WHERE dat=101;
8	COMMIT;

Потрібно вияснити, що буде з даними dat для id=1 після завершення транзакції.

Пояснення. На першому кроці виконання транзакції відбувається зміна значення dat на 101 для id=1. На другому кроці встановлюється точка відкату Point1. На третьому кроці значення dat змінюється на 102 для id=1. На четвертому кроці встановлюється точка відкату Point2. На п’ятому кроці значення dat змінюється на 103 для id=1. На шостому кроці виконується відкат до точки збереження Point2. Тобто виконується відміна всіх операцій, які виконувалися після точки відкату Point2. На даному кроці значення dat стає рівним 102 для id=1. На сьомому кроці подається запит на зміну значення dat на 101, якщо воно було рівне 100. В таблиці на даному кроці виконання транзакції немає значення dat яке рівне 101, тому змін ніяких не відбувається в таблиці. В результаті, після завершення транзакції на кроці 8, значення dat=102 для id=1.

Приклад 2.

Дано таблицю EXAMPLE в базі даних:

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай транзакція T виконується, в порядку поданому в таблиці:

Крок	Транзакція T
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE example SET dat=102 WHERE dat=104;
9	COMMIT;

Потрібно вияснити, що буде з даними dat для id=1 після завершення транзакції.

Пояснення. На першому кроці виконання транзакції відбувається зміна значення dat на 101 для id=1. На другому кроці встановлюється точка відкату Point1. На третьому кроці значення dat змінюється на 102 для id=1. На четвертому кроці встановлюється точка відкату Point2. На п'ятому кроці значення dat змінюється на 103 для id=1. На шостому кроці виконується відкат до точки збереження Point1. Тобто виконується відміна всіх операцій, які виконувалися після точки відкату Point1. На даному кроці значення dat стає рівним 101 для id=1. На сьомому кроці виконується відкат до точки збереження Point2. Але точка збереження Point2 не збереглася, так як її створення було відмінено на шостому кроці. Тому на даному кроці значення dat рівне 101. На восьмому кроці подається запит на зміну значення dat на 104, якщо воно було рівне 102. В таблиці на даному кроці виконання транзакції немає значення dat яке рівне 102, тому змін ніяких не відбувається в таблиці. В результаті, після завершення транзакції на дев'ятому кроці, значення dat=101 для id=1.

Приклад 3.

Дано таблицю EXAMPLE в базі даних:

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай транзакція T виконується, в порядку поданому в таблиці:

Крок	**Транзакція T**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE example SET dat=102 WHERE dat=104;
9	ROLLBACK;

Потрібно вияснити, що буде з даними dat для id=1 після завершення транзакції.

Пояснення. На дев'ятому кроці відбувається відкат всієї транзакції, тому, немає значення які дії з таблицею виконувалися під час виконання транзакції, вони всі були відмінені. Тому після завершення транзакції значення dat залишилося рівним 100 для id=1.

3. ВИЗНАЧЕННЯ ПАРАМЕТРІВ ТРАНЗАКЦІЇ

Для транзакції можна встановити параметри – режим і рівень ізоляції. З цією метою використовується оператор:

SET TRANSACTION Режим, ISOLATION LEVEL Рівень_Ізоляції;

Режим може приймати наступні два значення:

- READ WRITE (читання-запис) – значення за замовчуванням;
- READ ONLY (тільки читання).

Рівень ізоляції транзакції може приймати чотири значення:

- SERIALIZABLE (послідовне виконання) – значення за замовчуванням;
- REPEATABLE READ (повторне читання);
- READ COMMITED (підтверджене читання);
- READ UNCOMMITED (непідтверджене читання).

Транзакція має параметри за замовчуванням: READ WRITE і SERIALIZABLE. Режим READ WRITE дозволяє вносити зміни в базу даних, а рівень ізоляції SERIALIZABLE є найбільш безпечним. Якщо ж необхідно задати інші значення, то слід використовувати оператор SET TRANSACTION. При цьому він записується або на початку додатка, визначаючи першу транзакцію, або після оператора COMMIT АБО ROLLBACK, визначаючи наступну транзакцію. Якщо ж після COMMIT АБО ROLLBACK не викликати оператор SET TRANSACTION, то наступна транзакція буде мати параметри, прийняті за замовчуванням.

Розглянемо приклади.

Приклад 4.

Дано таблицю EXAMPLE в базі даних:

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Для транзакції T встановлено режим READ ONLY.

SET TRANSACTION READ ONLY;

Нехай транзакція T виконується, в порядку поданому в таблиці:

Крок	**Транзакція T**
1	UPDATE example SET dat=101 WHERE id=1;
2	INSERT INTO example VALUES (12, 210);
3	COMMIT;

Потрібно вияснити, що буде з даними в таблиці EXAMPLE після завершення транзакції.

Пояснення. Під час виконання даної транзакції ніяких змін в таблиці бази даних не відбудеться, так, як дана транзакція має режим тільки читання. При даному режимі не можуть виконуватися операції insert/delete/update. Транзакція може тільки читати дані в межах її виконання.

4. РІВНІ ІЗОЛЯЦІЇ ТРАНЗАКЦІЙ

Рівень ізоляції транзакції в багатокористувацькій системі визначається через відсутність певної аномалії доступу до бази даних, яка може в кінцевому результаті загрожувати цілісності даних. Існують класичні аномалії при паралельному виконанні транзакцій.

Втрачені зміни. Транзакція T1 читає дані. Транзакція T2 читає ті самі дані. T1 на основі прочитаного значення обчислює нове значення даних записує його в базу даних і завершується. T2 на основі прочитаного значення обчислює нове значення даних записує його в базу даних і завершується. В результаті значення, записане транзакцією T2, "витре" значення записане транзакцією T1.

Брудне читання. Транзакція T1 змінює деякі дані, але ще не завершується. Транзакція T2 читає ті ж дані (з змінами внесеними T1) і приймає на їх основі якесь рішення. Транзакція T1 виконує відкат. В результаті рішення, прийняте транзакцією T2 основане на неправильних даних.

