**İÇİNDEKİLER TABLOSU**

[**1.** **VERİTABANI TEMELLERİ** 5](#_Toc195467964)

[**1.1.** **VERİ NEDİR?** 5](#_Toc195467965)

[**1.2.** **VERİTABANI NEDİR?** 5](#_Toc195467966)

[**1.3.** **VERİTABANI TÜRLERİ** 5](#_Toc195467967)

[**1.4.** **İLİŞKİSEL VERİTABANLARINA GİRİŞ** 5](#_Toc195467968)

[**1.5.** **İLİŞKİSEL VERİTABANLARINI KARŞILAŞTIRILMASI** 6](#_Toc195467969)

[**1.6.** **VERİ TÜRLERİ NELERDİR?.** 6](#_Toc195467970)

[**1.7.** **VERİTABANI MODELLERİNİN KARŞILAŞTIRILMASI** 6](#_Toc195467971)

[**1.8.** **SQL-NOSQL VERİTABANLARI (DEVAM)** 6](#_Toc195467972)

[**1.9.** **EĞİTİMDE KULLANILACAK VERİTABANI: POSTGRESQL** 7](#_Toc195467973)

[**1.10.** **SONUÇ** 8](#_Toc195467974)

[**2.** **SQL'E GİRİŞ VE TEMEL KOMUTLAR** 8](#_Toc195467975)

[**2.1.** **SQL NEDİR?** 8](#_Toc195467978)

[**2.2.** **SQL YAZMA KURALLARI** 8](#_Toc195467979)

[**2.3.** **KOLONLARI SEÇME** 8](#_Toc195467980)

[**2.4.** **SQL İLE NELER YAPABİLİRİZ?** 8](#_Toc195467981)

[**2.5.** **SQL KOMUTLARI SINIFLANDIRMASI** 8](#_Toc195467982)

[**2.6.** **SELECT İFADESİ - SYNTAX** 8](#_Toc195467983)

[**2.7.** **TÜM KOLONLARI SEÇME** 9](#_Toc195467984)

[**2.8.** **BELİRLİ KOLONLARI SEÇME** 9](#_Toc195467985)

[**2.9.** **ARİTMETİK İFADELERİ SORGU İÇERİSİNDE KULLANMA** 9](#_Toc195467986)

[**2.10.** **BİRLEŞTİRME (CONCATENATİON) OPERATÖRÜ** 9](#_Toc195467987)

[**2.11.** **KOLONLARA ALİAS VERME** 9](#_Toc195467988)

[**2.12.** **ORDER BY KOMUTU İLE SATIRLARI SIRALAMA** 9](#_Toc195467989)

[**2.13.** **DISTINCT KOMUTU** 9](#_Toc195467990)

[**2.14.** **ÖZET** 10](#_Toc195467991)

[**3.** **SQL SATIR VE VERİ OPERATÖRLERİ** 10](#_Toc195467992)

[**3.1.** **SATIRLARI LİMİTLEME - WHERE İFADESİ** 10](#_Toc195467994)

[**3.2.** **OPERATÖR LİSTESİ** 10](#_Toc195467995)

[**3.3.** **KARŞILAŞTIRMA OPERATÖRLERİ** 10](#_Toc195467996)

[**3.4.** **MANTIKSAL OPERATÖRLER** 10](#_Toc195467997)

[**3.5.** **BETWEEN OPERATÖRÜ** 11](#_Toc195467998)

[**3.6.** **IN OPERATÖRÜ** 11](#_Toc195467999)

[**3.7.** **LIKE OPERATÖRÜ** 11](#_Toc195468000)

[**3.8.** **NOT OPERATÖRÜ** 11](#_Toc195468001)

[**3.9.** **NULL DEĞERİ** 11](#_Toc195468002)

[**3.10.** **IS NULL OPERATÖRÜ** 11](#_Toc195468003)

[**3.11.** **LIMIT KOMUTU İLE SATIRLARI SINIRLAMA** 11](#_Toc195468004)

