

VERİTABANI TASARIMI

Öğretim Görevlisi A. Berika VAROL MALKOÇOĞLU

İçindekiler

- TCL Nedir?
- TCL örnekleri
- A.C.I.D Nedir?
- Saklı Yordamlar Nedir?
- Saklı Yordam örnekleri

TCL

- **TCL (İşlem Kontrol Dili – Transaction Control Language):** DML ile gerçekleştirdiğimiz işlemleri yönettiğimiz ve kontrol ettiğimiz dildir.
 - **COMMIT:** Yapılanları kayıt eder.
 - **SAVEPOINT:** Daha sonra geri dönülecek bir dönüş noktası belirler.
 - **ROLLBACK:** Mevcut işlemi geri alır ve değişikliklerini iptal eder.

TRANSACTION Nedir?

- Daha küçük parçalara ayrılamayan en küçük işlem yığını
- Hepsi tek bir SQL ifadesiymiş gibi yürütülen SQL komutları kümesidir.
- Transaction'ın tamamlanabilmesi ve verilerdeki değişikliklerin kaydedilebilmesi için transaction'da yer alan tüm SQL komutlarının gerçekleştirilmesi gerekir.

TRANSACTION Nedir?

- Transaction bloğu ya hep ya hiç mantığı ile çalışır.
- Ya tüm işlemler düzgün olarak gerçekleşir ve geçerli kabul edilir.
- Ya da bir kısım işlemler yolunda gitse bile, blok sona ermeden bir işlem yolunda gitmese hiçbir işlem olmamış gibi kabul edilir.
- Eğer bir transaction herhangi bir nedenle tamamlanmazsa:
 - Transaction'dan ötürü veri kümesinde meydana gelen değişiklikler iptal edilir.
 - Veritabanı transaction başlamadan önceki orijinal durumuna geri döndürülür.

TRANSACTION Oluşturma

- Transaction başlatmak için:
 - START TRANSACTION;
 - BEGIN
 - BEGIN WORK
- Transaction'ı sonlandırmak için:
 - Commit(başarılı)
 - Rollback(başarısız)

AUTOCOMMIT

- AUTOCOMMIT özelliği default olarak 1 yani aktif olarak gelir.
- AUTOCOMMIT aktif olduğu için tablolar üzerindeki her hareketimiz otomatik olarak anında COMMIT edilmiş olur.
- Böylece yaptığımız işlemi geri alma şansımız olmaz.
- Yanlış yaptığımız bir UPDATE işlemi kalıcı olarak tablolara yazılmış olur.

AUTOCOMMIT

```
41
42 • SELECT @@AUTOCOMMIT;
43
44
```

< [Progress Bar]

Result Grid | [Grid Icon] [Refresh Icon] Filter Rows: [Input Box]

	@@AUTOCOMMIT
▶	1

```
45
46
47 • SET AUTOCOMMIT=0; -- otomatik commiti kapatır
48
49 • SELECT @@AUTOCOMMIT;
50
```

< [Progress Bar]

Result Grid | [Grid Icon] [Refresh Icon] Filter Rows: [Input Box] | Export: [Icon] | Wrap




	@@AUTOCOMMIT
▶	0

COMMIT

- Yapılan DML işlemlerini kalıcı hale getirmek için kullanılır.

```
7 • SELECT * FROM okul.danisman ;  
8
```

<

Result Grid |   Filter Rows: | Edit: 

	danisman_id	ad	soyad	departman
•	NULL	NULL	NULL	NULL

COMMIT

```
1  
2 • START TRANSACTION;  
3 • INSERT INTO okul.danisman VALUES (1,'ALİ','ÇAN','BİLGİSAYAR PROG.');  
4 • INSERT INTO okul.danisman VALUES (2,'Mehmet','GEZER','FİZİK');  
5 • INSERT INTO okul.danisman VALUES (3,'Aslı','ŞENTÜRK','MATEMATİK');  
6 • INSERT INTO okul.danisman VALUES (4,'Ayşe','DAL','YAZILIM MÜHENDİSLİĞİ');  
7 • COMMIT;
```

Result Grid | Filter Rows: | Edit: | Export/Import: |

	danisman_id	ad	soyad	departman
▶	1	ALİ	ÇAN	BİLGİSAYAR PROG.
	2	Mehmet	GEZER	FİZİK
	3	Aslı	ŞENTÜRK	MATEMATİK
	4	Ayşe	DAL	YAZILIM MÜHENDİSLİĞİ
•	NULL	NULL	NULL	NULL

