29

Using Lock-Free Reservation

This chapter explains how to use lock-free reservation in database applications.

Topics:

29.1 About Concurrency in Transaction Processing

Transaction processing usually involves flat transactions with data updates that replace old values with new. Most applications store numeric and aggregate values, such as quantity-on-hand of products, bank account balances, stocks, and number of seats available. These numeric aggregate data, although referenced as a single entity, are based on one or more same or similar data. Such data involve subtraction or addition of the values rather than an assignment of the form “data ← value.” For example, a bank account balance is based on the debit and credit transactions that happen on the account.

Applications that provide inventory control, supply chain, financial or investment banking, stocks, travel, and entertainment operate on numeric aggregate data. A numeric aggregate data is generally identified as a “hot” resource because applications continuously and repeatedly read or update such data. The likelihood of many transactions concurrently accessing such data is high. In conventional locking protocols, such as two-phase locking (2PL), execution of concurrent transactions is serialized after a transaction initiates an update on a row and locks it. The initiated update is complete only when the transaction finalizes with a commit or a rollback. Serialization blocks other concurrent transactions from accessing the row until the completion of the transaction that has locked the row. If you allow concurrent transactions to access data, you must control the transactions to preserve application correctness.

In long-running transactions such as microservices applications, a resource remains locked for an extended period, potentially making it a hot resource. The long-term resource locking limits concurrency. In microservices applications that offer services like trip booking services, you may have flight, hotel, and car bookings in long-running transactions. These transactions are typically handled as Sagas and span different services with databases holding locks until the Saga finalization.

Let us look at an example of an online shopping cart to understand how you would handle concurrency in a normal course. A shopping cart has items added to the cart before being sold. Your application must be able to handle multiple transactions at various states like: when a user puts an item into the cart, the item becomes unavailable to other buyers and yet unsold. For concurrent transactions, managing the inventory states and count becomes complex with multiple transactions adding items to the cart and checking out or abandoning the cart. The inventory or similar fields must be locked for a transaction much before a commit or rollback can change the quantity. Locking data for long periods prevents other concurrent transactions

from accessing the item, until the lock is released. Therefore, aiming for isolation and allowing the application to run transactions independently limits concurrency.

For many business applications, blocking concurrent accesses to “hot” data can severely impact performance and reduce user experience and throughput. Applications can benefit if there is improved concurrency coupled with reduced isolation, while also maintaining the atomicity, consistency, and durability properties of transactions. To improve concurrency, it is important to capture the state of a hot resource during a transaction lifecycle and enable data locking only when the resource value is being modified.

> **See Also:**
>
> - Lock-Free Reservation for more information about how reservable columns are used to control concurrency in transaction processing.

29.2 Lock-Free Reservation Terminology

- Numeric Aggregate Data
- Hot Data or Hot Resource
- Transaction Lifecycle
- Compensation or Compensating Transaction
- Reservable
- Reservable Column
- Reservable Update
- Reservation Journal
- Lock-free Reservation
- Optimistic Locking
- Saga

Numeric Aggregate Data

Data that involves subtraction (consume, decrement) or addition (replenish, increment) of the quantity (numeric data) rather than an assignment of the form “data ← value”. Operations on numeric aggregate data are commutative in nature.

Hot Data or Hot Resource

Data or a resource that receives high traffic from transactions requiring frequent reads and updates or a resource that is read or updated in a long-running transaction.

Transaction Lifecycle

The states in a business transaction as it transitions from its creation to its committed or rollback state.

Compensation or Compensating Transaction

A compensation or compensating transaction compensates (rolls back) for the already committed transactions of a Saga if any other transaction that is also a part of the Saga fails.

Reservable

Reservable is a column property that you can define for a column that has a numeric data type. A reservable column keeps journals of modification made to the column. These journals are used for concurrency control and auto-compensation.

Reservable Column

A column that contains a hot resource and is identified for lock-free reservation. These columns are declared with a `reservable` column property keyword.

Reservable Update

Numeric aggregate data updates made on a reservable column.

Reservation Journal

A reservation journal is a table associated with the user table that records the modification (increase or decrease by a delta amount with respect to the current value) in the reservable column.

