Temel SQL Eğitimi

Umut ÇAKIR

umutcakirbm@gmail.com

Temel SQL eğitiminde neler öğreneceğiz?

- Veri, veritabanı, veritabanı yönetim sistemi ve SQL kavramları, ilişkileri ve farkları
- PostgreSQL ve yardımcı araçların kurulumu
- SQL temelleri ve örneklerle sorguların incelenmesi
- Tablolarla çalışmak, verilerin oluşturulması, güncellenmesi ve silinmesi
- Temel SQL veri tiplerinin incelenmesi
- SQL'de JOIN yapıları ve kullanım örnekleri
- Alt sorgular ve JOIN yapısı ile birlikte kullanım örnekleri

Veri ve Veritabanı

Veri: Ölçüm, sayım, deney, gözlem veya araştırma sonucuyla elde edilen ham bilgilerdir.

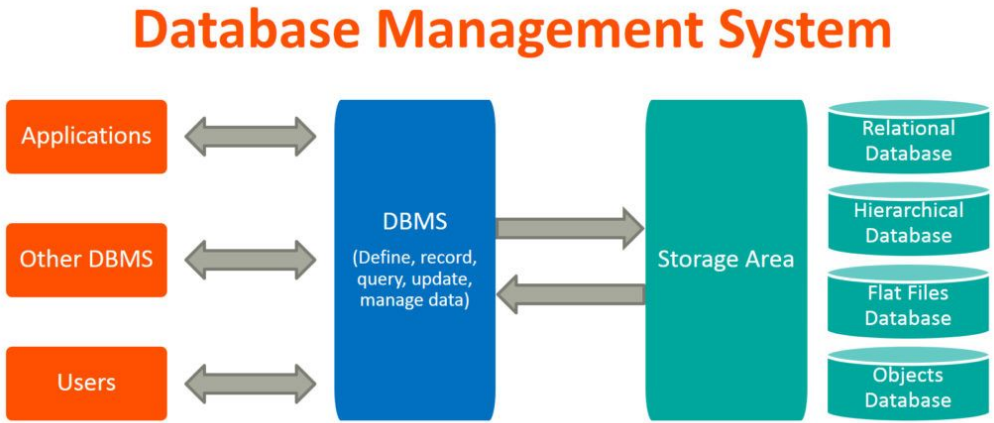
Veritabanı: Verilerin organize bir şekilde depolanmasını sağlayan sistemlerdir.

İlişkisel Veritabanı: İlişkisel veritabanı modeli, tablolardan ve bu tablolardaki veriler arasındaki ilişkilerden oluşur.

- **Verilerin Düzenli Saklanması:** Veritabanları, verileri yapılandırılmış bir şekilde saklama imkanı sağlar. Veriler, tablolar, sütunlar ve satırlar gibi yapılar içinde düzenlenir. Bu sayede verilerin mantıklı bir şekilde organize edilmesi ve daha kolay erişilebilmesi sağlanır.
- **Veri Bütünlüğü:** Veri tabanları, veri bütünlüğünü korumak için mekanizmalar sunar. Veri bütünlüğü, verilerin doğruluğunu ve tutarlılığını ifade eder.
- **Veri Erişimi:** Veritabanları, verilere hızlı ve etkili bir şekilde erişim sağlar. Veritabanlarına indeksleme gibi yöntemler uygulanarak verilere hızlı bir şekilde erişmek mümkün olur.
- **Veri Paylaşımı:** Veritabanları, birden çok kullanıcı arasında veri paylaşımını kolaylaştırır. Birden çok kullanıcı, aynı veritabanına erişebilir ve verilere güvenli bir şekilde erişim izinleri tanımlanabilir.
- **Veri Güvenliği:** Veri tabanları, verilerin güvenliğini sağlamak için çeşitli güvenlik mekanizmaları sunar. Verilere yetkilendirme ve erişim kontrolleri uygulanabilir.
- **Veri Yönetimi ve Analizi:** Veritabanları, verilerin etkin bir şekilde yönetilmesini ve analiz edilmesini sağlar. Veriler, sorgular ve raporlar kullanılarak işlenebilir, filtrelenir ve analiz edilir.

