GROUP PROJECT

Team Name: Cluster Crafters (Mobile Phone Supply Chain Management)
PART 2: Design and Implementation of a Distributed Database System

Fragmentation

Fragmentation in database systems refers to the process of dividing a database into smaller segments or fragments to enhance its manageability, performance, and efficiency. Fragmentation becomes particularly relevant in distributed database systems, where data is stored across multiple sites or locations. The primary goal of fragmentation is to store data in a way that is most convenient and efficient for the system and its users.

Horizontal Fragmentation

Horizontal fragmentation involves dividing a database table into smaller subsets or fragments, each containing a subset of the rows from the original table. This type of fragmentation is based on the tuples of a table. The division criteria often involve the value of one or more attributes. For example, a customer database might be horizontally fragmented based on geographical locations, with each fragment containing customers from a specific region.

Benefits of Horizontal Fragmentation:

- Improved Query Performance: Queries targeting a specific subset of rows can be executed more efficiently.
- Localized Data: Data relevant to a particular region or set of users can be stored closer to them.
- Enhanced Security: Sensitive data can be isolated in specific fragments.

We have achieved Horizontal Fragmentation in our implementation for the Inventory table as shown in Fig 1. Based on the column InventoryID we are performing range partitioning.

```
CREATE TABLE Inventory (
    InventoryID SERIAL,
    MobilePhoneID INT,
    WarehouseID INT,
    Quantity INT,
    PurchaseDate DATE,
    FOREIGN KEY (MobilePhoneID) REFERENCES MobilePhone(MobilePhoneID),
    FOREIGN KEY (WarehouseID) REFERENCES Warehouse(WarehouseID),
    PRIMARY KEY (InventoryID)
)
""",
"""
ALTER TABLE Inventory PARTITION BY RANGE (InventoryID) (
PARTITION Inventory_P1 VALUES FROM (MINVALUE) TO (100),
PARTITION Inventory_P2 VALUES FROM (101) TO (200)
);
""".
```

Fig 1. Snapshot of creating Horizontal Fragmentation for the table Inventory ID

Snapshots for Horizontal fragmentation on the Inventory table:

```
HORIZONTAL FRAGMENTATION

Table Inventory has been Horizontal Fragmented with RANGE PARTITIONING
Column InventoryID is used for partitioning

Database: defaultdb
Table: inventory
Partition: inventory_p1
Range: (MINVALUE) TO (100)
Config: range_min_bytes = 134217728,
range_max_bytes = 536870912,
gc.ttlseconds = 4500,
num_replicas = 3,
constraints = '[]',
lease_preferences = '[]'

Database: defaultdb
Table: inventory
Partition: inventory_p2
Range: (101) TO (200)
Config: range_min_bytes = 134217728,
range_max_bytes = 536870912,
gc.ttlseconds = 4500,
num_replicas = 3,
constraints = '[]',
lease_preferences = '[]'
```

Vertical Fragmentation

Vertical fragmentation divides a database table into smaller fragments based on columns rather than rows. In this approach, each fragment consists of a subset of columns from the original table. This type of fragmentation is useful when different applications or users require access to different subsets of columns from a table.

Benefits of Vertical Fragmentation:

- Reduced I/O: Queries accessing only a subset of columns need to read less data.
- Security and Privacy: Sensitive columns can be isolated and secured separately.
- Network Efficiency: Less data is transferred over the network when only the required columns are accessed.