Lock-free Reservation

Any reservable update is treated as a lock-free reservation, implying that the transactions making the updates to a qualifying row do not lock the row, but indicate their intention to modify the reservable column value by a delta amount. The modify operation is recorded in a reservation journal. Lock-free reservations are transformed to actual updates at the commit of the transaction.

Lock-free reservation ensures that the lock is obtained only at the time of commit to update a reservable column. Lock-free reservation is used for hot (high-concurrency) data and are suited for microservices transaction models because of its implicit compensation support.

Optimistic Locking

Optimistic locking is a concurrency control method that allows transactions to use data resources without acquiring locks on those resources. Before committing, each transaction verifies that no other transaction has modified the data that it last read.

Saga

A Saga encapsulates a long running business transaction that is composed of several independent microservices. Each microservice comprises of one or more local transactions that are part of the same Saga.

29.3 Lock-Free Reservation

Lock-free reservation provides an in-database infrastructure for transactions operating on reservable columns to:

- Enable concurrent transactions to proceed without being blocked on updates made to reservable columns
- Issue automatic compensations for reservable updates of successful transactions in a canceled Saga

Here's how lock-free reservation enables applications to include concurrency and auto-compensating features into their transactions:

Declaring a Reservable Column

You can use the `reservable` keyword to declare a reservable column when you create or alter a table. Lock-free reservation is offered on columns with numeric data types. To identify a potential reservable column, look for hot resources with numeric aggregate data that could benefit from improved concurrency.

Reserving a Transaction Update

- When a transaction issues an update operation on a reservable column, the reservable update is placed as a lock-free reservation in a reservation journal. All updates issued on a reservable column are treated as lock-free reservation.
- The transaction update does not lock the row (that has the reservable update) but indicates its intention to modify (add or subtract) the reservable column in the row by a delta amount. The modification amount is reserved and promised so that the transaction may proceed without waiting on other uncommitted, concurrent transactions that have made earlier reservations on the same row's reservable column. The reservation enables other concurrent transactions to issue reservable updates to the same row.
- The operation of modification (increase or decrease by the delta amount with respect to the current value of the reservable column) is recorded in a reservation journal. Instead of reading and writing the actual value of the reservable column, transactions issue operations to increment or decrement the reservable column value. Update requests made to the reservable column are recorded in a reservation journal.

Verifying and Deferring an Update

The transaction, based on the constraints placed on the reservable column, decides whether the quantity is sufficient to make the update. The transaction checks for the following:

- Verifies that the update can succeed. The update to the reservation journal is allowed to proceed if there is a sufficient balance. If the balance is not sufficient to fulfill the update (consume) request, the update request to the reservable column is failed, without relying on any active (not yet committed) replenishment to succeed.
- Checks for any constraints or bounds (`CHECK` constraints) on reservable columns to enforce business rules and to ensure application correctness. `CHECK` constraints can include checks for reservable and non-reservable columns.
- Defers the actual update to reservable column until the commit time to improve concurrency

Auto-compensating Successful Updates in a Failed Transaction

Lock-free reservation enables the tracking of the state transitions of a reservable column through its transaction lifecycle. If a transaction is canceled (after partial completions) due to reasons such as insufficient balance or cancellation of a Saga, lock-free reservation issues automatic compensation or rollback of the reservable updates.

Ensuring Durability of a Reservable Update

The lock-free reservations are transformed to the actual updates at the commit of the transaction. Rollback of the transaction voids all lock-free reservations that the transaction holds in the reservation journals. Rollback to a savepoint removes the lock-free reservations made by the transaction after the affected savepoint.

For Sagas in microservices, lock-free reservations enable automatic compensation of the reservable updates made for successful local transactions in an canceled Saga. Lock-free

reservation enables tracking the reservable updates within the database during the execution of transactions and retains the journals beyond the commit of the transaction until the Saga finalization.

> **Note:**
>
> The Lock-Free Reservation feature does not lock rows until commit time, hence Priority Transactions (another 23ai feature) is a no-op for transactions that only do lock-free reservations.