Неповторне читання. Транзакція T1 під час свого виконання декілька раз читає ті самі дані. Транзакція T2 в проміжках між читаннями транзакцією T1 змінює ці дані і завершується. В результаті, виявляється, що читання одних і тих же даних у T1 дає різні результати.

Фіктивне читання. Транзакція T1 під час виконання кілька разів вибирає множину рядків за одними і тими ж критеріями. Транзакція T2 в інтервалах між вибірками транзакції T1 додає або видаляє рядки або змінює стовпці деяких рядків, які використовуються в критерії відбору і фіксується. У результаті виходить, що одні і ті ж вибірки транзакції T1 вибирають різну кількість рядків.

Стандартний SQL/92 визначає рівень ізоляції транзакції в багатокористувацькій системі установка яких запобігає певним конфліктним ситуаціям (табл.1.). Введені наступні чотири рівня ізоляції.

Непідтверджене читання. Найнижчий рівень ізоляції – READ UNCOMMITED (непідтверджене читання або читання непідтверджених даних). При даному рівні зміни, внесені одним користувачем, можуть бути прочитані іншим, навіть якщо перший користувач ще не підтвердив їх за допомогою оператора COMMIT. Проблема виникне у випадку, якщо перший користувач виконає відкат своєї транзакції. Тоді всі наступні дії другого користувача виявляються заснованими на неправильних даних.

Таким чином, рівень READ UNCOMMITED не слід використовувати, коли потрібні точні результати. Він більш-менш підходить в тому випадку, коли достатньо отримати приблизні результати, наприклад, при статистичній обробці мало змінних даних.

Очевидно, що при рівні ізоляції READ UNCOMMITED обидві транзакції можуть виконуватися одночасно.

Підтверджене читання. Наступним по силі є рівень ізоляції READ COMMITED (підтверджене читання або читання підтверджених даних). У цьому випадку зміни, зроблені іншою транзакцією, залишаються невидимими у вашій транзакції до тих пір, поки інший користувач не завершить її оператором

COMMIT. Хоча до завершення іншої транзакції внесені нею зміни вам не видно, обидві транзакції можуть виконуватися одночасно.

У порівнянні з READ UNCOMMITED даний рівень ізоляції дає кращі результати, але не звільняє від неприємностей так званого неповторного читання. Розглянемо наступну ситуацію.

Припустимо, перший користувач збирається спочатку прочитати дані, а потім використовувати їх при внесенні змін в іншу таблицю. Наприклад, він хоче спочатку дізнатися, скільки є деякого товару на складі, а потім оформити замовлення на цей товар. Може статися так, що цей користувач прочитає застарілі дані, які вже змінені другим користувачем, але ще не підтверджені ним за допомогою оператора COMMIT. Наприклад, другий користувач вже "забрав" весь запас даного товару, а перший користувач, не знаючи про це, оформив замовлення на відсутній товар. Таким чином, проблема полягає в тому, що результати читання даних першим користувачем до і після виконання другим користувачем оператора COMMIT можуть відрізнятися. Це так звана проблема неповторного читання.

Повторюване читання. Рівень ізоляції REPATABLE READ (повторюване читання) виключає проблему неповторного читання. Тим не менш, цей рівень не усуває іншу неприємність, так зване фіктивне читання. Вона виникає тоді, коли користувач читає дані, що змінюються в результаті виконання іншої транзакції. При цьому зміни даних відбуваються під час їх читання. Розглянемо як приклад наступну ситуацію.

Припустимо, перший користувач виконує оператор вибірки даних (SELECT) З деякою умовою (WHERE або HAVING). В результаті він вибирає якусь множину записів. Другий користувач відразу ж за цим змінює дані в одному або декількох записах, які вибрав перший користувач. Більш того, другий користувач завершує свою транзакцію оператором COMMIT. Таким чином, виходить, що записи, які раніше відповідали умові вибірки, тепер перестали їй відповідати. Не виключено, що з'явилися нові записи, які раніше не задовольняли умові, а тепер відповідні їй. У цей час перший користувач, ще не завершивши свою транзакцію, не знає про зміни, що відбулися і відправляє на виконання ще один SQL-оператор з такою ж умовою вибірки, що і на початку. Однак цей оператор буде працювати не з тими самими записами, що попередній оператор. Дана проблема і є проблемою фіктивного читання.

Послідовне виконання. Уникнути неприємностей, пов'язаних з раніше розглянутими рівнями ізоляції, дозволяє рівень SERIALIZABLE (послідовне виконання). Транзакції з рівнем ізоляції SERIALIZABLE завжди виконуються не паралельно, а послідовно. Якщо одна з них вже розпочата, то інша буде чекати її закінчення. Апаратні та програмні відмови можуть призвести до невиконання транзакції, однак при даному рівні ізоляції можна бути впевненим, що результати роботи з базою даних будуть коректними.

Надаючи найбільшу надійність, рівень ізоляції SERIALIZABLE максимально знижує загальну продуктивність системи.

Таблиця. 1.
Визначення рівнів ізоляції

Рівні ізоляції SQL/92	**АНОМАЛІЇ**			
	Втрачені зміни	Грязне читання	Неповторне читання	фіктивне читання
READ UNCOMMITTED	ні	так	так	так
READ COMMITTED	ні	ні	так	так
REPEATABLE READ	ні	ні	ні	так
SERIALIZABLE	ні	ні	ні	ні

Нехай в базі даних існує таблиця EXAMPLE:

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Наведемо приклади сценаріїв перевірки небажаних ситуацій на прикладі таблиці EXAMPLE (табл. 2).