COMMIT

```
8
9 ● START TRANSACTION;
10 ● INSERT INTO okul.ogrenci VALUES (1000,'Şeyma','ÇAĞLAR',3);
11 ● INSERT INTO okul.ogrenci VALUES (1001,'Çiğdem','GEZER',3);
12 ● INSERT INTO okul.ogrenci VALUES (1002,'Süleyman','ARSLAN',4);
13 ● COMMIT;
14
15
```

<

Result Grid |   Filter Rows: | Edit:    | Export/Import: 

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	3
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	4
★	NULL	NULL	NULL	NULL

COMMIT

```
15 ● START TRANSACTION;  
16 ● UPDATE okul.danisman SET ad='Alican' WHERE danisman_id=1;  
17 ● COMMIT;  
18  
19 ● SELECT * FROM okul.danisman ;  
20
```

Result Grid | Filter Rows: | Edit: | Export

	danisman_id	ad	soyad	departman
▶	1	Alican	ÇAN	BİLGİSAYAR PROG.
	2	Mehmet	GEZER	FİZİK
	3	Aslı	ŞENTÜRK	MATEMATİK
	4	Ayşe	DAL	YAZILIM MÜHENDİSLİĞİ

SAVE POINT

- Tanımlayıcının bir adıyla adlandırılmış bir işlem kayıt noktası ayarlar.
- Geçerli işlemin aynı adı taşıyan bir kayıt noktası varsa, eski kayıt noktası silinir ve yenisi belirlenir.
- Yapılan işlemlerin START TRANSACTION içerisinde yapılması gerekir.

SAVE POINT save_name;

ROLLBACK TO SAVEPOINT save_name;

SAVE POINT

```
8 • INSERT INTO okul.ogrenci VALUES (1000,'Şeyma','ÇAĞLAR',3);
9 • INSERT INTO okul.ogrenci VALUES (1001,'Çiğdem','GEZER',3);
10 • INSERT INTO okul.ogrenci VALUES (1002,'Süleyman','ARSLAN',4);
11 • COMMIT;
12
13 • SELECT * FROM okul.ogrenci ;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	3
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	4
*	NULL	NULL	NULL	NULL

```
19
20 • START TRANSACTION;
21 • UPDATE okul.ogrenci SET danisman=1 WHERE ogrenci_no=1000;
22 • SAVEPOINT ogrenci_1000_danisman_atamasi;
23
24 • UPDATE okul.ogrenci SET danisman=1 WHERE ogrenci_no=1001;
25 • SAVEPOINT ogrenci_1001_danisman_atamasi;
26
27 • ROLLBACK TO SAVEPOINT ogrenci_1000_danisman_atamasi;
28
29 • UPDATE okul.ogrenci SET danisman=2 WHERE ogrenci_no=1002;
30 • SAVEPOINT ogrenci_1002_danisman_atamasi;
31 • COMMIT;
32
33 • SELECT * FROM okul.ogrenci ;
```

Result Grid | Filter Rows: | Edit: | Export:

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	2
*	NULL	NULL	NULL	NULL

ROLLBACK

- ROLLBACK ile yapılmış bir işlemi geri alabiliriz.
- İşlem COMMIT edilse bile bir önceki haline geri dönüş yapılabilir.
- Fakat AUTOCOMMIT'in kağıalı olması gerekir.
- Yapılan işlemlerin START TRANSACTION içerisinde yapılması gerekir.
- Aksi halde ROLLBACK ifadesi çalışmaz.

ROLLBACK

```
32
33 • SELECT * FROM okul.ogrenci ;
34
35
```

Result Grid | Filter Rows: | Edit: | Export

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	2
•	NULL	NULL	NULL	NULL

```
34
35 • START TRANSACTION;
36 • UPDATE okul.ogrenci SET danisman=1 WHERE ogrenci_no=1002;
37
38 • ROLLBACK;
39 • COMMIT;
40
41 • SELECT * FROM okul.ogrenci ;
42
```

Result Grid | Filter Rows: | Edit: | Export/Import

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	1
•	NULL	NULL	NULL	NULL

ROLLBACK