> **See Also:**
>
> - Priority Transactions for more information about the Priority Transactions feature

29.3.1 Comparing Optimistic Locking and Lock-Free Reservation

Lock-free reservation guarantees the outstanding reservations by operating within bounds as follows:

- Journals on-going requests.
- Introduces new reservation journal columns to track projected values based on the pending requests.
- Consults the journal to allow or reject new requests based on outstanding reservations.
- Rejects new requests that cannot be satisfied due to outstanding reservations or if the request quantity is greater than the availability.

In comparison, optimistic locking does not track or guarantee reservations.

Here are other drawbacks of the optimistic locking approach:

- Requests may not be satisfied at the commit time because of insufficient quantity.
- Possibility of a transaction getting canceled at the end. If a long-running transaction is canceled, work is wasted and the changes must be rolled back.

29.3.2 Creating a Reservable Column at Table Creation

When you create a new table, you can declare a reservable column using the `reservable` keyword.

1. Modify the `CREATE TABLE` command to enable lock-free reservation as follows:

```
Create_table_command::={relational_table | object_table | XMLType_table }
Relational_table::= CREATE TABLE [ schema. ] table …;
relational_properties::= [ column_definition ]
column_definition::= column_name datatype reservable [CONSTRAINT constraint_name
check_constraint]

CREATE TABLE Account( ID NUMBER PRIMARY KEY,
 Name VARCHAR2(10),
 Balance NUMBER reservable CONSTRAINT minimum_balance CHECK (Balance >= 50))
```

The creation of the "Account" table with the reservable column "Balance" creates an associated reservation journal table. The reservation journal table is created under the same user schema and in the same tablespace as the user table. The reservation journal table also has deferred segment creation enabled.

> **Note:**
>
> The `CREATE TABLE` statement syntax supports the `RESERVABLE` keyword but not the `NOT RESERVABLE` keyword. The `NOT RESERVABLE` keyword is supported in the `ALTER TABLE` command.

> **See Also:**
>
> Reservation Journal Table Columns for more information about the columns used in the reservation journal table.

2. Use the `ALL_CONSTRAINTS` view to get the constraint details.

```
SELECT table_name, constraint_name, search_condition
 FROM ALL_CONSTRAINTS
 WHERE table_name='ACCOUNT';
```

29.3.2.1 Reservation Journal Table Columns

A reservation journal table contains the following column information.

The `DESCRIBE` statement gives you the actual names of the columns in the reservation journal table associated with the earlier mentioned Account table:

```
SQL>desc SYS_RESERVJRNL_<object_number_of_base_table>;
```

Table 29-1 Reservation Table Columns

Name	Null?	Type	Description
ORA_SAGA_ID$		RAW(16)	Saga ID of the transaction (0 for non-saga transactions)
ORA_TXN_ID$		RAW(8)	Transaction ID of the transaction (containing usn, slot, seq)
ORA_STATUS$		VARCHAR2(11)	Status of the Txn ID, with values: {ACTIVE, COMMITTED, COMPENSATED}
ORA_STMT_TYPE$		VARCHAR2(6)	DML statement type {UPDATE}
ID	NOT NULL	NUMBER	Primary Key column of user table

Table 29-1 (Cont.) Reservation Table Columns

Name	Null?	Type	Description
BALANCE_OP		VARCHAR2(1)	Reservable column operation with operations having '+' or '-' for replenishment or consumption
BALANCE_RESERVED		NUMBER	Reservable column reserved and is the amount reserved from the reservable column

29.3.3 Adding or Modifying Reservable Columns

You can modify the `ALTER TABLE` statement to add a reservable column or change a reservable column into a non-reservable column.

1. To add a reservable column, modify the `ALTER TABLE` command as follows:

```
ALTER TABLE [ schema.]table
 [add [column_definition]]…;
 column_definition::= column_name datatype reservable [default <value>]
 [CONSTRAINT constraint_name check_constraint]

ALTER TABLE Account
 ADD (Balance NUMBER reservable default 50 CONSTRAINT minimum_balance CHECK (Balance
>= 50));
```

2. To change an existing column to a reservable column, modify the `ALTER TABLE` command as follows:

```
ALTER TABLE [ schema.]table
 [modify [column_definition]]…;
 column_definition::= column_name reservable [default <value>]
 [CONSTRAINT constraint_name check_constraint]
```

