ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ირაკლი გელენიძე

ტრანზაქციების მართვა თანამედროვე მონაცემთა ბაზებში

სამაგისტრო პროგრამა: ინფორმაციული ტექნოლოგიები

ნაშრომი შესრულებულია ინფორმაციული ტექნოლოგიების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: რევაზ ქურდიანი
ასოცირებული პროფესორი

თბილისი

2017

ანოტაცია

რელაციური მონაცემთა ბაზების სისტემების არსებობის დასაბამიდან ტრანზაქციების მართვა ერთ-ერთ მნიშვნელოვან ამოცანას წარმოადგენს როგორც მონაცემთა ბაზების შემქმნელებისათვის, ასევე მათი მომხმარებლებისთვის, მონაცემთა ბაზების დეველოპერების, არქიტექტორებისა თუ მონაცემთა ბაზების ადმინისტრატორებისთვის. მონაცემთა ბაზებში ფუნქციონალურად და წარმადობის თვალსაზრისით ტრანზაქციების მართებულად იმპლემენტაცია უმნიშვნელოვანესი საკითხია, როდესაც საქმე ეხება თანამედროვე, დიდი მოცულობისა და მრავალმომხმარებლიან მონაცემთა ბაზებს.

ნაშრომში განხილულია მონაცემთა ბაზებში ტრანზაქციათაშორისი კონკურენციის პრობლემის გადაჭრის გზები ძირითადად Oracle-ის მონაცემთა ბაზების მაგალითზე. ნაშრომში ასევე აღწერილია ტრანზაქციების მუშაობის ის ძირითადი პრინციპები და მიდგომები, რაზეც არის დაფუძნებული თანამედროვე მონაცემთა ბაზებში ტრანზაქციების მართვა.

Abstract

From the beginning of RDBMS (Relational Database Management Systems) transaction management has been one of the most important questions for Database vendors as well as for database developers, architects and database administrators. In fast growing modern databases one of the main problems is implementation of proper transactions, which should be well functional with efficient performance.

In this work we discuss the ways to solve concurrency problem across database transactions on Oracle's example and describe the main methods, theories and approaches of transaction management in modern databases.

შინაარსი

შესავალი 4
1. ტრანზაქციის მართვის მიზანი 5
2. ტრანზაქციული მონაცემთა ბაზები 5
3. მონაცემთა ბაზების ტრანზაქციების სტანდარტიზებული მართვა 7
3.1 ANSI/ISO ტრანზაქციის იზოლაციის დონეები 7
4. მონაცემთა კონკურენტულობა და კონსისტენტურობა 9
5. მონაცემთა ბაზების ტრანზაქციის იზოლაციის დონეები Oracle Database სისტემებში 13
5.1 Committed წაკითხვის იზოლაციის დონე 13
5.2 წაკითხვის კონსისტენტურობა Read Committed Isolation Level-ში 13
5.3 კონფლიქტური ჩაწერა Read Committed ტრანზაქციებში 14
5.4 სერიალიზებადი იზოლაციის დონე (Serializable Isolation Level) 17
5.5 მხოლოდ წამკითხველი იზოლაციის დონე (Read-Only Isolation Level) 23
6. ტანზაქციულობის მართვის მიდგომები თანამედროვე მონაცემთა ბაზებიში 23
6.1 ტრანზაქციების მართვა Oracle Database გარემოში 24
6.2 SQL ბრძანების შესრულება და ტრანზაქციის კონტროლი 25
7. Locking-ის მექანიზმი მონაცემთა ბაზებში 28
7.1 მონაცემთა ბაზების locking-ის ტექნიკა 29
7.2 Locking-ის ქცევის შეჯამება Oracle-ს მონაცემთა ბაზების მაგალითზე 30
7.3 Lock-ის მეთოდები Oracle Database-ში, მათი გარდაქმნა და ესკალაცია 31
7.4 deadlock-ები Oracle-ს მონაცემთა ბაზებში 32
დასკვნა 33
გამოყენებული ლიტერატურა 34

შესავალი

რელაციურ მონაცემთა ბაზებში ტრანზაქცია ასოცირდება ბაზის შესრულებული სამუშაოს ერთ მთლიან დანაყოფთან და წარმოგვიდგება, როგორც მონაცემთა ბაზებში წარმოქმნილი ქმედებების თანამიმდევრულლობისა და დამოუკიდებლობის დაცვის მთავარი და საიმედო საშუალება. ზოგადად, ტრანზაქციას წარმოადგენს ბაზაში არსებული ყველა სახის ქმედება. მონაცემთა ბაზების გარემოში ტრანზაქციას აქვს ორი ძირითადი მიზანი:

1. ბაზაში წარმოქმნილი ნებისმიერი პრობლემის შემთხვევაში, შექმნას შესრულებული სამუშაოს ერთეულისთვის სრულად და გამართულად აღდგენის საშუალება, ასევე, დაცული იქნეს მონაცემთა ბაზის კონსისტენტურობა, იმ შემთხვევაშიც კი, როდესაც წარმოიქმნება ბაზისგან დამოუკიდებელი სისტემური პრობლემა, რამაც შესაძლოა გამოიწვიოს შესრულების სრული ან ნაწილობრივი შეწყვეტა და მონაცემთა ბაზაში ოპერაციის დაუსრულებელი სახით, გაურკვეველ სტატუსში დარჩენა.
2. შექმნას შესაბამისი იზოლაცია მაშინ, როდესაც პროგრამები მიმართავენ მონაცემთა ბაზებს კონკურენტულად. თუ ბაზებში იზოლაცია არ არის წარმოდგენილი, ეს ქმნის გარდაუვალ საშიშროებას იმისას, რომ პროგრამის მუშაობისას მოსალოდნელი შედეგი იქნება მცდარი.

რელაციურ მონაცემთა ბაზებში მარტივი აქსიომაა, რომ ტრანზაქცია აკმაყოფილებდეს შემდეგ ოთხ ძირითად პირობას: ატომურობა, კონსისტენტურობა, იზოლირებულობა და გამძლეობა (atomic, consistent, isolated and durable). მონაცემთა ბაზების ამ თავისებურებას პრაქტიკოსები ხშირად მოიხსენიებენ აკრონიმით: ACID.

„ყველაფერი ან არაფერი“ - ასე შეიძლება აღიწეროს მიდგომა მონაცემთა ბაზებში, რომელსაც გვთავაზობს ბაზის ტრანზაქციულობა, რაც გულისხმობს შემდეგს: ნებისმიერი დაწყებული ქმედება ბაზაში ან უნდა შესრულდეს სრულად, როგორც ქმედების ერთი ერთეული, ან საერთოდ არ უნდა იქონიოს გავლენა. გარდა ამისა, სისტემამ უნდა მოახდინოს ერთი ტრანზაქციის იზოლირება ყველა სხვა დანარჩენისგან, შედეგი კი უნდა აკმაყოფილებდეს ბაზაში არსებულ ყველა შეზღუდვას (constraints) და შემდეგ, წარმატებულად დასრულებული ტრანზაქცია უნდა ჩაიწეროს მეხსიერების საიმედო სისტემაზე.

1. ტრანზაქციის მართვის მიზანი

ტრანზაქციული მონაცემთა ბაზებს და ასევე მონაცემთა სხვა სახის საცავებს, რომელთა უმთავრეს პრინციპსაც მონაცემთა მთლიანობა წარმოადგენს, აქვთ ტრანზაქციების მართვის შესაძლებლობა, რომელსაც იყენებენ მონაცემთა ერთიანობის პრინციპის დასაცავად. ერთი ტრანზაქცია შესაძლოა, შედგებოდეს ქმედებათა ერთი ან მეტი დამოუკიდებელი ნაწილისაგან, რომელთაგან თითოეული მათგანი აკეთებს ჩაწერის ან წაკითხვის ოპერაციას მონაცემთა ბაზებში ან სხვა ტიპის საცავებში. როდესაც ასეთი ქმედება ხდება, მნიშვნელოვანია, უზრუნველყოფილ იქნეს ყველა მსგავსი პროცესის დასრულება თანმიმდევრულად და სრულყოფილად.

მაგალითისათვის, ბუღალტერიის თეორიაში არსებული ორმაგი ჩაწერის პრინციპი კარგად ასახავს ტრანზაქციულობის კონცეფციას: ყველა სადებეტო ჩანაწერისთვის უნდა არსებობდეს საკრედიტო ჩანაწერიც. მაგალითად, როდესაც გვინდა ავსახოთ სურსათის შეძენის ოპერაცია, საბუღალტრო აღრიცხვის სისტემაში უნდა მოხდეს ორი სახის ჩანაწერის ასახვა, რომლებიც შინაარსობრივად წარმოადგენს ერთ ოპერაციას. ჩანაწერებს ექნება შემდეგი სახე:

1. დებეტი: 100₾ სურსათის ხარჯის ანგარიში
2. კრედიტი: 100₾ საბანკო/საჩეკო ანგარიში

ტრანზაქციულმა სისტემამ უნდა უზრუნველყოს, რომ ორივე ჩანაწერი წარმატებით ჩაიწეროს ან კიდევ - არცერთი. მიდგომით, რომლითაც ერთი ან მეტი ლოგიკური დანაყოფი წარმოდგენილია მუშაობის ერთ ატომურ ტრანზაქციულ ერთეულად, სისტემა ინარჩუნებს ჩაწერილი მონაცემების მთლიანობის დაცვას. ჩვენს მარტივ მაგალითზე რომ ვთქვათ, არასდროს უნდა მოხდეს სადებეტო ჩანაწერის ჩაწერა ისე, რომ მასთან კორესპონდენციაში მყოფი საკრედიტო ჩანაწერი არ არსებობდეს და პირიქით.

2. ტრანზაქციული მონაცემთა ბაზები

ტრანზაქციულ ბაზებს მოიხსენებენ აკრონიმით: DBMS. DBMS-ში არსებობს შესაძლებლობა, რომ დამუშავებული ტრანზაქცია გაუქმდეს და დაუბრუნდეს საწყის მდგომარეობას (Rollback) იმ შემთხვევაში, თუ იგი წარმატებით ვერ დასრულდა (მაგალითად, კვების წყაროს ან კავშირის გათიშვის გამო).

რელაციური მონაცემთა ბაზების მართვის თანამედროვე სისტემების უმეტესობა მოქცეული არიან მონაცემთა ბაზების იმ კატეგორიაში, სადაც ტრანზაქციულობა სრულად მხარდაჭერილი და იმპლემენტირებულია.

მონაცემთა ბაზებში ტრანზაქცია შესაძლოა შედგებოდეს ერთი ან რამდენიმე „მონაცემთა მანიპულაციის ენის" (DML- data manipulation language) ბრძანებისა და მოთხოვნისაგან, რომელთაგან თითოეული კითხულობს ან წერს ინფორმაციას მონაცემთა ბაზაში. მონაცემთა ბაზების სისტემების მომხმარებლები მონაცემების კონსისტენტურობას და მთლიანობას განიხილავენ უმნიშვნელოვანეს საკითხად. ჩვეულებრივ, მარტივი ტრანზაქცია მონაცემთა ბაზებში წარმოგვიდგება პროგრამული ენის სახით, როგორიცაა SQL (structured query language), რომელიც შაბლონურად შემდეგი სახით შეგვიძლია წარმოვადგინოთ:

1. ტრანზაქციის დაწყება
2. მონაცემთა მანიპულაციების ან/და მოთხოვნების ერთობლიობათა შესრულება.
3. თუ შეცდომა არ დაფიქსირდა, მაშინ ტრანზაქციის წარმატებით დასრულება- commit.
4. თუ შეცდომა დაფიქსირდა, ტრანზაქციის დაბრუნება საწყის წერტილზე და დასრულება - Rollback.

