İÇİNDEKİLER TABLOSU

1. VERİTABANI TEMELLERİ

Video Linki = https://www.youtube.com/watch?v=-QUn2lFFTAA&t=12s

1.1. VERİ NEDİR?

Veri, belirli bir bağlamda anlam taşıyan, işlenebilir ve analiz edilebilir bilgilerdir. Veri, sayılar, metinler, tarih ve zaman, görüntüler gibi çeşitli biçimlerde olabilir. Örneğin, bir müşteri kaydı, müşteri adı, adresi ve telefon numarasını içeren verilerden oluşur.

1.2. VERİTABANI NEDİR?

Veritabanı, verilerin düzenli bir şekilde saklandığı, yönetildiği ve erişildiği bir sistemdir. Veritabanları, verilerin depolanması, güncellenmesi ve sorgulanması için kullanılan yazılımlar ve yapılar içerir. Veritabanları, genellikle bir veritabanı yönetim sistemi (DBMS) tarafından yönetilir.

1.3. VERİTABANI TÜRLERİ

Veritabanları, çeşitli türlerde sınıflandırılabilir:

1. **İlişkisel Veritabanları:** Verileri tablolar halinde saklar ve tablolar arasındaki ilişkileri tanımlar. Örnek: PostgreSQL, MySQL.
2. **NoSQL Veritabanları:** Yapılandırılmamış veya yarı yapılandırılmış verileri saklamak için kullanılır. Örnek: MongoDB, Cassandra.
3. **Hiyerarşik Veritabanları:** Verileri ağaç yapısında saklar. Örnek: IBM Information Management System (IMS).
4. **Ağ Veritabanları:** Verileri grafik yapısında saklar ve daha karmaşık ilişkileri destekler.

1.4. İLİŞKİSEL VERİTABANLARINA GİRİŞ

İlişkisel veritabanları, verileri tablolar halinde düzenler. Her tablo, satırlar (kayıtlar) ve sütunlar (alanlar) içerir. Tablolar arasında ilişkiler kurarak verilerin entegrasyonunu sağlar. Örneğin, bir müşteri tablosu ve bir sipariş tablosu arasında bir ilişki kurulabilir.

```
CREATE TABLE customer (
customer_id SERIAL PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL
);
CREATE TABLE orders (
order_id SERIAL PRIMARY KEY,
customer_id INTEGER REFERENCES customer(customer_id) ON DELETE CASCADE,
order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1.5. İLİŞKİSEL VERİTABANLARINI KARŞILAŞTIRILMASI

İlişkisel veritabanları, verilerin tutarlılığını sağlamak için ACID (Atomicity, Consistency, Isolation, Durability) özelliklerini destekler. Diğer veritabanı türleri, genellikle daha esnek veri yapıları sunar, ancak ACID garantileri sağlamayabilir.

ACID, bir veritabanı işleminin ya tamamen başarılı olması ya da hiç gerçekleşmemesi (atomiklik), veritabanının her zaman tutarlı bir durumda kalması (tutarlılık), işlemlerin birbirinden bağımsız olarak çalışması (izolasyon) ve işlemlerin sonuçlarının kalıcı olması (dayanıklılık) gerektiğini belirten bir kavramdır.

1.6. VERİ TÜRLERİ NELERDİR?.

Veri türleri, veritabanında saklanacak verilerin türünü tanımlar. PostgreSQL'de yaygın veri türleri şunlardır:

- **String Veri Türleri:** CHAR, VARCHAR, TEXT
- **Sayısal Veri Türleri:** INTEGER, FLOAT, NUMERIC
- **Tarih-Zaman Veri Türleri:** DATE, TIME, TIMESTAMP
- **Boolean Veri Türü**: BOOLEAN

```
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100),
    price NUMERIC(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

1.7. VERİTABANI MODELLERİNİN KARŞILAŞTIRILMASI

Veritabanı modelleri, verilerin nasıl yapılandırılacağını ve saklanacağını belirler. İlişkisel veritabanları, verileri tablolar halinde düzenlerken, NoSQL veritabanları daha esnek yapılar sunar. Aşağıda bazı temel farklar bulunmaktadır:

Özellik	İlişkisel Veritabanları	NoSQL Veritabanları
Veri Yapısı	Tablolar	Belge, anahtar-değer, grafik
İlişkiler	Tanımlı	Esnek
ACID Garantileri	Var	Genellikle yok
Ölçeklenebilirlik	Dikey ölçeklenebilir	Yatay ölçeklenebilir

1.8. SQL-NOSQL VERİTABANLARI (DEVAM)

- **SQL Veritabanları**

SQL (Structured Query Language), ilişkisel veritabanları ile etkileşimde bulunmak için kullanılan bir sorgu dilidir. SQL veritabanları, verileri tablolar halinde düzenler ve bu tablolardaki veriler arasında ilişkiler kurar. SQL veritabanlarının bazı özellikleri şunlardır:

- **ACID Özellikleri:** Atomicity, Consistency, Isolation, Durability garantileri sunar.
- **Veri Bütünlüğü:** Veri tutarlılığını sağlamak için kısıtlamalar (örneğin, PRIMARY KEY, FOREIGN KEY, UNIQUE) kullanılır.
- **Sorgulama**: SQL kullanarak karmaşık sorgular yazmak mümkündür.

```
SELECT c.first_name, c.last_name, r.rental_date, f.title
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
WHERE r.rental_date > '2001-01-01';
```

- **NoSQL Veritabanları**

NoSQL (Not Only SQL), yapılandırılmamış veya yarı yapılandırılmış verileri saklamak için kullanılan bir veritabanı türüdür. NoSQL veritabanları, genellikle daha esnek veri yapıları sunar ve büyük veri uygulamaları için uygundur. NoSQL veritabanlarının bazı özellikleri şunlardır:

- **Esneklik:** Veri yapıları, uygulama gereksinimlerine göre değiştirilebilir.
- **Yüksek Ölçeklenebilirlik:** Yatay ölçeklenebilirlik sunar, yani daha fazla sunucu ekleyerek kapasite artırılabilir.
- **Farklı Veri Modelleri:** Belge tabanlı (MongoDB), anahtar-değer (Redis), sütun tabanlı (Cassandra) ve grafik tabanlı (Neo4j) gibi farklı veri modelleri vardır.

MongoDB Örneği: MongoDB'de bir belge oluşturma:

```
db.customers.insert({
  first_name: "John",
  last_name: "Doe",
  orders: [
    { order_id: 1, order_date: new Date("2023-01-15") },
    { order_id: 2, order_date: new Date("2023-02-20") }
  ]
});
```

1.9. EĞİTİMDE KULLANILACAK VERİTABANI: POSTGRESQL

PostgreSQL, açık kaynaklı bir ilişkisel veritabanı yönetim sistemidir. Güçlü özellikleri, geniş veri türü desteği ve esnekliği ile bilinir. Eğitimde kullanılacak veritabanı olarak PostgreSQL'in avantajları şunlardır:

- **Açık Kaynak:** Ücretsizdir ve geniş bir topluluk desteği vardır.
- **Gelişmiş Veri Türleri**: JSON, XML, ve dizi gibi karmaşık veri türlerini destekler.
- **Güçlü Sorgulama Dili:** SQL standardına uygun bir sorgulama dili sunar.
- **Genişletilebilirlik:** Kullanıcı tanımlı fonksiyonlar ve veri türleri oluşturma imkanı sunar.

PostgreSQL ile Temel Örnekler:

- **Tablo Oluşturma**:

```
CREATE TABLE students (
  student_id SERIAL PRIMARY KEY,
```

```
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    enrollment_date DATE
);
```

- **Veri Ekleme:**

```
INSERT INTO customers (first_name, last_name, create_date)
VALUES ('Alice', 'Smith', '2023-09-01');
```

- **Veri Sorgulama:**

```
SELECT * FROM customers WHERE create_date > '2023-01-01';
```

- **Veri Güncelleme:**

```
UPDATE customers
SET last_name = 'Johnson'
WHERE customer_id = 1;
```

- **Veri Silme:**

```
DELETE FROM customers WHERE customer_id = 1;
```

1.10. SONUÇ

Veri, veritabanları ve veritabanı türleri, modern yazılım geliştirme ve veri yönetimi için kritik öneme sahiptir. İlişkisel veritabanları, verilerin düzenli bir şekilde saklanmasını ve yönetilmesini sağlarken, NoSQL veritabanları daha esnek ve ölçeklenebilir çözümler sunar. PostgreSQL, eğitimde kullanılabilecek güçlü bir veritabanı yönetim sistemidir ve geniş veri türü desteği ile kullanıcıların ihtiyaçlarına uygun çözümler sunar.

2. SQL'E GİRİŞ VE TEMEL KOMUTLAR

Video Linki = https://www.youtube.com/watch?v=nryDf_hVO6s&t=3082s

2.1. SQL NEDİR?

SQL, veritabanları ile etkileşimde bulunmak için kullanılan bir dildir. Veritabanı yönetim sistemleri (DBMS) ile veri sorgulama, güncelleme, ekleme ve silme işlemleri yapmak için kullanılır. SQL, ilişkisel veritabanları için standart bir dildir.

2.2. SQL YAZMA KURALLARI

SQL yazarken dikkat edilmesi gereken bazı kurallar vardır:

- SQL komutları genellikle büyük harfle yazılır (örneğin, **SELECT, FROM, WHERE**), ancak küçük harfle yazılması da mümkündür.
- SQL ifadeleri genellikle noktalı virgül **(;)** ile sonlandırılır.
- Alan adları ve tablo adları genellikle alıntı işareti **(")** ile çevrilebilir.