   To change an existing QOH column to a reservable column and to optionally add a new constraint:

```
ALTER TABLE PRODUCTS
MODIFY (QOH reservable default 0 CONSTRAINT maxAmount CHECK (QOH <= 100));
```

3. To change a reservable column to a non-reservable column, modify the `ALTER TABLE` command as follows:

```
ALTER TABLE [ schema.]table
 [modify [column_definition]]…;
 column_definition::= column_name not reservable]
```

   To change an existing reservable column QOH to a non-reservable column:

```
ALTER TABLE PRODUCTS modify (QOH not reservable);
```

> **Note:**
>
> A reservable column can be converted to a non-reservable column using the earlier mentioned `ALTER TABLE` command. Although the lock-free reservations for the column are disabled, once a reservable column is changed to a non-reservable column, applications may choose to enforce the constraints. Hence, the constraints are not automatically dropped when a column is converted to a non-reservable column. You can choose to drop the constraints using the `ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>` statement.

29.3.4 About CHECK Constraints in Reservable Columns

A `CHECK` constraint lets you specify a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either `TRUE` or unknown (due to a `NULL`). When Oracle evaluates a check constraint condition for a particular row, any column names in the condition refer to the column values in that row.

> **Note:**
>
> Oracle does not verify that conditions of check constraints are not mutually exclusive. Therefore, if you create multiple check constraints for a column, design them carefully so their purposes do not conflict. Do not assume any particular order of evaluation of the conditions.

Table-level CHECK Constraints for Reservable Columns

You can have reservable columns in table-level `CHECK` constraints. If a table-level `CHECK` constraint that involves reservable and non-reservable columns fails validation at commit time, then the transaction is canceled. The cancel happens because the non-reservable column values cannot be guaranteed using the reservation mechanism.

For example:

```
CREATE TABLE Account(
 ID NUMBER PRIMARY KEY,
 Name VARCHAR2(10),
 Balance NUMBER reservable,
 Earmark NUMBER,
 Limit NUMBER,
 CONSTRAINT minimum_balance CHECK (Balance + Limit - Earmark >= 0))
```

You can also define reservable columns without constraints. For such reservable columns all lock-free reservations are successful.

No storage clause specification is supported for reservable columns.

29.3.5 Example: Conventional Locking and Lock-Free Reservation

The examples below provide a purchase transaction in conventional and lock-free reservation modes.

Conventional Locking (with Long-held Locks)

The following example uses traditional locking to allow a purchase of a $25 item while maintaining a $50 minimum balance:

- A `SELECT FOR UPDATE` is first issued to read and lock the balance.
- If the balance is at least 75, the item purchase is allowed.
- The `UPDATE` then debits the balance.
- The transaction then commits.
- An insufficient balance causes a canceling of the transaction.

```
CREATE TABLE Account (
 ID NUMBER PRIMARY KEY,
 Name VARCHAR2(10),
 Balance NUMBER CONSTRAINT minimum_balance CHECK (Balance >= 50));

DECLARE current NUMBER;

BEGIN
 -- Read and Lock account balance
 SELECT Balance INTO current
 FROM Account
 WHERE ID = 12345 FOR UPDATE;

 IF current >= 75 THEN
  -- Sufficient funds: Perform item purchase
  PurchaseItem();
  -- Debit account balance and commit
  UPDATE Account
  SET Balance = Balance - 25
  WHERE ID = 12345;
  COMMIT;
 ELSE
  ROLLBACK; -- Insufficient funds, so cancel
 END IF;
END;
```

Lock-Free Reservation (with Short-held Locks)

The following example uses lock-free reservation to allow a purchase of a $25 item while maintaining a $50 minimum balance. The reservable column constraint allows a reservation to be placed on a column value without locking the row.