Veritabanı Yönetim Sistemi

- **Veritabanı yönetim sistemleri (DBMS)**, veri tabanlarının yönetimi ve işletilmesi için kullanılan yazılım sistemleridir. Bu sistemler, veritabanının oluşturulmasını, güncellenmesini, saklanmasını, erişimini ve yönetimini kolaylaştırır. Veritabanı yönetim sistemleri, veritabanı ile etkileşimde bulunarak kullanıcıların verilere erişmelerini, sorgulamalarını ve veri manipülasyonu yapmalarını sağlar.

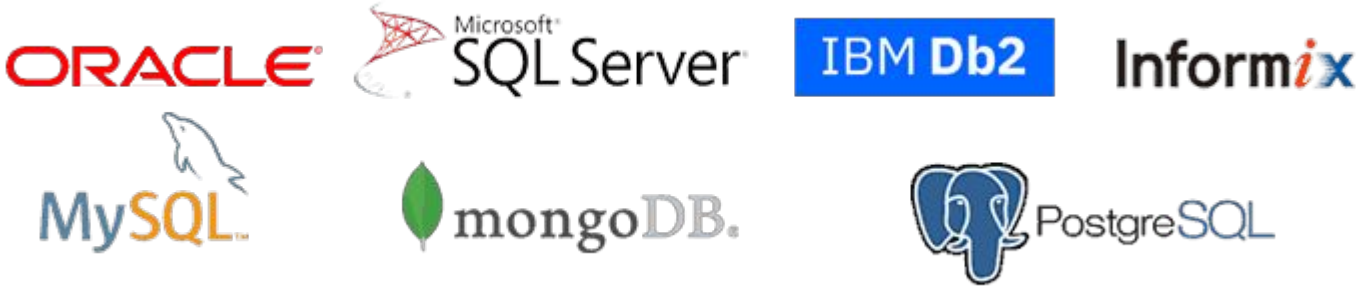


- Şekilde de gördüğümüz üzere farklı kaynaklardan gelen sorgular DBMS yazılımı sayesinde farklı veri tabanlarında kullanılır. Genel kullanım olarak pratikte veri, veritabanı ve veritabanı yönetim sisteminin tamamını **VERİTABANI** olarak adlandırmak şekilde bir eğilimimiz vardır.

Temel Veritabanı Yönetim Sistemleri

- **Hiyerarşik Veritabanı**
- **Ağ Veritabanı**
- **İlişkisel Veritabanı (RDBMS)**
- **Nesneye Yönelik Veritabanı**

Aşağıda popüler veritabanı yönetim sistemi yazılımlarını görebilirsiniz.



SQL Nedir?

SQL (Structured Query Language), ilişkisel veritabanı yönetim sistemlerinde (RDBMS) veri manipülasyonu ve veritabanı yönetimi için kullanılan bir sorgulama dilidir.

Verilerin sorgulanması, ekleme, güncelleme ve silme gibi işlemler SQL aracılığıyla gerçekleştirilir.

Veritabanı yönetim sistemleri ile etkileşim kurmak için kullanılır.

- **Veri Depolama ve Organizasyonu:** SQL, verileri düzenli bir şekilde depolamak için kullanılır. Veriler, tablolar halinde oluşturulur ve ilişkiler aracılığıyla birbiriyle bağlantılı hale getirilir.
- **Veri Sorgulama:** SQL, veriler üzerinde çeşitli sorgular yapabilmemizi sağlar. Veritabanında bulunan bilgilere erişmek, belirli kriterlere göre filtrelemek, verileri sıralamak ve gruplamak gibi işlemler SQL kullanılarak gerçekleştirilir.
- **Veri Manipülasyonu:** SQL, veritabanındaki verileri güncellemek, eklemek veya silmek için kullanılır.
- **Veritabanı Yönetimi:** SQL, veritabanı yönetim sistemleri (DBMS) tarafından kullanılır. Veritabanı yönetimi, verilerin depolanması, güvenliği, yedeklenmesi, kullanıcı erişim haklarının yönetimi gibi birçok görevi içerir.
- **Veri Analitiği:** SQL, büyük miktardaki veriler üzerinde karmaşık analizler yapabilmemizi sağlar. SQL sorguları kullanarak, verileri filtreleyebilir, gruplayabilir, birleştirebilir ve istatistiksel hesaplamalar yapabiliriz.