2.3. KOLONLARI SEÇME

SQL ile belirli kolonları seçmek için **SELECT** ifadesi kullanılır. Tüm kolonları veya belirli kolonları seçebilirsiniz.

2.4. SQL İLE NELER YAPABİLİRİZ?

SQL ile aşağıdaki işlemleri gerçekleştirebiliriz:

- Veritabanı oluşturma ve yönetme
- Tablo oluşturma, güncelleme ve silme
- Veri ekleme, güncelleme ve silme
- Veri sorgulama ve raporlama
- Veritabanı güvenliği ve erişim kontrolü

2.5. SQL KOMUTLARI SINIFLANDIRMASI

SQL komutları genellikle dört ana kategoriye ayrılır:

- **DML (Data Manipulation Language):** Veri manipülasyonu için kullanılır (örneğin, **SELECT, INSERT, UPDATE, DELETE**).
- **DDL (Data Definition Language):** Veritabanı yapısını tanımlamak için kullanılır (örneğin, **CREATE, ALTER, DROP**).
- **DCL (Data Control Language):** Erişim kontrolü için kullanılır (örneğin, **GRANT, REVOKE**).
- **TCL (Transaction Control Language):** Transaction yönetimi için kullanılır (örneğin, **COMMIT, ROLLBACK**).

2.6. SELECT İFADESİ - SYNTAX

SELECT ifadesi, veritabanından veri almak için kullanılır. Temel sözdizimi şu şekildedir:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

2.7. TÜM KOLONLARI SEÇME

Tüm kolonları seçmek için * kullanılır.

```
SELECT * FROM customer;
```

Bu sorgu, **customer** tablosundaki tüm kayıtları ve tüm kolonları döndürür.

2.8. BELİRLİ KOLONLARI SEÇME

Belirli kolonları seçmek için kolon adlarını virgülle ayırarak yazabilirsiniz.

```
SELECT first_name, last_name  FROM customer;
```

Bu sorgu, **customer** tablosundaki sadece **first_name ve last_name** kolonlarını döndürür.

2.9. ARİTMETİK İFADELERİ SORGU İÇERİSİNDE KULLANMA

SQL'de aritmetik ifadeleri kullanarak hesaplamalar yapabilirsiniz.

```
SELECT customer_id,amount, amount*1.1 AS new_amount FROM payment;
```

Bu sorgu, **payment** tablosundaki **amount** kolonundaki her verinin miktarını %10 artırarak yeni miktarları **new_amount** kolonunda döndürür.

2.10. BİRLEŞTİRME (CONCATENATİON) OPERATÖRÜ

PostgreSQL'de metin birleştirmek için || operatörünü kullanabilirsiniz.

```
SELECT first_name || ' ' || last_name AS full_name
FROM customer;
```

Bu sorgu, her müşterinin adını ve soyadını birleştirerek **full_name** olarak döndürür.

2.11. KOLONLARA ALİAS VERME

Alias, bir kolonun veya tablonun geçici adını belirlemek için kullanılır. **AS** anahtar kelimesi ile tanımlanır.

```
SELECT first_name AS "Ad", last_name AS "Soyad"
FROM customer;
```

Bu sorgu, **first_name** kolonunu "Ad" ve **last_name** kolonunu "Soyad" olarak döndürür.

2.12. ORDER BY KOMUTU İLE SATIRLARI SIRALAMA

ORDER BY ifadesi, sorgu sonuçlarını belirli bir kolona göre sıralamak için kullanılır.

```
SELECT * FROM payment
ORDER BY amount DESC;
```

Bu sorgu, **payment** tablosundaki kayıtları miktara göre azalan sırada sıralar.

```
SELECT * FROM payment
```

ORDER BY amount **ASC**;

Bu sorgu, **payment** tablosundaki kayıtları miktara göre artan sırada sıralar.**(!!!!Artan sıra için ASC yazmamıza gerek yok. ORDER BY default değeri ASC 'dir.)**

2.13. DISTINCT KOMUTU

DISTINCT ifadesi, sorgu sonuçlarında tekrarlanan kayıtları kaldırmak için kullanılır. Bu, belirli bir kolonun benzersiz değerlerini almak için yararlıdır.

SELECT DISTINCT staff_id **FROM** payment;

Bu sorgu, **payment** tablosundaki benzersiz **staff_id** değerlerini döndürür. Yani, her personel ID'si yalnızca bir kez gösterilecektir.

2.14. ÖZET

PostgreSQL'de SQL kullanarak veritabanı ile etkileşimde bulunmak için çeşitli komutlar ve ifadeler mevcuttur. Aşağıda özetlenen konular, SQL'in temel işlevlerini anlamanıza yardımcı olacaktır:

1. **SQL Nedir?:** Veritabanları ile etkileşimde bulunmak için kullanılan bir dil.
2. **SQL Yazma Kuralları:** SQL komutlarının yazım kuralları.
3. **Kolonları Seçme:** SELECT ifadesi ile veritabanından kolonları seçme.
4. **SQL ile Neler Yapabiliriz?:** Veritabanı yönetimi ve veri manipülasyonu.
5. **SQL Komutları Sınıflandırması**: DML, DDL, DCL ve TCL komutları.
6. **Select İfadesi - Syntax:** Temel SELECT sözdizimi.
7. **Tüm Kolonları Seçme:** SELECT * ile tüm kolonları alma.
8. **Belirli Kolonları Seçme:** Belirli kolonları seçme.
9. **Aritmetik İfadeleri Sorgu İçerisinde Kullanma:** Hesaplamalar yapma.
10. **Birleştirme (Concatenation) Operatörü:** Metin birleştirme.
11. **Kolonlara Alias Verme:** Kolonlara geçici adlar verme.
12. **Order By Komutu İle Satırları Sıralama:** Sonuçları sıralama.
13. **DISTINCT Komutu:** Tekrar eden kayıtları kaldırma.

3. SQL SATIR VE VERİ OPERATÖRLERİ

Video Linki = https://www.youtube.com/watch?v=deNhiFLynRo&t=3830s

3.1. SATIRLARI LİMİTLEME - WHERE İFADESİ

WHERE ifadesi, SQL sorgularında belirli koşullara uyan satırları filtrelemek için kullanılır. Bu, sorgunun yalnızca belirli bir koşulu karşılayan kayıtları döndürmesini sağlar.

SELECT * **FROM** payment

WHERE amount > 11;

Bu sorgu, miktarı 11'den büyük olan tüm ödemeleri döndürür.

3.2. OPERATÖR LİSTESİ

PostgreSQL'de çeşitli operatörler kullanarak verileri filtreleyebiliriz. Bu operatörler, karşılaştırma, mantıksal, aralık ve benzeri işlemler için kullanılır.

3.3. KARŞILAŞTIRMA OPERATÖRLERİ

Karşılaştırma operatörleri, iki değeri karşılaştırmak için kullanılır. En yaygın karşılaştırma operatörleri şunlardır:

=: Eşittir

!= veya <>: Eşit değildir

<: Küçüktür

>: Büyüktür

<=: Küçük veya eşittir

>=: Büyük veya eşittir

SELECT * **FROM** film

WHERE length >= 180

Bu sorgu, film uzunluğunun 180 veya daha fazla olan film tablosundaki tüm değerleri döndürür.

3.4. MANTIKSAL OPERATÖRLER

Mantıksal operatörler, birden fazla koşulu birleştirmek için kullanılır. En yaygın mantıksal operatörler şunlardır:

AND: Her iki koşulun da doğru olması gerekir.

OR: En az bir koşulun doğru olması yeterlidir.

NOT: Koşulu tersine çevirir.

```
SELECT * FROM film WHERE length >= 180 AND rating = 'R'
```

Bu sorgu, film uzunluğu 180'den büyük ve izlenme oranı 'R' olan film tablosundaki tüm değerleri döndürür.

3.5. BETWEEN OPERATÖRÜ

BETWEEN operatörü, bir değerin belirli bir aralıkta olup olmadığını kontrol etmek için kullanılır.

```
SELECT * FROM film WHERE length BETWEEN 80 AND 90;
```

Bu sorgu, film uzunluğu 80 ile 90 arasında olan tüm film tablosundaki tüm değerleri döndürür.

3.6. IN OPERATÖRÜ

IN operatörü, bir değerin belirli bir değer kümesinde olup olmadığını kontrol etmek için kullanılır.

```
SELECT * FROM customer
WHERE customer_id IN (1, 2, 3);
```

Bu sorgu, customer ID'si 1, 2 veya 3 olan tüm müşterileri döndürür.

3.7. LIKE OPERATÖRÜ

LIKE operatörü, bir metin değerinin belirli bir desene uyup uymadığını kontrol etmek için kullanılır. **%** karakteri, sıfır veya daha fazla karakteri temsil ederken, **_** karakteri tek bir karakteri temsil eder.

```
SELECT * FROM customer
WHERE first_name LIKE 'A%';
```

Bu sorgu, adı 'A' harfi ile başlayan tüm müşterileri döndürür.

3.8. NOT OPERATÖRÜ

NOT operatörü, bir koşulun tersini kontrol etmek için kullanılır.

```
SELECT * FROM customer
WHERE NOT store_id = 1;
```

Bu sorgu, store ID'si 1 olmayan tüm müşterileri döndürür.

3.9. NULL DEĞERİ

NULL, bir değerin mevcut olmadığını veya bilinmediğini belirtir. **NULL** değerleri ile çalışırken dikkatli olunmalıdır.