```
34
35 ● START TRANSACTION;
36 ● UPDATE okul.ogrenci SET danisman=1 WHERE ogrenci_no=1002;
37 ● ROLLBACK;
38 ● COMMIT;
39
40 ● SELECT * FROM okul.ogrenci ;
41
```

Result Grid				
Filter Rows:				
	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	2
•	NULL	NULL	NULL	NULL

VERİTABANI GÜVENLİĞİ İÇİN A.C.I.D

A.C.I.D (Atomicity, Consistency, Isolation, Durability)

- Güvenli bir veritabanı veriler üzerinde değişiklik yaparken ACID kuralını sağlamalıdır.
 1. Atomicity (Bölünmezlik)
 2. Consistency (Tutarlılık)
 3. Isolation (İzolasyon)
 4. Durability (Dayanıklılık)

Atomicity (Bölünmezlik)

- Transaction daha küçük parçalara ayrılamayan bir işlem birimi olarak ele alınır.
- Transaction bloğu yarım kalmaz.
- Ya hepsi gerçekleşmiş sayılır ya da hiçbir işlem gerçekleşmemiş gibi kabul edilerek en başa dönülür.
 - Veritabanları erişilemez olabilir.
 - Network problemi olabilir.
 - Herhangi bir hata oluşabilir.
- Bu durumda işlem geçersiz sayılacaktır.

Consistency (Tutarlılık)

- Transaction veritabanının yapısını bozmadan işlem bloğunu terk etmelidir.
- Yani ara işlemler yaparken ürettiği işlem parçacıklarının etkisini veritabanında bırakarak, transaction'ı sonlandıramaz.
- Örneğin, birinci hesaptan para azaltıp ikinci hesaba eklemekten transaction sonlandırılmaz.

Isolation (İzolasyon)

- Farklı transaction'lar birbirinden ayrık ele alınmalıdır.
- Aynı anda aynı veri üzerinde birden fazla Transaction değiştirme gereksinimi olabilir.
- Transaction'ların birbirlerinin işlemlerinden etkilenmemesi için işlemler Seri olarak yapılması gerekir.
- Her transaction için veritabanının yapısı ayrı ayrı korunmalıdır.
- İlk transaction tarafından yapılan değişiklikler, ikinci transaction'dan her an görülememeli, sadece bütün işlem gerçekleştiği anda ve bütünü bir anda görülmelidir.

Durability (Dayanıklılık)

- Tamamlanmış transaction'ın hatalara karşı esnek olması gerekir.
- Transaction sırasında fiziksel veya işlemsel bir hata olması durumunda sistemin kendisini bir önceki geçerli veri durumuna döndürebilme kabiliyetidir.
- Elektrik kesilmesi, CPU yanması, işletim sisteminin çökmesi bu kuralları uygulamaya engel olmamalıdır.
- Bunun içinde gerçekleşmiş ve başarılı olarak sonlanmış transaction'ın değişikliklerinin kalıcı olarak diske yansıtılması gerekir.

SAKLI YORDAMLAR (STORED PROCEDURES)

Saklı Yordamlar

- Veritabanında saklanan hazır derlenmiş sql kod bloklarıdır.
- Saklı yordamlar uygulamanın performansını yükseltir.
- Uygulama ve veritabanı sunucusu arasındaki trafiği azaltır.
- Saklı yordamlar tekrar kullanılabilir.
- Saklı yordamlar güvenlidir.

Saklı Yordamlar

- Oluşturulan veya var olan saklı yordamlar dışarıdan parametre alabilirler.
- Bir kez yazılıp, tekrar tekrar kullanıldığı için program modüler bir yapıda geliştirilmiş olur.
- Otomatik devreye giremezler.
- Uygulama ya da script tarafından çağırılmaları gereklidir.

Neden Kullanılır?

- Kodların yeniden kullanımı
 - Aynı veritabanını kullanan farklı uygulamalar tekrar aynı kodları yazmak yerine saklı yordamları kullanabilir.
- Daha kolay kodlama
 - Geliştirici sorgu ya da tabloların adını tam olarak bilmeden bir saklı yordamı çağırabilir.
 - Veritabanını daha kullanıcı dostu kılarlar.

Saklı Yordamlar Yapısı

- CREATE PROCEDURE ya da kısaca CREATE PROC ifadesi ile yaratılır.