Таблиця 2.
Сценарії перевірок

Крок	**Транзакція T1**	**Транзакція T2**
1. Втрачені зміни		
1	UPDATE example SET dat=101 WHERE id=1;	
2		UPDATE example SET dat=102 WHERE id=1;
4		COMMIT;
3	COMMIT;	
Якщо втрачені зміни допускаються, то сценарій виконається без помилок і блокувань. В базі даних збережуться зміни, зроблені на кроці 1.		
2. Брудне читання		
1	SELECT * FROM example WHERE id=2;	
2		UPDATE example SET dat=101 WHERE id=2;
3	SELECT * FROM example WHERE id=2;	
4		ROLLBACK;
5	SELECT * FROM example WHERE id=2;	
Якщо допускається незавершене читання, то сценарій виконається без помилок і блокувань. На кроці 1 будуть вибрані значення (2,110). На кроці 2 - (2,101). На кроці 3 - (2,110).		

3. Неповторне читання		
1	SELECT * FROM example WHERE id=3;	
2	[COMMIT]	UPDATE example SET dat=101 WHERE id=3;
3		COMMIT;
4	SELECT * FROM example WHERE id=3;	
5	COMMIT;	
Якщо допускається неповторне читання, то сценарій виконається без помилок і блокувань. Операцію COMMIT на кроці 2 виконувати не прийдеться. На кроці 1 будуть вибрані значення (1,120). На кроці 2 – (1,101).		
4. Фіктивне читання		
1	SELECT * FROM example WHERE dat>180;	
2	[COMMIT]	INSERT INTO example VALUES(12,210);
3		COMMIT;
4	SELECT * FROM example WHERE dat>180;	
5	COMMIT;	
Якщо допускається фіктивне читання, то сценарій виконається без помилок і блокувань. Операцію COMMIT на кроці 2 виконувати не прийдеться. На кроці 1 будуть вибрані значення (10,190), (11,200). На кроці 2 - (10,190), (11,200), (12,210).		
5. Безвихідь		
1	UPDATE example SET dat=101 WHERE id=1;	
2		UPDATE example SET dat=112 WHERE id=2;
3	UPDATE example SET dat=111 WHERE id=2;	
4		UPDATE example SET dat=102 WHERE id=1;
Якщо система не знаходить і не усуває безвихідних ситуацій, то після виконання кроку 4 транзакції повинні заблокуватися.		

Розглянемо приклади

Приклад 5.

Дано таблиця example бази даних

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	Транзакція T1	Транзакція T2
1	UPDATE example SET dat=101 WHERE id=1;	
2		UPDATE example SET dat=102 WHERE id=1;
3		COMMIT;
4	COMMIT;	

Потрібно вияснити, що буде з даними в таблиці EXAMPLE після завершення транзакцій, якщо для обох транзакцій встановлено рівень ізоляції SERIALIZABLE.

Пояснення. Першою розпочинається транзакція T1. На першому кроці транзакція T1 змінює дані dat на значення 101 для id=1. На другому кроці розпочинається транзакція T2, яка теж хоче змінити ці ж самі дані. Але, так як для обох транзакцій встановлено рівень ізоляції SERIALIZABLE, то друга транзакція не змінює дані, а чекає завершення першої транзакції. Тому на другому кроці значення dat залишається рівним 101 для id=1. На третьому кроці подано команду на завершення транзакції T2. Але транзакція ще не завершується, а стоїть в черзі і чекає завершення транзакції T1. Тобто команди на зміну і завершення транзакції T2 подані, але вони не виконуються, чекаючи завершення транзакції T1, яка розпочалася раніше. На четвертому кроці відбувається завершення транзакції T1, яка підтверджує зміни зроблені першою транзакцією, при цьому підтверджується зміна значення dat, яке стає рівне 101 для id=1. Після завершення транзакції T1 починає виконуватися транзакція T2, яка змінює значення dat на 102 для id=1 і завершується. Тобто при такому сценарії для рівня їзоляції транзакцій SERIALIZABLE в результаті отримаємо значення dat яке рівне 102 для id=1.

Приклад 6.

Нехай для обох транзакцій, які виконуються за сценарієм попереднього прикладу, встановлено рівень ізоляції READ COMMITTED (читання підтверджених даних). Потрібно вияснити, що буде з даними в таблиці EXAMPLE після завершення транзакцій.

Пояснення. У цьому випадку зміни, зроблені транзакцією T2, залишаються невидимими у транзакції T1 до тих пір, поки ця транзакція не завершиться. Але після завершення T2 на третьому кроці значення dat стане рівним 102 для id=1, при цьому перша транзакція, ще не підтвердила свої зміни зроблені на кроці 1. На четвертому кроці транзакція T1 підтверджує свої зміни і значення dat стає рівним 101 для id=1. Тобто перша транзакція T1 витре зміни, які зробила транзакція T2.

5. ОБМЕЖЕННЯ В ТРАНЗАКЦІЯХ

Обмеження, що накладаються на стовпці, таблиці та зв'язку між ними, впливають на виконання транзакцій. Наприклад, таблиця має стовпець, на який накладено обмеження NOT NULL (значення стовпця не можуть бути невизначеними), і потрібно додати нові дані. Як правило, в цьому випадку спочатку додають новий порожній запис, а потім заповнюють його значеннями. Однак даний спосіб не можна здійснити, навіть якщо спробувати його виконати до оператора завершення транзакції COMMIT. Справа в тому, що обмеження NOT NULL не дозволить створити рядок з невизначеними значеннями. У SQL: 2003 для вирішення даної проблеми можна надати обмеженням додаткові властивості:

- DEFERABLE (допускає затримку). Обмеження з даною властивістю можуть бути задані як IMMEDIATE (негайні) або як DEFERED (затримані);
- NOT DEFERABLE (не допускає затримку). Обмеження з цією властивістю виконуються негайно.

Якщо обмеження типу DEFERABLE задано як IMMEDIATE, то воно діє негайно, як і обмеження типу NOT DEFERABLE. Якщо обмеження типу DEFERABLE задано як DEFERED, то його дія може бути затримана.

