YILDIZ TEKNİK ÜNİVERSİTESİ

SQL ile Veri Bütünlüğü & Güvenlik

Öğr. Gör. Dr. Yasemin Topuz

Yıldız Teknik Üniversitesi

BLM3041 – Veritabanı Yönetimi

13 Kasım 2025 | YTÜ Bilgisayar Mühendisliği

Neler konuşacağız?

- Stored Procedure, SQL/PSM
- Triggers
- Güvenlik

Uygulama Programları: Yaklaşımlar

3) **Stored Procedure/Tetikleyici Dilleri**

- **Tanım:** DB-yerel dillerle (PL/pgSQL, PL/SQL, T-SQL vb.) yazılan stored procedure / function / trigger'lar veritabanında kalıcı olarak saklanır ve sunucuda çalışır.
- **Nasıl çalışır?** Uygulama sadece CALL/SELECT ile prosedürü/fonksiyon tetikler; işlem mantığı DB içinde, veriye en yakın yerde yürür (set-tabanlı optimizasyon, ağ turu yok).
- **Derleme zinciri:** DBMS'in kendi derleyicisi/yorumlayıcısı kullanılır (CREATE PROCEDURE/FUNCTION). Uygulama tarafında ek preprocessor yok
- **Taşınabilirlik:** Düşük–orta. Sözdizimi ve özellikler DBMS'e bağımlıdır (PL/pgSQL ≠ PL/SQL ≠ T-SQL). Taşıma genelde yeniden yazım ister.

Artı: En hızlı veri işleme (minimum network), atomiklik ve güvenlik (tek transaction bloğu, tablo yerine prosedür yetkisi), Impedans uyumsuzluğunu azaltır.

Eksi: Sınırlı esneklik (büyük mimarilerde iş mantığını DB'ye yükler). PL/pgSQL ≠ PL/SQL ≠ T-SQL. Bir DBMS'ten diğerine geçmek çoğu zaman yeniden yazım ister.

3) Database Stored Procedures

- Kalıcı prosedürler/fonksiyonlar (modüller) DB sunucuda saklanır ve sunucuda çalışır (istemci yerine).
  - Bugün ki ziyaretçi sayısı 5000'i geçmiş mi? Geçmişse tüm ziyaretçilere kutlama maili gönder.

Avantajları:

- Prosedür birçok uygulama tarafından ihtiyaç duyuluyorsa, herhangi biri tarafından çağrılabilir (böylece tekrarlar azalır.)
- Sunucu tarafından yürütülmesi iletişim maliyetlerini azaltır.
- Görünümlerin modelleme gücünü artırır.

Dezavantajları:

- Her DBMS'nin kendi sözdizimi vardır ve bu, sistemi daha az taşınabilir hale getirebilir.

3) Database Stored Procedures – Avantajlar

- **Geliştirilmiş Performans:** Saklı prosedürler önceden derlenmiştir, yani birden fazla ayrı sorguyu çalıştırmaktan daha hızlı yürütülürler.
- **Gelişmiş Güvenlik:** Kullanıcılara, alttaki tablolara doğrudan erişmeden saklı yordamları yürütme izni verilebilir.
- **Kodun Yeniden Kullanılabilirliği:** Saklı prosedürler yeniden kullanılabilirliğe izin verir, kodun bakımını ve güncellemesini kolaylaştırır.
- **Azaltılmış Ağ Trafiği:** Birden fazla SQL ifadesini tek bir çağrıda birleştirerek, saklı yordamlar ağ yükünü azaltır ve uygulama performansını iyileştirir.
- **Daha İyi Hata Yönetimi:** SQL saklı yordamları, blokları kullanarak hataları yönetmenin yapılandırılmış bir yolunu sağlar TRY...CATCH.

3) Database Stored Procedures – SQL/PSM nedir?

- **SQL/PSM (Persistent Stored Modules):** SQL standardının, kalıcı yordam/fonksiyon yazımı için tanımladığı ek yapılar (değişken, koşul, döngü, exception vb.).
- Yani: SQL + programlama yapıları → SQL'in ifade gücünü artırır (dallanma/tekrarlama, hata yönetimi, modülerlik).

Yordam (PROCEDURE): birden fazla adım içeren, veritabanı durumunu değiştirmeyi gerektiren veya işlem denetimi gerektiren görevler için daha uygundur.

```
CREATE PROCEDURE proc_name(param_decls)
BEGIN
  -- local-declarations
  -- procedure-body  (SQL deyimleri + IF/CASE/LOOP/WHILE + HANDLER)
END;
```

Çağırma

```
CALL proc_name(arg1, arg2, ...);
```

Fonksiyon (FUNCTION): özellikle sonucu bir SQL sorgusunun parçası olarak kullanmak istiyorsanız, giriş parametrelerine dayalı bir değer hesaplamanız ve döndürmeniz gerektiğinde en iyi seçenektir.

```
CREATE FUNCTION fun_name(param_decls) RETURNS return_type
BEGIN
  -- local-declarations
  -- function-body  (en sonda RETURN ifadesi)
END;
```

Çağırma

```
SELECT fun_name(arg1, arg2, ...);
```

SQL/PSM: Örnek 1

```
CREATE FUNCTION dept_size(deptno INTEGER)
RETURNS VARCHAR(7)
LANGUAGE plpgsql
AS $$      (Pltcl, plv8, Plpythonu vb.)
DECLARE
    tot_emps INTEGER;
BEGIN
    SELECT COUNT(*) INTO tot_emps FROM employee
    WHERE dno = deptno;

    IF    tot_emps > 100 THEN RETURN 'HUGE';
    ELSIF tot_emps > 50 THEN RETURN 'LARGE';
    ELSIF tot_emps > 30 THEN RETURN 'MEDIUM';
    ELSE RETURN 'SMALL';
    END IF;
END;
$$;
```

IN (default): Kullanıcıdan bu fonksiyonu çağırırken değer belirtmesi istenildiğinde

OUT: Programın akışı sırasında o değişkenin içindeki değerin değişmesi. Aslında Return değişkeni. Return komutu ile sadece 1 sonuç döndürülür ancak birden fazla **OUT** değişkeni döndürülebilir.

```
SELECT dept_size (5)
```

