**1. Build a Highly Scalable Database with Automated Backups**

**a. Choose Amazon Aurora**

* **Highly Scalable**: Aurora automatically scales up to accommodate high workloads and can handle thousands of concurrent connections.
* **High Availability**: Deploy the Aurora database in a Multi-AZ configuration, which ensures high availability and automatic failover.
* **Automated Backups**: Configure automated backups with a retention period of **2 days**. You can set this during the creation of the Aurora database or modify it later.

**2. Build a Database Architecture for Multi-Region Reads**

**a. Aurora Global Database**

* **Global Database**: Use Aurora Global Database to create a read replica in different regions. This allows your application to read from the closest replica, reducing latency and improving performance.
* **Failover Support**: In case of a regional failure, you can promote a read replica to be the new primary database, ensuring minimal downtime.

**b. DynamoDB Global Tables (if applicable)**

* If using DynamoDB, implement Global Tables for automatic multi-region replication. This allows your application to access data from the nearest region, enhancing performance.

**3. Establish Near Real-Time Data Processing**

**a. Amazon Aurora with Triggers and Stored Procedures**

* **Real-Time Processing**: Use database triggers and stored procedures in Aurora to process data in near real-time. Triggers can automatically execute actions in response to data modifications.
* **Materialized Views**: Create materialized views for frequently queried data to speed up read operations. These views can be refreshed periodically or based on specific events.

**b. DynamoDB Streams**

* If using DynamoDB, enable DynamoDB Streams to capture changes to items in your table. You can set up Lambda functions (or use other database services) to process these streams in near real-time.

**4. Resolve Latency Issues**

**a. Implement Read Replicas**

* Set up read replicas in regions closer to the branches. Modify your application logic to route read queries to these replicas, reducing latency associated with fetching data from the primary database.

**b. Database Caching Strategies**

* **Caching Frequently Accessed Data**: While primarily using database services, consider implementing query optimization and indexing strategies to reduce latency. For example:
  + **Indexing**: Create indexes on frequently queried columns to improve query performance.
  + **Partitioning**: If applicable, partition large tables to enhance performance and manageability.

**c. Query Optimization**

* Optimize SQL queries for performance. Analyze and refine queries to reduce execution time, which can help in alleviating latency when accessing the database.

**Implementation Steps:**

1. **Set Up Amazon Aurora**:
   * Create an Aurora database cluster in a Multi-AZ configuration with automated backups set to 2 days.
2. **Implement Global Database**:
   * Configure Aurora Global Database to set up read replicas in regions closest to the branches.
3. **Enable Triggers/Stored Procedures**:
   * Use triggers and stored procedures to handle near real-time processing.
4. **Optimize Queries**:
   * Analyze query performance and implement indexing and partitioning strategies as necessary.