Для додавання порожніх записів і виконання інших операцій, які можуть порушити обмеження типу DEFERABLE, використовується наступний оператор:

SET CONSTRAINS ALL DEFERED;

(Встановити всі обмеження як затримані).

Цей оператор визначає всі обмеження типу DEFERABLE як DEFERED, а на обмеження типу NOT DEFERABLE він не діє. Після виконання всіх операцій, які можуть порушити обмеження, і досягнення таблицею стану, в якому їх порушити вже не можна, всі обмеження можна відновити:

SET CONSTRAINS ALL IMMEDIATE;

(Встановити всі обмеження як негайні).

Якщо обмеження, раніше задані як DEFERED, ви не задаєте явно як IMMEDIATE, то при спробі завершити свою транзакцію оператором COMMIT будуть активовані всі затримані обмеження. Якщо до цього моменту виконуються не всі обмеження, транзакція буде скасована з повідомленням про помилку.

Отже, обмеження захищають базу даних від неправильного введення даних або від їх відсутності. Однак у вас є можливість усередині транзакції тимчасово відмінити обмеження , які заважають вам.

Для чого потрібні відкладені обмеження? Прикладів їх використання можна привести багато. Одна з причин використання відкладених обмежень – забезпечення каскадного оновлення, коли виникає необхідність оновлення первинного ключа в зв'язку "головний-підлеглий". Якщо ви робите зовнішній ключ відкладеним, але спочатку негайним, ви можете:

- перевести всі обмеження в відкладений режим;

- оновити первинний ключ – на даний момент дочірні обмеження цілісності перевірятися не будуть;
- оновити дочірні зовнішні ключі;
- виконати оператор COMMIT (він буде виконаний успішно, якщо всі дочірні записи, порушені оновленням, вказують на існуючий батьківський запис).

Без відкладених обмежень процес оновлення буде надмірно важким. Крім того, що відкладені обмеження ви можете використовувати в різних багатооператорних транзакціях, в процесі виконання яких вимагається тимчасово порушувати цілісність, але в кінці транзакцій цілісність відновлюється.

Розглянемо приклади.
Створимо таблицю test.

```
create table test (
x int constraint check_x check (x> 0) deferrable initially immediate,
y int constraint check_y check (y> 0) deferrable initially deferred);
```

Приклад 7.
Спробуємо добавити дані, які задовольняють обмеженням.

```
insert into test values (1,1);
commit;
```

Пояснення. Отже, коли обидва обмеження задовольняються, рядки вставляються без помилки.

Приклад 8.
Спробуємо добавити рядок, який порушує обмеження check_x.

```
insert into test values (-1,1);
commit;
```

Пояснення. Обмеження check_x типу deferrable (перевірку якого можна відкласти до завершення транзакції), але таке, що перевіряється негайно (immediate), тому рядок відразу ж перевіряється і виникне помилка, ще до давершення транзакції.

Приклад 9.
Спробуємо добавити рядок, який порушує обмеження check_ y.

```
insert into test values (1,-1);
commit;
```

Пояснення. Обмеження check_y, не тільки відкладене (deferrable), але і таке, перевірка якого відкладене до завершення транзакції. Тобто, обмеження не буде перевірятися до виконання оператора COMMIT або зміни стану обмеження на негайне. В даному випадку вставка виконається успішно (у всякому разі, зараз), так як ми відклали перевірку обмеження до завершення

транзакції. Але після виконання оператора COMMIT знову ж таки виникне повідомлення про помилку. У цей момент сервер бази даних виконає відкат транзакції, так як стався збій оператора COMMIT через порушення обмеження.

Отже, приклади 7–9 демонструють відмінності між спочатку негайними і спочатку відкладеними обмеженнями. Компонент “initially” в стані обмеження вказує, коли буде за замовчуванням перевірятися обмеження: або в кінці виконання оператора (immediate), або в кінці виконання транзакції (deferred).

Приклад 10.

Тепер виконаємо оператор, який робить всі обмеження відкладеними (всі, які можуть бути відкладеними, тобто всі типу deferrable). Зауважимо, ви можете також виконувати цей оператор для окремих обмежень, якщо це потрібно; ви не повинні обов’язково відкладати всі обмеження, які можуть бути відкладеними:

set constraints all deferred;

Після цього знову попробуємо вставити дані

insert into test values (-1,1);

Пояснення. Коли спочатку негайне обмеження переводиться в відкладений режим, здається, що операція вставки виконується успішно, але при спробі фіксації транзакції (commit) відбувається збій транзакції і виконується її відкат, оскільки перевірочне обмеження check_x перевіряється під час виконання оператора COMMIT.

Приклад 11.

Тепер змусимо спочатку відкладені обмеження діяти як “негайні” обмеження:

set constraints all immediate;

insert into test values (1, -1);

Пояснення. В такому випадку оператор, який працював до моменту фіксації транзакції, збивається відразу ж. Ми вручну змінили режим перевірки обмежень за замовчуванням.

Приклад 12.

Дано таблиця, створена за допомогою запиту SQL:

```
create table test(
x int check (x> 0) deferrable initially immediate,
y int check (y> 0) deferrable initially deferred,
z int check (z> 0) deferrable initially deferred);
```

Нехай виконується транзакція T, в порядку поданому в таблиці:

Крок	Транзакція T
1	INSERT INTO test VALUES (1, 1, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 3, -3);
5	SAVEPOINT Point2;
6	UPDATE test SET z=100 WHERE y=1;
7	ROLLBACK TO SAVEPOINT Point1;
8	COMMIT;

Вияснити, які дані будуть в таблиці test після завершення транзакції.