როგორც აღვნიშნეთ, თუ შეცდომა არ მოხდა სისტემა დაასრულებს ტრანზაქციას commit-ით. ტრანზაქციის commit-ის ოპერაცია გულისხმობს მონაცემთა მანიპულაციის ბრძანებით შეტანილი ცვლილებების მონაცემთა ბაზაში დამახსოვრებას. ტრანზაქციის მიმდინარეობის დროს თუ შეცდომა მოხდება ან მომხმარებელი თვითნებურად განსაზღვრავს Rollback-ის ოპერაციას, მაშინ მონაცემთა მანიპულაციის შედეგად მონაცემებში შეტანილი ცვლილებები საბოლოოდ არ აისახება მონაცემთა ბაზაში. არ არსებობს შემთხვევა, როდესაც ნაწილობრივ შეიძლება იქნეს შენახული ტრანზაქციის შედეგი, ვინაიდან ეს შემთხვევა გამოიწვევდა მონაცემთა ბაზის არაკონსისტენტურ მდგომარეობას.

ხშირად მრავალმომხმარებლიანი მონაცემთა ბაზები შიდა დონეზე ტრანზაქციებს ინახავს და ამუშავებს ტრანზაქციის იდენტიფიკატორების საშუალებით (Transaction ID/XID).

გარდა ზემოთ მარტივად ახსნილი ვარიანტისა, არსებობს მრავალი გზა ტრანზაქციის იმპლემენტაციის. მაგალითისათვის, **ჩადგმული ტრანზაქციები** (nested transactions) მოიაზრებს ტრანზაქციას, რომელიც შეიცავს ბრძანებას, სადაც იწყება ახალი ტრანზაქცია - ე. წ. ქვეტრანზაქცია (sub-transaction). **მრავალდონიანი ტრანზაქციები** არის ჩადგმული

ტრანზაქციების ნაირსახეობა, სადაც გვხვდება სხვადასხვა დონის ქვეტრანზაქციები. ასევე არსებობს **მაკომპენსირებელი ტრანზაქციები**, ეს ის შემთხვევაა, როდესაც ბიზნეს პროცესი შედგება ერთი ან მეტი ტრანზაქციისაგან, თითოეული მათგანი კი მოიცავს რამდენიმე ოპერაციას. მთლიანობაში ქმნიან ერთ პროცეს, თუმცა დამოუკიდებელი ერთეულები არიან. ამ სახის ტრანზაქციული ლოგიკა საშუალებას იძლევა შევამციროთ ტრანზაქციის დეტალიზაცია. ასეთი ტიპის ტრანზაქციები გამოიყენება გრძელვადიან ტრანზაქციებში, მაგალითისათვის როდესაც საიტი ელოდება მომხმარებლისგან შესაბამის ფორმაში მნიშვნელობის შეტანას.

გარდა ზემოთ აღნიშნულისა, კიდევ არსებობს დისტრიბუციული ტრანზაქციები, ფუნქციონირების არეალი ცდება ერთ სისტემას, ანუ ტრანზაქცია მუშავდება რამდენიმე აპლიკაციასა და ჰოსტს შორის ერთდროულად. დისტრიბუციულ ტრანზაქციებში ACID მიდგომა განხორციელებულია ერთდროულად მრავალ სისტემასა და მონაცემთა საცავებზე, რომლებიც შესაძლოა შედგებოდნენ მონაცემთა ბაზებისგან, ფაილური სისტემებისაგან, მესიჯ-სისტემებისაგან და სხვა აპლიკაციებისგან. ამ ტიპის ტრანზაქციებში მაკოორდინებელი სერვისი უზრუნველყოფს ტრანზაქციის ყველა დანაყოფის შესრულებას ყველა შესაბამის სისტემაში. დისტრიბუციულ სისტემაში თუ ტრანზაქციის ნებისმიერი ერთი ნაწილი შეცდომით დამთავრდა, მაშინ ეს ტრანზაქცია ყველა სისტემაში დაუბრუნდება საწყის წერტილს.

3. მონაცემთა ბაზების ტრანზაქციების სტანდარტიზებული მართვა

მრავალმომხმარებლიანი მონაცემთა ბაზები მონაცემთა კონკურენციის, კონსისტენტურობისა და მთლიანობის პრობლემის მოსაგვარებლად იყენებენ სხვადასხვა ფორმის მონაცემთა locking-ს. Lock არის ერთსა და იმავე რესურსზე წვდომის მიერ გამოწვეული უარყოფითი ეფექტის პრევენციის მექანიზმი. ვინაიდან ეს პრობლემა მონაცემთა ბაზების ძირითად ტიპებში არსებობს, მოხდა მისი locking-ის მექანიზმის სტანდარტიზება.

3.1 ANSI/ISO ტრანზაქციის იზოლაციის დონეები

SQL სტანდარტი, რომელიც ადაპტირებულია ANSI და ISO/IEC-ზე, განსაზღვრავენ ტრანზაქციის იზოლაციის ოთხ დონეს. ეს დონეები განსხვავდებიან ერთმანეთისგან ტრანზაქციის პროცესზე ზეგავლენის ხარისხით.

ეს იზოლაციის დონეები განსაზღვრულები არიან იმ შემთხვევებისათვის, რომელთა პრევენციაც უნდა მოხდეს კონკურენტულად შესრულებული ტრანზაქციების დროს. ასეთი მოვლენებია:

- **ბინძური წაკითხვა (dirty reads)** - ტრანზაქცია კითხულობს მონაცემებს, რომლებიც ჩაწერილია სხვა ტრანზაქციის მიერ, თუმცა ჯერ არ არის დაკომიტებული.
- **არაგანმეორებადი წაკითხვა (Nonrepeatable/fuzzy reads)** - ტრანზაქცია ხელმეორედ კითხულობს მონაცემებს, რომელიც ერთხელ უკვე წაიკითხა და აღმოჩნდება, რომ სხვა ტრანზაქციას განუხორციელებია ამ მონაცემებზე update ან delete ბრძანება და დაუკომიტებია.

```
1   /* SESSION 1 */
2   SELECT FirstName,LastName
3   FROM Person.Person
4   WHERE LastName = 'Jones';
5
6   | FIRSTNAME | LASTNAME |
7   | Peter     | Jones    |
8
9
10  /* SESSION 2 */
11  UPDATE Person.Person
12  SET FirstName = 'James'
13  WHERE LastName = 'Jones';
14  COMMIT;
15
16  /* SESSION 1 */
17  SELECT FirstName,LastName
18  FROM Person.Person
19  WHERE LastName = 'Jones';
20
21  | FIRSTNAME | LASTNAME |
22  | James     | Jones    |
23
```

- **Phantom reads** - ტრანზაქცია ხელმეორედ კითხულობს მონაცემებს, რომელიც ერთხელ უკვე წაიკითხა და აღმოჩნდება, რომ სხვა ტრანზაქციას განუხორციელებია ამ მონაცემებზე insert ბრძანება, დაუმატებია ახალი ჩანაწერი და დაუკომიტებია. მაგალითისათვის, თანამშრომლების ცხრილიდან ვიღებთ თანამშრომელთა რაოდენობას. 5 წუთის მერე ვასრულებთ იმავე ქმედებას და შედეგი სხვა გვხდება, ვინაიდან მოხდა ჩანაწერების დამატება სხვა ტრანზაქციის მიწერ. წინა შემთხვევისგან განსხვავებით, ყველა ის ჩანაწერი იგივე დარჩა რაც პირველ მოთხოვნაზე იყო, გაიზარდა მხოლოდ ჩანაწერების რაოდენობა.

SQL სტანდარტი ამ ტიპის შემთხვევებისათვის განსაზღვრავს იზოლაციის დონეებს. დონეები წარმოდგენილია შემდეგ ცხრილზე:

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

4. მონაცემთა კონკურენტულობა და კონსისტენტურობა

მონაცემთა კონკურენტულობის საკითხი დღესდღეობით ძალიან აქტუალურია თანამედროვე მონაცემთა ბაზებში, რომლებიც ემსახურებიან ერთდროულად ათასობით და შესაძლოა მილიონობით მომხმარებელსაც კი.

ერთმომხმარებლიან მონაცემთა ბაზაში მომხმარებელს შეუძლია განახორციელოს მონაცემთა მანიპულაცია ისე, რომ უგულებელყოს ისეთი სიტუაცია, როდესაც სხვა მომხმარებელი შესაძლოა იმავე მონაცემის ცვლილებას ახორციელებდეს.

მრავალმომხმარებლიანი მონაცემთა ბაზა უნდა უზრუნველჰყოფდეს შემდეგს:

- იძლეოდეს გარანტიას, რომ მომხმარებლებს ექნებათ წვდომა მონაცემებთან ერთდროულად (მონაცემთა კონკურენტულობა).
- იძლეოდეს გარანტიას, რომ თითოეული მომხმარებელი ხედავდეს მონაცემთა კონსისტენტურ, თანმიმდევრულ სახეს, ასევე უნდა ხედავდეს საკუთარი ტრანზაქციის ფარგლებში განხორციელებულ ცვლილებებს და სხვა მომხმარებლების შეტანილ ცვლილებებს commit-ის შემდეგ.

როდესაც ტრანზაქცია მოქმედებს კონკურენტულად, კონსისტენტური ტრანზაქციის ქცევის აღსაწერად მონაცემთა ბაზების მკვლევარები განსაზღვრავენ ტრანზაქციის იზოლაციის მოდელს სახელად „სერიალიზაცია“. სერიალიზირებული ტრანზაქცია მოქმედებს გარემოში, რომელიც წარმოაჩენს ტრანზაქციას ისე, რომ თითქოს სხვა მომხმარებლები არ ახორციელებდნენ მონაცემთა მანიპულაციას ბაზაში.

ბევრი აპლიკაციის მუშაობას სერიალიზებულ რეჟიმში მონაცემთა ბაზაში შეუძლია გამოიწვიოს მნიშვნელოვანი პრობლემები. კონკურენციის სრული იზოლაციის დროს შესაძლოა მივიჩნიოთ, რომ ერთ ტრანზაქციას არ შეიძლია დაასრულოს insert-ის პროცესი ცხრილში, თუ ამ ცხრილზე არსებობს მოთხოვნა სხვა ტრანზაქციიდან. მოკლედ რომ ვთქვათ, რეალურ სამყაროში, ჩვეულებრივ, გვიწევს კომპრომისი გავაკეთოთ ტრანზაქციის იზოლაციასა და მოთხოვნათა წარმადობას შორის.

