**APEX CODES**

**APEX TRİGGER UpdateIdAsVolume**

/\*Bu tetikleyici, özel Books\_\_c nesnesi üzerinde gerçekleşen before insert ve before update olaylarını ele almak üzere tasarlanmıştır.

Bir yeni kayıt veya güncellenen kaydın Volume\_\_c alanı 1'den büyükse, bu tetikleyici devreye girer.

Her ekstra hacim için, orijinal ile aynı detaylara sahip yeni bir kayıt oluşturur, ancak hacim 1 olarak ayarlanır.

Tetikleyici aynı zamanda orijinal kaydın hacmini 1 olarak günceller ve çoğaltılmış kayıtları ekler.\*/

trigger UpdateIdAsVolume on Books\_\_c (before insert,before update) {

List<Books\_\_c> booksToUpdate = new List<Books\_\_c>();

for (Books\_\_c book : Trigger.new) {

if (book.Volume\_\_c > 1) {

// For new records with more than one volume, create new records with the same name

for (Integer i = 1; i < book.Volume\_\_c; i++) {

Books\_\_c updatedBook = new Books\_\_c(

Name\_\_c = book.Name\_\_c,

Author\_\_c = book.Author\_\_c, // Set the author from the original record

Price\_\_c = book.Price\_\_c,

Volume\_\_c = 1, // Set the volume to 1 for the duplicated records

Pages\_\_c=book.Pages\_\_c,

Genre\_\_c=book.Genre\_\_c,

Published\_By\_\_c=book.Published\_By\_\_c,

Language\_\_c=book.Language\_\_c

);

booksToUpdate.add(updatedBook);

System.debug(booksToUpdate);

}

}

}

// Update the volume of the original records

for (Books\_\_c book : Trigger.new) {

book.Volume\_\_c = 1;

}

// Insert the duplicated records

insert booksToUpdate;

}

**APEX TEST CLASS TestUpdateIdAsVolume**

@isTest

public class TestUpdateIdAsVolume {

@isTest

static void testTrigger() {

List<Author\_\_c> authors = new List<Author\_\_c>();

for (Integer i = 0; i < 5; i++) {

Author\_\_c author = new Author\_\_c(

Name\_\_c = 'Author ' + i,

Surname\_\_c = 'Surname ' + i

);

authors.add(author);

}

insert authors;

// Create a test record

Books\_\_c testBook = new Books\_\_c(

Name\_\_c = 'Test Book',

Author\_\_c = authors[0].id,

Price\_\_c = 29.99,

Volume\_\_c = 3,

Pages\_\_c = 200,

Genre\_\_c = 'Roman',

Language\_\_c = 'English'

);

// Insert the test record

insert testBook;

// Check if additional records were created

List<Books\_\_c> duplicatedBooks = [SELECT Name\_\_c, Author\_\_c,

Price\_\_c, Volume\_\_c, Pages\_\_c, Genre\_\_c, Published\_By\_\_c, Language\_\_c

FROM Books\_\_c WHERE Name\_\_c = 'Test Book'

AND Volume\_\_c = 1];

System.assertEquals(3, duplicatedBooks.size(), 'Two additional records should have been created');

// Update the test record's volume

testBook.Volume\_\_c = 2;

update testBook;

// Check if additional records were created after update

duplicatedBooks = [SELECT Name\_\_c, Author\_\_c, Price\_\_c,

Volume\_\_c, Pages\_\_c, Genre\_\_c, Published\_By\_\_c, Language\_\_c

FROM Books\_\_c WHERE Name\_\_c = 'Test Book' AND

Volume\_\_c = 1];

System.assertEquals(4, duplicatedBooks.size(), 'One additional record should have been created after update');

}

}

**APEX TRİGGER UpdateMemberLastBookReadDate**

/\*Bu tetikleyici, Book\_Tracking\_\_c özel nesnesi üzerinde gerçekleşen before insert ve before update olaylarını ele almak üzere tasarlanmıştır.

Tetikleyici, üyelerin en son kitap satın alma tarihini güncellemekten sorumludur.

Yeni bir Book\_Tracking\_\_c kaydı eklenirken veya güncellenirken, ilgili üyenin en son satın alma tarihi kontrol edilir.

Her üye için, ilgili tüm Book\_Tracking\_\_c kayıtları arasında en son Issue\_Date\_\_c bulunur.