3.10. IS NULL OPERATÖRÜ

IS NULL operatörü, bir değerin NULL olup olmadığını kontrol etmek için kullanılır.

```
SELECT * FROM film
WHERE title IS NULL;
```

Bu sorgu, film başlıkları olmayan tüm filmleri döndürür.(!!Sonuç boş dönecektir. Tüm filmlerin başlığı vardır.!!)

3.11. LIMIT KOMUTU İLE SATIRLARI SINIRLAMA

LIMIT komutu, sorgunun döndüreceği satır sayısını sınırlamak için kullanılır. Bu, büyük veri setleri ile çalışırken performansı artırmak ve yalnızca belirli sayıda kayıt almak için yararlıdır.

```
SELECT * FROM film

ORDER BY film_id DESC

LIMIT 5;
```

Bu sorgu, film ID'ye göre azalan sırada sıralanmış **film** tablosundaki en son 5 filmi döndürür.

3.12. OFFSET İLE BİRLİKTE LIMIT KULLANIMI

LIMIT komutunu **OFFSET** ile birleştirerek, belirli bir başlangıç noktasından itibaren kayıtları alabilirsiniz. Bu, sayfalama (pagination) işlemleri için kullanışlıdır.

```
SELECT * FROM film

ORDER BY film_id DESC

LIMIT 5 OFFSET 10;
```

Bu sorgu, **film ID'ye** göre azalan sırada sıralanmış **film** tablosundaki 986. ile 990. arasındaki (toplamda 5 kayıt) filmi döndürür. Yani, ilk 10 kaydı atlayarak sonraki 5 kaydı alır.

3.13. ÖZET

PostgreSQL'de satırları limitleme ve filtreleme işlemleri, veritabanı sorgularında önemli bir rol oynar. Aşağıda özetlenen operatörler ve ifadeler, verileri daha etkili bir şekilde yönetmek için kullanılır:

- **WHERE İfadesi:** Belirli koşullara uyan satırları filtreler.
- **Karşılaştırma Operatörleri**: Eşitlik ve büyüklük karşılaştırmaları yapar.
- **Mantıksal Operatörler**: Birden fazla koşulu birleştirir.
- **BETWEEN Operatörü:** Bir değerin belirli bir aralıkta olup olmadığını kontrol eder.
- **IN Operatörü:** Bir değerin belirli bir değer kümesinde olup olmadığını kontrol eder.
- **LIKE Operatörü**: Metin değerlerinin belirli bir desene uyup uymadığını kontrol eder.
- **NOT Operatörü**: Koşulun tersini kontrol eder.
- **IS NULL Operatörü:** NULL değerleri kontrol eder.
- **LIMIT Komutu**: Sorgunun döndüreceği satır sayısını sınırlar.
- **OFFSET ile LIMIT**: Belirli bir başlangıç noktasından itibaren kayıtları alır.

Bu operatörler ve ifadeler, PostgreSQL'de veri sorgulama ve yönetme işlemlerini daha etkili hale getirir.

4. SQL FONKSİYONLARI

Video Linki = https://www.youtube.com/watch?v=m31WqHvVrSQ&t=3s

4.1. SQL FONKSİYONLARI NEDİR?

SQL fonksiyonları, veritabanı sorgularında veri işleme ve manipülasyon işlemleri için kullanılan yerleşik işlevlerdir. Bu fonksiyonlar, verileri dönüştürmek, hesaplamak veya belirli bir koşula göre işlem yapmak için kullanılır.

4.2. STRİNG FONKSİYONLARI: BÜYÜK-KÜÇÜK HARF

String fonksiyonları, metin verileri üzerinde işlem yapmamıza olanak tanır. Büyük-küçük harf dönüşümleri için kullanılan fonksiyonlar:

- **UPPER**: Metni büyük harfe çevirir.
- **LOWER**: Metni küçük harfe çevirir.

```
SELECT UPPER(first_name) AS UppercaseName, LOWER(last_name) AS LowercaseName

FROM customer;
```

Bu sorgu, müşterilerin adlarını büyük harfle ve soyadlarını küçük harfle döndürür.

4.3. STRİNG FONKSİYONLARI: KARAKTER İŞLEME

Karakter işleme fonksiyonları, metin verileri üzerinde daha karmaşık işlemler yapmamıza olanak tanır.

- **SUBSTR**: Metnin belirli bir kısmını alır.
- **LENGTH**: Metnin uzunluğunu döndürür.
- **TRIM**: Metnin başındaki ve sonundaki boşlukları kaldırır.

```
SELECT first_name,SUBSTRING(first_name FROM 1 FOR 3) AS short_name,last_name, LENGTH(last_name) AS
last_name_length

FROM customer;
```

Bu sorgu, müşterilerin adlarının ilk üç karakterini ve soyadlarının uzunluğunu döndürür.

4.4. MATEMATİK FONKSİYONLARI

Matematik fonksiyonları, sayısal veriler üzerinde hesaplamalar yapmamıza olanak tanır.

- **ROUND**: Sayıyı belirli bir ondalık basamağa yuvarlar.
- **FLOOR**: Sayıyı aşağıya yuvarlar.
- **CEIL**: Sayıyı yukarıya yuvarlar.

```
SELECT ROUND(amount, 2) AS RoundedSalary, FLOOR(amount) AS FloorSalary

FROM payment
```

Bu sorgu, ödem miktarlarını iki ondalık basamağa yuvarlar ve aşağıya yuvarlanmış değerlerini döndürür.

4.5. TARİH FONKSİYONLARI

Tarih fonksiyonları, tarih ve saat verileri üzerinde işlem yapmamıza olanak tanır.

- **CURRENT_DATE:** Geçerli tarihi döndürür.
- **CURRENT_TIMESTAMP**: Geçerli tarih ve saati döndürür.
- **AGE**: İki tarih arasındaki farkı döndürür.

```
SELECT CURRENT_DATE AS current_date, CURRENT_TIMESTAMP AS current_timestamp,

        AGE('2023-10-01'::date, CURRENT_DATE) AS age_difference;
```

Bu sorgu, geçerli tarihi ve üç ay sonrasını döndürür.

4.6. DÖNÜŞÜM FONKSİYONLARI

Dönüşüm fonksiyonları, bir veri türünü başka bir veri türüne dönüştürmek için kullanılır.

TO_DATE: String veriyi tarih formatına dönüştürür.

TO_TIMESTAMP: String veriyi zaman damgası formatına dönüştürür.

TO_NUMBER: String veriyi sayısal formata dönüştürür.

TO_CHAR: Tarih veya sayıyı string formata dönüştürür.

```
SELECT TO_DATE('2023-10-01', 'YYYY-MM-DD') AS converted_date,

        TO_CHAR(CURRENT_TIMESTAMP, 'DD-MON-YYYY') AS formatted_date;
```

Bu sorgu, bir string tarihi tarih formatına dönüştürür ve geçerli tarihi belirli bir formatta döndürür.

4.7. DÖNÜŞÜM FONKSİYONLARI: CAST FONKSİYONU

CAST fonksiyonu, bir veri türünü başka bir veri türüne dönüştürmek için kullanılır.

```
SELECT CAST(Amount AS TEXT) AS AmountAsString

FROM payment;
```

Bu sorgu, ödeme miktarlarını string formatına dönüştürür.

4.8. TARİHLER İLE ARİTMETİK İŞLEMLER

Tarih verileri üzerinde aritmetik işlemler yaparak tarihleri toplamak veya çıkarmak mümkündür. SQL'de tarih aritmetiği, tarih değerlerine gün, ay veya yıl eklemek veya çıkarmak için kullanılır.

```
SELECT CURRENT_DATE AS current_date,
```

```
    CURRENT_DATE + INTERVAL '7 days' AS date_after_seven_days,
    CURRENT_DATE - INTERVAL '30 days' AS date_thirty_days_ago;
```

Bu sorgu, geçerli tarihi, 7 gün sonrasını ve 30 gün öncesini döndürür.

4.9. COALESCE FONKSİYONU

COALESCE fonksiyonu, birden fazla değeri alır ve NULL olmayan ilk değeri döndürür. Bu, NULL değerleri kontrol etmek ve varsayılan değerler sağlamak için kullanışlıdır.

```
SELECT COALESCE(address2,'adres yok') FROM address
```

Bu sorgu,adres tablosundaki adres2 kolonunu döndürür; eğer address2 NULL ise "adres yok" ifadesini döndürür.

4.10. NULLIF FONKSİYONU

NULLIF fonksiyonu, iki değeri karşılaştırır ve eğer değerler eşitse NULL döner; aksi takdirde ilk değeri döner. Bu, belirli koşullara göre NULL değerler oluşturmak için kullanılır.

```
SELECT NULLIF(amount, 0.99) AS NonZeroSalary
FROM payment;
```

Bu sorgu, ödeme miktarı 0,99 olan çalışanlar için NULL döndürür; diğerleri için ödeme miktarlarının değerini döndürür.

4.11. CASE İFADESİ

CASE ifadesi, SQL'de koşullu mantık uygulamak için kullanılır. Belirli koşullara göre farklı değerler döndürmek için kullanılır.

```
SELECT title,
    CASE
        WHEN length < 90 THEN 'Low'
        WHEN length BETWEEN 90 AND 160 THEN 'Medium'
        ELSE 'High'
    END AS length_Category
FROM film;
```

Bu sorgu, film uzunluklarını belirli aralıklara göre "Low", "Medium" veya "High" olarak kategorize eder.