მრავალვერსიული წაკითხვის კონსისტენტურობა. Oracle-ს მონაცემთა ბაზებში, მრავალვერსიულობა იძლევა შესაძლებლობას, ერთდროულად არსებობდეს მონაცემთა ერთზე მეტი ვერსია.

Oracle-ში მოთხოვნებს გააჩნიათ შემდეგი მახასიათებლები:

- Read-consistent queries - მოთხოვნის მიერი დაბრუნებული მონაცემები არიან სრულად ჩაწერილი ბაზაში (commited) და კონსისტენტური დროის კონკრეტულ

```
1  /* SESSION 1 */
2  BEGIN TRANSACTION;
3  UPDATE Person.Person
4  SET FirstName = 'James'
5  WHERE LastName = 'Jones';
6  WAITFOR DELAY '00:00:05.000';
7  ROLLBACK TRANSACTION;
8  SELECT FirstName,LastName
9  FROM Person.Person
10 WHERE LastName = 'Jones';
11
12 /* SESSION 2 */
13 SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
14 SELECT FirstName ,LastName
15 FROM Person.Person
16 WHERE LastName = 'Jones';
17
```

მონაკვეთისთვის.

როგორც ცნობილა, Oracle-ის მონაცემთა ბაზები არასდროს უშვებს „ბინძური წაკითხვის" (dirty read) არსებობას, რომელიც ხდება მაშინ, როდესაც ტრანზაქცია კითხულობს სხვა ტრანზაქციის მიერ ჯერ კიდევ დაუსრულებელ მონაცემებს - uncommitted data. პირველი სესია აკეთებს Rollback ბრძანებას და შეცვლილი მონაცემი უბრუნდება საწყის, ძველ მნიშვნელობას, თუმცა მეორე ტრანზაქცია აგრძელებს განახლებული მონაცემების გამოყენებას, ამ დროს წარმოიქმნება dirty read. ეს კი საფრთხეს უქმნის მონაცემთა კონსისტენტურობას, foreign keys-ს და ასევე უნიკალურობის შეზღუდვას. განვიხილოთ შემდეგი მარტივი მაგალითი MS SQL server-ის მაგალითზე:

პირველი სესია ხსნის ტრანზაქციას, აახლებს LastName სვეტის მნიშვნელობას და rollback-მდე ელოდება 5 წამი. ამ 5 წამის განმავლობაში მეორე სესია თუ გამოიყენებს "SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED" ბრძანებას და მოითხოვს იგივე მონაცემებს რაზეც პირველი სესია აკეთებს მანიპულაციას, მაშინ დაინახავს ბაზაში ჩაუწერელ ინფორმაციას, სწორედ ამას გულისხმობს Dirty Read.

- Nonblocking queries - მონაცემების წამკითხველი და ჩამწერი სესიები არ ბლოკავენ ერთმანეთს.

წაკითხვის კონსისტენტურობა ბრძანების დონეზე (Statement-Level Read Consistency) – Oracle ყოველთვის უზრუნველყოფს იმის გარანტიას, რომ მოთხოვნის მიერ დაბრუნებული მონაცემები არის „committed" და კონსისტენტური დროის კონკრეტული პერიოდისთვის. დროის ეს პერიოდი დამოკიდებულია ტრანზაქციის იზოლაციის ხარისხსა და მოთხოვნის ბუნებაზე:

- „read committed isolation level"-ის დროს ამ მომენტად მიიჩნევა ბრძანების გახსნის დროს. მაგალითისათვის, select-ის ბრძანება გაიხსნა SCN 1000-ზე, მაშინ მონაცემები კონსისტენტური იქნება SCN 1000 ვარიანტზე.
- სერიალიზებულ ან მხოლოდ წამკითხველი ტრანზაქციების დროს ამ მომენტად მიიჩნევა ტრანზაქციის დაწყების დრო. მაგალითისათვის, თუ ტრანზაქცია დაიწყო SCN 1000-ზე, ყველა ბრძანება ამ ტრანზაქციის ფარგლებში იქნება კონსისტენტური SCN 1000 ვერსიის.
- Flashback მოთხოვნების (SELECT ... AS OF) დროს მანუალურად ხდება კონკრეტული პერიოდის განსაზღვრა დროის ან SCN-ის მითითებით.

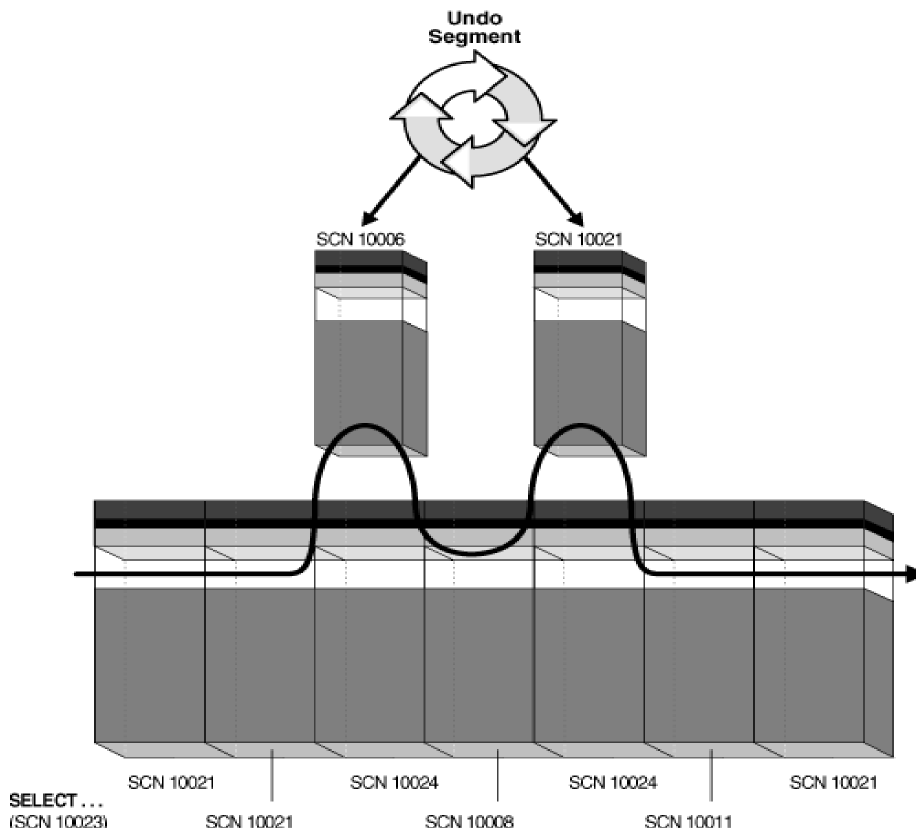
წაკითხვის კონსისტენტურობა ტრანზაქციის დონეზე - Oracle ასევე გვთავაზობს წაკითხვის კონსისტენტურობას ტრანზაქციის ჭრილში ყველა მოთხოვნისათვის, ამ მიდგომას ეწოდება წაკითხვის კონსისტენტურობა ტრანზაქციის დონეზე. ამ შემთხვევაში ტრანზაქციის ყველა ბრძანება მონაცემებს ხედავს დროის ერთი პერიოდისთვის, როდესაც ტრანზაქცია დაწყო.

სერიალიზებული ტრანზაქციების მოთხოვნები ხედავენ ამ ტრანზაქციაშივე მომხდარ მონაცემთა მანიპულაციებს. მაგალითისათვის, იმ ტრანზაქციის მოთხოვნა, რომელში შეიცვალა მონაცემი, ხედავს ამ შეცვლილ მონაცემებს. კონსისტენტურობის ეს დონე გვაძლებს „განმეორებადი წაკითხვის" (repeatable read) საშუალებას და გვიცავს phantom read-ისაგან.

წაკითხვის კონსისტენტურობა და ანულირების სეგმენტი (Undo Segment) - მრავალვერსიული წაკითხვის კონსისტენტურობის მოდელის გამართულად მუშაობისათვის მონაცემთა ბაზამ უნდა შექმნას მონაცემთა კონსისტენტური სიმრავლეები, როდესაც ხდება ცხრილის ერთდროული წაკითხვა და ცვლილება.

Oracle-ს მონაცემთა ბაზები ამ მიზნის მისაღწევად იყენებს Undo Segment-ს.

როდესაც მომხმარებელი ცვლის მონაცემებს, მონაცემთა ბაზაში იქნება undo ერთეულები, რომლებიც იწერება undo segment-ში. იგი შეიცავს მონაცემთა ძველ მნიშვნელობებს, რომლებიც შეცვლილია დაუკომიტებელი ან ბოლოს დაკომიტებული ტრანზაქციის მიერ. ერთი და იმავე მონაცემის მრავალი ვერსია შესაძლოა არსებობდეს ბაზაში. მონაცემთა ბაზას შეუძლია გამოიყენოს მონაცემთა სხვადასხვა დროის სნეფშოტები, რათა უზრუნველყოს მონაცემთა კონსისტენტური ხედვის საშუალება და არამბლოკავი მოთხოვნების არსებობა.



5. მონაცემთა ბაზების ტრანზაქციის იზოლაციის დონეები Oracle Database სისტემებში

როგორც ნაშრომის მე-3 თავში ვნახეთ ANSI სტანდარტი განსაზღვრავს ტრანზაქციის იზოლაციის დონეებს. სტანდარტი განსაზღვრულია შემთხვევებისათვის, რომლებიც ან დაშვებული ან აკრძალული უნდა იყოს სხვადასხვა იზოლაციის დონეზე. Oracle-ის მონაცემთა ბაზებში განსაზღვრულია ტრანზაქციის შემდეგი იზოლაციის დონეები:

- Committed წაკითხვის იზოლაციის დონე (Read Committed Isolation Level)
- სერიალიზებადი წაკითხვის დონე (Serializable Isolation Level)
- მხოლოდ წამკითხავი იზოლაციის დონე (Read-Only Isolation Level)

5.1 Committed წაკითხვის იზოლაციის დონე

Oracle-ის მონაცემთა ბაზა ამ ტიპის იზოლაციის დონეს იყენებს გულისხმობის პრინციპით. Read Committed-ის დროს თითოეული მოთხოვნა, რომელიც სრულდება ტრანზაქციის ფარგლებში, ხედავს მხოლოდ მანამდე მომხდარი მოთხოვნების committed მონაცემებს. ეს იზოლაციის დონე კარგად ერგება ისეთ მონაცემთა ბაზის გარემოს, რომელშიც ნაკლებადაა მოსალოდნელი მონაცემთა კონფლიქტების დიდი რაოდენობა.