- The balance update reserves $25 without locking the account.
- If the reservation succeeds, the item purchase is allowed to proceed.
- The final commit locks the account row and applies the balance debit of $25 as recorded in the reservation.
- If the reservation fails due to insufficient funds, the update statement fails with the `CHECK` constraint violation.

```
CREATE Table Account(
 ID NUMBER PRIMARY KEY,
 Name VARCHAR2(10),
 Balance NUMBER RESERVABLE CONSTRAINT minimum_balance CHECK (Balance >= 50));

BEGIN
 -- Reserve 25 from account balance
 UPDATE Account SET Balance = Balance - 25
 WHERE ID = 12345;
 -- If reservation succeeds perform item purchase
 PurchaseItem();
 -- The commit finalizes the balance update
 COMMIT; -- This gets the account row lock
 EXCEPTION WHEN Check_Constraint_Violated
 -- This indicates that the reservation failed
 THEN ROLLBACK;
END;
```

29.3.6 Querying Reservable Column Views

You can run queries on the dictionary views of reservable columns to get information about reservable columns.

You can run queries on the `DBA_TAB_COLUMNS`, `USER_TAB_COLUMNS` and `ALL_TAB_COLUMNS` views to check if a column is declared as a reservable column.

```
SELECT table_name, column_name , reservable_column
 FROM user_tab_columns
 WHERE table_name = <table name>;
```

You can run queries on the `DBA_TAB_COLS`, `USER_TAB_COLS`, and `ALL_TAB_COLS` views to check if a column is declared as a reservable column.

```
SELECT table_name, column_name , reservable_column
 FROM user_tab_cols
 WHERE table_name = <table name>;
```

Example Query:

```
SQL> SELECT table_name, column_name , reservable_column
 FROM user_tab_cols
 WHERE table_name = 'ACCOUNT';
```

Result:

```
TABLE_NAME     COLUMN_NAME    RES
-----------------------------------
ACCOUNT         NAME          NO
ACCOUNT         BALANCE       YES
ACCOUNT         ID            NO

3 rows selected
```

You can run queries on the `DBA_TABLES`, `USER_TABLES`, and `ALL_TABLES` views to check if the user table has one or more reservable columns.

```
SELECT table_name, has_reservable_column
 FROM user_tables
 WHERE table_name = <table name>;
```

Example Query:

```
SQL> SELECT table_name, has_reservable_column
 FROM user_tables
 WHERE table_name = 'ACCOUNT';
```

Result:

```
TABLE_NAME     HAS
--------------------
ACCOUNT        YES

1 row selected
```

29.4 Benefits of Using Lock-Free Reservation

Improved User Experience and Concurrency

Lock-free reservation holds locks on hot data for short intervals of time, thereby improving user experience and concurrency. Transactions reserve an amount from a reservable column value without locking it. The locking happens only when the value is modified during the commit of the transaction.

Automatic Compensation

When using compensating functions, you need to track the dependencies to be able to transition the database into a consistent state. For Sagas, these dependencies need to be tracked for a long period (until the change is confirmed). You can use lock-free reservation in your Saga implementations. With lock-free reservation, if a Saga transaction is canceled, implicit compensating transactions are automatically issued to take care of the rollback of other transactions of the Saga that have already committed. You do not need to go through the complexity of writing compensating functions. Lock-free reservation enables auto-compensation through the journals to reverse any successful update on failed Sagas.

Efficient Resource Usage

When you use lock-free reservation, multiple transactions can run in parallel and use resources without blocking each other out. Efficiency in resources usage improves with less waiting and response times.

Broader Scope

Reservable updates are done on numeric aggregate data, which is integral to many applications that operate on a wide variety of data. Improved concurrency using lock-free reservation can benefit applications wherever there are a high rate of rows with reservable updates. Applications that have reservable updates in long-running transactions can benefit the most from improved concurrency in transactions. Some of these applications are those that deal in banking, inventory control, ticketing, and event reservation.

29.5 Guidelines and Restrictions for Lock-Free Reservation

This section provides the guidelines and restrictions for lock-free reservation.