SQL/PSM: Örnek 2

- Gönderilen isim ve soyisime sahip çalışan tabloda yoksa ekleyen varsa 'zaten var' mesajını yazdıran procedure yazınız.

```
CREATE PROCEDURE employee_add(fname_add VARCHAR, lname_add VARCHAR, ssn_add CHAR(9), dno_add INT)
LANGUAGE plpgsql
AS $$
DECLARE
    sayi INTEGER;
BEGIN
    -- Aynı isim ve soyisimde çalışan var mı kontrol et
    SELECT COUNT(*) INTO sayi
    FROM EMPLOYEE
    WHERE LOWER(Fname) = LOWER(fname_add)
      AND LOWER(Lname) = LOWER(lname_add);
    IF sayi = 0 THEN
        INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)
        VALUES (fname_add, lname_add, ssn_add, dno_add);
        RAISE NOTICE 'Yeni çalışan eklendi: % %', fname_add, lname_add;
    ELSE
        RAISE NOTICE 'Bu çalışan zaten var: % %', fname_add, lname_add;
    END IF;
END;
$$;
```

```
CALL employee_add('Ali', 'Veli', '987654320', 5);
```

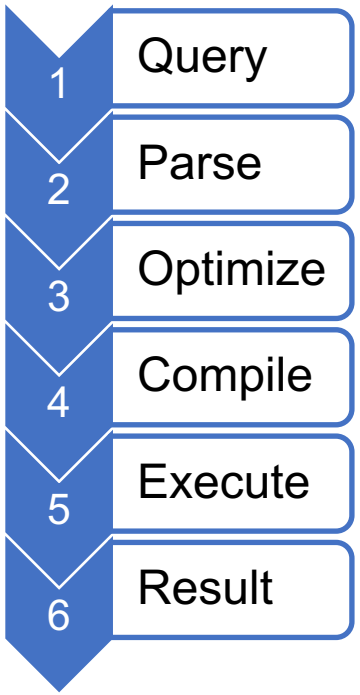
```
NOTICE:  Yeni çalışan eklendi: Ali Veli
NOTICE:  Bu çalışan zaten var: Ali Veli
```