Committed წაკითხვის ტრანზაქციებში მოთხოვნა დაცულია იმ მონაცემების წაკითხვისაგან, რომელიც commit-დება ამ მოთხოვნის შესრულების დროს. მაგალითისათვის, მოთხოვნას ნახევრად უკვე დასკანერებული აქვს მილიონ ჩანაწერიანი ცხრილი, სხვა ტრანზაქცია აკეთებს update ბრძანებას ამ ცხრილის ჯერ კიდე დაუსკანერებლ ნაწილზე. ამ დროს პირველი მოთხოვნა ვერ ხედავს შეტანილ ცვლილებებს. თუმცა, ვინაიდან მონაცემთა ბაზა არ ზღუდავს ცვლილების უფლებას იმ დროს, როდესაც ხდება წაკითხვა იმავე ცხრილზე, სხვა ტრანზაქცია შესაძლოა ცვლიდეს მონაცემებს მოთხოვნის შესრულებებს შორის. აქედან გამომდინარე, ტრანზაქცია, რომელშიც ერთი და იგივე მოთხოვნა სრულდება ორჯერ, შეძლება გადააწყდეს dirty read-ისა და phantom read-ის პრობლემას.

5.2 წაკითხვის კონსისტენტურობა Read Committed Isolation Level-ში

Oracle-ს მონაცემთა ბაზა უზრუნველყოფს კონსისტენტურ შედეგს ყველა მოთხოვნისათვის ისე, რომ მომხმარებლის ჩარევა არ გახდეს საჭირო. იმპლიციტური მოთხოვნა, რომელსაც მიეკუთვნება “update მოთხოვნა where განყოფილებით”, ასევე გარანტირებულად აბრუნებს შედეგების კონსისტენტურ სიმრავლეს.

იმ შემთხვეაში თუ SELECT-ის სია შეიცავს PL/SQL ფუნქციას, მაშინ მონაცემთა ბაზა განსაზღვრავს წაკითხვის კონსისტენტურობას ბრძანების დონეზე იმ მოთხოვნისთვის, რომელიც მუშაობს PL/SQL-ის ფუნქციის ფარგლებში. მაგალითისათვის, ფუნქციას აქვს წვდომა იმ ცხრილზე, რომლის მონაცემიც შეიცვალა და დაკომიტდა სხვა მომხმარებლის მიერ. ფუნქციაში SELECT-ის ყველა შესრულებისათვის განისაზღვრება წაკითხვის კონსისტენტური სნეფშოტი.

5.3 კონფლიქტური ჩაწერა Read Committed ტრანზაქციებში

Read Committed ტრანზაქციებში კონფლიქტური ჩაწერა ხდება მაშინ, როდესაც ერთი ტრანზაქცია ცდილობს სტრიქონის ცვლილებას, რომელიც უკვე შეცვლილია სხვა ტრანზაქციის მიერ თუმცა uncommitted სტატუსშია, ზოგ შემთხვევაში ასეთ ტრანზაქციას უწოდებენ მბლოკავ ტრანზაქციას (blocking transaction). Read Committed ტრანზაქცია ელოდება მბლოკავ ტრანზაქციას მანამდე, სანამ არ მოხსნის ბლოკირებას მოთხოვნილ სტრიქონს.

მოვლენები შესაძლოა განვითარდეს ორი სახით:

- თუ მბლოკავი ტრანზაქცია roll back-ს შეასრულებს, მაშინ ტრანზაქცია, რომელიც ელოდება, გააგრძელებს მუშაობას და შეცვლის შესაბამის სტრიქონს ისე, თითქოს სხვა კონკურენტული ტრანზაქცია არც არსებულა.
- თუ მბლოკავი ტრანზაქცია commit-ს შეასრულებს და მოხსნის ბლოკირებას, მაშინ ტრანზაქცია, რომელიც ელოდება, გააგრძელებს მუშაობას და შეცვლის უკვე შეცვლილ სტრიქონს ახალი მნიშვნელობით.

ქვემოთ მოყვანილი მაგალითი აჩვენებს, ერთი ტრანზაქცია, რომელიც შესაძლოა იყოს სერიალიზებული ან read committed, როგორ ურთიერთქმედებს მეორე read committed ტრანსაქციასთან. ნაჩვენებია კლასიკური მაგალითი, რომელიც ცნობილია “დაკარგული განახლების“ (lost update) სახელით. პირველი ტრანზაქციის მიერ განხორციელებულ ცვლილება არ აისახება ცხრილში იმ შემთხვევაშიც, თუ პირველი ტრანზაქცია დაკომიტდება. ამ პრობლემაზე გამკლავება პროგრამული დეველოპმენტის მნიშვნელოვანი ნაწილია.

სესია 1	სესია 2	აღწერა
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 6200 Lasha 9500	ქმედება არ ხდება.	სესია 1 ითხოვს გიორგის, ლაშას და გივის ხელფას. სახელით "გივი" ჩანაწერი არ მოიძებნა.
SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Giorgi';	ქმედება არ ხდება.	სესია 1 იწყებს ტრანზაქცია 1-ს გიორგის ხელფასის განახლებით. ნაგულისხმები ტრანზაქციის იზოლაციის დონე არის READ COMMITTED.
ქმედება არ ხდება.	SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	სესია 2 იწყებს ტრანზაქცია 2-ს და ტრანზაქციის იზოლაციის დონეს ხელოვნური ჩარევის საშუალებით აყენებს READ COMMITTED-ზე.
ქმედება არ ხდება.	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 6200 Lasha 9500	ტრანზაქცია 2 ითხოვს გიორგის, ლაშას და გივის ხელფასებს. Oracle-ს მონაცემთა ბაზა იყენებს წაკითხვის კონსისტენტურობას რათა აჩვენოს იმ დგომარეობით, სანამ მოხდებოდა ტრანზაქცია 1-ის მიერ uncommitted ცვლილებები.
ქმედება არ ხდება.	SQL> UPDATE employees SET salary = 9900 WHERE last_name='Lasha';	ტრანზაქცია 2 ცვლის ლაშას ხელაფასს ბაზაში წარმატებით, ვინაიდან ტრანზაქცია 1 ბლოკავს მხოლოდ გიორგის ჩანაწერს.
SQL> INSERT INTO	ქმედება არ ხდება.	

employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Givi', JGivi', SYSDATE, 'SH_CLERK');		ტრანზაქცია 1 ამატებს ახალ ჩანაწერს გივისთვის, თუმცა არ აკომიტებს.
ქმედება არ ხდება.	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 6200 Lasha 9900	ტრანზაქცია 2 ითხოვს ინფორმაციას გიორგის, ლაშას და გივის ხელფასის შესახევ. ტრანზქცია 2 ხედავს თავის შეცვლილ ჩანაწერს ლაშასთვის და ვერ ხედავს uncommited ცვლილებეს გიორგისა და გივისთვის რომელიც განახრციელა ტრანზაქცია 1-მა.
ქმედება არ ხდება.	SQL> UPDATE employees SET salary = 6300 WHERE last_name = 'Giorgi'; -- prompt does not return	ტრანზაქცია 2 ცდილობს შეცვალოს ჩანაწერი გიორგისთვის, რომელიც ამჟამად ბლოკირებულია ტრანზაქცია 1-სგან, ამით კი იქნება ჩაწერის კონფლიქტი. ტრანზაქცია 2 ელოდება იქამდე, სანამ ტრანზაქცია 1 არ დაასრულებს მუშაობას
SQL> COMMIT;	ქმედება არ ხდება.	ტრანზაქცია 1 აკეთებს commit-ს და ასრულებს მუშაობას.
ქმედება არ ხდება.	1 row updated. SQL>	გიორგის სტრიქონზე ბლოკი მოიხსნა, ტრანზაქცია 2 აგრძელებს მოქმედებას და ცვლის გიორგის ხელფასის მნიშვნელობას
ქმედება არ ხდება.	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', Givi');	ტრანზაქცია 2 ითხოვს გიორგის, ლაშას და გივის ხელფასებს. გივის ჩანაწერი, რომელიც დაემატა და დაკომიტრა ტრანზაქცია 1-ის მიერ უკვე ხედვადია ტანზაქცია 2-ში. ტრანზაქცია

	LAST_NAME SALARY ------------- ---------- Giorgi 6300 Lasha 9900 Givi	2 ასევე ხედავს საკუთარ ცვლილებებს გიორგის ხელფასზე.
ქმედება არ ხდება.	COMMIT;	ტრანზაქცია 2 აკეთებს commit-ს თავისი ქმედებების და სრულდება ტრანზაქცია 2.
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 6300 Lasha 9900 Givi	ქმედება არ ხდება.	სესია 1 ითხოვს ჩანაწერებს გიორგიზე, ლაშაზე და გივიზე. გიორგის ხელფასი არის 6300, რომელიც განახლებულია ტრანზაქცია 2-ის მიერი. ტრანზაქცია 1-ის განახლება, რომელმაც გიორგის ხელფასს მიანიჭა მნიშვნელობა 7000, დაიკარგა.

5.4 სერიალიზებადი იზოლაციის დონე (Serializable Isolation Level)

სერიალიზებადი იზოლაციის დონის დროს ტრანზაქცია ხედავს committed ცვლილებებს ტრანზაქციის და არა - მოთხოვნის შესრულების დაწყებამდე და ასევე ცვლილებებს, რომლებიც მოხდა ამ ტრანზაქციის ჭრილში. სერიალიზებული ტრანზაქცია მოქმედებს ისეთ გარემოში, თითქოს სხვა მომხმარებლები არაფერ ცვლილებებს არ ახორციელებდნენ მონაცემთა ბაზაში.

სერიალიზებადი იზოლაციის დონე კარგად ერგება შემდეგი ტიპის გარემოებს:

- დიდი მონაცემთა ბაზები მცირეოდენი ტრანზაქციებით, სადაც ხდება მხოლოდ მცირე რაოდენობით update-ები მონაცემებზე.
- მონაცემთა ბაზები, სადაც მცირეა იმისი შანსი, რომ ორი კონკურენტული ტრანზაქცია ცვლიდეს ერთსა და იმავე ჩანაწერს.

- მონაცემთა ბაზები, სადაც დიდი ხანი მომუშავე ტრანზაქციები უმეტეს წილად აკეთებენ მხოლოდ წაკითხვის ოპერაციას.

სერიალიზებად იზოლაციის დონეში წაკითხვის კონსისტენტურობა, ჩვეულებრივ, მიიღწევა ბრძანების დონეზე და ხდება მისი განვრცობა მთიან ტრანზაქციაზე. ტრანზაქციის მიერ წაკითხული ყველა სტრიქონი უცვლელი უნდა დარჩეს, როდესაც ხდება მისი ხელახალი წაკითხვა. ყველა ქუერი გარანტირებულად უნდა აბრუნებდეს იდენტურ შედეგს ტრანზაქციის განმავლობაში, ასე რომ, სხვა ტრანზაქციის მიერ განხორციელებული ცვლილებები ვერ უნდა დაინახოს პირველი ტრანზაქციის მოთხოვნამ, იმისდა მიუხედავად, თუ რამდენ ხანს იმუშავებს მოთხოვნა. სერიალიზებად ტრანზაქციაში ვერ შევხვდებით მონაცემთა ბაზის შემდეგ ფენომენებს: dirty reads, fuzzy reads, or phantom reads.