[**3.12.** **OFFSET İLE BİRLİKTE LIMIT KULLANIMI** 12](#_Toc195468005)

[**3.13.** **ÖZET** 12](#_Toc195468006)

[**4.** **SQL FONKSİYONLARI** 12](#_Toc195468007)

[**4.1.** **SQL FONKSİYONLARI NEDİR?** 12](#_Toc195468009)

[**4.2.** **STRİNG FONKSİYONLARI: BÜYÜK-KÜÇÜK HARF** 12](#_Toc195468010)

[**4.3.** **STRİNG FONKSİYONLARI: KARAKTER İŞLEME** 12](#_Toc195468011)

[**4.4.** **MATEMATİK FONKSİYONLARI** 13](#_Toc195468012)

[**4.5.** **TARİH FONKSİYONLARI** 13](#_Toc195468013)

[**4.6.** **DÖNÜŞÜM FONKSİYONLARI** 13](#_Toc195468014)

[**4.7.** **DÖNÜŞÜM FONKSİYONLARI: CAST FONKSİYONU** 13](#_Toc195468015)

[**4.8.** **TARİHLER İLE ARİTMETİK İŞLEMLER** 13](#_Toc195468016)

[**4.9.** **COALESCE FONKSİYONU** 13](#_Toc195468017)

[**4.10.** **NULLIF FONKSİYONU** 14](#_Toc195468018)

[**4.11.** **CASE İFADESİ** 14](#_Toc195468019)

[**4.12.** **NESTED (İÇ İÇE) FONKSİYONLAR** 14](#_Toc195468020)

[**5.** **GRUP FONKSİYONLARI VE GROUP BY** 14](#_Toc195468021)

[**5.1.** **GRUP FONKSİYONLAR LİSTESİ** 14](#_Toc195468023)

[**5.2.** **AVG FONKSİYONU** 14](#_Toc195468024)

[**5.3.** **SUM FONKSİYONU** 15](#_Toc195468025)

[**5.4.** **COUNT FONKSİYONU** 15](#_Toc195468026)

[**5.5.** **MIN-MAX FONKSİYONLARI** 15](#_Toc195468027)

[**5.6.** **GROUP BY İFADESİ** 15](#_Toc195468028)

[**5.7.** **HAVING İFADESİ** 15](#_Toc195468029)

[**6.** **BİRDEN FAZLA TABLO ÜZERİNDE SORGULAMA** 15](#_Toc195468030)

[**6.1.** **ER DİYAGRAMLARI** 15](#_Toc195468032)

[**6.2.** **TABLOLARA ALİAS VERME** 16](#_Toc195468033)

[**6.3.** **JOİN TİPLERİ** 16](#_Toc195468034)

[**6.4.** **INNER JOİN** 16](#_Toc195468035)

[**6.5.** **JOİN İŞLEMİ - USING İFADESİ İLE** 16](#_Toc195468036)

[**6.6.** **JOİN İŞLEMİ - KLASİK YÖNTEM** 16](#_Toc195468037)

[**6.7.** **LEFT JOİN - RİGHT JOİN** 16](#_Toc195468038)

[**6.8.** **FULL OUTER JOİN** 16](#_Toc195468039)

[**6.9.** **SELF JOİN** 17](#_Toc195468040)

[**6.10.** **CROSS JOİN** 17](#_Toc195468041)

[**6.11.** **NATURAL JOİN** 17](#_Toc195468042)

[**6.12.** **NON EQUAL JOİN** 17](#_Toc195468043)

[**6.13.** **ÖZET** 17](#_Toc195468044)

[**7.** **ALT SORGULARI KULLANMA** 18](#_Toc195468045)

[**7.1.** **ALT SORGULARIN KULLANIM ALANLARI** 18](#_Toc195468047)

[**7.2.** **ALT SORGU TİPLERİ** 18](#_Toc195468048)

[**7.3.** **TEK SATIR ALT SORGULAR** 18](#_Toc195468049)