4.12. NESTED (İÇ İÇE) FONKSİYONLAR

İç içe fonksiyonlar, bir fonksiyonun içinde başka bir fonksiyon kullanarak daha karmaşık işlemler yapmamıza olanak tanır.

```
SELECT UPPER(SUBSTRING(first_name FROM 1 FOR 1)) || LOWER(SUBSTRING(first_name FROM 2)) AS formatted_name
FROM customer;
```

Bu sorgu, müşterilerin adlarının ilk harfini büyük, geri kalanını küçük harf yaparak döndürür.

5. GRUP FONKSİYONLARI VE GROUP BY

Video Linki = https://www.youtube.com/watch?v=uEY6CRQKo4M&t=1s

5.1. GRUP FONKSİYONLAR LİSTESİ

Grup fonksiyonları, belirli bir grup üzerinde toplu hesaplamalar yapmamıza olanak tanır. En yaygın grup fonksiyonları şunlardır:

AVG: Ortalama hesaplar.

SUM: Toplam hesaplar.

COUNT: Kayıt sayısını döndürür.

MIN: En küçük değeri döndürür.

MAX: En büyük değeri döndürür.

5.2. AVG FONKSİYONU

AVG fonksiyonu, belirli bir sütunun ortalamasını hesaplar.

```
SELECT AVG(amount) AS AverageAmount
FROM payment;
```

Bu sorgu, **payment** tablosundaki ödeme miktarlarını ortalamasını döndürür.

5.3. SUM FONKSİYONU

SUM fonksiyonu, belirli bir sütundaki değerlerin toplamını hesaplar.

```
SELECT SUM(amount) AS TotalAmount
FROM payment;
```

Bu sorgu, **payment** tablosundaki tüm ödeme miktarılarının toplamını döndürür.

5.4. COUNT FONKSİYONU

COUNT fonksiyonu, belirli bir sütundaki kayıt sayısını döndürür. **COUNT(*)** tüm kayıtları sayarken, **COUNT(column_name)** belirli bir sütundaki NULL olmayan değerleri sayar.

```
SELECT COUNT(*) AS TotalPayment
FROM payment;
```

Bu sorgu, **payment** tablosundaki toplam ödeme sayısını döndürür.

5.5. MIN-MAX FONKSİYONLARI

MIN ve **MAX** fonksiyonları, belirli bir sütundaki en küçük ve en büyük değerleri döndürür.

```
SELECT MIN(amount) AS MinimumAmount, MAX(amount) AS MaximumAmount
FROM payment;
```

Bu sorgu, **amount** tablosundaki en düşük ve en yüksek ödeme miktarını döndürür.

5.6. GROUP BY İFADESİ

GROUP BY ifadesi, verileri belirli bir sütuna göre gruplamak için kullanılır. Grup fonksiyonları genellikle **GROUP BY** ile birlikte kullanılır.

```
SELECT customer_id, AVG(amount) AS AverageAmount
FROM payment
GROUP BY customer_id;
```

Bu sorgu, her müşterinin ödeme miktarının ortalamasına döndürür.

5.7. HAVING İFADESİ

HAVING ifadesi, **GROUP BY** ile oluşturulan gruplar üzerinde koşul belirlemek için kullanılır. **HAVING, WHERE** ifadesinden farklı olarak, grup fonksiyonları ile birlikte kullanılabilir.

```
SELECT customer_id, AVG(amount) AS AverageAmount
FROM payment
GROUP BY customer_id
HAVING COUNT(*) > 20;
```

Bu sorgu, 20'den fazla ödeme yapmış müşterilerin ödeme ortalmasını döndürür.

6. BİRDEN FAZLA TABLO ÜZERİNDE SORGULAMA

Video Linki = https://www.youtube.com/watch?v=QgQ7A5EEu24&t=1s

6.1. ER DİYAGRAMLARI

ER diyagramları, veritabanı tasarımında kullanılan bir araçtır. Varlıklar (entities), bu varlıkların özellikleri (attributes) ve varlıklar arasındaki ilişkileri (relationships) gösterir. Örneğin, bir "Müşteri" ve "Sipariş" varlığı arasındaki ilişkiyi gösterebiliriz.

Örnek:

- **Müşteri**: MüşteriID, Ad, Soyad
- **Sipariş**: SiparişID, MüşteriID, Tarih

Müşteri ile Sipariş arasında bir "1-N" ilişkisi vardır; yani bir müşteri birden fazla sipariş verebilir.

6.2. TABLOLARA ALİAS VERME

Alias, SQL sorgularında tabloları veya sütunları daha okunabilir hale getirmek için kullanılan takma adlardır.

```
SELECT c.first_name AS "Ad", c.last_name AS "Soyad"
FROM customer c;
```

Burada **customer** tablosuna **c** alias'ı verilmiştir.

6.3. JOİN TİPLERİ

Join işlemleri, birden fazla tabloyu birleştirerek veri çekmek için kullanılır. Farklı join türleri vardır:

6.4. INNER JOİN

Inner join, iki tablo arasında eşleşen kayıtları getirir.

```
SELECT c.first_name, s.last_update
FROM customer c
INNER JOIN store s ON c.store_id = s.store_id;
```

Bu sorgu, müşterilerin depolarının son güncellenme tarihlerini getirir.

6.5. JOİN İŞLEMİ - USING İFADESİ İLE

USING ifadesi, iki tablo arasında ortak bir sütun adı kullanarak **join** yapar.

```
SELECT c.first_name, s.last_update
FROM customer c
JOIN store s USING (store_id);
```

Bu sorgu, **store_id** sütununu kullanarak iki tabloyu birleştirir.

6.6. JOİN İŞLEMİ - KLASİK YÖNTEM

Klasik yöntem, **join** işlemi için **WHERE** ifadesini kullanır.

```
SELECT c.first_name, s.last_update
FROM customer c, store s
WHERE c.store_id = s.store_id;
```

Bu sorgu, **WHERE** ifadesi ile iki tabloyu birleştirir.

6.7. LEFT JOİN - RİGHT JOİN

- **Left Join**: Sol tablodaki tüm kayıtları ve sağ tablodaki eşleşen kayıtları getirir.
- **Right Join:** Sağ tablodaki tüm kayıtları ve sol tablodaki eşleşen kayıtları getirir.

```
-- Left Join
SELECT c.first_name, s.last_update
FROM customer c
LEFT JOIN store s ON c.store_id = s.store_id;
```

```
-- Right Join

SELECT c.first_name, s.last_update

FROM customer c

RIGHT JOIN store s ON c.store_id = s.store_id;
```

6.8. FULL OUTER JOİN

Full outer join, her iki tablodaki tüm kayıtları getirir; eşleşmeyen kayıtlar için NULL değerler döner.

```
SELECT c.first_name, s.last_update

FROM customer c

FULL OUTER JOIN store s ON c.store_id = s.store_id;
```

6.9. SELF JOİN

Self join, bir tablonun kendisiyle birleştirilmesidir. Genellikle hiyerarşik verilerde kullanılır.

```
SELECT e1.first_name AS "Çalışan", e2.last_name AS "Yönetici"

FROM employees e1

JOIN employees e2 ON e1.manager_id = e2.employee_id;
```

Bu sorgu, çalışanların adlarını ve yöneticilerinin adlarını getirir.

6.10. CROSS JOİN

Cross join, iki tablonun kartesyen çarpımını alır; yani her bir kayıt için diğer tablodaki tüm kayıtları eşleştirir.

```
SELECT e.first_name, d.department_name

FROM employees e

CROSS JOIN departments d;
```

6.11. NATURAL JOİN

NATURAL JOIN, iki tablo arasında ortak olan sütunları kullanarak otomatik birleştirme yapar. Ortak sütunlar, her iki tabloda da aynı isimde olan sütunlardır. Bu tür bir join, genellikle daha az yazım gerektirir, ancak dikkatli kullanılmalıdır çünkü hangi sütunların ortak olduğunu bilmek önemlidir.

```
SELECT *

FROM employees

NATURAL JOIN departments;
```

Bu sorgu, **employees** ve **departments** tablolarındaki ortak sütunları kullanarak birleştirilmiş sonuçları döndürür.

6.12. NON EQUAL JOİN

Non-equal join, iki tabloyu birleştirirken eşitlik dışındaki koşulları kullanır. Bu tür bir join, genellikle **JOIN ... ON** ifadesi ile birlikte kullanılır ve belirli bir koşula göre kayıtları birleştirir.

```
SELECT e.first_name, e.salary, d.department_name

FROM employees e

JOIN departments d ON e.salary > d.location_id;
```

Bu sorgu, çalışanların maaşlarının departman bütçesinden büyük olduğu durumlarda çalışanların adlarını ve departman adlarını döndürür.

6.13. ÖZET

PostgreSQL'de join işlemleri, birden fazla tabloyu birleştirerek veri çekmek için kullanılır. Farklı join türleri, verilerin nasıl birleştirileceğini belirler:

- **INNER JOIN**: Eşleşen kayıtları getirir.
- **LEFT JOIN**: Sol tablodaki tüm kayıtları ve sağ tablodaki eşleşen kayıtları getirir.
- **RIGHT JOIN**: Sağ tablodaki tüm kayıtları ve sol tablodaki eşleşen kayıtları getirir.
- **FULL OUTER JOIN**: Her iki tablodaki tüm kayıtları getirir.
- **SELF JOIN**: Bir tablonun kendisiyle birleştirilmesidir.
- **CROSS JOIN**: Kartesyen çarpım alır.
- **NATURAL JOIN**: Ortak sütunları kullanarak birleştirir.
- **NON EQUAL JOIN**: Eşitlik dışındaki koşulları kullanarak birleştirir.