Oracle-ს მონაცემთა ბაზაში სერიალიზებად ტრანზაქციის დაწყება შესაძლებელია მხოლოდ მაშინ, თუ მოთხოვნილ სტრიქონებზე ცვლილებები, რომელთაც სხვა ტრანზაქციები ახორციელებენ, უკვე დაკომიტებულია. მონაცემთა ბაზა აგენერირებს შეცდომას იმ შემთხვევაში, თუ სერიალიზებადი ტრანზაქცია ცდილობს შეცვალოს ან წაშალოს ჩანაწერი, რომელიც შეიცვალა სხვა ტრანზაქციის მიერ და დაკომიტდა მას შემდეგ, რაც სერიალიზებად ტრანზაქცია დაიწყო. გენერირდება შემდეგი სახის შეცდომა:

```
ORA-08177: Cannot serialize access for this transaction
```

როდესაც სერიალიზებადი ტრანზაქცია გადის `ORA-08177` შეცდომაზე, აპლიკაციაში შესაძლოა მოხდეს რამდენიმე სახის ქმედება:

- Commit იმ ქმედებების რაც მოხდა ამ მდგომარეობამდე.
- შესრულდეს დამატებითი ბრძანება, მაგალითად დაუბრუნდეს ტრანზაქციაში შეცდომამდე არსებულ savepoint-ს.
- მოხდეს სრული ტრანზაქციის roll back.

მომდევნო ცხრილი გვიჩვენებს იმას, თუ როგორ მოქმედებს სერიალიზებადი ტრანზაქცია სხვა ტრანზაქციებთან. თუ სერიალიზებადი ტრანზაქცია არ ცდილობს იმ ჩანაწერის შეცვლას, რომელიც დაკომიტდა სხვა ტრანზაქციის მიერ სერიალიზებადი ტრანზაქციის დაწყების შემდეგ, მაშინ სერიალიზებული წვდომის პრობლემა მოხსნილი იქნება.

სესია 1	სესია 2	აღწერა
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', Lasha', 'Givi'); LAST_NAME SALARY ------------- --------- - Giorgi 6200 Lasha 9500	ქმედება არ ხდება.	სესია 1 აკეთებს მოთხოვნა გიორგის, ლაშას და გივის ხელფასებზე. გივის სახელით ჩანაწერი არ მოიძებნა.
SQL> UPDATE employees SET salary = 7000 WHERE last_name='Giorgi';	ქმედება არ ხდება.	სესია 1 ტრანზაქცია 1-ს იწყებს გიორგის ხელფასის ცვლილებით. ნაგულისხმები იზოლაციის დონე არის READ COMMITTED.
ქმედება არ ხდება.	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	სესია 2 იწყებს ტრანზაქცია 2-ს და იზოლაციის დონეს აყენებს SERIALIZABLE-ზე
ქმედება არ ხდება.	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------ ---------- Giorgi 6200 Lasha 9500	ტრანზაქცია 2 ითხოვს გიორგის, ლაშას და გივის ხელფასებს. Oracle-ს მონაემთა ბაზა იყენებს წაკითხვის კონსისტენტურობას და მონაცემს აჩვენებს იმ დგომარეობით, სანამ ტრანზაქცია 1 განახორციელებდა uncommitted ცვლილებას.
ქმედება არ ხდება.	SQL> UPDATE employees SET salary = 9900	ტრანზაქცია 2 ცვლის ლაშას ხელფას წარმატებით, რადგან

	WHERE last_name = 'Lasha';	მხოლოდ გიორგის ჩანაწერია დაბლოკილი.
SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Givi', JGivi', SYSDATE,'SH_CLERK');	ქმედება არ ხდება.	ტრანზაქცია 1 ამატებს ჩანაწერს გივისთვის.
SQL> COMMIT;	ქმედება არ ხდება.	ტრანზაქცია 1 იძახებს commit-ს თავისი ქმედებებისთვის და ასრულებს ტრანზაქციას
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 7000 Lasha 9500 Givi	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------- --------- Giorgi 6200 Lasha 9900	სესია 1 ითხოვს გიორგის, ლაშას და გივის ხელფასებს და ხედავს ტრანზაქცია 1-ის შეცვლილ და დაკომიტებულ მონაცემებს. სესია 1 ვერ ხედავს ტრანზაქცია 2-ის uncommited ცვლილებას ლაშას ჩანაწერზე. ტრანზაქცია 2 ითხოვს გიორგის, ლაშას და გივის ხელფასებს. Oracle-ს მონაცემთა ბაზა წაკითხვის კონსისტენტურობის მიზნით ტრანზაქცია 2-ის მიერ გივიზე დამატებულ და გიორგიზე შეცვლილ და დაკომიტებულ ცვლილებებს არ აჩვენებს. ტრანზაქცია 2 ხედავს მხოლოდ საკუთარ შეტანილ ცვლილებებს ლაშას ჩანაწერზე.
ქმედება არ ხდება.	COMMIT;	ტრანზაქცია 2 იძახებს commit-ს თავისი ქმედებებისთვის და ასრულებს ტრანზაქციას
SQL> SELECT last_name, salary FROM employees	SQL> SELECT last_name, salary FROM employees	ორივე სესია ითხოვს გიორგის, ლაშას, და გივის ხელფასებს. თითოეული ხედავს ყველა დაკომიტებულ

WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------- --------- - Giorgi 7000 Lasha 9900 Givi	WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------- --------- Giorgi 7000 Lasha 9900 Givi	ცვლილებას განხორციელებულს ტრანზაქცია 1-სა და ტრანზაქცია 2-ის მიერ.
SQL> UPDATE employees SET salary = 7100 WHERE last_name = 'Givi';	ქმედება არ ხდება.	სესია 1 იწყებს ტრანზაქცია 3-ს გივის ხელფასის ცვლილებით. ნაგულისხმები იზოლაციის დონე არის READ COMMITTED.
ქმედება არ ხდება.	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	სესია 2 იწყებს ტრანზაქცია 4-ს და იზოლაციის დონეს აყენებს SERIALIZABLE-ზე
ქმედება არ ხდება.	SQL> UPDATE employees SET salary = 7200 WHERE last_name = 'Givi'; -- prompt does not return	ტრანზაქცია 4 ცდილობს რომ განაახლოს გივის ხელფასი, მაგრამ მოთხოვნილი სტრიქონი დაბლოკილია ტრანზაქცია 3-ის მიერ. ტრანზაქცია 4 დგება რიგში.
SQL> COMMIT;	ქმედება არ ხდება.	ტრანზაქცია 3 იძახებს commit-ს გივის ხელფასის განახლებისთვის და ასრულებს ტრანზაქციას
ქმედება არ ხდება.	UPDATE employees SET salary = 7200 WHERE last_name = 'Givi' * ERROR at line 1:	commit რომელიც ასრულებს ტრანზაქცია 3-ში გივის განახლების ჩაწერას, იწვევს ORA-08177 შეცდომას. პრობლემა წარმოიქმნა იმიტომ, რომ ტრანზაქცია 3-ის commit ბრძანება მოხდა

	ORA-08177: can't serialize access for this transaction	მას შემდეგ რაც სერიალიზაბადი ტრანზაქცია 4 დაიწყო.
ქმედება არ ხდება.	SQL> ROLLBACK;	სესია 2 აკეთებს ტრანზაქცია 4-ის roll back-ს, რაც ასრულებს ტრანზაქციას.
ქმედება არ ხდება.	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	სესია 2 იწყებს ტრანზაქცია 5-ს და იზოლაციის დონეს აყენებს SERIALIZABLE-ზე
ქმედება არ ხდება.	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Giorgi', 'Lasha', 'Givi'); LAST_NAME SALARY ------------- ---------- Giorgi 7000 Lasha 9500 Givi 7100	ტრანზაქცია 5 კვლავ ითხოვს გიორგის, ლაშას და გივის ხელფასებზე ინფორმაციას. გივის ხელფასი, რომელიც ტრანზაქცია 3-მა შეცვალა და დააკომიტა, უკვე ხილვადა მიმდიანრე ტრანზაქციაში.
ქმედება არ ხდება.	SQL> UPDATE employees SET salary = 7200 WHERE last_name='Givi'; 1 row updated.	ტრანზაქცია 5 ცვლის გივის ხელფასს. ვინაიდან გივის ხელფასი შეცვლილი და დაკომიტებული ტრანზაქცია 3-ის მიერ სანამ ტრანზაქცია 5 დაიწყებოდა, მოხდა სერიალიზებული წვდომის პრობლემის თავიდან არიდება. შენიშვნა: სხვა ტრანზაქციას რომ შეეცვალა გივის ხელფასის მნიშვნელობა ტრანზაქცია 5 დაწყების შემდეგ, კვლავ იმავე პრობლემას გადავაწყდებოდთ.

ქმედება არ ხდება.	`SQL> COMMIT;`	სესია 2 იძახებს commit-ს მომხდარი ცვლილებებისთ ყოველგვარი პრობლემის გარეშე და ასრულებს ტრანზაქციას.

5.5 მხოლოდ წამკითხველი იზოლაციის დონე (Read-Only Isolation Level)

Read-Only იზოლაციის დონე მსგავსია სერიალიზებადი იზოლაციის, მაგრამ მხოლოდ წამკითხველ ტრანზაქციაში მონაცემთა ცვლილება არ არის ნებადართული გარდა იმ შემთხვევებისა, როცა SYS მომხმარებელი აკეთებს ცვლილებას. მხოლოდ წამკითხველი ტრანზაქციის დროს ზემოთ ხსენებული შეცდომა ORA-08177-ის მოხდენა შეუძლებელია. მხოლოდ წამკითხველი იზოლაციის დონის გამოყენება ხელსაყრელია მაშინ, როდესაც გვესაჭიროება ისეთი რეპორტის ამოღება, რომელიც უნდა შედგებოდა იმ დროის კონსისტენტური მონაცემებისგან, როდისაც დაიწყო ტრანზაქცია.

Oracle-ის მონაცემთა ბაზა წაკითხვის კონსისტენტურობას აღწევს მონაცემების რეკონსტრუქციით Undo Segment-იდან. ვინაიდან undo segment-ების გამოყენება ხდება წრიული წესით, მონაცემთა ბაზას შეუძლია გადააწეროს undo მონაცემები ამ სეგმენტებს. დიდხანს მომუშავე რეპორტების დგანან რისკის წინაშე, რომ მოთხოვნილი undo მონაცემების გამოყენება ხდებოდეს ასევე სხვა ტრანზაქციის მიერ, რამაც შესაძლოა გამოიწვიოს „snapshot too old“ შეცდომა. Undo-ს შენარჩუნების პერიოდის კონფიგურაცია შესაძლებელია სისტემური პარამეტრების საშუალებით, რომლითაც ხდება მინიმუმი დროის განსაზღვრა იმისათვის, თუ რამდენი ხანი უნდა მოხდეს undo-ს შენარჩუნება, სანამ მოხდება მასზე ახალი მონაცემების გადაწერა.