Bu tarih, üyenin Last\_Book\_Read\_Date\_\_c alanına yazılır.\*/

trigger UpdateMemberLastBookReadDate on Book\_Tracking\_\_c (before insert, before update) {

// Collect the Member\_\_c record Ids for all inserted Book\_Tracking\_\_c records

Set<Id> memberIds = new Set<Id>();

for (Book\_Tracking\_\_c bookTracking : Trigger.new) {

if (bookTracking.Borrowed\_By\_\_c != null) {

memberIds.add(bookTracking.Borrowed\_By\_\_c);

}

}

// Query the latest Issue\_Date\_\_c for each Member\_\_c record

Map<Id, Date> memberLastBookReadDates = new Map<Id, Date>();

for (AggregateResult result : [

SELECT Borrowed\_By\_\_c, MAX(Issue\_Date\_\_c) maxIssueDate

FROM Book\_Tracking\_\_c

WHERE Borrowed\_By\_\_c IN :memberIds

GROUP BY Borrowed\_By\_\_c

]) {

Id memberId = (Id)result.get('Borrowed\_By\_\_c');

Date maxIssueDate = (Date)result.get('maxIssueDate');

memberLastBookPurchaseDates.put(memberId, maxIssueDate);

}

// Update the Member\_\_c records with the latest Issue\_Date\_\_c

List<Member\_\_c> membersToUpdate = new List<Member\_\_c>();

for (Id memberId : memberIds) {

if (memberLastBookPurchaseDates.containsKey(memberId)) {

Member\_\_c member = new Member\_\_c(Id = memberId);

member.Last\_Book\_Read\_Date\_\_c =

memberLastBookPurchaseDates.get(memberId);

membersToUpdate.add(member);

}

}

// Perform the updates

update membersToUpdate;

}

**APEX TEST CLASS TestUpdateMemberLastBookReadDate**

@isTest

public class TestUpdateMemberLastBookReadDate {

@isTest

static void testUpdateLastBookPurchaseDate() {

// Create a test Member\_\_c record

Member\_\_c testMember = new Member\_\_c(Name\_\_c = 'mehmet',

Surname\_\_c = 'gel', Email\_\_c = 'mhm@gl.tr');

insert testMember;

// Create a test Author\_\_c record

Author\_\_c testAuthor = new Author\_\_c();

insert testAuthor;

// Create a test Book\_\_c record

List<Books\_\_c> books = new List<Books\_\_c>();

books.add(new Books\_\_c(Name\_\_c = 'Book1', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

books.add(new Books\_\_c(Name\_\_c = 'Book2', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

books.add(new Books\_\_c(Name\_\_c = 'Book3', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

insert books;

// Create test Book\_Tracking\_\_c records associated with the test

Member\_\_c record

Book\_Tracking\_\_c book1 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id, Book\_Name\_\_c= books[0].id,

Issue\_Date\_\_c =

Date.newInstance(2023,12,1), Status\_\_c='Issued');

insert book1;

// Introduce a delay (e.g., 2 seconds) before inserting the second

Book\_Tracking\_\_c record

Test.startTest();

Long delayMillis = 2000; // 2 seconds

Long startTime = System.now().getTime();

while (System.now().getTime() < startTime + delayMillis) {

// Wait for the specified delay

}

Book\_Tracking\_\_c book2 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id, Book\_Name\_\_c= books[1].id,

Issue\_Date\_\_c =

Date.newInstance(2023,12,15),Status\_\_c='Issued');

insert book2;

// Introduce a delay (e.g., 2 seconds) before inserting the second

Book\_Tracking\_\_c record

Long delayMilliss = 2000; // 2 seconds

Long startTimee = System.now().getTime();

while (System.now().getTime() < startTime + delayMillis) {

// Wait for the specified delay

}

Test.stopTest();

Book\_Tracking\_\_c book3 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id,

Book\_Name\_\_c= books[2].id,

Issue\_Date\_\_c = Date.newInstance(2023,12,30),

Status\_\_c='Issued');

insert book3;

// Trigger should have updated the Last\_Book\_Read\_Date\_\_c on the

Member\_\_c record

testMember = [SELECT Id, Last\_Book\_Read\_Date\_\_c FROM

Member\_\_c WHERE Id = :testMember.Id];

Date expectedLastBookReadDate = Date.newInstance(2023,12,15);

System.assertEquals(expectedLastBookReadDate,

testMember.Last\_Book\_Read\_Date\_\_c, 'Last\_Book\_Read\_Date\_\_c should be updated');

}

}

**APEX CLASS LibraryOperations**

/\*

Bu Class, kütüphane işlemleri için çeşitli metotlar içerir.

Bu metotlar ziyaretçi üyelik kontrolü yapmak ve yanlış yazılmış yazar adını düzeltmek gibi işlemleri gerçekleştirir.