7. ALT SORGULARI KULLANMA

Video Linki = https://www.youtube.com/watch?v=zV3WCU36Zro&t=3129s

7.1. ALT SORGULARIN KULLANIM ALANLARI

Alt sorgular, genellikle veri filtreleme, veri gruplama veya belirli bir koşula göre veri çekme işlemlerinde kullanılır. Örneğin, bir tablodan belirli bir koşula uyan verileri çekmek için alt sorgular kullanılabilir.

7.2. ALT SORGU TİPLERİ

Alt sorgular, genel olarak iki ana tipe ayrılır:

- Tek Satır Alt Sorgular
- Birden Fazla Satır Alt Sorgular

7.3. TEK SATIR ALT SORGULAR

Tek satır alt sorgular, yalnızca bir satır ve bir veya daha fazla kolon döndüren sorgulardır.

7.4. WHERE İFADESİNDE KULLANIM - TEK KOLON

```
SELECT *
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees);
```

Bu sorgu, en yüksek maaşı alan çalışanları getirir.

7.5. WHERE İFADESİNDE KULLANIM - BİRDEN FAZLA KOLON

```
SELECT *
FROM Employees
WHERE (first_name, last_name) = (SELECT first_name, last_name FROM Employees WHERE employee_id = 1);
```

Bu sorgu, **employee_id'si** 1 olan çalışanın adını ve soyadını kullanarak aynı adı ve soyadı taşıyan diğer çalışanları getirir.

7.6. KOLON OLARAK KULLANIM

```
SELECT employee_id, (SELECT MAX(Salary) FROM Employees) AS MaxSalary
FROM Employees;
```

Bu sorgu, her çalışanın id'si ile birlikte en yüksek maaşı döndürür.

7.7. HAVİNG İFADESİNDE KULLANIM

```
SELECT DepartmentID, COUNT(*) AS EmployeeCount
FROM Employees
GROUP BY DepartmentID
HAVING COUNT(*) > (SELECT AVG(EmployeeCount) FROM (SELECT COUNT(*) AS EmployeeCount FROM Employees GROUP BY DepartmentID) AS DeptCounts);
```

Bu sorgu, çalışan sayısı ortalamanın üzerinde olan departmanları getirir.

7.8. BİRDEN FAZLA SATIR ALT SORGULAR

Birden fazla satır döndüren alt sorgular, genellikle **IN**, **EXISTS** veya **NOT EXISTS** gibi operatörlerle birlikte kullanılır.

7.9. TANIM

Birden fazla satır alt sorgular, birden fazla sonuç döndüren sorgulardır. Örneğin, bir tablodan belirli bir koşula uyan birden fazla değeri döndürebilir.

7.10. FROM İFADESİNDE KULLANIM

```
SELECT *
FROM (SELECT first_name, last_name FROM Employees WHERE department_id = 1) AS DeptEmployees;
```

Bu sorgu, belirli bir departmandaki tüm çalışanların adlarını ve soyadlarını getirir.

7.11. IN OPERATÖRÜ

```
SELECT *
FROM Employees
WHERE department_id IN (SELECT department_id FROM Departments WHERE location_id = 1400);
```

Bu sorgu, lokasyon id'si 1400 olan departmanlara ait çalışanları getirir.

7.12. EXISTS OPERATÖRÜ

```
SELECT *
FROM Departments d
WHERE EXISTS (SELECT * FROM Employees e WHERE e.department_id = d.department_id);
```

Bu sorgu, en az bir çalışana sahip olan departmanları getirir. **EXISTS** operatörü, bir alt sorgudaki satırların varlığını kontrol eden bir Boolen operatörüdür.

7.13. NOT EXISTS OPERATÖRÜ

```
SELECT *
FROM Departments d
WHERE NOT EXISTS (SELECT * FROM Employees e WHERE e.department_id = d.department_id);
```

Bu sorgu, hiç çalışana sahip olmayan departmanları getirir.

8. DML - VERİLERİ DEĞİŞTİRME KOMUTLARI

Video Linki = https://www.youtube.com/watch?v=GmM3kQ3fS3o&t=3438s

8.1. INSERT - TEK SATIR

Tek bir satır eklemek için kullanılan **INSERT** ifadesidir.

```
INSERT INTO employees (first_name, last_name, age)
VALUES ('John', 'Doe', 30);
```

Bu ifade, **employees** tablosuna **first_name**, **last_name** ve **age** sütunlarına sahip bir satır ekler.

8.2. INSERT - RETURNİNG İFADESİ

Ekleme işlemi sonrası eklenen satırın bazı bilgilerini döndürmek için kullanılır.

```
INSERT INTO employees (first_name, last_name, age)
VALUES ('Jane', 'Smith', 25)
RETURNING id, first_name;
```

Bu ifade, **employees** tablosuna yeni bir satır ekler ve eklenen satırın **id** ve **first_name** değerlerini döndürür.

8.3. INSERT - BİRDEN FAZLA SATIR

Birden fazla satır eklemek için kullanılır.

```
INSERT INTO employees (first_name, last_name, age)
VALUES
  ('Alice', 'Johnson', 28),
  ('Bob', 'Brown', 35),
  ('Charlie', 'Davis', 40);
```

Bu ifade, **employees** tablosuna üç yeni satır ekler.

8.4. INSERT - SATIRLARI KOPYALAMA

Bir tablodan diğerine satır kopyalamak için kullanılır.

```
INSERT INTO employees_archive (first_name, last_name, age)
SELECT first_name, last_name, age FROM employees WHERE age > 30;
```

Bu ifade, **employees** tablosundaki yaşı 30'dan büyük olan çalışanları **employees_archive** tablosuna kopyalar.

8.5. UPDATE - TEK SATIR

Tek bir satırı güncellemek için kullanılır.

```
UPDATE employees
SET age = 31
WHERE first_name = 'John' AND last_name = 'Doe';
```

Bu ifade, **employees** tablosundaki **John Doe**'nun yaşını 31 olarak günceller.

8.6. UPDATE - BİRDEN FAZLA SATIR

Birden fazla satırı güncellemek için kullanılır.

```
UPDATE employees
SET age = age + 1
WHERE age < 30;
```

Bu ifade, yaşı 30'dan küçük olan tüm çalışanların yaşını bir artırır.

8.7. UPDATE - RETURNİNG

Güncelleme sonrası güncellenen satırların bazı bilgilerini döndürmek için kullanılır.

```
UPDATE employees
SET age = 32
WHERE first_name = 'Jane'
RETURNING id, age;
```

Bu ifade, **Jane**'in yaşını 32 olarak günceller ve güncellenen satırın **id** ve **age** değerlerini döndürür.

8.8. UPDATE - JOİN

Birden fazla tabloyu birleştirerek güncelleme yapmak için kullanılır.

```
UPDATE employees e
SET age = e.age + 1
FROM departments d
WHERE e.department_id = d.id AND d.name = 'Sales';
```

Bu ifade, **Sales** departmanındaki tüm çalışanların yaşını bir artırır.

8.9. DELETE - TEK SATIR

Tek bir satırı silmek için kullanılır.

```
DELETE FROM employees
WHERE first_name = 'John' AND last_name = 'Doe';
```

Bu ifade, **employees** tablosundaki **John Doe**'yu siler.

8.10. DELETE - BİRDEN FAZLA SATIR

Birden fazla satırı silmek için kullanılır.

```
DELETE FROM employees
WHERE age < 25;
```

Bu ifade, yaşı 25'ten küçük olan tüm çalışanları siler.

8.11. DELETE - RETURNİNG

Silme işlemi sonrası silinen satırların bazı bilgilerini döndürmek için kullanılır.

```
DELETE FROM employees
WHERE first_name = 'Jane'
RETURNING id, first_name;
```

Bu ifade, **Jane'i** siler ve silinen satırın **id** ve **first_name** değerlerini döndürür.

8.12. DELETE - JOİN

Birden fazla tabloyu birleştirerek silme işlemi yapmak için kullanılır.

```
DELETE FROM employees e
USING departments d
WHERE e.department_id = d.id AND d.name = 'HR';
```

Bu ifade, **HR** departmanındaki tüm çalışanları siler.

9. VERİTABANI TRANSACTİON YÖNETİMİ

Video Linki = https://www.youtube.com/watch?v=qEhGTQVuI4A&t=3516s

9.1. VERİTABANI TRANSACTİON'I NEDİR?

Veritabanı transaction'ı, bir dizi SQL işleminin (örneğin, **INSERT**, **UPDATE**, **DELETE**) bir bütün olarak ele alındığı bir işlemdir. Transaction, ya tamamen başarılı bir şekilde tamamlanır (commit), ya da herhangi bir hata durumunda tüm işlemler geri alınır (rollback). Bu, veritabanının tutarlılığını ve bütünlüğünü korumak için önemlidir.

9.2. VERİTABANI TRANSACTİON TİPLERİ

Otomatik Transaction: Her SQL komutu kendi başına bir transaction olarak kabul edilir.

Manuel Transaction: Kullanıcı tarafından başlatılan ve yönetilen transaction'lardır. **BEGIN**, **COMMIT**, **ROLLBACK** gibi komutlarla kontrol edilir.