6. ტანზაქციულობის მართვის მიდგომები თანამედროვე მონაცემთა ბაზებიში

როგორც მოგეხსენებათ, დღესდღეობით მონაცემთა ბაზების ბაზარზე რამდენიმე დიდი მიმწოდებელი მოქმედებს, მათგან ორი ყველაზე დიდი წარმომადგენელია Oracle და Microsoft. Oracle ავითარებს Oracle Database-ს, რომლის უკანასკნელი ვერსიაც ამ პერიოდისათვის გახლავთ Oracle 12c. Microsoft კი ცდილობს არ დათმოს ბაზრის

პოზიციები და ავითარებს Microsoft SQL Server-ს, რომლის ამჟამინდელი ბოლო ვერსია არის: Microsoft SQL Server 2016. ისინი ცდილობენ, მომხმარებელს მიაწოდონ სანდო პროდუქტი, რომელიც დააკმაყოფილებს მათ მოთხოვნებს ინფორმაციის საცავების მიმართ. საკმაოდ მოკლე ინტერვალებით გამოდის სისტემური განახლებები ამ პროდუქტებისათვის ახლად დამატებული ფუნქციონალითა თუ ძველის გაუმჯობესებული ვარიანტებით. იცვლება არსებული პრობლემებისადმი მიდგომები, გვხვდება ახალი გამარტივებული გადაჭრის გზები. იხვეწება ასევე კონცეფციებიც. თუმცა ძირითადი არსი და კონცეფციები იგივე რჩება. ვინაიდან ტრანზაქციულობა დღესაც ერთ-ერთი უმნიშვნელოვანესი ფაქტორია მონაცემთა ბაზებში, მისადმი მოთხოვნა თანამედროვე ბაზებშიც დიდია მომხმარებლების მხრიდან.

6.1 ტრანზაქციების მართვა Oracle Database გარემოში

თანამედროვე მონაცემთა ბაზებში ვხვდებით ბაზების ტრანზაქციულობის გადაწყვეტის სხვადასხვა გზებს. როგორც ზემოთ აღვნიშნეთ, ტრანზაქციას Oracle-ც განიხილავს, როგორც შესრულებული სამუშაოს ლოგიკურ დანაყოფს, რომელიც შედგება ერთი ან მეტი SQL ბრძანებისაგან. ტრანზაქცია არის ატომური ერთეული. SQL ბრძანებების შედეგები ან უნდა დასრულდეს commit-ით ან უნდა მოხდეს მთლიანი Roll back.

Oracle SQL-ის იმპლემენტაციას ახორციელებს ისეთი ძლიერი პროგრამული ენის საშულებით, როგორიცაა პროცედურული ენა PL/SQL. ამ ენაში ტრანზაქციულობის პრობლემის გადაჭრის რამდენიმე ვარიანტს ვხვდებით.

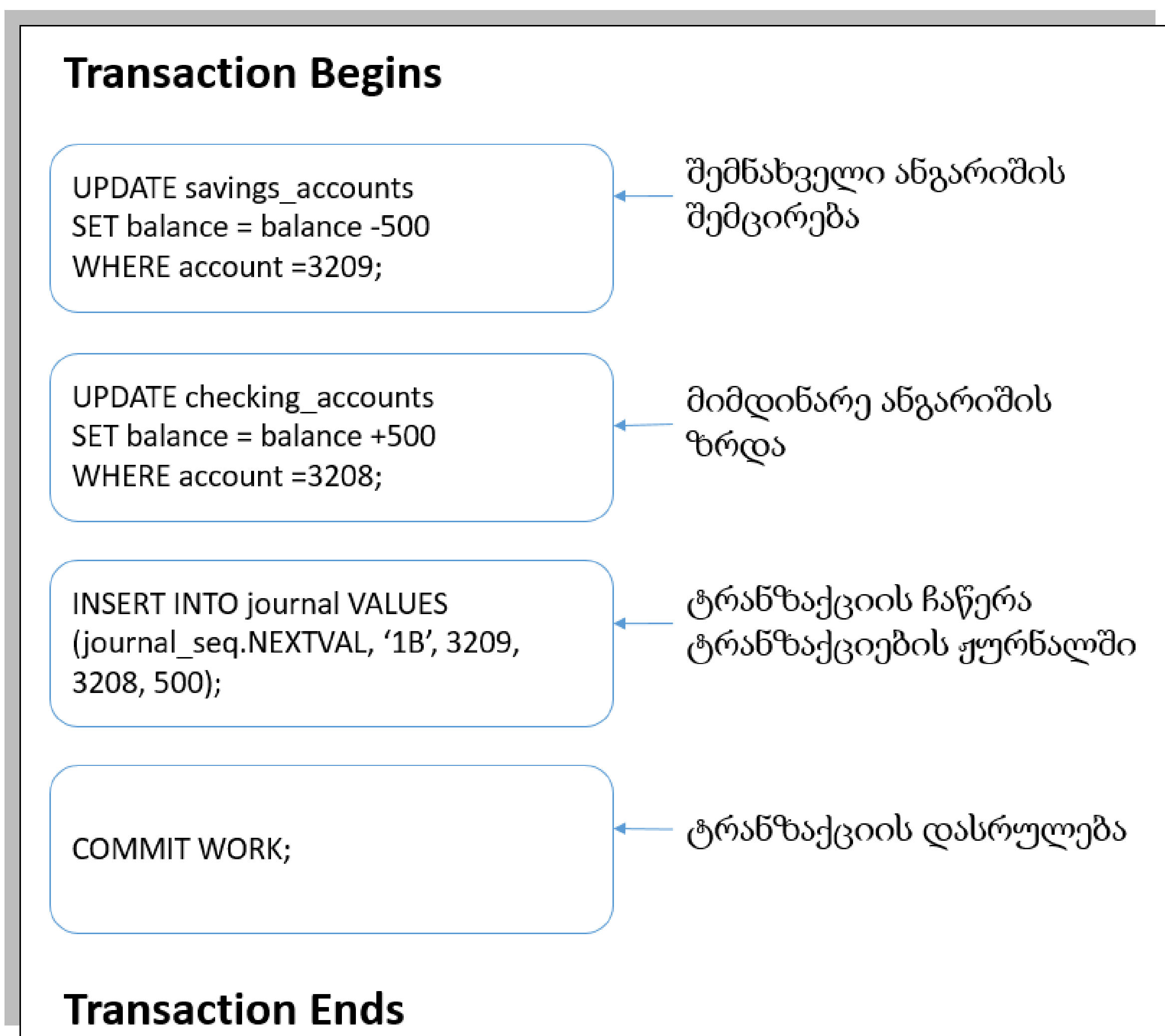
Oracle-ს პროგრამულ ბლოკში ტრანზაქცია იწყება პირველი შესრულებადი SQL ბრძანებით. ტრანზაქცია მთავრდება როდესაც ხდება commit ან rollback ბრძანება, არ აქვს მნიშვნელობა ეს მოხდება ხელოვნური ჩარევის საშუალებით თუ დამოუკიდებლად, როდესაც DDL ბრძანება იქნება გამოყენებული ტრანზაქციაში, ამ უკანასკნელ შემთხვევაში, თუ სტრუქტურის ცვლილების ბრძანება იქნება გამოყენებული, oracle-ს სისტემაში ავტომატურად ხდება commit ბრძანება და სრულდება ტრანზაქცია.

Oracle-ში მონაცემთა ბაზებში ტრანზაქციის კონცეფციის ილუსტრაციისათვის განვიხილოთ ბანკის მონაცემთა ბაზა. როდესაც ბანკის მომხმარებელი ახორციელებს თანხის გადარიცხვას შემნახველი ანგარიშიდან მიმდინარე ანგარიშზე, წარმოიქმნება სამი დამოუკიდებელი ოპერაცია:

1. შემნახველი ანგარიშის შემცირება

2. მიმდინარე ანგარიშის ზრდა
3. ტრანზაქციის ჩაწერა ტრანზაქციების ჟურნალში

Oracle განიხილავს ორ შესაძლო ვარიანტს. თუ სამივე SQL ბრძანება შესაძლოა შესრულდეს და ანგარიშებიც შესაბამისად დაბალანსდეს, მაშინ შედეგი შესაძლოა ჩაწერილ იქნეს მონაცემთა ბაზაში. თუმცა თუ წარმოიქმნება პრობლემა, მაგალითად, არასაკმარისი თანხა, არასწორი ანგარიშის ნომერი, სისტემის ფიზიკური პრობლემა oracle იძლევა გარანტიას, რომ მხოლოდ პირველი ან მეორე ბრძანება არ შესრულდება დამოუკიდებლად, არამედ სრული ტრანზაქცია, ანუ სამივე ქმედება იქნება დაბრუნებული საწყის მდგომარეობაზე, ისე რომ ბალანსები ყველა ანგარიშზე იქნება სწორი.



6.2 SQL ბრძანების შესრულება და ტრანზაქციის კონტროლი

SQL ბრძანება, რომელიც წარმატებით შესრულდა განსხვავდება დასრულებული ტრანზაქციისაგან. წარმატებით შესრულება გულისხმობს, რომ ტრანზაქცია იყო:

- პარსირებული

- სწორად კონსტრუირებული SQL
- იმუშავა შეცდომის გარეშე, როგორც ატომურმა ერთეულმა. მაგალითად, მრავალსტრიქონიანი update ბრძანების ყველას სტრიქონი არის განახლებული.

მიუხედავად ამისა, სანამ ტრანზაქციაში არ მოხდება commit ბრძანება, ტრანზაქციას შესაძლებელია გავუკეთოთ roll back და ყველა ცვლილება გავაუქმოთ.

მომდევნო მაგალითში სრულდება UPDATE ბრძანება შემდეგ ROLLBACK და შემდეგ კვლავ UPDATE ბრძანება. დინამიურ view-ში კი ჩანს, თუ როგორ ენიჭება ROLLBACK-იმ შემდეგ გახსნილ ტრანზაქციას ახალი უნიკალური იდენტიფიკატორი.

```
1   SQL> UPDATE hr.employees SET salary=salary;
2   107 rows updated.
3
4   SQL> SELECT XID, STATUS FROM V$TRANSACTION;
5
6   XID              STATUS
7   ---------------- ----------------
8   0800090033000000 ACTIVE
9
10  SQL> ROLLBACK;
11
12  Rollback complete.
13
14  SQL> SELECT XID FROM V$TRANSACTION;
15
16  no rows selected
17
18  SQL> UPDATE hr.employees SET last_name=last_name;
19
20  107 rows updated.
21
22  SQL> SELECT XID, STATUS FROM V$TRANSACTION;
23
24  XID              STATUS
25  ---------------- ----------------
26  0900050033000000 ACTIVE
27
```

Committing ნიშნავს, რომ მომხმარებელმა პირდაპირი ჩარევით ბაზას მოთხოვა, ტრანზაქციის ფარგლებში განხორციელებული ცვლილებები გამხდარიყო პერმანენტული. პირდაპირი ჩარევით მოთხოვნა გულისხმობს, რომ მომხმარებელმა გამოიძახა commit ბრძანება. დამოუკიდებელი მოთხოვნა commit-ის ხდება მაშინ, როდესაც ხდება

აპლიკაციის კავშირის სტანდარტული დასრულება ან როდესაც DDL ბრძანება მოხდება. ცვლილებები SQL ბრძანებების მიერ პერმანენტული და ხილვადი ხდება მხოლოდ commit ბრძანების შემდეგ.