Stored Procedure vs AD HOC Query

Stored Prosedürlerin doğru yer ve zamanda kullanıldığında performansa etkisi

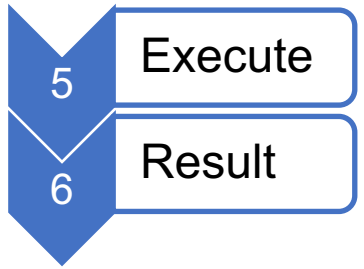
AD HOC QUERY

SELECT * FROM EMPLOYEE

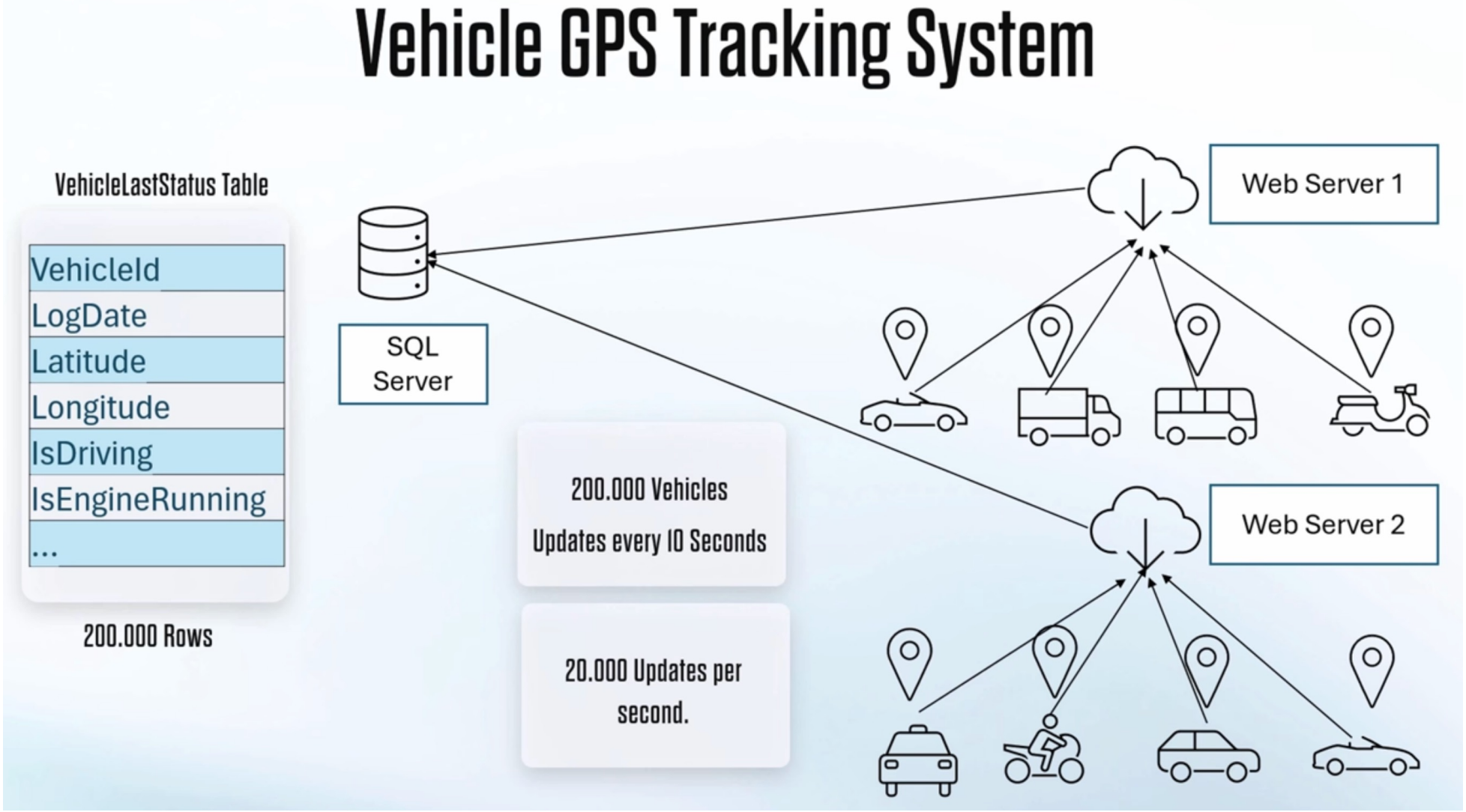


STORED PROCEDURE

CALL SP_GETEMPLOYEE



Stored Procedure vs AD HOC Query

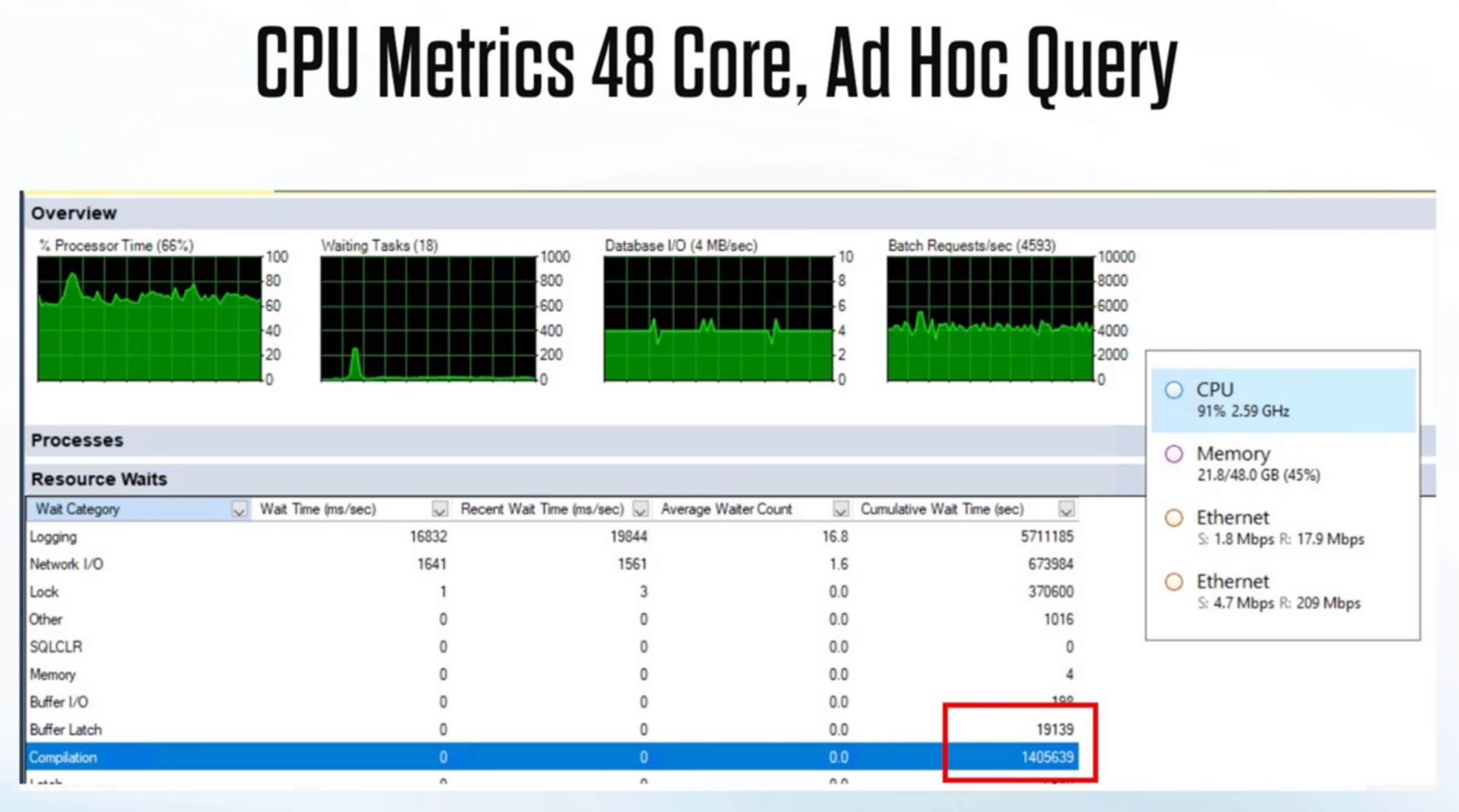


Stored Procedure vs AD HOC Query

Vehicle GPS Tracking System

EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_executesql N'UPDATE top(1)							21
RPC:Completed	exec sp_executesql N'UPDATE top(1)							24
RPC:Completed	exec sp_executesql N'UPDATE top(1)							26
RPC:Completed	exec sp_executesql N'UPDATE top(1)							43
RPC:Completed	exec sp_executesql N'UPDATE top(1)							31
RPC:Completed	exec sp_executesql N'UPDATE top(1)							48
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_executesql N'UPDATE top(1)							2
RPC:Completed	exec sp_executesql N'UPDATE top(1)							3
RPC:Completed	exec sp_executesql N'UPDATE top(1)							3
RPC:Completed	exec sp_executesql N'UPDATE top(1)							2
RPC:Completed	exec sp_executesql N'UPDATE top(1)							4
RPC:Completed	exec sp_executesql N'UPDATE top(1)							5
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_executesql N'UPDATE top(1)							0
RPC:Completed	exec sp_executesql N'UPDATE top(1)							4
RPC:Completed	exec sp_executesql N'UPDATE top(1)							22
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_executesql N'UPDATE top(1)							1
RPC:Completed	exec sp_reset_connection							0
RPC:Completed	exec sp_reset_connection							0