9.3. TRANSACTİON'LARIN ÖZELLİKLERİ

Transaction'lar, ACID özelliklerine sahiptir:

- **Atomicity (Atomiklik)**: Transaction içindeki tüm işlemler ya tamamen başarılı olur ya da hiçbiri.
- **Consistency (Tutarlılık)**: Transaction tamamlandığında veritabanı tutarlı bir durumda olmalıdır.
- **Isolation (İzolasyon)**: Bir transaction'ın etkileri, diğer transaction'lar tarafından görünmez olmalıdır.
- **Durability (Dayanıklılık)**: Transaction tamamlandığında, veritabanındaki değişiklikler kalıcıdır.

9.4. TRANSACTİON YÖNETME - COMMIT

COMMIT, bir transaction içindeki tüm işlemleri kalıcı hale getirir.

```
BEGIN;
INSERT INTO accounts (account_id, balance) VALUES (1, 1000);
UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
COMMIT;
```

Bu örnekte, **BEGIN** ile bir transaction başlatılır, ardından bir **INSERT** ve bir **UPDATE** işlemi yapılır. **COMMIT** ile bu işlemler kalıcı hale getirilir.

9.5. TRANSACTİON YÖNETME - ROLLBACK

ROLLBACK, bir transaction içindeki tüm işlemleri geri alır.

```
BEGIN;
```

```
INSERT INTO accounts (account_id, balance) VALUES (2, 500);

UPDATE accounts SET balance = balance - 200 WHERE account_id = 1;

ROLLBACK;
```

Bu örnekte, **ROLLBACK** komutu ile yapılan tüm işlemler geri alınır ve veritabanı önceki durumuna döner.

9.6. TRANSACTİON YÖNETME - SAVEPOINT

SAVEPOINT, bir transaction içinde belirli bir noktaya geri dönmek için kullanılır.

```
BEGIN;

INSERT INTO accounts (account_id, balance) VALUES (3, 300);

SAVEPOINT sp1;

UPDATE accounts SET balance = balance - 50 WHERE account_id = 3;

ROLLBACK TO sp1;  -- sp1'e geri dön

COMMIT;
```

Bu örnekte, **SAVEPOINT** ile **sp1** adında bir nokta belirlenir. **ROLLBACK TO sp1** komutu ile bu noktaya geri dönülür ve **INSERT** işlemi kalıcı hale gelirken, **UPDATE** işlemi geri alınır.

9.7. COMMIT & ROLLBACK SİMULASYON

Bir transaction simülasyonu yapalım:

```
BEGIN;

INSERT INTO orders (order_id, amount) VALUES (1, 100);

INSERT INTO orders (order_id, amount) VALUES (2, 200);

-- Hata oluşursa

ROLLBACK;  -- Tüm işlemler geri alınır

-- Eğer hata yoksa

COMMIT;  -- Tüm işlemler kalıcı hale gelir
```

Bu simülasyonda, iki **INSERT** işlemi yapılır. Eğer bir hata oluşursa, **ROLLBACK** ile tüm işlemler geri alınır. Hata yoksa, **COMMIT** ile işlemler kalıcı hale gelir.

9.8. COMMIT SONRASI

COMMIT komutundan sonra, yapılan değişiklikler kalıcı hale gelir ve veritabanında görünür hale gelir. Artık bu değişiklikler geri alınamaz. Yani, **COMMIT** işleminden sonra, veritabanı durumu, yapılan tüm işlemleri yansıtır.

```
BEGIN;  -- Yeni bir transaction başlat

INSERT INTO employees (first_name, last_name, salary) VALUES ('Hannah', 'Taylor', 14000);COMMIT;  -- Yapılan değişiklikleri kalıcı hale getir

-- Şimdi employees tablosunu kontrol edelim

SELECT * FROM employees;  -- 'Hannah Taylor' kaydı görünmelidir
```

Bu örnekte, **Hannah Taylor** adlı çalışan ekleniyor ve **COMMIT** ile bu değişiklik kalıcı hale getiriliyor. **SELECT** sorgusu ile employees tablosunu kontrol ettiğimizde, bu kaydın görünmesi gerekir.

9.9. SATIR KİLİTLEME

PostgreSQL'de satır kilitleme, bir transaction sırasında belirli satırların diğer transaction'lar tarafından değiştirilmesini engellemek için kullanılır. Bu, veri tutarlılığını sağlamak için önemlidir. Satır kilitleme, **SELECT ... FOR UPDATE** veya **SELECT ... FOR SHARE** gibi ifadelerle yapılır.

```
BEGIN;  -- Yeni bir transaction başlat

-- Satırı kilitle

SELECT * FROM employees WHERE employee_id = 1 FOR UPDATE;
```

```
-- Bu satır üzerinde değişiklik yapabiliriz

UPDATE employees SET salary = salary + 1000 WHERE employee_id = 1;

COMMIT;  -- Yapılan değişiklikleri kalıcı hale getir
```

Bu örnekte**, employee_id** 1 olan satır kilitleniyor ve bu satır üzerinde güncelleme yapılıyor. Diğer transaction'lar bu satırı değiştiremez.

9.10. ROLLBACK SONRASI

ROLLBACK komutundan sonra, yapılan tüm değişiklikler geri alınır ve veritabanı, **ROLLBACK** komutundan önceki durumuna döner. Bu, transaction sırasında yapılan tüm işlemlerin iptal edildiği anlamına gelir.

```
BEGIN;  -- Yeni bir transaction başlat

INSERT INTO employees (first_name, last_name, salary) VALUES ('Ivy', 'Clark', 15000);

INSERT INTO employees (first_name, last_name, salary) VALUES ('Jack', 'White', 16000);

ROLLBACK;  -- Yapılan değişiklikleri geri al

-- Şimdi employees tablosunu kontrol edelim

SELECT * FROM employees;  -- 'Ivy Clark' ve 'Jack White' kayıtları görünmemelidir
```

Bu örnekte, iki yeni çalışan ekleniyor, ancak **ROLLBACK** ile bu eklemeler geri alınıyor. **SELECT** sorgusu ile employees tablosunu kontrol ettiğimizde, bu kayıtların görünmemesi gerekir.

9.11. ÖZET

PostgreSQL'de transaction yönetimi, veritabanı işlemlerinin güvenilirliğini ve tutarlılığını sağlamak için kritik öneme sahiptir. **COMMIT**, **ROLLBACK**, ve **SAVEPOINT** gibi komutlar, transaction'ların nasıl yönetileceğini belirler.

- **COMMIT:** Yapılan değişiklikleri kalıcı hale getirir.
- **ROLLBACK:** Yapılan değişiklikleri geri alır.
- **SAVEPOINT:** Transaction içinde belirli bir noktaya geri dönmeyi sağlar.
- **Satır Kilitleme:** Belirli satırların diğer transaction'lar tarafından değiştirilmesini engeller**.**

Bu özellikler, veritabanı uygulamalarında veri bütünlüğünü ve tutarlılığını sağlamak için kullanılır.

10. VERİ TİPLERİ VE TABLO OLUŞTURMA

Video Linki = https://www.youtube.com/watch?v=0d5jKpMzKv8

10.1. VERİ TİPLERİ

- **String Veri Tipleri**

PostgreSQL'de string veri tipleri arasında **CHAR**, **VARCHAR**, ve **TEXT** bulunur.

- **CHAR(n):** Sabit uzunlukta karakter dizisi. Örneğin, CHAR(10) her zaman 10 karakter uzunluğunda bir dizi tutar.
- **VARCHAR(n):** Değişken uzunlukta karakter dizisi. Örneğin, VARCHAR(50) en fazla 50 karakter tutabilir.
- **TEXT:** Sınırsız uzunlukta karakter dizisi. Herhangi bir uzunluk sınırlaması yoktur.

```
SELECT first_name, last_name FROM actor WHERE LENGTH(first_name) > 5;
```

- **Numeric Veri Tipleri**

PostgreSQL'de sayısal veri tipleri arasında **INTEGER, FLOAT, NUMERIC,** ve **DECIMAL** bulunur.

- **INTEGER:** Tam sayılar için kullanılır.
- **FLOAT:** Kesirli sayılar için kullanılır.
- **NUMERIC**: Kesirli sayılar için tam hassasiyet sağlar.

```
SELECT rental_id, amount FROM payment WHERE amount > 10.00;
```

- **Date-Time Veri Tipleri**

PostgreSQL'de tarih ve zaman veri tipleri arasında **DATE, TIME, TIMESTAMP**, ve **INTERVAL** bulunur.

- **DATE:** Tarih bilgisi (YYYY-MM-DD formatında).
- **TIME:** Zaman bilgisi (HH:MM:SS formatında).

- **TIMESTAMP:** Tarih ve zaman bilgisi.

```
SELECT rental_date FROM rental WHERE rental_date > '2000-01-01';
```

10.2. TABLO OLUŞTURMA METODLARI

- **Bir Tablo Oluşturma - Script Yazarak**

PostgreSQL'de bir tablo oluşturmak için **CREATE TABLE** komutunu kullanabilirsiniz.

```
CREATE TABLE new_actor (
    actor_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    last_update TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Bir Tablo Oluşturma - SELECT INTO**

Mevcut bir tablodan yeni bir tablo oluşturmak için **SELECT INTO** ifadesini kullanabilirsiniz.

```
SELECT * INTO new_payment FROM payment WHERE amount > 10.00;
```

- **Bir Tablo Oluşturma - CREATE TABLE AS (Sorgu)**

CREATE TABLE AS ifadesi ile bir sorgunun sonucunu yeni bir tabloya yazabilirsiniz.