Пояснення. На першому кроці виконання транзакції в таблицю додаються дані (1, 1, -1), на другому – (2, 1, -2). На третьому кроці встановлюється точка відкату Point1. На четвертому кроці в таблицю додаються дані (3, 3, -3). На п'ятому кроці встановлюється точка відкату Point2. Всі дані, які додавалися до шостого кроку не задовольняють обмеженню цілісності, але є такими, перевірка яких відкладена до завершення транзакції. Тобто в таблиці test на даному кроці знаходяться значення (1, 1, -1); (2, 1, -2); (3, 2, -3). На шостому кроці дані змінюються і в базі будуть знаходяться значення (1, 1, 100); (2, 1, 100); (3, 2, -3). На сьомому кроці відбувається відміна всіх операцій, які виконувалися після точки відкату Point1, тобто відміняються всі дії до кроку 3. В такому випадку в таблиці бази даних залишаються тільки значення (1, 1, -1); (2, 1, -2). На восьмому кроці відбувається підтвердження виконання транзакції, під час якого перевіряються всі відкладені обмеження. А так, як значення z порушують обмеження накладені на даний стовпець, то значення не будуть додані в таблицю бази даних. Тобто, якщо таблиця була порожньою до виконання транзакції, то вона буде порожньою і після завершення транзакції.

Приклад 13.

Дано таблиця, створена за допомогою запиту SQL:

```
create table test(
x int check (x> 0) deferrable initially immediate,
y int check (y> 0) deferrable initially deferred,
z int check (z > 0) deferrable initially deferred);
```

Нехай виконується транзакція T, в порядку поданому в таблиці:

Крок	Транзакція T
1	INSERT INTO test VALUES (1, 1, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	UPDATE test SET z=100 WHERE y=1;
5	SAVEPOINT Point2;
6	INSERT INTO test VALUES (3, 3, -3);
7	ROLLBACK TO SAVEPOINT Point2;
8	COMMIT;

Вияснити, які дані будуть в таблиці test після завершення транзакції.

Пояснення. На першому кроці виконання транзакції в таблицю додаються дані (1, 1, -1), на другому – (2, 1, -2). На третьому кроці встановлюється точка відкату Point1. Всі дані, які додалися до третього кроку не задовольняють обмеженню цілісності, але є такими, перевірка яких відкладена до завершення транзакції. Тобто в таблиці test на даному кроці знаходяться значення (1, 1, -1); (2, 1, -2). На четвертому кроці дані змінюються і в базі будуть знаходитися значення (1, 1, 100); (2, 1, 100). На п'ятому кроці встановлюється точка відкату Point2. На шостому кроці в таблицю додаються дані (3, 3, -3). Дані не задовольняють обмеженню цілісності, але є такими, перевірка яких відкладена до завершення транзакції. Тобто в таблиці test на даному кроці знаходяться (1, 1, 100); (2, 1, 100); (3, 3, -3). На сьомому кроці відбувається відміна всіх операцій, які виконувалися після точки відкату Point2, тобто відміняються всі дії до кроку 5. В такому випадку в таблиці бази даних залишаються тільки значення (1, 1, 100); (2, 1, 100). На восьмому кроці відбувається підтвердження виконання транзакції, під час якого перевіряються всі відкладені обмеження. А так, як значення, які на даному кроці знаходяться в таблиці бази даних не порушують обмеження накладені на дані, то значення будуть додані в таблицю бази даних. Тобто, якщо таблиця була порожньою до виконання транзакції, то після завершення транзакції в таблиці будуть значення (1, 1, 100); (2, 1, 100).

Приклад 14.

Дано таблиця, створена за допомогою запиту SQL:

```
create table test(
x int check (x> 0) deferrable initially immediate,
y int check (y> 0) deferrable initially deferred,
z int check (z > 0) deferrable initially deferred);
```

Нехай виконується транзакція T, в порядку поданому в таблиці:

Крок	Транзакція T
1	INSERT INTO test VALUES (1, 1, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	UPDATE test SET z=100 WHERE y=1;
5	SAVEPOINT Point2;
6	INSERT INTO test VALUES (3, 3, -3);
7	ROLLBACK TO SAVEPOINT Point2;
8	ROLLBACK;

Вияснити, які дані будуть в таблиці test після завершення транзакції.

Пояснення. На восьмому кроці відбувається відкат всієї транзакції, тому, немає значення які дії з таблицею виконувалися під час виконання транзакції, вони всі були відмінені. Тобто, якщо таблиця була порожньою до виконання транзакції, то після завершення транзакції таблиця теж буде порожньою.

6. ТЕСТОВІ ЗАВДАННЯ

1. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK TO SAVEPOINT Point1;
6	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

2. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK TO SAVEPOINT Point2;
6	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

3. **Дано таблиця example бази даних:**

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK TO SAVEPOINT Point2;
6	ROLLBACK;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

4. **Дано таблиця example бази даних:**

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK;
6	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

5. **Дано таблиця example бази даних:**

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK;
6	ROLLBACK TO SAVEPOINT Point2;
7	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

6. **Дано таблиця example бази даних:**

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	ROLLBACK TO SAVEPOINT Point1;
6	UPDATE example SET dat=100 WHERE dat=101;
7	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

7. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE example SET dat=100 WHERE dat=101;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

8. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE example SET dat=100 WHERE dat=101;
8	ROLLBACK;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

9. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE example SET dat=100 WHERE dat=102;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

10. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE example SET dat=100 WHERE dat=103;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

11. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK;
7	UPDATE example SET dat=103 WHERE dat=101;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

12. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK;
7	UPDATE example SET dat=104 WHERE id=1;
8	ROLLBACK TO SAVEPOINT Point2;
9	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

13. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
	UPDATE example SET dat=104 WHERE dat=102;
7	ROLLBACK TO SAVEPOINT Point2;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. Null

14. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
	UPDATE example SET dat=104 WHERE dat=102;
7	ROLLBACK TO SAVEPOINT Point2;
8	ROLLBACK;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

15. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
	UPDATE example SET dat=104 WHERE dat=101;
7	ROLLBACK TO SAVEPOINT Point2;
8	COMMIT;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 100
b. 101
c. 102
d. 103
e. 104
f. Null

16. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	UPDATE example SET dat=101 WHERE id=1;
2	SAVEPOINT Point1;
3	UPDATE example SET dat=102 WHERE id=1;
4	SAVEPOINT Point2;
5	UPDATE example SET dat=103 WHERE id=1;
6	ROLLBACK TO SAVEPOINT Point1;
	UPDATE example SET dat=104 WHERE dat=101;
7	ROLLBACK TO SAVEPOINT Point2;
8	ROLLBACK;

Після виконання даної транзакції для id=1 значення dat буде рівним:

a. 0
b. 100
c. 101
d. 102
e. 103
f. 104
g. Null

17. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	UPDATE example SET dat=101 WHERE id=1;	
2		UPDATE example SET dat=102 WHERE id=1;
3		COMMIT;
4	COMMIT;	

Якщо для обох транзакцій встановлено рівень ізоляції SERIALIZABLE після завершення першої транзакції (крок 4), для id=1 значення dat,буде рівним:

a. 100
b. 101
c. 102
d. 103
e. Null
f. Транзакції заблокуються

18. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	SELECT * FROM example WHERE id=2	
2		UPDATE example SET dat=101 WHERE id=2
3	SELECT * FROM example WHERE id=2	
4		ROLLBACK
5	SELECT * FROM example WHERE id=2	

Якщодля обох транзакцій встановлено рівень ізоляції SERIALIZABLE на кроці 3 буде вибрано значення dat, яке буде рівним:

a. 100
b. 101
c. 110
d. 120
e. Null
f. Транзакції заблокуються

19. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10
	dat INTEGER	100	110	120	130	140	150	160	170	180	190

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	SELECT * FROM example WHERE id=3;	
2	[COMMIT]	UPDATE example SET dat=101 WHERE id=3;
3		COMMIT;
4	SELECT * FROM example WHERE id=3;	
5	COMMIT;	

Якщо для обох транзакцій встановлено рівень ізоляції SERIALIZABLE на кроці 4 буде вибрано значення dat, яке буде рівним:

a. 100
b. 101
c. 110
d. 120
e. Транзакції заблокуються

20. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10
	dat INTEGER	100	110	120	130	140	150	160	170	180	190

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	SELECT * FROM example WHERE id=3;	
2	[COMMIT]	UPDATE example SET dat=101 WHERE id=3;
3		COMMIT;
4	SELECT * FROM example WHERE id=3;	
5	COMMIT;	

Якщо для обох транзакцій встановлено рівень ізоляції SERIALIZABLE то транзакції заблокуються на кроці:

a. 1
b. 2
c. 3
d. 4
e. 5
f. транзакції не заблокуються

21. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10
	dat INTEGER	100	110	120	130	140	150	160	170	180	190

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	SELECT * FROM example WHERE dat>180;	
2	[COMMIT]	INSERT INTO example VALUES(12,210);
3		COMMIT;
4	SELECT * FROM example WHERE dat>180;	
5	COMMIT;	

Якщодля обох транзакцій встановлено рівень ізоляції SERIALIZABLE на кроці 4 буде виведено кількість рядків:

a. 1
b. 2
c. 3
d. 4
e. 5
f. транзакції заблокуються

22. Дано таблиця example бази даних:

Таблиця example	**id INTEGER**	1	2	3	4	5	6	7	8	9	10
	dat INTEGER	100	110	120	130	140	150	160	170	180	190

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

Крок	**Транзакція T1**	**Транзакція T2**
1	SELECT * FROM example WHERE id=3;	
2	[COMMIT]	UPDATE example SET dat=101 WHERE id=3;
3		COMMIT;
4	SELECT * FROM example WHERE id=3;	
5	COMMIT;	

Якщодля обох транзакцій встановлено рівень ізоляції SERIALIZABLE після завершення першої транзакції (крок 5) для id=3, значення dat буде рівним:

a. 100
b. 101
c. 110
d. 120
e. Транзакції заблокуються

23. Дано таблиця, створена за допомогою запиту SQL:

```
create table test(
    x int constraint check_x check (x> 0) deferrable initially immediate,
    y int constraint check_y check (y> 0) deferrable initially deferred);
```

При спробі добавити дані insert into test values (1,-1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

24. Дано таблиця, створена за допомогою запиту SQL:

```
create table test (
    x int constraint check_x check (x> 0) deferrable initially immediate,
    y int constraint check_y check (y> 0) deferrable initially deferred);
```

При спробі добавити дані insert into test values (-1,1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

25. Дано таблиця, створена за допомогою запиту SQL:

```
create table test (
    x int constraint check_x check (x> 0) deferrable initially immediate,
    y int constraint check_y check (y> 0) deferrable initially deferred);
```

При спробі добавити дані insert into test values (-1,-1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

26. Дано таблиця, створена за допомогою запиту SQL:

```
create table test (
        x int constraint check_x check (x> 0) deferrable initially immediate,
        y int constraint check_y check (y> 0) deferrable initially deferred);
set constraints all deferred;
```

При спробі добавити дані insert into test values (-1,1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

27. Дано таблиця, створена за допомогою запиту SQL:

```
create table test (
        x int constraint check_x check (x> 0) deferrable initially immediate,
        y int constraint check_y check (y> 0) deferrable initially deferred);
set constraints all immediate;
```

При спробі добавити дані insert into test values (1,-1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

28.Дано таблиця, створена за допомогою запиту SQL:

```
create table test (
        x int constraint check_x check (x> 0) deferrable initially immediate,
        y int constraint check_y check (y> 0) deferrable initially deferred);
        z int constraint check_y check (z> 0) deferrable initially immediate);
set constraints all deferred;
```

При спробі добавити дані insert into test values (1,-1,-1);

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.
e. транзакція заблокується.