In our implementation, we are performing the vertical fragmentation on the table “Supplier” as shown in Fig 2. When our application displays a supplier for a product, we only need to access the 'Supplier_Name' table to get the supplier name. Since this table is smaller (containing only 'SupplierID' and 'SupplierName'), queries to retrieve supplier names will be faster compared to querying a larger table with more columns. This is particularly beneficial in distributed environments where minimizing data transfer is crucial.

```
# Insert data into Supplier_Name and Supplier_Contact during vertical fragmentation
cursor.execute("INSERT INTO Supplier_Name (SupplierID, SupplierName) SELECT SupplierID, SupplierName FROM Supplier")
cursor.execute("INSERT INTO Supplier_Contact (SupplierID, ContactInfo, Address) SELECT SupplierID, ContactInfo, Address FROM Supplier")
```

Fig 2. Inserting data into vertically fragmented Supplier_Name and Supplier_Contact

Snapshots for vertically fragmented data:

```
VERTICAL FRAGMENTATION
Supplier_Name Table (Vertical Fragmentation):
(1, 'Apple')
(2, 'Samsung')
(3, 'Google Pixel')
(4, 'Xiaomi')
(5, 'Huawei')
(6, 'OnePlus')
(7, 'Oppo')
(8, 'Vivo')
(9, 'Sony')
(10, 'Realme')
(11, 'Motorola')
(12, 'Asus')
(13, 'Tecno Mobile')
(14, 'LG')
(15, 'ZTE')
(16, 'Nokia')
(17, 'HTC')
(18, 'Panasonic')
(19, 'Honor')
(20, 'Alcatel')
```

```
Supplier_Contact Table (Vertical Fragmentation):
(1, '652-522-7230', '02833 Prairie Rose Drive')
(2, '403-664-2259', '3 Crowley Junction')
(3, '470-791-7654', '85 Judy Avenue')
(4, '375-705-5225', '19820 Pine View Alley')
(5, '127-455-0889', '82 Stoughton Avenue')
(6, '183-384-5876', '951 Twin Pines Alley')
(7, '430-346-8865', '625 Acker Avenue')
(8, '175-184-2287', '42 Fallview Court')
(9, '570-859-8105', '03774 Shoshone Pass')
(10, '768-637-7856', '9027 Sachtjen Court')
(11, '678-918-2262', '20136 Dayton Court')
(12, '773-137-1276', '3083 Raven Point')
(13, '724-862-7737', '22417 Thackeray Pass')
(14, '660-237-1142', '38 Fallview Street')
(15, '201-465-9076', '387 American Pass')
(16, '850-350-1743', '4 Rusk Center')
(17, '544-256-3677', '31 Hudson Hill')
(18, '712-471-6445', '4 Rowland Drive')
(19, '397-538-2353', '56668 Hudson Alley')
(20, '363-529-7108', '438 Buell Way')
```

Replication Strategies

Replication in database systems involves maintaining copies of data or fragments across different servers or locations. This strategy is used to enhance data availability, fault tolerance, and performance. There are several replication strategies:

- Full Replication: Every site holds a copy of the entire database. This maximizes availability but requires more storage and effort to maintain consistency.
- Partial Replication: Only certain fragments or portions of the database are replicated across different sites. This strategy balances storage costs and availability.
- No Replication: Each fragment exists in only one location. While this minimizes storage requirements, it may affect availability and performance.

In our application, we are using CockroachDB. By default, **CockroachDB adheres to partial replication**. It combines the benefits of automatic replication, range-based data division, geo-partitioning for data locality, and a strong consistency model to provide a robust, scalable, and efficient distributed database solution.

Snapshot to display the configuration of the number of replicas used for the database. Further, our database is configured for the Default zone.

```
DATABASE REPLICATION
Table: RANGE default
Configuration: ALTER RANGE default CONFIGURE ZONE USING
        range_min_bytes = 134217728,
        range_max_bytes = 536870912,
        gc.ttlseconds = 4500,
        num_replicas = 3,
        constraints = '[]',
        lease_preferences = '[]'
```

Conclusion

In this chapter, we have comprehensively explored the concepts of Fragmentation and Replication Strategies in distributed database systems, crucial for optimizing the structure and accessibility of data. This knowledge forms a solid foundation for the next phase in our project: enhancing the Mobile Phone Supply Chain Management system using CockroachDB. As we transition to our next focus, Query Processing, and Optimization Techniques, we aim to build upon this groundwork to further improve query efficiency and database performance, ensuring a robust, efficient, and effective data management solution.