Bir Programlama Dili Olarak SQL

- SQL üzerine konuşulurken ilk olarak şu soru akla gelir. SQL bir programlama dili midir? Evet, SQL ilişkisel veritabanı yönetim sistemleri ile ilişki kurmamızı sağlayan bir **declarative bildirimsel** bir programlama dilidir.

SELECT title FROM book
WHERE page_number > 200;

- Yukarıdaki sorgumuzda, veritabanındaki book tablosundan sayfa sayısı 200 den daha fazla olan kitapları görmek istiyoruz. Burada biz işin sonuç kısmıyla ilgileniyoruz. SQL, DBMS ile nasıl çalışır, arka tarafta yapılan işlemin bizim açımızdan önemi yoktur. Bundan dolayı SQL declarative yani bildirimsel, beyan edici bir yaklaşıma sahiptir.

Dördüncü Nesil Programlama Dili

- SQL daha az kod yazarak ve daha çok belirli şablonlar kullanan bir programlama dili olarak dördüncü nesil bir programlama dilidir. Yapılması istenen işlemin her basamağının ayrıca kodlanmasına gerek duyulmaz.

PostgreSQL Kurulumu

Bilgisayarlarımıza PostgreSQL ve yardımcı araçların kurulumuna geçebiliriz.

PostgreSQL

Çalışmalarımız boyunca veritabanı yönetim sistemi olarak PostgreSQL kullanacağız. PostgreSQL açık kaynak kodlu ve popüler bir ilişkisel veritabanı yönetim sistemidir.

PostgreSQL, geniş bir veri tipleri yelpazesini destekler ve birçok ileri düzey özelliğe sahiptir.

- **Açık kaynak kodlu**
- **Çeşitli veri tipi desteği**
- **Güçlü dökümantasyon ve topluluk desteği**
- **Tüm işletim sistemlerine uygunluk**
- **ACID uyumluluk**
    - **Atomicity (Atomiklik):** Atomiklik, bir veritabanı işleminin tüm veya hiç olarak kabul edilmesini ifade eder.
    - **Consistency (Tutarlılık):** Tutarlılık, veritabanının belli bir kural setine uygun olmasını ifade eder.
    - **Isolation (İzolasyon):** İzolasyon, eş zamanlı olarak çalışan işlemlerin birbirlerini etkilememesini sağlar.
    - **Durability (Dayanıklılık):** Dayanıklılık, bir işlemin tamamlandıktan sonra yapılan değişikliklerin kalıcı olarak saklanacağını garanti eder.
- **Güvenlidir**
- **Büyük verilerle kolay çalışma**

SQL Temelleri I

Bu bölümde temel SQL sorgu sözdizimlerini ve komutlarını öğreneceğiz.

SELECT

SELECT komutu, veritabanından **veri seçmek ve sorgulamak** için kullanılan SQL'in en temel komutlarından biridir. SELECT komutunu kullanarak belirli sütunları **seçebilir**, **filtreleme** yapabilir, **sıralama** yapabilir ve veri tabanı üzerinde çeşitli işlemler gerçekleştirebilirsiniz. İşte SELECT komutunun kullanımına ilişkin örnekler:

- **Tüm Sütunları Seçmek:**
  SELECT * FROM tablo_adı;
  Bu komut, belirtilen tablodaki tüm sütunları seçer.

- **Belirli Sütunları Seçmek:**
  SELECT sütun1, sütun2 FROM tablo_adı;
  Bu komut, belirtilen tablodaki sadece belirli sütunları seçer.

İpucu: İhtiyaç duyulan verileri seçin: SELECT sorgusuyla yalnızca ihtiyaç duyulan sütunları seçin. Tüm sütunları seçmek yerine, yalnızca gerekli verileri seçmek, sorgunun performansını artırır ve ağ trafiğini azaltır.

WHERE ve Karşılaştırma Operatörleri

WHERE ifadesi, SQL sorgularında kullanılan ve belirli bir koşula uyan verileri seçmek için kullanılan bir yapıdır. Karşılaştırma operatörleri ise WHERE ifadesinde kullanılan koşulları belirlemek için kullanılan operatörlerdir.

Eşitlik Karşılaştırma Operatörleri

- **"=":** İki değer birbirine eşitse, koşul doğru olur.
  Örneğin: WHERE sütun = değer
  SELECT * FROM employees WHERE name = 'John';
  Bu sorgu, "name" sütunu 'John' olan çalışanları seçer.

