**ტრანზაქციები**

ტრანზაქცია დამოუკიდებლად მუშაობს. თუკი ტრანზაქცია წარმატებით დასრუდება, ყველა ცვლილება, რაც მის შიგნით არის განხორციელდება და მონაცემთა ბაზის მუდმივი ნაწილი გახდება. ხოლო თუ ტრანზაქცია წარუმატებლად დასრულდება, შესაძლებელი იქნება ამ ცვლილების მოთხოვნის დაბრუნება და ყველა ცვლილება რაც მასში იყო თავმოყრილი წაიშლება.

SQL Server-ს გააჩნია რამდენიმე ტრანზაქციის რეჟიმი:

**Autocommit Transactions**

ყველა ცალკეული განაცხადი ტრანზაქციაა, რომელიც ავტომატურ ტრანზაქციად შეიძლება ჩავთვალოთ.

**Explicit transactions**

ყველა ტრანზაქცია ცალსახად იწყება BEGIN TRANSACTION- ით და ცალსახად მთავრდება COMMIT ან ROLLBACK-ით.

**Implicit transactions**

ამ შემთხვევაში ახალი ტრანზაქცია იწყება არაცალსახად, როცა წინა ტრანზაქცია სრულდება, მაგრამ თითოეული ტრანზაქცია სრულდება COMMIT ან ROLLBACK-ით.

**Batch-scoped transactions**

გამოიყენება მხოლოდ მრავალჯერადი აქტიური შედეგების კომპლექტთან ერთად (MARS multiple active result sets), Transact-SQL- ს აშკარა ან არაცალსახა ტრანზაქციები, რომელიც იწყება MARS- ს გაშვებისას, ხდება Batch-scoped ტრანზაქცია. Batch-scoped ტრანზაქცია, რომელიც არ არის დაკომიტებული ან დაროლბექებული პორცია , SQL Server ავტომატურად აბრუნებს ცვლილებას.



**ტრანზაქციის მახასიათებლები**

ტრანზაქციებს გააჩნია 4 მახასიათებელი თვისება, რომლებსაც მოკლედ უწოდებენ ACID -ს. (**Atomicity, Consistency, Isolation, Durability**)

* **ატომურობა**-რწმუნდება, რომ ყველა ოპერაცია რომელიც სრულდება, სრულდება წარმატებით. წინააღმდეგ შემთხვევაში, ტრანზაქცია გაუქმდება „მარცხის“ მომენტში, და ყველა ძველი ოპერაცია გაუქმდება და დაუბრუნდება საწყის ფორმას.
* **კონსისტენციალურობა -** უზრუნველჰყოფს მონაცემთა ბაზის სწორად შეცვლას წარმატებით შესრულებული ტრანზაქციის საფუძველზე
* **იზოლაცია-**  საშუალებას აძლევს ტრანზაქციებს რომ დამოუკიდებლად და გამჭვირვალედ იმოქმედონ.
* **ხანგრძლივობა -** რწმუნდება, რომ შედეგი ან შეცვლილი ტრანზაქციის შედეგი არსებობს სისტების წარუმატებლობის შემთხვევაშიც.

**ტრანზაქციული მაკონტროლებელი ბრძანებები**

ტრანზაქციული მაკონტროლებელი ბრძანებები გამოიყენება მხოლოდ DML ბრზანებებში, როგორიცაა **D**elete, Insert და Update. ისინი არ გამოიყენება ცხრილების შექმნისას ან წაშლისას, რადგან ეს ბრძანებები ავტომატურად დასტურდება მონაცემთა ბაზაში.

### **ტრანზაქციის კონტროლი**

ქვემოთ ჩამოთვლილი ბრძანებები გამოიყენება ტრანზაქციების გასაკონტროლებლად.

* **COMMIT** − ცვლილების შესანახად.
* **ROLLBACK** − ცვლილების დასაბრუნებლად.
* **SAVEPOINT** − ქმნის პუნქცებს ტრანზაქციების ჯგუფებში თუ სად უნდა მოხდეს მონაცემების დაბრუნება, მიუხედავად იმისა rollback გამოყენებულია თუ არა. (commit ის შემთხვევაშიც დაბრუნდება აღნიშნული მონაცემები, რაც savepoint -მდე წერია).
* **SET TRANSACTION** – სახელს არქმევს ტრანზაქციას.