[**7.4.** **WHERE İFADESİNDE KULLANIM - TEK KOLON** 18](#_Toc195468050)

[**7.5.** **WHERE İFADESİNDE KULLANIM - BİRDEN FAZLA KOLON** 18](#_Toc195468051)

[**7.6.** **KOLON OLARAK KULLANIM** 18](#_Toc195468052)

[**7.7.** **HAVİNG İFADESİNDE KULLANIM** 18](#_Toc195468053)

[**7.8.** **BİRDEN FAZLA SATIR ALT SORGULAR** 18](#_Toc195468054)

[**7.9.** **TANIM** 18](#_Toc195468055)

[**7.10.** **FROM İFADESİNDE KULLANIM** 19](#_Toc195468056)

[**7.11.** **IN OPERATÖRÜ** 19](#_Toc195468057)

[**7.12.** **EXISTS OPERATÖRÜ** 19](#_Toc195468058)

[**7.13.** **NOT EXISTS OPERATÖRÜ** 19](#_Toc195468059)

[**8.** **DML - VERİLERİ DEĞİŞTİRME KOMUTLARI** 19](#_Toc195468060)

[**8.1.** **INSERT - TEK SATIR** 19](#_Toc195468062)

[**8.2.** **INSERT - RETURNİNG İFADESİ** 19](#_Toc195468063)

[**8.3.** **INSERT - BİRDEN FAZLA SATIR** 19](#_Toc195468064)

[**8.4.** **INSERT - SATIRLARI KOPYALAMA** 20](#_Toc195468065)

[**8.5.** **UPDATE - TEK SATIR** 20](#_Toc195468066)

[**8.6.** **UPDATE - BİRDEN FAZLA SATIR** 20](#_Toc195468067)

[**8.7.** **UPDATE - RETURNİNG** 20](#_Toc195468068)

[**8.8.** **UPDATE - JOİN** 20](#_Toc195468069)

[**8.9.** **DELETE - TEK SATIR** 20](#_Toc195468070)

[**8.10.** **DELETE - BİRDEN FAZLA SATIR** 20](#_Toc195468071)

[**8.11.** **DELETE - RETURNİNG** 20](#_Toc195468072)

[**8.12.** **DELETE - JOİN** 21](#_Toc195468073)

[**9.** **VERİTABANI TRANSACTİON YÖNETİMİ** 21](#_Toc195468074)

[**9.1.** **VERİTABANI TRANSACTİON'I NEDİR?** 21](#_Toc195468076)

[**9.2.** **VERİTABANI TRANSACTİON TİPLERİ** 21](#_Toc195468077)

[**9.3.** **TRANSACTİON'LARIN ÖZELLİKLERİ** 21](#_Toc195468078)

[**9.4.** **TRANSACTİON YÖNETME - COMMIT** 21](#_Toc195468079)

[**9.5.** **TRANSACTİON YÖNETME - ROLLBACK** 21](#_Toc195468080)

[**9.6.** **TRANSACTİON YÖNETME - SAVEPOINT** 21](#_Toc195468081)

[**9.7.** **COMMIT & ROLLBACK SİMULASYON** 22](#_Toc195468082)

[**9.8.** **COMMIT SONRASI** 22](#_Toc195468083)

[**9.9.** **SATIR KİLİTLEME** 22](#_Toc195468084)

[**9.10.** **ROLLBACK SONRASI** 23](#_Toc195468085)

[**9.11.** **ÖZET** 23](#_Toc195468086)

[**10.** **VERİ TİPLERİ VE TABLO OLUŞTURMA** 23](#_Toc195468087)

[**10.1.** **VERİ TİPLERİ** 23](#_Toc195468089)

[o **String Veri Tipleri** 23](#_Toc195468090)

[o **Numeric Veri Tipleri** 23](#_Toc195468091)

[o **Date-Time Veri Tipleri** 23](#_Toc195468092)

[**10.2.** **TABLO OLUŞTURMA METODLARI** 23](#_Toc195468093)