\*/

public class LibraryOperations {

// Verilen ad ve soyad ile bir ziyaretçinin üyeliğini kontrol eder

public static void **checkVisitorMembership**(String firstName, String lastName) {

List<Member\_\_c> members = [SELECT Id, Name FROM Member\_\_c

WHERE Name\_\_c = :firstName AND Surname\_\_c = :lastName];

if (!members.isEmpty()) {

List<Map<String, String>> booksRead = new List<Map<String,String>>();

List<Book\_Tracking\_\_c> bookTrackings = [SELECT Id, Book\_Name\_\_r.Name\_\_c, Book\_Name\_\_r.Author\_\_r.Name\_\_c, Book\_Name\_\_r.Author\_\_r.Surname\_\_c, Borrowed\_By\_\_r.Name\_\_c,Borrowed\_By\_\_r.Surname\_\_c FROM Book\_Tracking\_\_c

WHERE Borrowed\_By\_\_c = :members];

for (Book\_Tracking\_\_c bookTracking : bookTrackings) {

Map<String, String> bookInfo = new Map<String, String>();

bookInfo.put('Kitap Adı', bookTracking.Book\_Name\_\_r.Name\_\_c);

bookInfo.put('Yazar Adı ve Soyadı', bookTracking.Book\_Name\_\_r.Author\_\_r.Name\_\_c +' '+

bookTracking.Book\_Name\_\_r.Author\_\_r.Surname\_\_c);

booksRead.add(bookInfo);

}

System.debug(firstName + ' ' + lastName + ' kütüphane üyesidir.');

System.debug('Üye tarafından okunan kitaplar:'+booksRead);

} else {

System.debug(firstName + ' ' + lastName + ' kütüphane üyesi değildir.');

}

}

// Yanlış yazılmış yazar adını düzeltir

public static void **correctAuthorName**(String incorrectName, String correctName) {

List<Author\_\_c> authorsToUpdate = [SELECT Id, Name\_\_c FROM Author\_\_c

WHERE Name\_\_c = :incorrectName];

for (Author\_\_c author : authorsToUpdate) {

author.Name\_\_c = correctName;

}

update authorsToUpdate;

}

}

**APEX TEST CLASS TestLibraryOperations**

@isTest

public class TestLibraryOperations{

@isTest

static void testCheckVisitorMembership() {

// Create test data: Member\_\_c and Author\_\_c records

Member\_\_c testMember = new Member\_\_c(Name\_\_c = 'John',Surname\_\_c = 'Doe', Email\_\_c = 'john.doe@example.com');

insert testMember;

system.debug(testMember);

Author\_\_c testAuthor = createTestAuthor();

insert testAuthor;

system.debug(testAuthor);

// Create a test Book\_\_c record

Books\_\_c testBook = new Books\_\_c(

Name\_\_c = 'Test Book',

Author\_\_c = testAuthor.Id,

Price\_\_c = 50.0,

Volume\_\_c = 15

);

insert testBook;

system.debug(testBook);

// Create a test Book\_Tracking\_\_c record associated with the test Book\_\_c record

Book\_Tracking\_\_c testBookTracking = new Book\_Tracking\_\_c(

Book\_Name\_\_c = testBook.Id,

Borrowed\_By\_\_c = testMember.Id,

Status\_\_c='Issued',

Issue\_Date\_\_c= date.today()

);

insert testBookTracking;

system.debug(testBookTracking);

// Test the checkVisitorMembership method

Test.startTest();

System.debug('Test Case 1: Existing Member');

LibraryOperations.checkVisitorMembership('John', 'Doe');

System.debug('Test Case 2: Non-existing Member');

LibraryOperations.checkVisitorMembership('Jane', 'Smith');

Test.stopTest();

}

@isTest

static void testCorrectAuthorName() {

// Create test data: Author\_\_c record

Author\_\_c testAuthor = createTestAuthor();

insert testAuthor;

// Test the correctAuthorName method

Test.startTest();

System.debug('Test Case 1: Correct the author name');

LibraryOperations.correctAuthorName(testAuthor.Name\_\_c, 'CorrectedAuthor');

Test.stopTest();

// Verify that the author's name is updated correctly

Author\_\_c updatedAuthor = [SELECT Id, Name\_\_c FROM Author\_\_c WHERE Id = :testAuthor.Id];

System.assertEquals('Corrected Author', updatedAuthor.Name\_\_c);

}

// Helper method to create a test Author\_\_c record

static Author\_\_c createTestAuthor() {

return new Author\_\_c(Name\_\_c = 'Test Author');

}

}

**APEX CLASS CapitalizeMemberNamesBatch AND CapitalizeMemberNamesScheduler**

/\*

\* **CapitalizeMemberNamesBatch Class**

Bu Class, üye isimlerini büyük harfle düzenlemek için bir toplu işlem gerçekleştirir.