```
CREATE PROCEDURE procedure_name  
BEGIN  
    sql_statement;  
END;
```

- Saklı yordamı yürütme için;

```
CALL procedure_name;
```

Saklı Yordamlar Örnek

```
CREATE PROCEDURE GetAllClients()  
BEGIN  
    SELECT * FROM Clients;  
END;  
  
CALL GetAllClients();
```

Saklı Yordam İçerisinde Değişkenler Tanımlamak

- Saklı yordamlar içerisinde kullanabileceğimiz değişkenler tanımlayabiliriz.
- Değişken tanımlama DECLARE direktifi ile yapılır.

```
DECLARE degisken_adi veritipi(boyut) DEFAULT varsayilan_deger;
```

```
DECLARE totalBandwidth INT(20);
```

Saklı Yordam İçerisinde Değişkene Değer Atamak

- Stored procedure içerisinde tanımlanmış bir değişkene iki yolla değer atanabilir.
- Sql cumlecisi ile;

```
SELECT count(*) INTO totalBandwidth FROM NetworkReport;
```

- Set direktifi ile;

```
SET totalBandwidth = 3412314124;
```

Değişken Ömrü

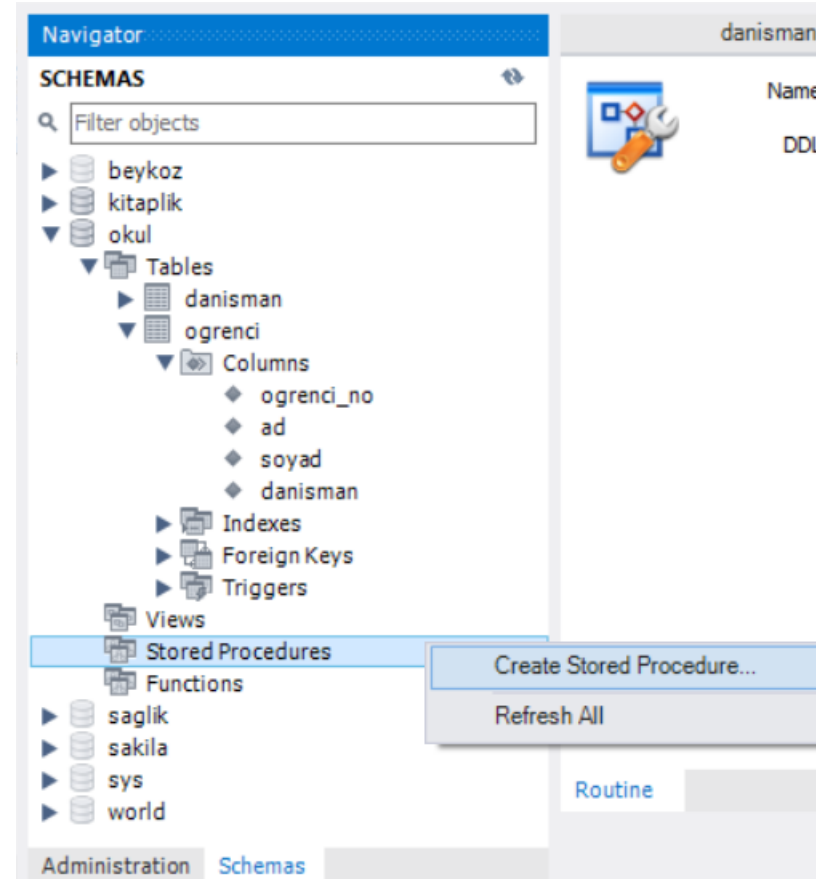
```
DELIMITER//  
CREATE PROCEDURE TestProcedure()  
BEGIN  
    DECLARE totalBandwidth int(20);  
    BEGIN  
        DECLARE totalHit int(20);  
    END  
END //  
DELIMITER;
```


Saklı Yordamlar Örnek

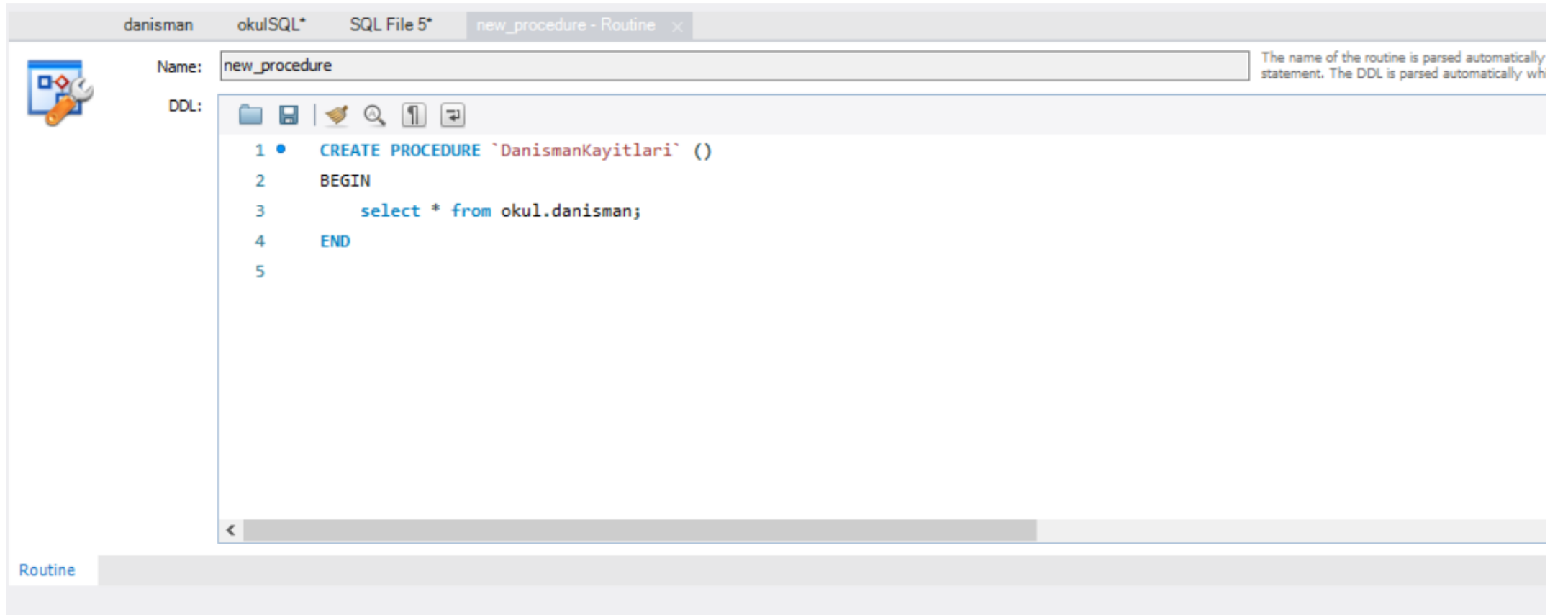
```
DECLARE benim_adim VARCHAR(50) DEFAULT "";  
SET benim_adim = 'berika';
```