Stored Procedure vs AD HOC Query



Stored Procedure vs AD HOC Query

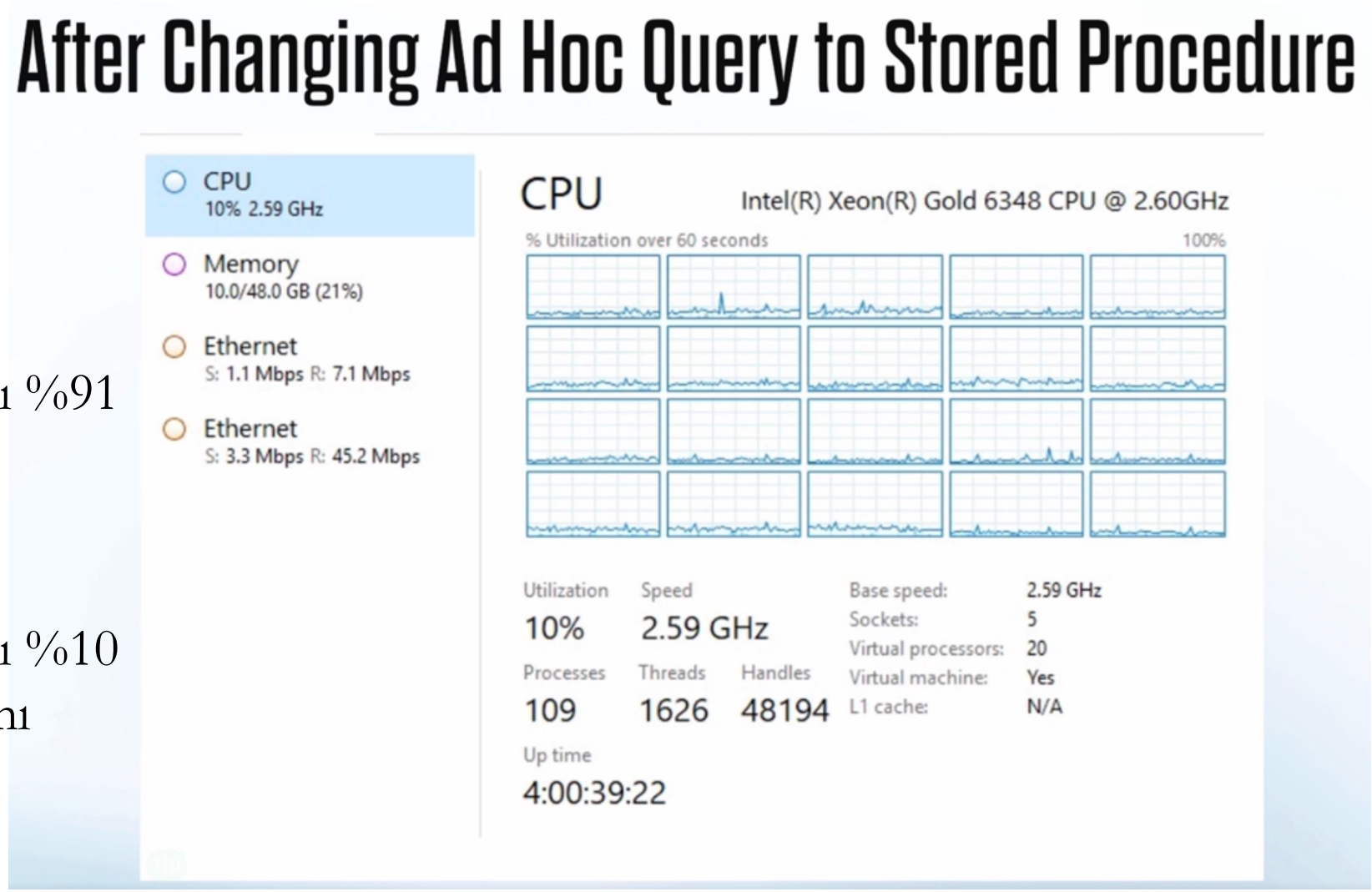
After Changing Ad Hoc Query to Stored Procedure

EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_update				0	2	0	3
RPC:Completed	exec sp_update				0	2	0	4
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				16	2	0	3
RPC:Completed	exec sp_update				0	2	0	4
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_update				0	2	0	2
RPC:Completed	exec sp_update				0	2	0	0
RPC:Completed	exec sp_reset_				0	0	0	0
RPC:Completed	exec sp_update				0	2	0	1
RPC:Completed	exec sp_reset_				0	0	0	0

Stored Procedure vs AD HOC Query

Ad Hoc Query:
48 core CPU, CPU kullanımı %91
200 Mbit network kullanımı

Stored Procedure
20 core CPU, CPU kullanımı %10
20-30 Mbit network kullanımı



Stored Procedure vs AD HOC Query

Senaryo	Önerilen Yaklaşım
Yüksek trafikli	🥇 **Stored Procedure**
Güvenlik kritik	🥇 **Stored Procedure**
Dinamik raporlama	🥈 **Ad Hoc (Parametreli)**
Hızlı prototip	🥈 **Ad Hoc Query**
Cross-database	🥈 **Ad Hoc Query**

Kural: Performans ve güvenlik için **Stored Procedure**, esneklik için **Ad Hoc Query**! 🎯

Constraints as Assertions

General Constraints: Temel SQL kısıtlamalarına (örneğin UNIQUE, PRIMARY KEY, NOT NULL, FOREIGN KEY vb.) uymayan daha karmaşık kurallardır.

- Bu tür kısıtlamalar, veritabanında ilişkisel bütünlüğü sağlamak için kullanılır.

Mekanizma: CREATE ASSERTION

- Bir assertion (genel kısıtlama) oluşturmak için şu yapı kullanılır:
  - Kısıtlama adı yazılır,
  - Ardından CHECK ifadesi gelir,
  - CHECK ifadesini bir koşul (condition) izler.
- "Bir çalışanın maaşı, çalıştığı departmanın yöneticisinin maaşından **daha yüksek olmamalıdır**."