```
CREATE TABLE high_rentals AS
SELECT * FROM rental WHERE rental_duration > 5;
```

- **Bir Tablo Oluşturma - CREATE TABLE AS (Tablo)**

Mevcut bir tablodan yeni bir tablo oluşturmak için **CREATE TABLE AS** ifadesini kullanabilirsiniz.

```
CREATE TABLE actor_copy AS
SELECT * FROM actor;
```

- **SERIAL Pseudo Tipi - Tanım**

SERIAL veri tipi, otomatik artan bir tamsayı oluşturmak için kullanılır. Genellikle birincil anahtar olarak kullanılır.

```
CREATE TABLE example (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100)
);
```

Bu örnekler, PostgreSQL'de veri tipleri ve tablo oluşturma yöntemleri hakkında temel bir anlayış sağlamaktadır. dvdrental veritabanı üzerinden yapılan sorgular ve tablo oluşturma işlemleri, veritabanı yönetimi ve veri analizi için yaygın olarak kullanılan yöntemlerdir.

10.3. VERİ TİPLERİ DETAYLARI

- **String Veri Tipleri**
  - **CHAR(n):** Eğer bir dizi n karakterden daha kısa ise, geri kalan alanlar boşluklarla doldurulur.
  - **VARCHAR(n):** n karakterden daha uzun bir dizi girilirse hata verir. Ancak, boş diziler için bir sınırlama yoktur.

- **TEXT:** Uzun metinler için idealdir. Örneğin, açıklamalar veya yorumlar gibi.

```
CREATE TABLE movie_description (
  movie_id SERIAL PRIMARY KEY,
  title VARCHAR(100),
  description TEXT
);
```

- **Numeric Veri Tipleri**
  - **INTEGER:** -2,147,483,648 ile 2,147,483,647 arasında değer alabilir.
  - **FLOAT:** Kesirli sayılar için kullanılır, ancak tam hassasiyet sağlamaz.
  - **NUMERIC:** Kesirli sayılar için tam hassasiyet sağlar ve genellikle finansal veriler için tercih edilir.

```
CREATE TABLE financial_records (
  record_id SERIAL PRIMARY KEY,
  amount NUMERIC(10, 2)  -- 10 basamak, 2 ondalık
);
```

- **Date-Time Veri Tipleri**
  - **DATE**: Sadece tarih bilgisi tutar.
  - **TIME:** Sadece zaman bilgisi tutar.
  - **TIMESTAMP:** Hem tarih hem de zaman bilgisi tutar. Zaman dilimi bilgisi içermeyen versiyonu **TIMESTAMP WITHOUT TIME ZONE** olarak adlandırılır.

```
CREATE TABLE event_schedule (
  event_id SERIAL PRIMARY KEY,
  event_name VARCHAR(100),
  event_date TIMESTAMP
);
```

10.4. TABLO OLUŞTURMA METODLARI DETAYLARI

- **Bir Tablo Oluşturma - Script Yazarak**

Tablo oluştururken, veri tiplerini ve kısıtlamaları belirlemek önemlidir. Örneğin, **NOT NULL** kısıtlaması ekleyerek belirli alanların boş olmasını engelleyebilirsiniz.

```
CREATE TABLE customer (
  customer_id SERIAL PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **Bir Tablo Oluşturma - SELECT INTO**

SELECT INTO ifadesi, mevcut bir tablodan yeni bir tablo oluşturur ve yeni tabloya verileri kopyalar.

```
SELECT * INTO high_value_customers
FROM customer
```

```
WHERE customer_id IN (SELECT customer_id FROM payment WHERE amount > 100);
```

- **Bir Tablo Oluşturma - CREATE TABLE AS (Sorgu)**

Bu yöntem, belirli bir sorgunun sonucunu yeni bir tabloya kaydeder.

```
CREATE TABLE top_movies AS

SELECT title, release_year, rating

FROM film

WHERE rating > 8.0;
```

- **Bir Tablo Oluşturma - CREATE TABLE AS (Tablo)**

Mevcut bir tablodan yeni bir tablo oluşturmak için kullanılır.

```
CREATE TABLE actor_backup AS

SELECT * FROM actor;
```

- **SERIAL Pseudo Tipi - Tanım**

SERIAL veri tipi, otomatik artan bir tamsayı oluşturmak için kullanılır. Bu, genellikle birincil anahtar olarak kullanılır ve yeni bir kayıt eklendiğinde otomatik olarak bir değer alır.

```
CREATE TABLE orders (

   order_id SERIAL PRIMARY KEY,

   customer_id INTEGER REFERENCES customer(customer_id),

   order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

10.5. EKSTRA ÖRNEKLER VE KULLANIM SENARYOLARI

- **Veri Tipi Kontrolü**

Veri tiplerini kontrol etmek için pg_type sistem tablosunu kullanabilirsiniz.

```
SELECT typname, typlen, typcategory

FROM pg_type

WHERE typname IN ('varchar', 'integer', 'timestamp');
```

- **Tablo Yapısını Görüntüleme**

Bir tablonun yapısını görüntülemek için **\d** komutunu kullanabilirsiniz.

```
\d actor
```

- **Tablo Silme**

Bir tabloyu silmek için **DROP TABLE** komutunu kullanabilirsiniz.

```
DROP TABLE IF EXISTS
```

11. ALTER KOMUTLARI İLE TABLO YAPILARINDA DEĞİŞİKLİK

Video Linki: https://www.youtube.com/watch?v=kP6AsLGxrOE

11.1. BİR KOLON EKLEME (ADD COLUMN)

Mevcut bir tabloya yeni sütun ekler. PostgreSQL'de aynı anda birden fazla sütun eklenebilir.

```
-- office tablosuna yeni sütunlar ekleyelim
ALTER TABLE office
ADD COLUMN manager_email VARCHAR(100),
ADD COLUMN budget DECIMAL(12,2) DEFAULT 100000.00,
ADD COLUMN established_date DATE;

-- rentaldvd tablosuna yeni sütunlar ekleyelim
ALTER TABLE rentaldvd
ADD COLUMN dvd_condition VARCHAR(20) DEFAULT 'Good',
ADD COLUMN last_maintenance_date TIMESTAMP;
```

11.2. BİR KOLON İSMİNİ DEĞİŞTİRME (RENAME COLUMN)

Sütun adını değiştirir. Tablo yeniden oluşturulmadan ad değişikliği yapılabilir.

```
-- office tablosunda sütun adlarını değiştirelim
ALTER TABLE office
RENAME COLUMN manager_email TO director_email;

-- rentaldvd tablosunda sütun adlarını değiştirelim
ALTER TABLE rentaldvd
RENAME COLUMN dvd_condition TO media_condition;
```

11.3. DEFAULT DEĞERİ DEĞİŞTİRME (ALTER COLUMN SET DEFAULT)

Sütunun varsayılan değerini değiştirir. Var olan kayıtları etkilemez.

```
-- office tablosunda varsayılan değerleri güncelleyelim
ALTER TABLE office
ALTER COLUMN budget SET DEFAULT 150000.00,
ALTER COLUMN established_date SET DEFAULT CURRENT_DATE;

-- rentaldvd tablosunda varsayılan değerleri güncelleyelim
ALTER TABLE rentaldvd
ALTER COLUMN media_condition SET DEFAULT 'Excellent';
```

11.4. KOLON TİPİNİ DEĞİŞTİRME (ALTER COLUMN TYPE)

Sütunun veri tipini değiştirir. PostgreSQL tip dönüşümlerini otomatik yapmaya çalışır.

```
-- office tablosunda veri tiplerini değiştirelim
ALTER TABLE office
ALTER COLUMN budget TYPE NUMERIC(15,2),
ALTER COLUMN director_email TYPE TEXT;
```

```
-- rentaldvd tablosunda veri tiplerini değiştirelim
ALTER TABLE rentaldvd
ALTER COLUMN rental_price TYPE DECIMAL(6,2);
```

11.5. BİR COMMENT (YORUM) EKLEME (COMMENT)

Tablo veya sütunlara açıklayıcı yorum ekler.

```
-- office tablosuna ve sütunlarına yorum ekleyelim
COMMENT ON TABLE office IS 'Şirket ofis bilgilerini içeren ana tablo';
COMMENT ON COLUMN office.director_email IS 'Ofis direktörünün resmi email adresi';

-- rentaldvd tablosuna ve sütunlarına yorum ekleyelim
COMMENT ON TABLE rentaldvd IS 'DVD kiralama işlemlerinin kayıtları';
COMMENT ON COLUMN rentaldvd.media_condition IS 'DVD fiziksel durumu: Poor, Good, Excellent';
```

11.6. NOT NULL KISITLAMASI EKLEME-KALDIRMA

Sütunun NULL değer kabul edip etmeyeceğini kontrol eder.

```
-- office tablosunda NOT NULL kısıtlamaları
ALTER TABLE office
ALTER COLUMN director_email SET NOT NULL,
ALTER COLUMN established_date DROP NOT NULL;

-- rentaldvd tablosunda NOT NULL kısıtlamaları
ALTER TABLE rentaldvd
ALTER COLUMN media_condition SET NOT NULL;
```

11.7. KOLONLARI KALDIRMA (DROP COLUMN)

Tablodan sütun siler. PostgreSQL'de CASCADE seçeneği ile bağımlı nesneler de silinebilir.

```
-- office tablosundan sütun silme
ALTER TABLE office
DROP COLUMN budget,
DROP COLUMN established_date;

-- rentaldvd tablosundan sütun silme
ALTER TABLE rentaldvd
DROP COLUMN last_maintenance_date;
```

11.8. BİR CONSTRAİNT EKLEME (ADD CONSTRAINT)

Tabloya kısıtlama ekler (PK, FK, CHECK vb.).