### **SAVEPOINT ბრძანება**

SAVEPOINT არის პუნქტი ტრანზაქციაში, როდესაც შენ შეგიძლია დააბრუნო ტრანზაქცია გარკვეულ პუნქტში, მთლიანი ტრანზაქციის გაუქმების გარეშე.

მისი სინტაქსი ასეთია :

SAVEPOINT SAVEPOINT\_NAME;

შეგვიძლია SAVEPOINT შევქმნათ მონაცემების დაბრუნებისთვისაც, მისი სინტაქსი ასეთია : ROLLBACK TO SAVEPOINT\_NAME;

ამ მახასიათბლის მთავარი არსი მდგომარეობს შემდეგში, შეგვიძლია ერთ დიდ ტრანზაქციაში რასაც შევცვლით დავაბრუნოთ მომენტალურად. დასაბრუნებელი ტანი უნდა იჯდეს ერთნაირსახელიან savepoint ებს შორის (შექმნას და დაბრუნას შორის).

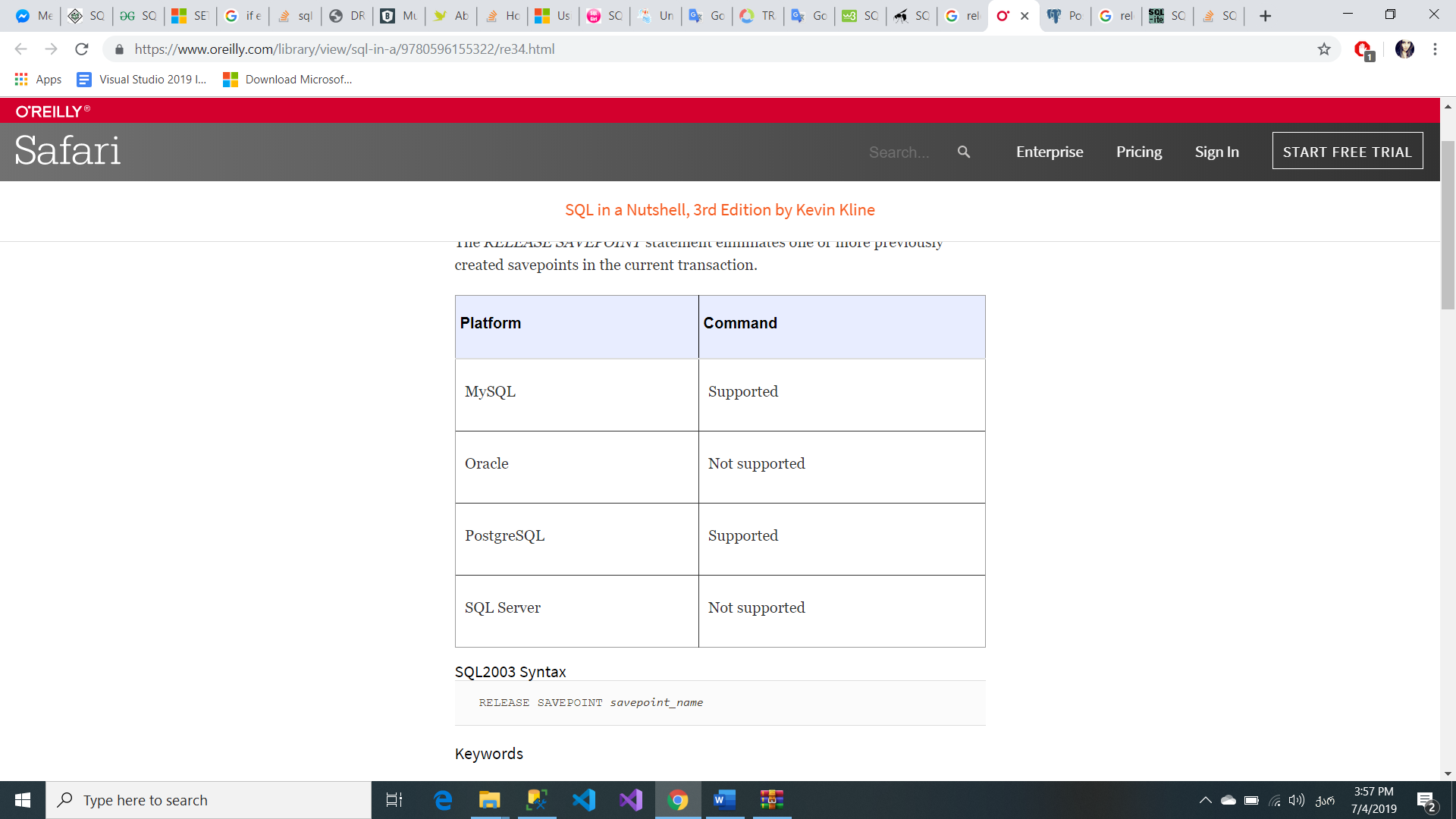
### **განყენებული savepoint ბრძანება( The RELEASE SAVEPOINT Command)**