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
        (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
         WHERE E.SALARY > M.SALARY AND E.DNO = D.NUMBER
         AND D.MGRSSN = M.SSN));
```

NOT !

CREATE ASSERTION ifadesi SQL standardında tanımlıdır ancak bazı veritabanı sistemleri (örneğin PostgreSQL, MySQL) tarafından doğrudan desteklenmez.

Bu durumda aynı mantık **TRIGGER** (tetikleyici) kullanılarak uygulanabilir.

SQL Triggers (Tetikleyiciler)

- **Amaç (Objective):** Bir veritabanını izlemek ve belirli bir durum oluştuğunda otomatik bir işlem başlatmak.
  - Satış yapılan ürünün stoktaki adedi 50 tanedir. Üründen 10 adet satıldı. Ürünün stoktaki sayısının otomatik olarak 10 tane azalması için kullanılır.

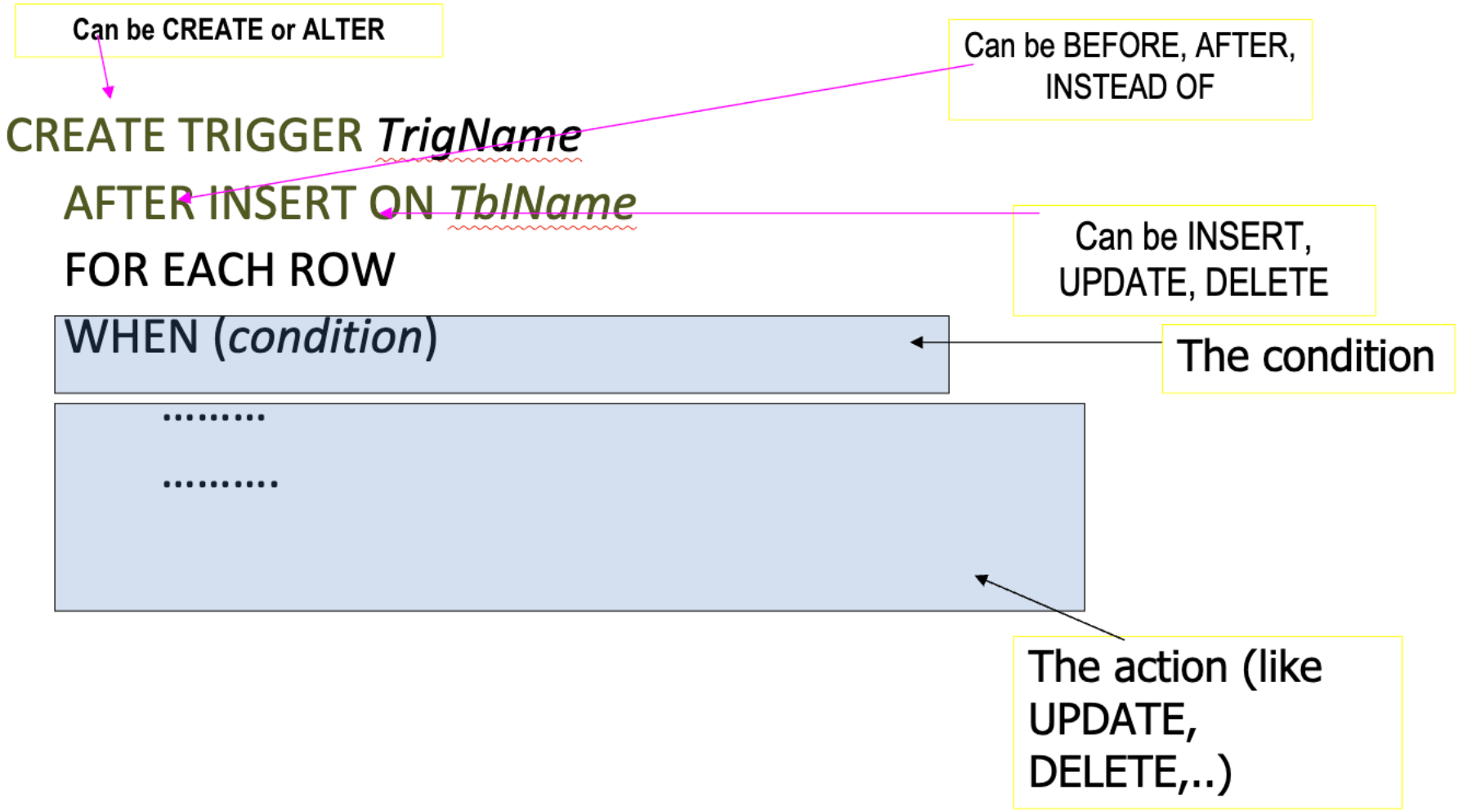
- **TRIGGER**, bir tablo üzerinde **INSERT, UPDATE** veya **DELETE** işlemi gerçekleştiğinde otomatik olarak çalışan bir komuttur.
  - Hiç elle çağrılmaz — sistem kendisi çalıştırır.
  - Bir olay ("event") gerçekleştiğinde tetikleyici otomatik olarak "ateşlenir" (çalışır).

- Tetikleyiciler, bir sözdiziminde ifade edilir ve şunları içerir:
  - **Olay (Event):** Ekleme, silme veya güncelleme işlemi gibi
  - **Koşul (Condition):** İsteğe bağlı
  - **Eylem (Action):** Koşul sağlandığında gerçekleştirilecek

SQL Triggers (Tetikleyiciler)

- **CREATE TRIGGER** <name> : Bir tetikleyiciyi oluştur
- **ALTER TRIGGER** <name> : Bir tetikleyiciyi değiştirir (varsa)
- **CREATE OR ALTER TRIGGER** <name>
  - Tetikleyici yoksa oluşturur.
  - Tetikleyici varsa değiştirir.

SQL Triggers (Tetikleyiciler)



BEFORE → Asıl işlemden önce tetiklenir. (Örneğin, veri eklenmeden önce kontrol yapılabilir.)

AFTER → Asıl işlemden sonra tetiklenir. (Genelde log tutmak veya başka tabloları güncellemek için kullanılır.)

INSTEAD OF → Asıl işlem yerine tetiklenir. (Özellikle view'larda kullanılır.)

Geçerli Trigger Türleri

- **Triggered by INSERT:**
  - BEFORE INSERT statement-level.
  - BEFORE INSERT row-level.
  - AFTER INSERT statement-level.
  - AFTER INSERT row-level.
- **Triggered by UPDATE:**
  - BEFORE UPDATE statement-level.
  - BEFORE UPDATE row-level.
  - AFTER UPDATE statement-level.
  - AFTER UPDATE row-level.
- **Triggered by DELETE:**
  - BEFORE DELETE statement-level.
  - BEFORE DELETE row-level.
  - AFTER DELETE statement-level.
  - AFTER DELETE row-level.
- **To replace the triggering event:**
  - INSTEAD OF statement-level.
  - INSTEAD OF row-level.

Özellik	🧩 Deyim Düzeyi (Statement-Level)	🔬 Satır Düzeyi (Row-Level)
Çalışma Sıklığı	Tetikleyici, SQL ifadesi başına **yalnızca 1 kez** çalışır.	Her etkilenen satır için **ayrı ayrı** çalışır.
Erişilebilir Veriler	`:NEW` veya `:OLD` değerlerine (yani satır verilerine) **erişemez**.	`:NEW` ve `:OLD` ile satırın **önceki ve yeni hâline** erişilebilir.
Performans	Çok hızlıdır, çünkü yalnızca 1 kez tetiklenir.	Çok satır varsa **her satırda** tetikleneceği için daha **yavaş** olabilir.
Kullanım Alanı	- Toplu işlemlerde log veya sayaç tutmak - İşlem başlamadan önce genel bir kontrol yapmak	- Satır bazında değişiklikleri izlemek - Her kayıt için doğrulama, log veya hesaplama yapmak
Avantajı	Basit, hızlı, düşük maliyetli.	Esnek, satır verisine erişim sağlar, detaylı kontrol imkânı sunar.
Dezavantajı	Satır bazında işlem yapılamaz, hangi satır değiştiği bilinmez.	Büyük veri kümelerinde performans düşebilir.

Örnek Trigger