29.5.1 Guidelines and Restrictions for Reservable Columns

When using lock-free reservation, follow these guidelines for reservable columns and user tables with reservable columns:

- The Schema definition of user tables declare the reservable columns with the `reservable` keyword. Reservable columns provide lock-free reservations.
- A reservable column can be specified for Oracle numeric data type (`NUMBER`, `INTEGER`, and `FLOAT`) columns only.
- A reservable column cannot be a `Primary Key` or an identity column (or virtual column) because the reservable column is an aggregate type.
- A user table can have at most ten reservable columns.
- User tables that have reservable columns must have a `Primary Key`.
- Indexes are not supported on reservable columns.
- Composite reservable columns are not allowed.
- Reservable columns are not allowed in Block Chain tables and Sharded tables.
- Reservable columns can be included only in `CHECK` constraints expression. The `CHECK` constraint can be at the column-level or table-level. User-defined operational constraints are used for reservable columns to ensure application correctness.
  - Optional `CHECK` constraints can be specified on the reservable columns.
  - Reservable columns cannot be involved in any other types of constraints. Non-`CHECK` constraints involving the reservable columns are not allowed. Reservable columns cannot be part of a foreign key constraint.
- External, Cluster, IOT and temporary tables cannot have reservable columns.
- Partitioning cannot be made on reservable columns.
- Two-phase online DDL optimization is not provided for user tables with reservable columns.
- Dropping a reservable column from a user table removes the corresponding lock-free reservation tracking columns from the reservation journal table. If the last reservable column is removed from a user table, the reservation journal table is dropped. Transactions with pending reservations must finalize before you can drop the reservable column or mark the column as `UNUSED`.

29.5.2 Guidelines and Restrictions for Update Statements

When using lock-free reservation, follow these guidelines for the `UPDATE` statements:

- Updates to reservable columns must be specified as one of the following:

```
UPDATE <table_name>
SET <reservable_column_name> = <reservable_column_name> + (<expression>)
WHERE <primary_key_column> = <expression>

UPDATE <table_name>
SET <reservable_column_name> = <reservable_column_name> - (<expression>)
WHERE <primary_key_column> = <expression>
```

> **Note:**
>
> Do not use `SET <reservable_column_name> = <value>`. Direct assignment of a value raises an error.

> **Note:**
>
> For composite primary keys, all the primary key columns must be specified in the `WHERE` clause of the reservable update.

- Multiple reservable columns of a table can be updated in a single update statement.
- Mixing reservable and non-reservable column updates in the same update statement is not allowed. Also, DML returning clause is not supported in reservable column update statements.
- Updates to reservable columns do not lock the row until the commit of the transaction. Instead, reservable columns provide for lock-free reservations. Lock-free reservations allow other concurrent transactions updating the reservable columns without being blocked.
- A reservable update issued on a locked row (locked by either the same or a different transaction that has updated a non-reservable column of the row) is allowed to obtain lock-free reservation.
- The updates that are transformed to pending reservations on the reservable columns ensure that the constraints on the reservable columns are not violated after considering the pending reservations.
- A transaction can read its own lock-free reservations by selecting from the reservation journal tables (associated with the user tables) on which the transaction has issued a reservable update. Reservations made by other transactions are not visible.

29.5.3 Guidelines for Inserts and Deletes

When using lock-free reservation, follow these guidelines for the `INSERT` and `DELETE` statements issued on tables with reservable columns:

- Transactions can insert an entire row with values for the reservable columns without any change in behavior.
- The inserted row is not visible to other transactions for any reservable updates until the inserting transaction commits.
- If a `DELETE` statement is issued when there are pending reservations for the rows of a user table that are to be deleted, the transactions with active lock-free reservations against those rows should complete before the delete can proceed. The `DELETE` statement is internally retried for an interval of 5 seconds and is allowed if there are no more pending reservations for the affected rows. After the timeout period, if the `DELETE` statement could not proceed, an error with resource busy message is raised, and the `DELETE` statement can be issued at a later time.

29.5.4 Guidelines for Concurrent DDL Statements

When using lock-free reservation, follow these guidelines for the concurrent DDL statements issued on tables with reservable columns:

- If a DDL statement is in progress on a table that has a reservable column, no reservable updates can operate on the user table until the DDL completes.
- If a DDL statement is issued when there are pending reservations for any rows of a user table, the DDL statement is blocked until the transactions with active reservations complete.

29.5.5 Restrictions for Reservation Journal Table

Observe these restrictions for reservation journal table:

User DML and DDL operations are not permitted on a reservation journal table. You cannot create or modify a reservation journal table using DML. You cannot use SQL to drop, truncate, rename, or change the reservation table's definition.