[o **Bir Tablo Oluşturma - Script Yazarak** 23](#_Toc195468094)

[o **Bir Tablo Oluşturma - SELECT INTO** 24](#_Toc195468095)

[o **Bir Tablo Oluşturma - CREATE TABLE AS (Sorgu)** 24](#_Toc195468096)

[o **Bir Tablo Oluşturma - CREATE TABLE AS (Tablo)** 24](#_Toc195468097)

[o **SERIAL Pseudo Tipi - Tanım** 24](#_Toc195468098)

[**10.3.** **VERİ TİPLERİ DETAYLARI** 24](#_Toc195468099)

[o **Numeric Veri Tipleri** 25](#_Toc195468100)

[o **Date-Time Veri Tipleri** 25](#_Toc195468101)

[**10.4.** **TABLO OLUŞTURMA METODLARI DETAYLARI** 25](#_Toc195468102)

[o **Bir Tablo Oluşturma - Script Yazarak** 25](#_Toc195468103)

[o **Bir Tablo Oluşturma - SELECT INTO** 25](#_Toc195468104)

[o **Bir Tablo Oluşturma - CREATE TABLE AS (Sorgu)** 25](#_Toc195468105)

[o **Bir Tablo Oluşturma - CREATE TABLE AS (Tablo)** 26](#_Toc195468106)

[o **SERIAL Pseudo Tipi - Tanım** 26](#_Toc195468107)

[**10.5.** **EKSTRA ÖRNEKLER VE KULLANIM SENARYOLARI** 26](#_Toc195468108)

[o **Veri Tipi Kontrolü** 26](#_Toc195468109)

[o **Tablo Yapısını Görüntüleme** 26](#_Toc195468110)

[o **Tablo Silme** 26](#_Toc195468111)

[**11.** **ALTER KOMUTLARI İLE TABLO YAPILARINDA DEĞİŞİKLİK** 26](#_Toc195468112)

[**11.1.** **BİR KOLON EKLEME (ADD COLUMN)** 26](#_Toc195468114)

[**11.2.** **BİR KOLON İSMİNİ DEĞİŞTİRME (RENAME COLUMN)** 27](#_Toc195468115)

[**11.3.** **DEFAULT DEĞERİ DEĞİŞTİRME (ALTER COLUMN SET DEFAULT)** 27](#_Toc195468116)

[**11.4.** **KOLON TİPİNİ DEĞİŞTİRME (ALTER COLUMN TYPE)** 27](#_Toc195468117)

[**11.5.** **BİR COMMENT (YORUM) EKLEME (COMMENT)** 28](#_Toc195468118)

[**11.6.** **NOT NULL KISITLAMASI EKLEME-KALDIRMA** 28](#_Toc195468119)

[**11.7.** **KOLONLARI KALDIRMA (DROP COLUMN)** 28](#_Toc195468120)

[**11.8.** **BİR CONSTRAİNT EKLEME (ADD CONSTRAINT)** 28](#_Toc195468121)

[**11.9.** **CONSTRAİNT KALDIRMA (DROP CONSTRAINT)** 29](#_Toc195468122)

[**11.10.** **TABLO İSMİNİ DEĞİŞTİRME (RENAME TO)** 29](#_Toc195468123)

[**11.11.** **BİR TABLOYU KALDIRMA (DROP TABLE)** 29](#_Toc195468124)

[**11.12.** **TRUNCATE KOMUTU** 29](#_Toc195468125)

[**11.13.** **POSTGRESQL ÖZEL NOTLARI:** 29](#_Toc195468126)

[**12.** **VERİTABANI KISITLAMALARI: CONSTRAİNTS** 29](#_Toc195468127)

[**12.1.** **PRİMARY KEY CONSTRAİNT (BİRİNCİL ANAHTAR KISITLAMASI)** 30](#_Toc195468129)

[**12.2.** **FOREİGN KEY CONSTRAİNT (YABANCI ANAHTAR KISITLAMASI)** 30](#_Toc195468130)