29. Дано таблиця, створена за допомогою запиту SQL:

```
create table test    (x int check (x> 0) deferrable initially immediate,
                      y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1,-1);
2	INSERT INTO test VALUES (2, 2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3,-3);
5	SAVEPOINT Point2;
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET y=100 WHERE x=1;
8	ROLLBACK TO SAVEPOINT Point2;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, -1); (2, 2); (3, -3);
b. (1, 1); (2, 2);
c. (1,100); (2, 2);
d. (1,100); (2, 100);
e. (1,100); (2, 100); (3, 100);
f. Таблиця буде порожньою.

30. Дано таблиця, створена за допомогою запиту SQL:

```
create table test    (x int check (x> 0) deferrable initially immediate,
                      y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1,-1);
2	INSERT INTO test VALUES (2, -2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3,-3);
5	SAVEPOINT Point2;
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET y=100 WHERE x=1;
8	ROLLBACK TO SAVEPOINT Point2;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, -1); (2, -2); (3, -3);
b. (1,100); (2, -2);
c. (1,100); (2, 100);
d. (1,100); (2, 100); (3, 100);
e. Таблиця буде порожньою.

31. Дано таблиця, створена за допомогою запиту SQL:

```
create table test    (x int check (x> 0) deferrable initially immediate,
                      y int check (y> 0) deferrable initially deferred,
                      z int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1, 1, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 2, -3);
5	SAVEPOINT Point2;
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET z=100 WHERE y=1;
8	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, 1, -1); (2, 1, -2); (3, 2, -3);
b. (1, 1, 1); (2, 1,-2);
c. (1, 1,100); (2, 1, -2);
d. (1, 1, 100); (2, 1, 100);
e. (1, 1, 100); (2, 1, 100); (3, 2, 100);
f. Таблиця буде порожньою.

32. Дано таблиця, створена за допомогою запиту SQL:

```
create table test    (x int check (x> 0) deferrable initially immediate,
                      y int check (y> 0) deferrable initially deferred,
                      z int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1, 2, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 2, -3);
5	SAVEPOINT Point2;
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET z=100 WHERE y=1;
8	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, 1, -1); (2, 1, -2); (3, 2, -3);
b. (1, 1,100); (2, 1, -2);
c. (1, 1, 100); (2, 1, 100);
d. (1, 1, 100); (2, 1, 100); (3, 2, 100);
e. Таблиця буде порожньою.

33. Дано таблиця, створена за допомогою запиту SQL:

```
create table test   (x int check (x> 0) deferrable initially immediate,
                     y int check (y> 0) deferrable initially deferred,
                     z int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1, 1, -1);
2	INSERT INTO test VALUES (2, 1, -2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 3, -3);
5	SAVEPOINT Point2;
6	UPDATE test SET z=100 WHERE y=1;
7	ROLLBACK TO SAVEPOINT Point2;
8	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, 1, -1); (2, 1, -2); (3, 2, -3);
b. (1, 1, 1); (2, 1,-2);
c. (1, 1,100); (2, 1, -2);
d. (1, 1, 100); (2, 1, 100);
e. (1, 1, 100); (2, 1, 100); (3, 2, 100);
f. Таблиця буде порожньою.

34. Дано таблиця, створена за допомогою запиту SQL:

```
create table test   (x int check (x> 0) deferrable initially immediate,
                     y int check (y> 0) deferrable initially deferred,
                     z int check (z> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1, 1,-1);
2	INSERT INTO test VALUES (2, 1,-2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 1,-3);
5	SAVEPOINT Point2;
6	UPDATE test SET z=100 WHERE y=1;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE test SET z=200 WHERE y=1;
	ROLLBACK TO SAVEPOINT Point2;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, 1,-1); (2, 1,-2); (3, 2, -3);
b. (1, 1, -1); (2, 1, -2);
c. (1, 1, 100);
d. (1, 1, 100); (2, 1, 100);
e. (1, 1, 100); (2, 1, 100); (3, 1, 100);
f. (1, 1, 200);
g. (1, 1, 200); (2, 1, 200);
h. (1, 1, 200); (2, 1, 200); (3, 1, 200);
i. Таблиця буде порожньою.

35. Дано таблиця, створена за допомогою запиту SQL:

```
create table test   (x int check (x> 0) deferrable initially immediate,
                     y int check (y> 0) deferrable initially deferred,
                     z int check (z> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	**Транзакція T1**
1	INSERT INTO test VALUES (1, 1,-1);
2	INSERT INTO test VALUES (2, 1,-2);
3	SAVEPOINT Point1;
4	INSERT INTO test VALUES (3, 1,-3);
5	SAVEPOINT Point2;
6	UPDATE test SET z=100 WHERE y=1;
7	ROLLBACK TO SAVEPOINT Point1;
8	UPDATE test SET z=200 WHERE y=1;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, 1,-1); (2, 1,-2); (3, 2, -3);
b. (1, 1, -1); (2, 1, -2);
c. (1, 1, 100);
d. (1, 1, 100); (2, 1, 100);
e. (1, 1, 100); (2, 1, 100); (3, 1, 100);
f. (1, 1, 200);
g. (1, 1, 200); (2, 1, 200);
h. (1, 1, 200); (2, 1, 200); (3, 1, 200);
i. Таблиця буде порожньою.

36. Дано таблиця, створена за допомогою запиту SQL:

```
create table test   (x int check (x> 0) deferrable initially immediate,
                     y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (3,-3);
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET y=100 WHERE x>1;
8	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1,-1); (2,-2); (3,-3);
b. (1,1); (2,-2);
c. (1,1);
d. (1,100);
e. (1,100); (2,100);
f. (1,100); (2,100); (3,100);
g. Таблиця буде порожньою.