Veritabanı kayıtlarını toplu olarak alır, isim ve soyisim alanlarını büyük harfe çevirir ve günceller.

\*/

public class CapitalizeMemberNamesBatch implements

Database.Batchable<sObject> {

// Adım 1: İşlem başladığında yürütülen sorgu belirlenir

public Database.QueryLocator start(Database.BatchableContext context) {

String query = 'SELECT Id, Name\_\_c, Surname\_\_c FROM Member\_\_c';

return Database.getQueryLocator(query);

}

// Adım 2: Belirtilen kapsam içindeki üye isimlerini büyük harfe çevirir ve günceller

public void execute(Database.BatchableContext context,

List<Member\_\_c> scope) {

for (Member\_\_c member : scope) {

member.Name\_\_c = member.Name\_\_c.toUpperCase();

member.Surname\_\_c = member.Surname\_\_c.toUpperCase();

}

update scope; // Güncelleme işlemi

}

// Adım 3: İşlem tamamlandığında çalışır, isteğe bağlı olarak son işlemler yapılabilir

public void finish(Database.BatchableContext context) {

// İsteğe bağlı: Gerekirse herhangi bir son işlem mantığı ekleyebilirsiniz.

}

}

/\*

\* **CapitalizeMemberNamesScheduler Class**

Bu Class, CapitalizeMemberNamesBatch sınıfını bir zamanlayıcı ile çalıştırmak için tasarlanmıştır.

Zamanlayıcı tetiklemesi gerçekleştiğinde, toplu işlemi başlatır ve üye isimlerini büyük harfe dönüştürür.

\*/

public class CapitalizeMemberNamesScheduler implements Schedulable {

// Adım 1: Zamanlayıcı tetiklemesi gerçekleştiğinde yürütülen metot

public void execute(SchedulableContext context) {

// CapitalizeMemberNamesBatch sınıfından bir örnek oluştur

CapitalizeMemberNamesBatch batchJob = new CapitalizeMemberNamesBatch();

// Toplu işlemi yürüt

Database.executeBatch(batchJob);

}

}

**APEX TEST CLASS TestCapitalizeMemberNamesBatch**

@isTest

public class TestCapitalizeMemberNamesBatch {

@isTest

static void test\_BatchJob() {

// Create test data

List<Member\_\_c> testMembers = new List<Member\_\_c>();

for (Integer i = 0; i < 200; i++) {

testMembers.add(new Member\_\_c(Name\_\_c = 'Name' + i, Surname\_\_c = 'Surname' + i, Email\_\_c='test@test.tr' +i));

}

insert testMembers;

Test.startTest();

// Enqueue the batch job

CapitalizeMemberNamesBatch batchJob = new CapitalizeMemberNamesBatch();

Database.executeBatch(batchJob);

Test.stopTest();

// Retrieve the updated records and assert the changes

List<Member\_\_c> updatedMembers = [SELECT Id, Name\_\_c, Surname\_\_c FROM Member\_\_c WHERE Id IN :testMembers];

for (Member\_\_c member : updatedMembers) {

// Assert that names have been capitalized

System.assertEquals(member.Name\_\_c, member.Name\_\_c.toUpperCase());

System.assertEquals(member.Surname\_\_c, member.Surname\_\_c.toUpperCase());

}

}

}

**APEX TEST CLASS TestCapitalizeMemberNamesScheduler**

@isTest

public class TestCapitalizeMemberNamesScheduler {

@isTest

static void test\_Scheduler() {

// Create test data

List<Member\_\_c> testMembers = new List<Member\_\_c>();

for (Integer i = 0; i < 200; i++) {

testMembers.add(new Member\_\_c(Name\_\_c = 'Name' + i, Surname\_\_c = 'Surname' + i, Email\_\_c='test@test.tr'+i));

}

insert testMembers;

system.debug(testMembers);

String cronExp = '0 0 0 \* \* ?'; // Set the cron expression to a date in thefuture

Test.startTest();

CapitalizeMemberNamesScheduler capName= new CapitalizeMemberNamesScheduler();

system.schedule('CapitalizeMemberNamesScheduler', cronExp, capName);

Test.stopTest();

system.debug(capName);

}

}