Oracle-ის მონაცემთა ბაზების ასევე რის შესაძლებლობა, რომ გამოვიყენოთ სახელობითი ტრანზაქციები შემდეგი სინტაქსის საშუალებით: SET TRANSACTION … NAME, მანამდე სანამ დაიწყება ტრანზაქცია. იგი ამარტივებს დიდიხანი მომუშავე ტრანზაქციების მონიტორინგს და მართვას. ტრანზაქციას შეგვიძლია დავარქვათ მარტივი და დასამახსოვრებელი ტექსტის ტიპის სახელი. ეს სახელი იქნება შემხსენებელი იმის თაობაზე, თუ რას უკავშირდება ტრანზაქცია. ტრანზაქციის სახელი ანაცვლებს commit-ის კომენტარს დისტრიბუციული ტრანზაქციებისთვის და აქვს შემდეგი დადებითი თვისებები:

- მარტივია მონიტორინგი ხანგრძლივი ტრანზაქციების, განსაკუთრებით დისტრიბუციული ტრანზაქციების დროს.
- შესაძლებელია ტრანზაქციის სახელის ნახვა მის უნიკალურ იდენტიფიკატორთან ერთად შესაბამის აპლიკაციაში. მაგალითად, მონაცემთა ბაზის ადმინისტრატორს შეუცლია ნახოს ტრანზაქციის სახელი როდესაც სისტემის მონიტორინგს ახორციელებს „Enterprise Manager"-დან.
- ტრანზაქციის სახელი იწერება „transaction auditing redo record"-ში Oracle9i-სა და უფრო ახალ ვერსიებში.
- LogMiner-ის საშუალებით კონკრეტული ტრანზაქცია თავისი სახელით transaction auditing redo record log-ში.
- ტრანზაქციის სახელით შეგვიძლია მოვძებნოთ კონკრეტული ტრანზაქცია data dictionary view_ში, კონკრეტულად კი V$TRANSACTION-ში.

ტრანზაქციისათვის სახელის დარქმევის პრინციპი. როგორც ზემოთ აღვნიშნეთ, ტრანზაქციის სახელის გამოცხადება ხდება ტრანზაქციის დასაწყისში. როდესაც სახელს ვარქმევთ ტრანზაქციას, ამ სახელის ასოცირება ხდება შესაბამისი ტრანზაქციის უნიკალურ იდენტიფიკატორთან. არაა აუცილებელი, რომ ტრანზაქციის სახელი იყოს უნიკალური. განსხვავებულ ტრანზაქციებს შესაძლოა, რომ ჰქონდეთ იდენტური სახელები ერთსა და იმავე სქემაში. ამგვარად, შეგვიძლია გამოვიყენოთ ნებისმიერი სახელი, რომელიც საშუალებას მოგვცემს, კონკრეტული ტრანზაქცია გამოვარჩიოთ სხვა დანარჩენებისაგან.

Commit Comment ამ მეთოდის საშუალებით კომენტარი შეგვიძლია დავაკავშიროთ კონკრეტულ ტრანსაქციასთან. თუმცა კომენტარი შესაძლოა გამოყენებულ იქნეს მხოლოდ იმ შემთხვევაში როდესაც ტრანზაქცია commit-დება.

```
COMMIT
   COMMENT 'In-doubt transaction Code 36, Call (415) 555-2637';
```

Commit Comment ახალ ვერსიაში არის მხარდაჭერილი მხოლოდ backward compatibility-ის თვალსაზრისით. თუმცა Oracle-სგან მკაცრი რეკომენდაცია არსებობს, რომ არ გამოვიყენოთ ეს ფუნქცია. Commit Comment გამოყენება შეუძლებელია სახელიან ტრანზაქციებში. მომავალ ვერსიაში ეს ფუნქცია სავარაუდოდ ამოღებული იქნება.

7. Locking-ის მექანიზმი მონაცემთა ბაზებში

მონაცემთა ბაზის lock-ის მიზანია, დაბლოკოს მონაცემები ბაზაში ისე, რომ მონაცემთა ბაზის მხოლოდ ერთ მომხმარებელს/სესიას შეეძლოს ამ კონკრეტული მონაცემის შეცვლა. აქედან გამომდინარე, lock-ები არსებობენ იმისათვის, რომ გამორიცხონ ორი ან მეტი მომხმარებლის მეშვეობით ერთი და იმავე მონაცემის ცვლილება ზუსტად ერთსა და იმავე დროს. როდესაც მონაცემი დაბლოკილია, ეს იმას ნიშნავს, რომ მონაცემთა ბაზის სხვა სესიამ ვერ უნდა მოახერხოს ამ მონაცემის ცვლილება მანამდე, სანამ არ მოხდება ამ ბლოკის მოხსნა. როგორც ზემოთ აღვნიშნეთ, ბლოკის მოხსნა ხდება roll back ან commit ბრძანების საშუალებით.

რა ხდება, როდესაც სხვა სესია ცდილობს შეცვალოს დაბლოკილი მონაცემი - წარმოიდგინეთ რომ სესია 1 ცდილობს, რომ გააკეთოს მანიპულაცია ისეთ მონაცემზე, რომელიც დაბლოკილია სესია 2-ის მიერ. რა ხდება ამ დროს სესია 1-ში? სესია 1-ში ხდება მოვლენა, რომელსაც lock wait-ის მდგომარეობას უწოდებენ, სესია 1 არის მოლოდინის რეჟიმში და ყველა მომდევნო ბრძანება ამ სესიის ფარგლებში ელოდება ბლოკის მოხსნას. სხვადასხვა მომწოდებლის მონაცემთა ბაზებში განსხვავებული მიდგომებია locking-ის მექანიზმში - ისინი ერთმანეთისგან განსხვავებულად აგვარებენ lock wait-ის მდგომარეობას პრობლემას. ზოგიერთ მონაცემთა ბაზებში, მაგალითად DB2-ში, თუ სესია ელოდება lock-ის მოხსნას ძალიან დიდი ხანი, ხდება time out კონკრეტული დროის გასვლის შემდეგ და ლოდინის ნაცვლად ბრუნდება შეცდომა, შესაბამისად მანიპულაციის მოთხოვნაც იკარგება. თუმცა Oracle-ის მონაცემთა ბაზებში სხვაგვარი მიდგომა გვხვდება. Oracle-ში შესაძლებელია სესია lock wait-ის მდგომარეობაში დარჩეს უსასრულოდ.

7.1 მონაცემთა ბაზების locking-ის ტექნიკა

მონაცემთა ბაზის lock-ები შესაძლოა მოხდეს სხვადასხვა დონეზე, ეს საკითხი ცნობილია, როგორც „ბლოკირების გრანულაცია“ (lock granularity).

ქვემოთ განვიხილავთ ბლოკირების დონეებს და ტიპებს და ასევე იმას, თუ რა იგულისხმება ამ ტექნიკებში:

მონაცემთა ბაზის დონის ბლოკირება (Database level locking) - ამ ტიპის lock-ის დროს მთლიანი მონაცემთა ბაზა იბლოკება. რაც გულისხმობს იმას, რომ მხოლოდ მონაცემთა ბაზის ერთ სესიას შეუძლია განახორციელოს მონაცემების მანიპულაცია. Lock-ის ეს ტიპი ძალიან იშვიათად გამოიყენება, ვინაიდან ეს რეჟიმი ყველა მომხმარებელს, გარდა ერთისა, უკრძალავს მონაცემთა ბაზაში რაიმე სახის ცვლილებას. თუმცა მონაცემთა ბაზის ბლოკირების დონე შესაძლოა ხელსაყრელი გამოდგეს, როდესაც მონაცემთა ბაზაში ხდება მნიშვნელოვანი სისტემური განახლებები, როგორიცაა მაგალითად ბაზის ვერსიის განახლება და ა.შ. მაგალითისათვის, Oracle-ის მონაცემთა ბაზას აქვს ბლოკირების ექსკლუზიური რეჟიმი, რომელიც მხოლოდ ერთი სესიის არსებობას უშვებს ბაზაში, ეს რეჟიმი შესაძლოა მიჩნეული იქნეს მონაცემთა ბაზის lock-ის დონედ.

ფაილის დონის ბლოკირება (File level locking) - ამ ტიპის ბლოკირების დროს ხდება მონაცემთა ბაზის კონკრეტული ფაილის დაბლოკვა. მონაცემთა ბაზის ფაილი შესაძლოა, შეიცავდეს ბევრი ტიპის ინფორმაციას: მთლიან ცხრილს, ცხრილის ნაწილს ან სხვადასხვა ცხრილების ნაწილებს. ვინაიდან ბევრი ტიპის მონაცემი შესაძლოა ინახებოდეს მონაცემთა ბაზის ფაილში, ამ დონის ბლოკირება ძალიან იშვიათად გამოიყენება.

Page ან block-ის დონი ბლოკირება (Page or block level locking) - ამ დონის ბლოკირება ხდება როდესაც მონაცემთა ბაზის ფაილის page-ის ბლოკირება ხდება. როგორც ფაილის დონის ბლოკირების დროს, ვინაიდან ბევრი ტიპის მონაცემი შესაძლოა ინახებოდეს მონაცემთა ბაზის page-ში, ამ დონის ბლოკირება მიზანშეწონილი არ არის.

ბლოკირება სვეტის დონეზე (Column level locking) - ამ ტიპის ბლოკირება გულისხმობს, რომ ცხრილის სვეტი, რომელზეც ხდება მანიპულაცია, მთლიანად ექცევა ბლოკში. ბლოკირება სვეტის დონეზე იშვიათად გამოიყენება დღევანდელ მონაცემთა ბაზებში, ვინაიდან უმეტეს შემთხვევაში დიდ რესურს მოითხოვს. ამიტომ მონაცემთა ბაზების უმრავლესობაში ასეთი ტიპის lock არ არის მხარდაჭერილი.

ბლოკირება სტრიქონის დონეზე (Row level locking) - სტრიქონის დონი ბლოკირება ვრცელდება კონკრეტული ცხრილის განსაზღვრულ სტრიქონზე ან სტრიქონებზე. ბლოკირების ეს ტიპი გამოიყენება მონაცემთა ბაზებში ყველაზე ხშირად.

7.2 Locking-ის ქცევის შეჯამება Oracle-ს მონაცემთა ბაზების მაგალითზე

Oracle-ს მონაცემთა ბაზები შესასრულებელი ოპერაციის მიხედვით განსაზღვრავენ lock-ის რამდენიმე ტიპს.

ზოგადად, მონაცემთა ბაზებში გვხვდები შემდეგი ორი ტიპის lock: ექსკლუზიური ბლოკირება (exclusive lock) და საზიარო ბლოკირება (share lock). მხოლოდ ერთი exclusive lock შეიძლება წარმოიქმნას კონკრეტულ რესურსზე, როგორც შეიძლება იყოს სტრიქონი ან ცხრილი. Share lock-ის შემთხვევა განსხვავებულია, ვინაიდან ბევრი საზიარო ბლოკირება შესაძლოა, წარმოიქმნას ერთ კონკრეტულ რესურსზე.