აღნიშნული ბრძანება გამოიყენება savepoint ის წასაშლელად. სინტაქსი ასეთია :

RELEASE SAVEPOINT SAVEPOINT\_NAME;

თუკი ერთხელ გამოვიყენებთ release ბრძანებას SAVEPOINT ის წასაშლელად, უკვე ვეღარანაირად შევძლებთ მონაცემების დაბრუნებას ბოლო SAVEPOINT მდე.

რადგანაც მაგალითები sql სერვერზეა შესრულებული, ამ გარემოში არაა release savepoint ების გამოყენების საჭიროება, რადგანან savepoint- ები ავტომატურად იშლებიან საბოლოო commit ან roll back ის დროს.



### **SET ტრანზაქციის ბრძანება**

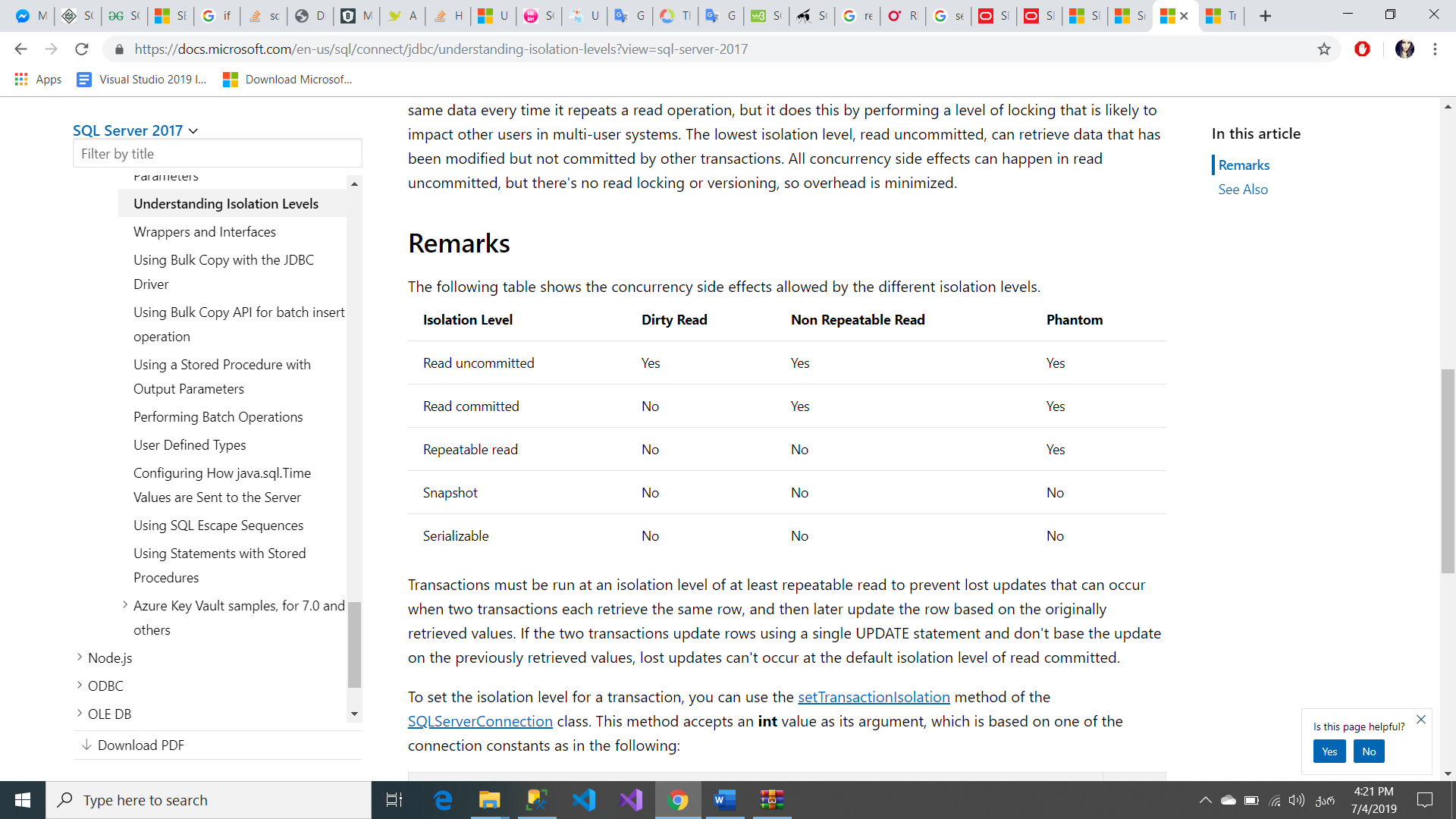
SET TRANSACTION command შეგვიძლია გამოვიყენოთ მონაცემთ ბაზის ტრანზაქციის ინიციალიზაციისთვის. ეს ბრძანება გამოიყენება რომ მოვახდინოთ დიფერენციერბა თვისობრივი, თუ როგორი ტრანზაქცია ეშვება. მაგალითად, შეგვიძლია გავუშვათ ტრანზაქცია მხოლოდ კითხვის, ან მხოლოდ წერის უფლებით.