```
DELIMITER //  
CREATE PROCEDURE SakliYordam()  
BEGIN  
    DECLARE benim_adim VARCHAR(50) DEFAULT "";  
    SET benim_adim = 'berika';  
    SELECT CHAR_LENGTH(benim_adim) AS Uzunluk;  
END//  
DELIMITER ;
```

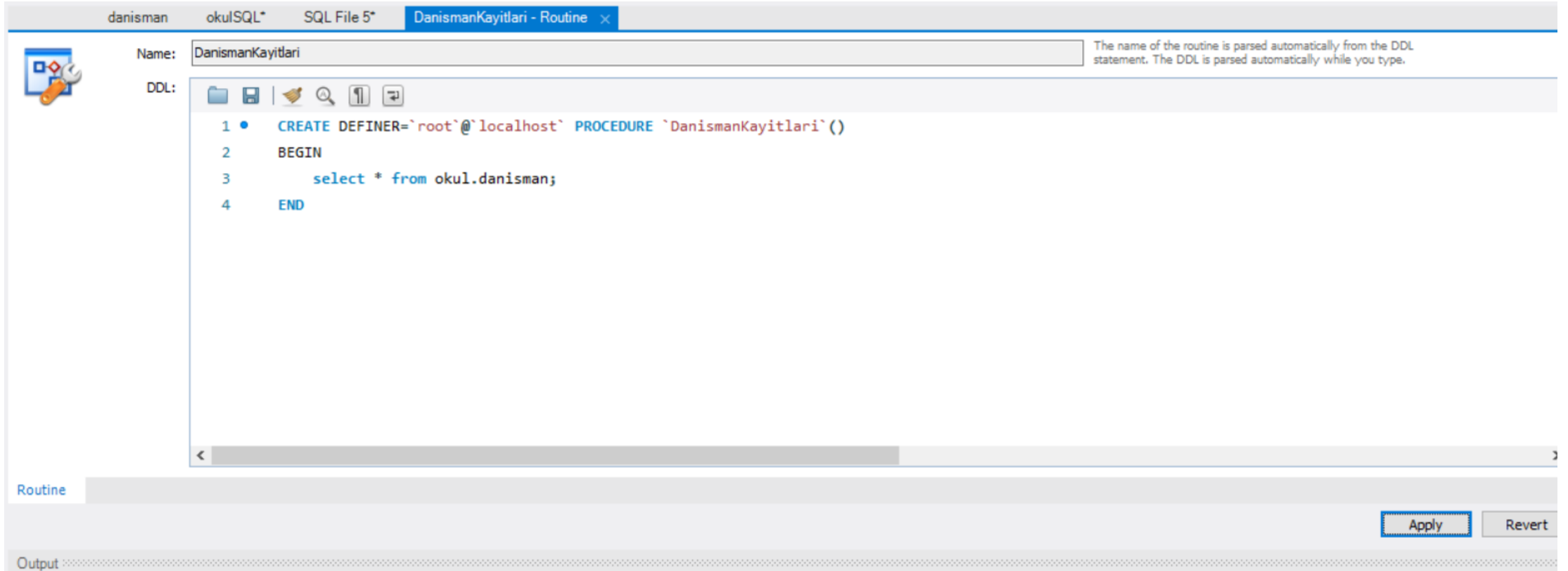
Örnek



Örnek



Örnek

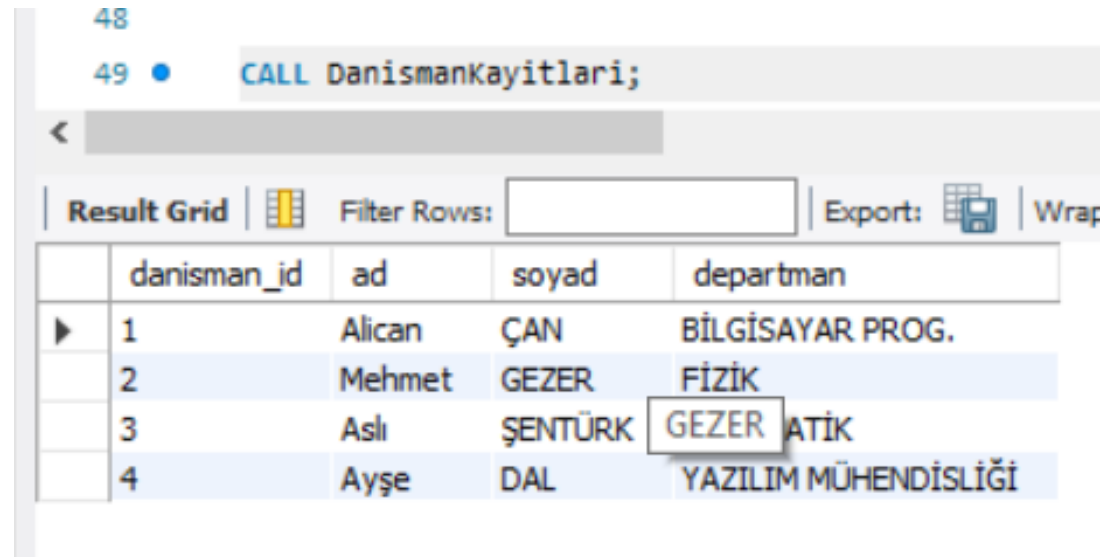
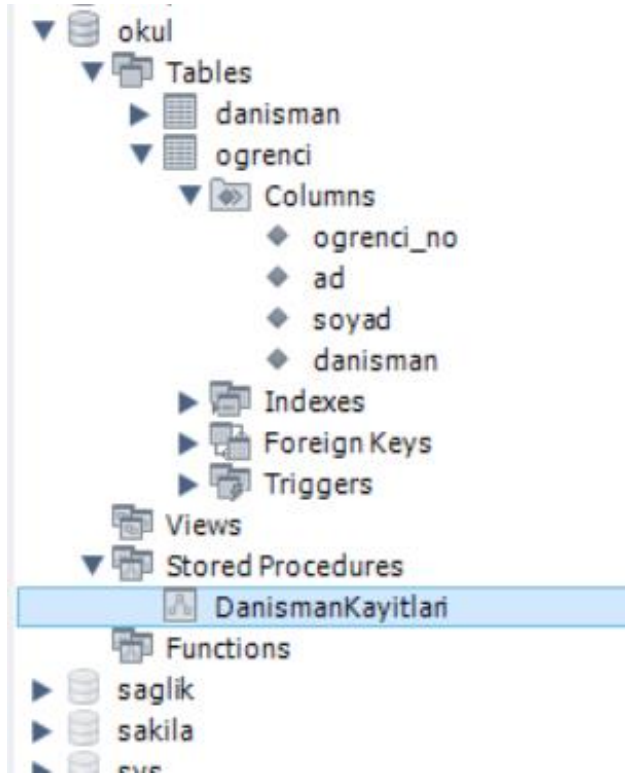


The screenshot displays a MySQL IDE window with the following components:

- Tab Bar:** Contains tabs for 'danisman', 'okulSQL*', 'SQL File 5*', and the active 'DanismanKayitlari - Routine'.
- Name Field:** Labeled 'Name:', it contains the text 'DanismanKayitlari'. A tooltip on the right states: 'The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.'
- DDL Field:** Labeled 'DDL:', it contains a SQL script:

```
1 • CREATE DEFINER=`root`@`localhost` PROCEDURE `DanismanKayitlari`()  
2 BEGIN  
3     select * from okul.danisman;  
4 END
```
- Buttons:** 'Apply' and 'Revert' buttons are located at the bottom right of the DDL field.
- Output:** An 'Output' section is visible at the bottom of the window.

Örnek



Örnek

```
51 • USE okul;  
52 • DROP procedure IF EXISTS Ogrencikayitlari;  
53  
54 DELIMITER $$  
55 • USE okul$$  
56 • CREATE PROCEDURE Ogrencikayitlari ()  
57 BEGIN  
58     select * from okul.ogrenci;  
59 END$$  
60  
61 DELIMITER ;  
62  
63  
64 • CALL Ogrencikayitlari;
```

< Result Grid | Filter Rows: | Export: | Wrap

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1001	Çiğdem	GEZER	3
	1002	Süleyman	ARSLAN	1

Örnek

```
66
67 DELIMITER $$
68 ● USE okul$$
69 ● CREATE PROCEDURE Adinda_ey_GecenOgrenciler ()
70 BEGIN
71     SELECT * FROM okul.ogrenci WHERE ad LIKE '%ey%';
72 END$$
73
74 DELIMITER ;
75
76 ● CALL Adinda_ey_GecenOgrenciler;
```

<

Result Grid | Filter Rows: | Export: | Wrap Cell Co

	ogrenci_no	ad	soyad	danisman
▶	1000	Şeyma	ÇAĞLAR	1
	1002	Süleyman	ARSLAN	1

Parametrelili Saklı Yordamlar

- Yarattığımız saklı yordamlar genellikle parametrelili olurlar.
- Parametre ile saklı yordama değer gönderebilir, değer alabiliriz.
- Mysql'de saklı yordam parametreleri 3 yöntem belirtecinden birini alabilir.
 - IN
 - OUT
 - INOUT

IN Belirteci

- Parametreye sadece değer göndereceksek kullanırız.

```
CREATE PROCEDURE GetAge (IN p_personId INT(3))  
BEGIN  
    SELECT age FROM Persons WHERE personId = p_personId  
END
```

Call GetAge (345);

OUT Belirteci

- Parametreye dışarıdan bir değişken gönderip saklı yordamın bu değişkene bir değer atamasını sağlarız.

```
CREATE PROCEDURE StudentTotal (IN s_name VARCHAR(45),  
OUT total INT)  
BEGIN  
    SELECT COUNT(*) INTO total FROM Student  
    WHERE name = s_name  
END
```

Call StudentTotal ('Ayşe'@total);

INOUT Belirteci

- Hem saklı yordama değer gönderilebilir hem de saklı yordamdan bir değer alınabilir.

```
CREATE PROCEDURE StudentUpdate (INOUT id INT)
BEGIN
    UPDATE Student SET s_id=s_id+5 WHERE s_id=id
END;
```

SET @id=1;

Call StudentUpdate (@id);