Lock-ები ზეგავლენას ახდენენ როგორც წამკითხველ, ასევე ჩამწერ მოთხოვნებზე. რესურსებზე ჩამწერი მოთხოვნა წარმოიქმნება მაშინ, როდესაც ჩამწერი ბრძანება ცვლის რესურსის მნიშვნელობას. შემდეგი წესების საშუალებით შეგვიძლია, შევაჯამოთ locking-ის ქცევა Oracle-ის მონაცემთა ბაზებში წამკითხველი და ჩამწერი ბრძანებებისთვის:

- სტრიქონი იბლოკება მხოლოდ მაშინ, როდესაც ხდება მისი ცვლილება ჩამწერი მოთხოვნის მეშვეობით.

როდესაც ბრძანება მნიშვნელობას უცვლის ერთი სტრიქონს, ტრანზაქცია წარმოქნის lock-ს მხოლოდ ამ კონკრეტული სტრიქონისათვის. ცხრილის ბლოკირება სტრიქონის დონეზე მინიმალურს ხდის კონკურენციას მოთხოვნილი რესურსისთვის. ჩვეულებრივ, მონაცემთა ბაზა არ აკეთებს row lock-ის ესკალაციას page ან table level-ზე.

- სტრიქონის ჩამწერ ბრძანება ბლოკავს იმავე სტრიქონზე ჩამწერ კონკურენტულ მოთხოვნას.

თუ ერთი ტრანზაქცია ცვლის სტრიქონის მნიშვნელობას, მაშინ row lock იცავს ამ ტრანზაქციას, რომ სხვა ტრანზაქციის ბრძანებამ არ შეცვალოს იგივე სტრიქონი მასთან ერთად.

- წამკითხველი მოთხოვნა არასდროს ბლოკავს ჩამწერ მოთხოვნას.

ვინაიდან წამკითხველი არ ბლოკავს სტრიქონს, ჩამწერ ტრანზაქციას შეუძლია განახორციელოს მასე ცვლილებები. გამონაკლისია SELECT ... FOR UPDATE ბრძანება, რომელიც ერთ-ერთი განსაკუთრებული სახეობაა SELECT ბრძანების. ამ სახით მოთხოვნილი ბრძანება ბლოკავს ყველა მოთხოვნილ სტრიქონს და იცავს ცვლილებებისაგან. FOR UPDATE ბრძანების გამოყენება ხელსაყრელია, როდესაც ჯერ მომხმარებელი აკეთებს წაკითხვის მოთხოვნას, შემდეგ წაკითხულ მონაცემებზე აკეთებს დამუშავებებს და ბოლოს ცვლის მოთხოვნილი სტრიქონების მნიშვნელობებს.

- ჩამწერი მოთხოვნა არასდროს ბლოკავს წაკითხვის მოთხოვნებს.

როდესაც სტრიქონი იცვლება ჩამწერი ტრანზაქციის მიერ, მონაცემთა ბაზა იყენებს undo მონაცემებს, რათა წამკითხველმა მოთხოვნებმა დაინახონ კონსისტენტური მონაცემები შეცვლილ სტრიქონებზე მოთხოვნილი ინფორმაციისათვის. (შენიშვნა: არსებობს გამონაკლისი შემთხვევები, როდესაც მონაცემების წამკითხველები ელოდებიან ჩაწერის მოთხოვნის დასრულებას. ეს შემთხვევა ხდება მხოლოდ დისტრიბუციული ტრანზაქციების დროს).

7.3 Lock-ის მეთოდები Oracle Database-ში, მათი გარდაქმნა და ესკალაცია

Oracle-ის მონაცემთა ბაზა ავტომატურად იყენებს შეზღუდვის ყველაზე დაბალ შესაძლო დონეს, რათა უზრუნველყოს მაღალი ხარისხის კონკურენცია და ასევე მონაცემთა დაცული მთლიანობა (fail-safe data integrity). რაც დაბალია შეზღუდვის დონე, მით უფრო ხელმისაწვდომია მონაცემები მომხმარებლებისთვის. ამის საპირისპიროდ. რაც უფრო მაღალია შეზღუდვის დონე, მით უფრო ლიმიტირებულია ბევრი ტრანზაქციისაგან მონაცემებთან წვდომა.

მრავალმომხმარებლიან მონაცემთა ბაზაში როგორც სხვა მონაცემთა ბაზები, ასევე Oracle-ც იყენებს ექსკლუზიური ბლოკირების და საზიარო ბლოკირების მეთოდებს.

Oracle-ის მონაცემთა ბაზები საჭიროების შემთხევაში აკეთებენ „ბლოკირების გარდაქმნას“ (lock conversion). ასეთი ქმედების დროს მონაცემთა ბაზა დაბალი დონის შეზღუდვის ცხრილის lock-ს ცვლის რომელიმე უფრო მაღალი დონის შეზღუდვით.

მაგალითისათვის, წარმოვიდგინოთ, რომ ტრანზაქცია იძახებს SELECT ... FOR UPDATE ბრძანებას რომელიმე ჩანაწერისთვის და შემდეგ ცვლის ბლოკირებული სტრიქონის მნიშვნელობას. ამ შემთხვევაში მონაცემთა ბაზა row share table lock-ს ავტომატურად გარდაქმნის row exclusive table lock-ად. ექსკლუზიური ბლოკირება შენარჩუნებული იქნება ყველა დამატებული, განახლებული და წაშლილი სტრიქონისთვის, რომელიც მოხდა მიმდინარე ტრანზაქციაში. ვინაიდან row lock-ის ყველაზე მაღალი დონე იქნება მიღწეული, ბლოკირების მეთოდის მეტი ცვლილების აუცილებლობა აღარ იქმნება ტრანზაქციაში.

Lock-ის გარდაქმნა განსხვავდება lock-ის ესკალაციისაგან, რომელიც ხდება მაშინ, როდესაც ძალიან ბევრი lock იქმნება ერთი დონის ერთეულზე (მაგალითად, სტრიქონზე) და მონაცემთა ბაზა წარმოქმნის lock-ს უფრო მახალი დონის ერთეულისთვის (მაგალითად, მთლიან ცხრილზე). თუ მომხმარებელი ბლოკავს ბევრ სტრიქონს ერთ ცხრილზე, მაშინ ზოგიერთმა მონაცემთა ბაზამ შესაძლოა გადაწყვიტოს ავტომატურად, რომ ესკალაცია მომგებიანი იქნება და მთლიანი ცხრილზე გაავრცელოს ბლოკირება _ lock-ების რაოდენობა მცირდება, თუმცა შეზღუდვის ხარისხი, რა თქმა უნდა, იზრდება.

Oracle-ს მონაცემთა ბაზა არასდროს იყენებს ბლოკირების ესკალაციას, ვინაიდან ეს ზრდის deadlock-ების წარმოქმნის შესაძლებლობას. წარმოვიდგინოთ, რომ სისტემა ცდილობს ტრანზაქცია 1-ის ბლოკირების ესკალაციას, მაგრამ ეს შეუძლებელია ტრანზაქცია 2-ის უკვე არსებული ბლოკირების გამო. ასეთ შემთხვევაში წარმოიქმნება deadlock.

7.4 deadlock-ები Oracle-ს მონაცემთა ბაზებში

როგორც ზემოთ აღვნიშნეთ, deadlock არის სიტუაცია, როდესაც ორი ან მეტი მომხმარებელი ელოდება ერთმანეთის მიერ დაბლოკილ მონაცემებს. Deadlock-ები იცავს ტრანზაქციებს ასეთი ტიპის ბლოკირების დროს უსასრულო მუშაობისაგან.

Oracle-ს მონაცემთა ბაზა ავტომატურად აკეთებს deadlock-ების აღმოჩენას და მათ აღმოფხვრას deadlock-ში ჩართული ერთ-ერთი ტრანზაქციის roll back-ის საშუალებით, ასე ხსნის კონფლიქტური lock-ების სიმრავლეს. მონაცემთა ბაზა ტრანზაქციას, რომელსაც roll back-ს უკეთებს, უბრუნებს შესაბამის შეტყობინებას და ატყობინებს deadlock-ის მოხდენას. ჩვეულებრივ, ასეთი ტრანზაქციას ავტომატურად უკეთდება roll back, თუმცა შესაძლოა ხელმეორედ გაკეთდეს იგივე მოთხოვნა ბლოკირების დასრულების შემდეგ.

Deadlock-ები უმეტეს შემთხვევაში ხდება მაშინ, როდესაც ტრანზაქცია ხელოვნური ჩარევის საშუალებით თავს არიდებს Oracle-ს მონაცემთა ბაზის ნაგულისხმევ ბლოკირების მეთოდს. ვინაიდან Oracle არ აკეთებს ბლოკირების ესკალაციას და არ იყენებს მოთხოვნისთვის წაკითხვის ბლოკირებას, არამედ მუშაობს სტრიქონის დონის ბლოკირებაზე, Deadlock-ები Oracle-ს სისტემაში არც ისე ხშირია.

დასკვნა

როგორც ნაშრომში ჩანს, მონაცემთა ბაზებში ტრანზაქციების მართვა საინფორმაციო ტექნოლოგიების მიმართულებაში, კერძოდ კი მონაცემთა ბაზებში ერთ-ერთი ყველაზე კომპლექსური საკითხია. როგორც მონაცემთა ბაზების მწარმოებლები, ასევე მათი მომხმარებლები დიდი გამოწვევების წინაშე დგანან, გამომდინარე იქიდან, რომ მონაცემთა ბაზების ზომები, მათზე მიმართვიანობა და შესაბამისად, დატვირთვა ექსპონენციალურად იზრდება. ასეთ პირობებში კი ტრანზაქციების მართვაში ახალი მიდგომების დანერგვა ხდება საჭირო. როგორც მაგალითებიდან ჩანს, ნაპოვნია გზები, რომლითაც პრობლემები ნაწილობრივ აღმოფხვრილია, თუმცა ბოლომდე სრულყოფილი არ არის და ამ ეტაპზე ვერც იქნება. მონაცემთა ბაზების დამპროექტებლებს და დიზაინერებს კი ისღა დარჩენიათ, სწორად განსაზღვრონ თავიანთი სისტემის მოთხოვნები და იმის შესაბამისად გამოიყენონ მონაცემთა ბაზების მწარმოებლების მიერ შემოთავაზებული ტრანზაქციების მართვის მიდგომები.

გამოყენებული ლიტერატურა

1. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. M. J. Hernandez. *Addison-Wesley Professional*; 2013 წ.
2. Oracle Essentials: Oracle Database 12c. R. Greenwald. *O'Reilly Media*. 5 edition. 2013 წ.
3. Database Systems: Design, Implementation, & Management. C. Coronel. *Course Technology*. 12 edition. 2016 წ.
4. Training Kit Querying Microsoft SQL Server 2012. D. Sarka, I. Ben-Gan. *Microsoft Press*. 2012 წ.
5. OCP Oracle Database 12c Advanced Administration Exam Guide. B. Bryla. *McGraw-Hill Education*. 2015 წ.
6. Database Systems: A Practical Approach to Design, Implementation, and Management. T. Connolly, C. Begg. *Pearson*. 2014 წ.
7. Modern Systems Analysis and Design. J. A. Hoffer, J. George, J. S. Valacich. *Pearson*. 7 edition. 2013 წ.