**ტრანზაქციის იზოლაციის დონეები**

ტრანზაქციის იზოლაციის დონეები არის იმ ხარისხის საზომი კრიტერიუმი, თუ ტრანზაქციის რომელი დონე უნდა იყოს გამოყენებული. როგორც წესი, ტრანზაქციის იზოლაციის დონის გამოყენება დამოკიდებულია შემდეგი ფენომენების არსებობით ან არარსებობით.

* **Dirty Reads** - გვხვდება მაშინ, როდესაც ტრანზაქცია კითხულობს ისეთ მონაცემებს რომლების შეცვლაც ჯერ არ დადასტურებულა სხვა ტრანზაქციის მიერ. მაგალითად: ერთი ტრანზაქცია ცვლის მონაცემებს ცხრილში , მეორე კი კითხულობს ამ მონაცემებს, თუმცა 1 ტრანზაქცია უქმდება. გამოდის, რომ, მეორე ტრანზაქციით არასწორი მონაცემები იქნა წაკითხული. მონაცემები, რომლებიც შესაძლოა არასდროს არსებობდა ამ ფორმით.
* **Nonrepeatable Reads** -გვხვდება მაშინ, როდესაც ტრანზაქცია ერთსადაიმავე მონაცემებს კითხულობს ორჯერ, თუმცა განსხვავებულ მონაცემებს იღებს ორივე შემთხვევაში.  მაგალითად, პირველი ტრანზაქცია კითხულობს მონაცემებს, ამასობაში მეორე ტრანზაქციით იცვლება ეს მონაცემები, ცვლილებაც ინახება (კომიტდება). პირველი ტრანზაქცია ხელახლა ნახულობს იმავე მონაცემებს და ხვედება მონაცემების განსხვავებული რაოდენობა, ან მნიშვნელობები ხვდება შეცვლილი.
* **Phantoms** - არის ხაზი, რომელიც შეესაბამება ძებნის კრიტერიუმს , მაგრამ თავდაპირველად არ ჩანს. მაგალითად, ვთქვათ, პირველი ტრანზაქციით ვკითხულობთ მონაცემებს, რომლებიც რაღაც კრიტერიუმებს აკმაყოფილებენ. მეორე ტრანზაქციით ვაგენერირებთ ახალ ხაზს, რომელიც შეესაბამება პირველი ტრანზაქციის ძიების კრიტერიუმებს. თუ პირველ ტრანზაქციას ხელახლა გავუშვებთ, განხვავებული რაოდენობის მონაცემებს წამოიღებს.