```
CREATE TRIGGER LogSalaryChange
AFTER UPDATE OF Salary ON EMPLOYEE
FOR EACH ROW
REFERENCING OLD ROW AS oldrow NEW ROW AS newrow
WHEN (oldrow.Salary <> newrow.Salary)
INSERT INTO SALARY_LOG (UserName, DateChanged, EmpID, OldSalary, NewSalary)
VALUES (CURRENT_USER, CURRENT_DATE, newrow.EmpID, oldrow.Salary, newrow.Salary);
```

📊 Örnek Log Tablosu (SALARY_LOG)

UserName	DateChanged	EmpID	OldSalary	NewSalary
HR_Admin	2025-11-13	101	8000	8800
HR_Admin	2025-11-13	102	7200	7500

Örnek Trigger

EMPLOYEE (FNAME, SSN, SALARY, DNO, SUPERVISOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER_SSN)

- Bir çalışan eklenirken veya maaşı güncellenirken, çalışanın maaşı kendi süpervizöründen (yöneticisinden) fazla olmasın.
  Eğer fazla olursa, yöneticiyi bilgilendiren bir işlem (örneğin INFORM_SUPERVISOR) tetiklensin.

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.SALARY >
      (SELECT SALARY FROM EMPLOYEE
       WHERE SSN = NEW.SUPERVISOR_SSN))
  INFORM_SUPERVISOR(NEW.SUPERVISOR_SSN, NEW.SSN);
```

Örnek Trigger

EMPLOYEE (FNAME, SSN, SALARY, DNO, SUPERVİSOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER _SSN)

- **Amaç:** Departman tablosundaki TOTAL_SAL (departman toplam maaş) alanını güncel tutmak.
- Herhangi bir çalışan eklenip, silinip veya maaşı/departmanı değiştiğinde, ilgili departmanın toplam maaş değeri otomatik olarak güncellenmelidir.

Tetikleyiciyi (Trigger) Çalıştıran 4 Olay

T1: Total_sal1 – Yeni Çalışan Eklendiğinde (Bir çalışanın bir departmana eklenmesi durumunda, o departmanın toplam maaşına çalışanın maaşı eklenir.)

- **Çalışma Zamanı:** AFTER INSERT ON EMPLOYEE (EMPLOYEE tablosuna veri eklendikten sonra tetiklenir.)
- **Çalışma Şekli:** FOR EACH ROW (Her Satır için)
- **Koşul:** WHEN NEW.Dno IS NOT NULL
- **İşlem:**
  - DEPARTMENT tablosu güncellenir.
  - İlgili departmanın TOTAL_SAL değeri aşağıdaki şekilde değiştirilir:
    - NEW.Total_sal = OLD.Total_sal + NEW.Salary
    - WHERE koşulu: Dno = NEW.Dno

Örnek Trigger

EMPLOYEE (FNAME, SSN, SALARY, DNO, SUPERVİSOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER _SSN)

T2: Total_sal2 – Çalışanın Maaşı Değiştiğinde

- Bir çalışanın maaşı güncellendiğinde, ilgili departmanın toplam maaşı da bu değişikliğe göre ayarlanır.

T3: Çalışan Silindiğinde

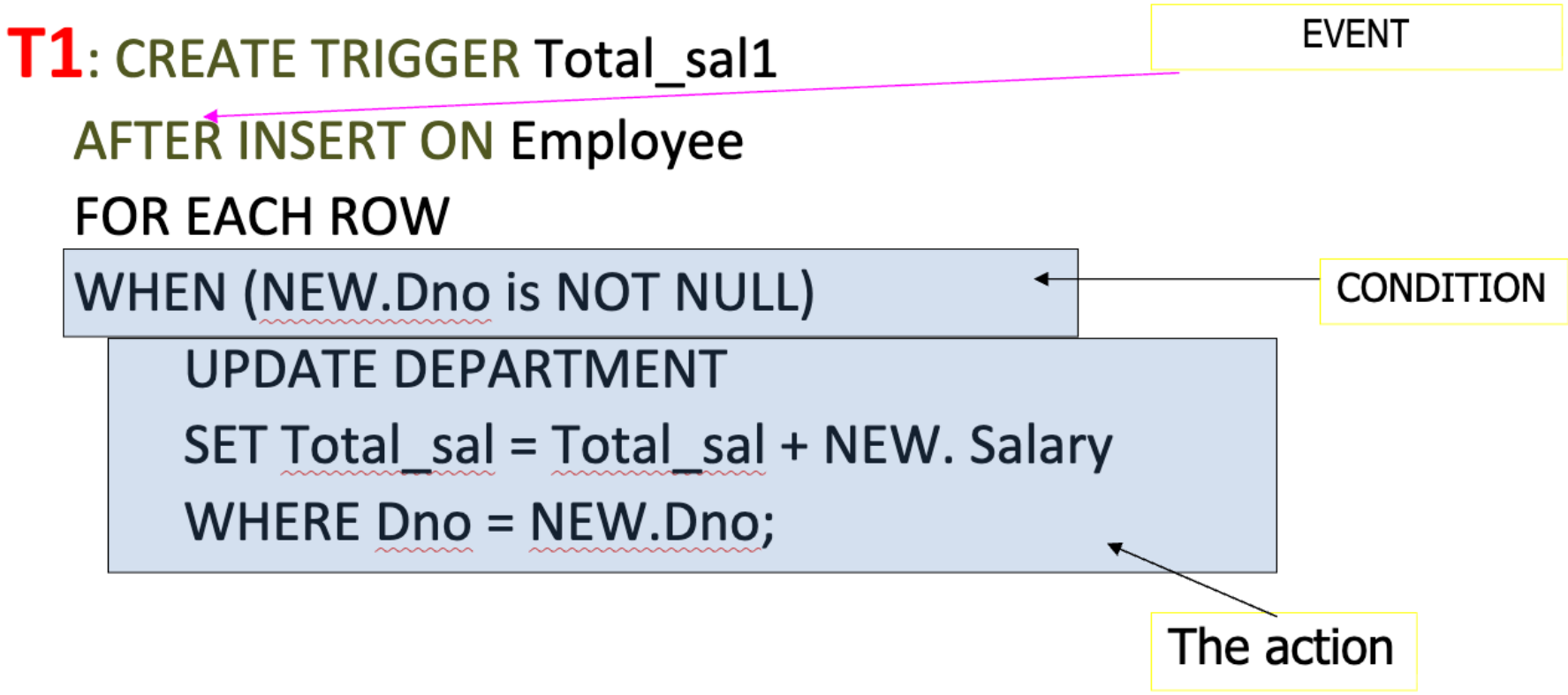
- Bir çalışan sistemden silindiğinde, o kişinin maaşı departmanın toplam maaşından çıkarılır.