```
-- office tablosuna kısıtlama ekleme
ALTER TABLE office
ADD CONSTRAINT pk_office PRIMARY KEY (office_id),
```

```
ADD CONSTRAINT chk_budget CHECK (budget >= 0);

-- rentaldvd tablosuna kısıtlama ekleme
ALTER TABLE rentaldvd
ADD CONSTRAINT fk_office FOREIGN KEY (office_id) REFERENCES office(office_id),
ADD CONSTRAINT chk_rental_price CHECK (rental_price > 0);
```

11.9. CONSTRAİNT KALDIRMA (DROP CONSTRAINT)

Var olan bir kısıtlamayı kaldırır.

```
-- office tablosundan kısıtlama kaldırma
ALTER TABLE office
DROP CONSTRAINT chk_budget;

-- rentaldvd tablosundan kısıtlama kaldırma
ALTER TABLE rentaldvd
DROP CONSTRAINT chk_rental_price;
```

11.10. TABLO İSMİNİ DEĞİŞTİRME (RENAME TO)

Tablonun adını değiştirir.

```
-- office tablosunun adını değiştirme
ALTER TABLE office
RENAME TO company_branches;

-- rentaldvd tablosunun adını değiştirme
ALTER TABLE rentaldvd
RENAME TO dvd_rentals;
```

11.11. BİR TABLOYU KALDIRMA (DROP TABLE)

Tabloyu ve tüm verilerini tamamen siler.

```
-- company_branches tablosunu silme
DROP TABLE company_branches;

-- dvd_rentals tablosunu silme
DROP TABLE dvd_rentals;
```

11.12. TRUNCATE KOMUTU

Tablodaki tüm verileri siler, tablo yapısı korunur.

```
-- company_branches tablosundaki tüm verileri silme
TRUNCATE TABLE company_branches;

-- dvd_rentals tablosundaki tüm verileri silme (identity sütunlarını sıfırlar)
TRUNCATE TABLE dvd_rentals RESTART IDENTITY;
```

11.13. POSTGRESQL ÖZEL NOTLARI:

- Birden fazla ALTER işlemi tek komutta virgülle ayrılarak yapılabilir
- TYPE değişikliğinde USING ifadesiyle özel dönüşüm kuralları belirtilebilir
- TRUNCATE komutunda CASCADE seçeneği ile bağımlı tablolardaki veriler de silinebilir
- PostgreSQL'de tablo yorumları pg_description sistem kataloğunda saklanır

12. VERİTABANI KISITLAMALARI: CONSTRAİNTS

Video Linki: https://www.youtube.com/watch?v=IlMAG2XNhTs&t=3162s

12.1. PRİMARY KEY CONSTRAİNT (BİRİNCİL ANAHTAR KISITLAMASI)

Bir tabloda her satırı benzersiz şekilde tanımlayan sütun veya sütun grubudur. NULL değer alamaz ve her tabloda yalnızca bir tane bulunur.

Özellikleri:

- Benzersiz (unique) olmalıdır
- NULL değer içeremez
- Tabloda tek olmalıdır (ancak birden fazla sütundan oluşabilir)

```
-- Tablo oluştururken primary key tanımlama
CREATE TABLE employees (
  employee_id SERIAL PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50)
);

-- Var olan tabloya primary key ekleme
ALTER TABLE employees ADD PRIMARY KEY (employee_id);
-- Composite primary key (birden fazla sütundan oluşan)
CREATE TABLE order_items (
  order_id INT,
  product_id INT,
  quantity INT,
  PRIMARY KEY (order_id, product_id) );
```

12.2. FOREİGN KEY CONSTRAİNT (YABANCI ANAHTAR KISITLAMASI)

Bir tablodaki sütunun değerlerinin, başka bir tablonun primary key'ine referans vermesini sağlar. İlişkisel veritabanlarında tablolar arası ilişki kurmak için kullanılır.

Özellikleri:

- Referans bütünlüğünü sağlar
- Parent tabloda olmayan bir değer eklenemez
- Silme ve güncelleme işlemlerinde davranışı tanımlanabilir

```
-- Tablo oluştururken foreign key tanımlama
CREATE TABLE orders (
  order_id SERIAL PRIMARY KEY,
  customer_id INT REFERENCES customers(customer_id),
  order_date DATE
);
```

```
-- Var olan tabloya foreign key ekleme
ALTER TABLE orders
ADD CONSTRAINT fk_customer
FOREIGN KEY (customer_id) REFERENCES customers(customer_id);

-- ON DELETE ve ON UPDATE davranışlarıyla
CREATE TABLE order_items (
    item_id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(order_id) ON DELETE CASCADE,
    product_id INT,
    quantity INT
);
```

Foreign Key Davranış Seçenekleri:

- **ON DELETE CASCADE**: Parent kayıt silinirse child kayıtlar da silinir
- **ON DELETE SET NULL**: Parent kayıt silinirse child'daki foreign key NULL yapılır
- **ON DELETE SET DEFAULT**: Parent kayıt silinirse child'daki foreign key default değere ayarlanır
- **ON DELETE RESTRICT**: Parent kayıt silinemez (child varsa)
- **ON UPDATE** için de benzer seçenekler vardır

12.3. CHECK CONSTRAİNT (KONTROL KISITLAMASI)

Bir sütuna girilebilecek değerleri belirli bir koşula göre kısıtlar. Boolean bir ifade ile kontrol sağlar.

Özellikleri:

- Özel koşullar tanımlamak için kullanılır
- Sütun seviyesinde veya tablo seviyesinde tanımlanabilir
- INSERT veya UPDATE işlemlerinde koşul sağlanmazsa hata verir

```
-- Tablo oluştururken check constraint ekleme
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100),
    price DECIMAL(10,2) CHECK (price > 0),
    discount_price DECIMAL(10,2),
    CONSTRAINT valid_discount CHECK (discount_price < price)
);

-- Var olan tabloya check constraint ekleme
ALTER TABLE employees
ADD CONSTRAINT valid_salary
CHECK (salary > 0 AND salary < 100000);

-- Tarih kontrolü
CREATE TABLE events (
    event_id SERIAL PRIMARY KEY,
```

```
    event_name VARCHAR(100),
    start_date DATE,
    end_date DATE,
    CONSTRAINT valid_dates CHECK (end_date >= start_date)
);
```

12.4. UNİQUE CONSTRAİNT (BENZERSİZ KISITLAMA)

Bir sütundaki veya sütun grubundaki tüm değerlerin benzersiz olmasını sağlar. Primary key'den farkı, NULL değerleri kabul edebilmesidir (PostgreSQL'de birden fazla NULL değere izin verilir).

Özellikleri:

- Sütundaki değerlerin benzersiz olmasını sağlar
- Birden fazla sütuna uygulanabilir (composite unique)
- Bir tabloda birden fazla unique constraint olabilir

```
-- Tablo oluştururken unique constraint ekleme
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    email VARCHAR(100) UNIQUE,
    password VARCHAR(100)
);


-- Var olan tabloya unique constraint ekleme
ALTER TABLE departments
ADD CONSTRAINT unique_dept_name UNIQUE (department_name);


-- Composite unique constraint (birden fazla sütunun kombinasyonu benzersiz olmalı)
CREATE TABLE student_courses (
    student_id INT,
    course_id INT,
    enrollment_date DATE,
    UNIQUE (student_id, course_id)
);
```

12.5. NOT NULL CONSTRAİNT (BOŞ OLMAZ KISITLAMASI)

Bir sütunun NULL değer içermemesini sağlar. Veri bütünlüğü için sık kullanılan basit bir kısıtlamadır.

Özellikleri:

- Sütunun zorunlu (required) olmasını sağlar
- Default olarak sütunlar NULL değer alabilir
- CHECK constraint ile de benzer sonuç elde edilebilir

```
-- Tablo oluştururken not null constraint ekleme
CREATE TABLE customers (
    customer_id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
```

```
    email VARCHAR(100)
);

-- Var olan tabloya not null constraint ekleme
ALTER TABLE products
ALTER COLUMN product_name SET NOT NULL;

-- Not null kaldırma
ALTER TABLE employees
ALTER COLUMN middle_name DROP NOT NULL;
```

12.6. CONSTRAİNT'LERİ YÖNETME

1. CONSTRAİNT İSİMLENDİRME

Constraint'lere isim verilmezse PostgreSQL otomatik isim oluşturur. Ancak best practice olarak kendi anlamlı isimlerinizi vermeniz önerilir.

```
-- İsimli constraint örneği
CREATE TABLE accounts (
    account_id SERIAL,
    account_number VARCHAR(20) NOT NULL,
    balance DECIMAL(15,2) DEFAULT 0,
    CONSTRAINT pk_account PRIMARY KEY (account_id),
    CONSTRAINT chk_balance CHECK (balance >= 0)
);
```

2. CONSTRAİNT SİLME

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

3. CONSTRAİNT DİSABLE/ENABLE ETME

```
-- Geçici olarak devre dışı bırakma
ALTER TABLE table_name DISABLE TRIGGER ALL;
-- Sonra tekrar aktif etme
ALTER TABLE table_name ENABLE TRIGGER ALL;
```

Constraint'ler veri bütünlüğünü sağlamak için çok önemli araçlardır. Doğru kullanıldığında veritabanınızda tutarsız veri oluşmasını engellerler.