[**12.3.** **CHECK CONSTRAİNT (KONTROL KISITLAMASI)** 31](#_Toc195468131)

[**12.4.** **UNİQUE CONSTRAİNT (BENZERSİZ KISITLAMA)** 32](#_Toc195468132)

[**12.5.** **NOT NULL CONSTRAİNT (BOŞ OLMAZ KISITLAMASI)** 32](#_Toc195468133)

[**12.6.** **CONSTRAİNT'LERİ YÖNETME** 33](#_Toc195468134)

1. **VERİTABANI TEMELLERİ**

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

* 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. **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. **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. **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. **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. **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. **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. **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. **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. **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.

1. **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.

* 1. **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.
  1. **KOLONLARI SEÇME**

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

* 1. **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
  1. **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**).
  1. **SELECT İFADESİ - SYNTAX**

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

**SELECT** column1, column2, ...

**FROM** table\_name

**WHERE** condition;

* 1. **TÜM KOLONLARI SEÇME**

Tüm kolonları seçmek için **\*** kullanılır.

**SELECT** \* **FROM** customer;

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

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.)**

* 1. **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.

* 1. **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.
14. **SQL SATIR VE VERİ OPERATÖRLERİ**

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

1. 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.

* 1. **OPERATÖR LİSTESİ**

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

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **NULL DEĞERİ**

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

* 1. **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.!!)

* 1. **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.

* 1. **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.

* 1. **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.

1. **SQL FONKSİYONLARI**

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

1. 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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

1. **GRUP FONKSİYONLARI VE GROUP BY**

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

1. 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.

* 1. **AVG FONKSİYONU**

**AVG** fonksiyonu, belirli bir sütunun ortalamasını hesaplar.

**SELECT AVG**(amount) **AS** AverageAmount

**FROM** payment;

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

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

1. **BİRDEN FAZLA TABLO ÜZERİNDE SORGULAMA**

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

1. 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.

* 1. **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.

* 1. **JOİN TİPLERİ**

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

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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;

* 1. **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;

* 1. **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.

* 1. **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;

* 1. **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.

* 1. **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.

* 1. **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.

1. **ALT SORGULARI KULLANMA**

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

1. 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.

* 1. **ALT SORGU TİPLERİ**

Alt sorgular, genel olarak iki ana tipe ayrılır:

* Tek Satır Alt Sorgular
* Birden Fazla Satır Alt Sorgular
  1. **TEK SATIR ALT SORGULAR**

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

* 1. **WHERE İFADESİNDE KULLANIM - TEK KOLON**

**SELECT** \*

**FROM** Employees

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

Bu sorgu, en yüksek maaşı alan çalışanları getirir.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **BİRDEN FAZLA SATIR ALT SORGULAR**

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

* 1. **TANIM**

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

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

1. **DML - VERİLERİ DEĞİŞTİRME KOMUTLARI**

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

1. 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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

1. **VERİTABANI TRANSACTİON YÖNETİMİ**

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

1. 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.

* 1. **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.

* 1. **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.
  1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

* 1. **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.

1. **VERİ TİPLERİ VE TABLO OLUŞTURMA**

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

1. 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';

* 1. **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.

* 1. **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**

);

* 1. **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**

);

* 1. **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**

1. **ALTER KOMUTLARI İLE TABLO YAPILARINDA DEĞİŞİKLİK**

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

1. 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**;

* 1. **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;

* 1. **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';

* 1. **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);

* 1. **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';

* 1. **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**;

* 1. **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;

* 1. **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);

* 1. **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;

* 1. **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;

* 1. **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;

* 1. **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**;

* 1. **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

1. **VERİTABANI KISITLAMALARI: CONSTRAİNTS**

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

* 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) );

* 1. **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
  1. **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)

);

* 1. **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)

);

* 1. **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**;

* 1. **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)

);

1. CONSTRAİNT SİLME

**ALTER** **TABLE** table\_name **DROP** **CONSTRAINT** constraint\_name;

1. 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.