T4: Çalışanın Departmanı Değiştirdiğinde

- Bir çalışan farklı bir departmana geçtiğinde:
  - Eski departmanın toplam maaşından bu çalışanın maaşı çıkarılır.
  - Yeni departmanın toplam maaşına bu maaş eklenir.

EMPLOYEE (NAME, SSN, SALARY, DNO, SUPERVİSOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER _SSN)

T1'in Uygulanması: «Yeni Bir Çalışan Ekleme»



EMPLOYEE (NAME, SSN, SALARY, DNO, SUPERVİSOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER _SSN)

T2'nin Uygulanması: «Mevcut Çalışanların Maaşlarının Değiştirilmesi»

```
T2: CREATE TRIGGER Total_sal2
    AFTER UPDATE OF Salary ON EMPLOYEE
    REFERENCING OLD ROW AS O, NEW ROW AS N
    FOR EACH ROW
    WHEN  ( N.Dno IS NOT NULL )
    UPDATE DEPARTMENT
    SET Total_sal = Total_sal + N.salary – O.salary
    WHERE Dno = N.Dno;
```

```
T2: CREATE TRIGGER Total_sal2
    AFTER UPDATE OF Salary ON EMPLOYEE
    REFERENCING OLD TABLE AS O, NEW TABLE AS N
    FOR EACH STATEMENT
    WHEN  EXISTS  ( SELECT * FROM N WHERE N.Dno IS NOT NULL ) OR
          EXISTS  ( SELECT * FROM O WHERE O.Dno IS NOT NULL )
    UPDATE DEPARTMENT AS D
    SET D.Total_sal = D.Total_sal
    + ( SELECT SUM (N.Salary) FROM N WHERE D.Dno=N.Dno )
    – ( SELECT SUM (O.Salary) FROM O WHERE D.Dno=O.Dno )
    WHERE Dno IN ( ( SELECT Dno FROM N ) UNION ( SELECT Dno FROM O ) );
```

EMPLOYEE (NAME, SSN, SALARY, DNO, SUPERVİSOR_SSN)
DEPARTMENT (DNAME, DNO, TOTAL_SAL, MANAGER _SSN)

T2'nin Uygulanması: «Mevcut Çalışanların Maaşlarının Değiştirilmesi»

	Salary	DNO
	200	1
	300	2

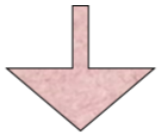
EVENT: ONLY «SAL» changes !

Old O

	Salary	DNO
	200	1
	300	2

New N

	Salary	DNO
	700	1
	900	2



ACTION !

Trigger ile Loglama

Log Bilgisi	PostgreSQL Komutu / Fonksiyonu
Kim tarafından?	`CURRENT_USER` veya `SESSION_USER`
Ne zaman?	`CURRENT_TIMESTAMP`
Hangi bilgisayardan?	`inet_client_addr()`
Hangi kullanıcı ile?	`current_user`
Hangi program ile?	`current_setting('application_name')`
Hangi SQL cümlesi?	`current_query()` (PG 13 öncesi), `pg_stat_activity.query` (alternatif)
Önceki değerler neydi?	Trigger içinde `OLD.*`
Sonraki değerler neydi?	Trigger içinde `NEW.*` (UPDATE için)

Güvenlik

- **Authentication (Kimlik doğrulama),** kullanıcının iddia ettiği kişi olduğunun ve DBMS'nin de kullanılmak istenen sistem olduğunun doğrulanması.
  - Kimlik doğrulama, belirli görevler için herhangi bir ayrıcalık/yetki vermez. **Authorization (Yetkilendirme)** için bir önkoşuldur.

- **Authorization (yetkilendirme);** hangi kullanıcının, hangi veriler üzerinde, neler yapabileceğinin, belirlenmesi.

- **Authentication «SERVER», Authorization «SERVER», Auditing «SERVER», …**

Authorization Methods (Yetkilendirme Metodları)

Veri tabanlarında kullanıcıların hangi kaynaklara erişebileceğini belirleyen iki temel yetkilendirme modeli vardır:

1. Discretionary Access Control (DAC – Takdire Dayalı Erişim Kontrolü)

- Yetkiler tablolar veya diğer veri tabanı nesneleri ile ilişkilendirilir.
- Sistemde kullanıcıların güvenilir olduğu varsayılır.
- Bir kullanıcı sahip olduğu yetkileri başka bir kullanıcıya devredebilir (propagate).
- SQL tarafından doğal olarak desteklenir ve oldukça esnektir.

Avantajları

- Esnek ve kullanımı kolaydır.
- GRANT/REVOKE yapısı ile hızlı yetki yönetimi sağlar.

Dezavantajları

- Yetkiler kolayca devredildiği için güvenlik açıkları oluşabilir.
- Kullanıcı hatalarına veya kötü niyetli kullanıma karşı daha zayıftır.