|  |  |
| --- | --- |
| ტრანზაქციის იზოლაციის დასახელება | შესაძლო იმპლემენტაცია |
| Read uncommitted | ტრანზაქციები არ არის იზოლირებული ერთმანეთისგან. თუ DBMS (Database Management System) მხარს უჭერს ტრანზაქციის სხვა იზოლაციის დონეებს, ის ყურადღებას არ აქცევს თუ რა მექანიზმს იყენებს ამ დონეების იმპლემენტაციისთვის. ასე რომ, ისინი უარყოფითად არ მოქმედებენ სხვა ტრანზაქციებზე, ტრანზაქციები რომლებიც გაშვებულია Read Uncommited დონის დროს, როგორც წესი არის მხოლოდ წაკითხვის (read-only). |
| Read committed | ტრანზაქცია ელოდება სანამ სხვა ტრანზაქცია არ დასრულდება, რითაც თავს არიდებს ისეთ პრობლემას, როგორიცაა “dirty read”. |
| Repeatable read | ტრანზაქცია ელოდება სანამ სხვა ტრანზაქცია არ დასრულდება, რითაც თავს არიდებს ისეთ პრობლემას, როგორიცაა “dirty read”. ტრანზაქცია ლოქავს წაკითხვას ყველა მონაცეემისთვის, რასაც ის აბრუნებს და ლოქავს ჩაწერას ყველა იმ მონაცემზე (ხაზზე), რასაც ის ცვლის , ამატებს ან შლის.  აქედან გამომდინარე, რადგანაც სხვა ტრანზაქციებს არ შეუძლიათ შეცვალონ ან წაშალონ ასეთი მონაცემები, ზემოთ აღნიშნული ტრანზაციის იზოლაციის დონე თავიდან ირიდებს ყველა „არაგამეორებადი წაკითხვის“ (**Nonrepeatable Reads** )პრობლემას. |
| Serializable | მას გააჩნია ყველა ის თვისება, რაც Read Committed და Repeatable read იზოლაციის დონეებს, იმ განსხვავებით, რომ ამ შემთხვევაში ტრანზაქცია ბლოკავს ყველა იმ მონაცემების ხაზს, რომლებზეც ის ზემოქმედებას ახდენს. მაგალითად: სელექთის შემთხვევაში ილოქება მთლიანი ცხრილი, რა მონაცემებიც მოაქვს სელექთს. შეცვლის შემთხვევაში, მაგალითად თუ ვშლით ცხრილიდან ყველა მონაცემს, რომლის სტატუსია „დახურული“, ტრანზაქცია ბლოკავს ყველა იმ მონაცემს რომელიც ამ სტატუსს შეიცავს, გარდა ამისა, ამ დროს შეუძლებელია ისეთი ჩანაწერის წაშლა ან დამატება რომელიც ასეთ სტატუსს შეიცავს.  აქედან გამომდინარე, იზოლაციის ეს დონე თავიდან ირიდებს სამივე პრობლემას : **Dirty Reads** , **Nonrepeatable Reads** და **Phantoms** -ს. |
| Snapshot | ტერმინი "Snapshot" ასახავს იმ ფაქტს, რომ ყველა მოთხოვნა ხედავს მონაცემთა ბაზის ერთსა და იმავე ვერსიას, ან მონაცემთა ბაზის იმ გაყინულ მდგომარეობას ხედავს , რომლის დროსაც ტრანზაქციის იწყება.არც ერთი ბლოკი (lock) ემატება,რაც ნებას რთავს სხვა ტრანზაქციებს განახორციელონ ცვლილებები.ტრანზაქციები რომლებიც ცვლიან მონაცემებს, არ ბლოკავენ ისეთ ტრანზაქციებს რომლებიც მხოლოდ მონაცემებს კითხულობენ და პირიქით, ისეთი ტრანზაქციები რომლებიც მხოლოდ კითხულობენ მონაცემებს, არ ბლოკავენ ცვლილების შემცველ ტრანზაქციებს, რადგან ისინი ჩვეულებრივ ავტომატურად ეშვებიან SQL Server- ში. ეს არა-ბლოკვადი სქემა ასევე მნიშვნელოვნად ამცირებს dead lock ების ჩამოყალიბებას. |