37. Дано таблиця, створена за допомогою запиту SQL

```
create table test    (x int check (x> 0) deferrable initially immediate,
                      y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (3,-3);
6	ROLLBACK TO SAVEPOINT Point2;
7	UPDATE test SET y=100 WHERE x>1;
8	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1,-1); (2,-2); (3,-3);
b. (1,1); (2,-2);
c. (1,1);
d. (1,100);
e. (1,100); (2,100);
f. (1,100); (2,100); (3,100);
g. Таблиця буде порожньою.

38. Дано таблиця, створена за допомогою запиту SQL:

```
create table test     (x int check (x> 0) deferrable initially immediate,
                       y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (3,-3);
6	UPDATE test SET y=100 WHERE x>1;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE test SET y=200 WHERE x>0;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (1, -1); (2, -2); (3, -3);
b. (1, 1); (2, -2);
c. (1, 1);
d. (1, 100);
e. (1, 100); (2, 100);
f. (1, 100); (2,100); (3, 100);
g. (1, 200);
h. (1, 200); (2 ,200);
i. (1, 200); (2, 200); (3, 200);
j. Таблиця буде порожньою.

39. Дано таблиця, створена за допомогою запиту SQL:

```
create table test     (x int check (x> 0) deferrable initially deferred,
                       y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (-1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (-2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (-3,-3);
6	UPDATE test SET y=100 WHERE x<0;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE test SET x=200 WHERE y>0;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (-1, -1); (-2, -2); (-3, -3);
b. (-1, -1); (-2, -2);
c. (-1, -1);
d. (-1, 100);
e. (-1, 100); (-2, 100);
f. (-1, 100); (-2,100); (-3, 100);
g. (200, 100);
h. (200, 100); (200, 100);
i. (200, 100); (200, 100); (200, 100);
j. Таблиця буде порожньою.

40. Дано таблиця, створена за допомогою запиту SQL:

```
create table test     (x int check (x> 0) deferrable initially immediate,
                       y int check (y> 0) deferrable initially immediate);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (-1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (-2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (-3,-3);
6	UPDATE test SET y=100 WHERE x<0;
7	ROLLBACK TO SAVEPOINT Point2;
8	UPDATE test SET x=200 WHERE y>0;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (-1, -1); (-2, -2); (-3, -3);
b. (-1, -1); (-2, -2);
c. (-1, -1);
d. (-1, 100);
e. (-1, 100); (-2, 100);
f. (-1, 100); (-2,100); (-3, 100);
g. (200, 100);
h. (200, 100); (200, 100);
i. (200, 100); (200, 100); (200, 100);
j. Таблиця буде порожньою.

41. Дано таблиця, створена за допомогою запиту SQL:

```
create table test     (x int check (x> 0) deferrable initially deferred,
                       y int check (y> 0) deferrable initially deferred);
```

Нехай виконується транзакція T1, в порядку поданому в таблиці:

Крок	Транзакція T1
1	INSERT INTO test VALUES (-1,-1);
2	SAVEPOINT Point1;
3	INSERT INTO test VALUES (-2,-2);
4	SAVEPOINT Point2;
5	INSERT INTO test VALUES (-3,-3);
6	ROLLBACK TO SAVEPOINT Point1;
7	UPDATE test SET y=100 WHERE x<0;
8	UPDATE test SET x=200 WHERE y>0;
9	COMMIT;

Які дані будуть в таблиці test після виконання даної транзакції, якщо до її виконання таблиця була порожня.

a. (-1, -1); (-2, -2); (-3, -3);
b. (-1, -1); (-2, -2);
c. (-1, -1);
d. (-1, 100);
e. (-1, 100); (-2, 100);
f. (-1, 100); (-2,100); (-3, 100);
g. (200, 100);
h. (200, 100); (200, 100);
i. (200, 100); (200, 100); (200, 100);
j. Таблиця буде порожньою.

Список використаних джерел

1. Булатецька Л. В. Бази даних та розподілені інформаційно-аналітичні системи : програма нормативної навчальної дисципліни / Л. В. Булатецька, В. В. Булатецький ; Східноєвропейський національний університет імені Лесі Українки, кафедра прикладної математики та інформатики . - Луцьк , 2019. – 12 с. – [Електронний ресурс] – Доступний з : http://evnuir.vnu.edu.ua/handle/123456789/16423
2. Дунаев В. В. Базы даннях. Язык SQL / В. В. Дунаев. — СПб. : БХВ-Петербург, 2006. — 288 с.
3. Андон Ф. Язык запросов SQL. Учебный курс. / Ф. Андон, В. Резниченко. — СПб. : Питер; Киев: Издательская группа BHV, 2006. — 416 с.
4. Астахова И.Ф. SQL в примерах и задачах; Учеб. пособие / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников. — М. : Новое знание, 2002. — 176 с.
5. Булатецька Л. В. Мова запитів SQL : текст лекцій нормативної навчальної дисципліни “Бази даних та розподілені інформаційно-аналітичні системи” / Булатецька Леся Віталіївна, Булатецький Віталій Вікторович. – Луцьк, 2018. – 92 с. – [Електронний ресурс] – Доступний з : http://evnuir.vnu.edu.ua/handle/123456789/17722
6. Реляційна алгебра. Реляційне числення [Електронний ресурс] : методичні вказівки для підготовки до контрольної роботи з нормативних навчальних дисциплін “Бази даних та розподілені інформаційно-аналітичні системи”, “Організація баз даних та знань”/ В. В. Булатецький, Л. В. Булатецька; ВНУ ім. Лесі Українки. – Електронні текстові данні (1 файл: 992 КБ). – Луцьк : ВНУ ім. Лесі Українки, 2020. – 36 с. – [Електронний ресурс] – Доступний з : https://evnuir.vnu.edu.ua/handle/123456789/18857
7. Управление транзакциями. – [Електронний ресурс] – Доступний з: http://khpi-iip.mipk.kharkiv.edu/library/dbms/lab2/ref/app6.html

УДК 004.655

Електронне мережне навчально-методичне видання

Булатецька Леся Віталіївна

Булатецький Віталій Вікторович

ТРАНЗАКЦІЇ В SQL

Тестові завдання з
нормативних навчальних дисциплін
“Бази даних та розподілені
інформаційно-аналітичні системи”
“Організація баз даних та знань”