Authorization Methods (Yetkilendirme Metodları)

Veri tabanlarında kullanıcıların hangi kaynaklara erişebileceğini belirleyen iki temel yetkilendirme modeli vardır:

2. Mandatory Access Control (MAC – Zorlayıcı Erişim Kontrolü)

- Yetkiler veri ile ilişkilendirilir (ör. sınıflandırma seviyeleri: gizli, çok gizli vb.).
- DAC modelinde olan açıkları kapatır; daha sıkı kontrol sağlar.
- Genellikle yüksek güvenlik gerektiren sistemlerde kullanılır. (ör. ordu, devlet kurumları).
- Esnekliği daha düşüktür, kurallar katıdır.

Avantajları

- Daha güvenlidir.
- Kullanıcılar yetkileri birbirine aktaramaz → veri sızıntısı engellenir.

Dezavantajları

- Yönetimi zordur, bürokratik kurallar gerektirir.
- Esneklik azdır. SQL'in içinde "gizlilik seviyesi ata", "erişim sınıfı tanımla" gibi hazır komutlar yoktur.

Authorization Methods (Yetkilendirme Metodları)

Özellik	DAC (Takdire Dayalı)	MAC (Zorlayıcı)
Yetki türü	Tabloya bağlı	Veriye bağlı
Güvenlik seviyesi	Orta	Yüksek
Esneklik	Yüksek	Düşük
Yetki devri	Var	Yok
Kullanım alanı	Genel sistemler	Askeri / devlet
SQL desteği	Doğal olarak desteklenir	Ek mekanizmalar gerekir

Company DB – Örnek Yetkiler

EMPLOYEE(Fname, Minit, Lname, Ssn, Bdate, Address, Sex, Salary, SuperSsn, Dno)
DEPARTMENT(Dname, Dnumber, MgrSsn, MgrStartDate)
DEPT_LOCATIONS(Dnumber, Dlocation)
PROJECT(Pname, Pnumber, Plocation, Dnum)
WORKS_ON(Essn, Pno, Hours)
DEPENDENT(Essn, Dependent_name, Sex, Bdate, Relationship)

```
grant select on EMPLOYEE to hr_manager, accountant
grant insert on EMPLOYEE to hr_manager
grant delete on EMPLOYEE to hr_manager
grant update on EMPLOYEE to hr_manager, accountant

grant select on DEPARTMENT to public
grant insert on DEPARTMENT to hr_manager
grant delete on DEPARTMENT to hr_manager
grant update on DEPARTMENT to hr_manager
```

```
grant select on DEPT_LOCATIONS to public
grant insert on DEPT_LOCATIONS to hr_manager
grant delete on DEPT_LOCATIONS to hr_manager
grant update on DEPT_LOCATIONS to hr_manager

grant select on PROJECT to public
grant insert on PROJECT to project_manager
grant delete on PROJECT to project_manager
grant update on PROJECT to project_manager

grant select on WORKS_ON to hr_manager, project_manager
grant insert on WORKS_ON to project_manager
grant delete on WORKS_ON to project_manager
grant update on WORKS_ON to hr_manager, project_manager
```

Fine-tuning: Column Privileges (Sütun Bazlı Yetkiler)

- Normalde bir tabloya yetki verildiğinde (ör: GRANT SELECT ON ENROLL TO dean), kullanıcı tablodaki tüm sütunları görebilir.
- Ama bazı bilgiler gizli olmalıdır (ör. öğrencinin notu, maaş, sağlık bilgisi).
- Bu yüzden "all or nothing" yerine, sütun düzeyinde izin vermek daha güvenlidir.

Sütun Bazlı Yetkilendirme Neden Kullanılır?

- Hassas bilgileri saklamak için.
- Kullanıcılara sadece ihtiyaç duydukları sütunları göstermek için.
- Daha esnek ve güvenli bir erişim modeli sağlar.

Fine-tuning: Column Privileges (Sütun Bazlı Yetkiler)

```
GRANT SELECT ON EMPLOYEE TO analyst_user;
```

Belirli sütunları seçme izni

```
GRANT SELECT(Fname, Lname, Salary) ON EMPLOYEE TO analyst_user;
```

Sadece Salary sütununu güncelleme

```
GRANT UPDATE(Salary) ON EMPLOYEE TO hr_user;
```

Sadece Fname ve Lname ekleme

```
GRANT INSERT(Fname, Lname) ON EMPLOYEE TO analyst_user;
```

SQL İfadelerinin Gerektirdiği Ayrıcalıklar

- **Select**, sorguda belirtilen her alan için "seçme ayrıcalığı" gerektirir.
- **Insert (update)**, alanın eklenmesi (değiştirilmesi) için "Ekleme (güncelleme) ayrıcalığı" ve where ifadesinde belirtilen her alan için "seçme ayrıcalığı" gerektirir.
- **Delete**, alanın silinmesi için "silme ayrıcalığı" ve where ifadesinde belirtilen her alan için "seçme ayrıcalığı" gerektirir.

```
GRANT UPDATE(Grade) ON ENROLL TO professor;
```

```
UPDATE ENROLL SET Grade = Grade + 1
```

❌

```
UPDATE ENROLL SET Grade = 100
```

✅

Mandatory Access Control (Zorunlu Erişim Kontrolü)

- Discretionary access control, kullanıcıların güvenilir olduğunu varsayar;
  - Kötü niyetli bir profesör, kendi oluşturduğu tabloya gizli notları kopyalayıp herkese SELECT yetkisi vererek gizliliği kolayca ihlal edebilir.

- **Mandatory access control** ise yetkileri kullanıcılara değil doğrudan verinin kendisine atadığı için, özel veriler başka tablolara kopyalansa bile gizlilik korunur;
  - Bu model sınıflandırma seviyeleri ile uygulanır ve böylece hassas bilgilere erişim her durumda zorunlu kurallarla kontrol altında tutulur.
  - Güvenlik veritabanı yazılımları ve isteğe bağlı ek SQL araçlarıyla oluşturulabilir.

YILDIZ TEKNİK ÜNİVERSİTESİ

Yasemin Topuz

Yıldız Teknik Üniversitesi

ytopuz@yildiz.edu.tr