- **"<>":** İki değer birbirine eşit değilse, koşul doğru olur.
  Örneğin: WHERE sütun <> değer
  SELECT * FROM employees WHERE age <> 30; veya;
  SELECT * FROM employees WHERE age != 30;
  Bu sorgu, "age" sütunu 30 olmayan çalışanları seçer.

```
id | name    | age | salary
---+---------+-----+-------
1  | John    | 30  | 5000
2  | Emma    | 28  | 6000
3  | Michael | 35  | 4500
4  | Sarah   | 32  | 5500
```

WHERE ve Karşılaştırma Operatörleri

İlişkisel Karşılaştırma Operatörleri

- **"<":** Sol tarafındaki değer, sağ tarafındaki değerden küçükse, koşul doğru olur. Örneğin: WHERE sütun < değer
  SELECT * FROM employees WHERE age < 30;
  Bu sorgu, "age" sütunu 30'dan küçük olan çalışanları seçer.

- **">":** Sol tarafındaki değer, sağ tarafındaki değerden büyükse, koşul doğru olur. Örneğin: ***WHERE sütun > değer***

```
id | name    | age | salary
---+---------+-----+-------
1  | John    | 30  | 5000
2  | Emma    | 28  | 6000
3  | Michael | 35  | 4500
4  | Sarah   | 32  | 5500
```

- **"<=":** Sol tarafındaki değer, sağ tarafındaki değere eşit veya küçükse, koşul doğru olur.
  Örneğin: ***WHERE sütun <= değer***

- **">=":** Sol tarafındaki değer, sağ tarafındaki değere eşit veya büyükse, koşul doğru olur.
  Örneğin: ***WHERE sütun >= değer***

WHERE ve Mantıksal Operatörler

WHERE ifadesiyle birlikte kullanılan mantıksal operatörler, SQL sorgularında birden fazla koşulu birleştirmek veya koşulları daha karmaşık şekillerde kontrol etmek için kullanılır.

- **"AND":** İki veya daha fazla koşulu birleştirmek için kullanılır ve tüm koşulların doğru olması gerektiğini belirtir. Örneğin:
  SELECT * FROM products
  WHERE category = 'Fruits' AND price < 2.00;

- **"OR":** İki veya daha fazla koşulu birleştirmek için kullanılır ve herhangi bir koşulun doğru olması durumunda sorgunun sonucunu döndürür.
  SELECT * FROM products
  WHERE category = 'Fruits' OR category = 'Snacks';

```
id | name       | category    | price
---+------------+-------------+-------
1  | Apple      | Fruits      | 2.50
2  | Banana     | Fruits      | 1.50
3  | Carrot     | Vegetables  | 0.75
4  | Tomato     | Vegetables  | 1.20
5  | Chocolate  | Snacks      | 3.00
```

- **"NOT":** Bir koşulun tersini almak için kullanılır. Yani, bir koşulun doğru olmadığı durumu kontrol etmek için kullanılır.
  SELECT * FROM products WHERE NOT (category = 'Vegetables' OR category = 'Snacks');

Uygulama 1

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan title ve description sütunlarındaki verileri sıralayınız.
- **film** tablosunda bulunan tüm sütunlardaki verileri film uzunluğu (length) 60 dan büyük VE 75 ten küçük olma koşullarıyla sıralayınız.
- **film** tablosunda bulunan tüm sütunlardaki verileri rental_rate 0.99 VE replacement_cost 12.99 VEYA 28.99 olma koşullarıyla sıralayınız.
- **customer** tablosunda bulunan first_name sütunundaki değeri 'Mary' olan müşterinin last_name sütunundaki değeri nedir?
- **film** tablosundaki uzunluğu(length) 50 ten büyük OLMAYIP aynı zamanda rental_rate değeri 2.99 veya 4.99 OLMAYAN verileri sıralayınız.

BETWEEN Operatörü

BETWEEN operatörü, bir değerin belirli bir aralıkta yer alıp almadığını kontrol etmek için kullanılan bir karşılaştırma operatörüdür. BETWEEN operatörü, belirtilen alt ve üst sınırlar arasında (sınırlar dahil) olan değerleri seçer.

- ***SELECT * FROM products WHERE price BETWEEN 1.00 AND 2.00;***
  Bu sorgu, "price" sütunu 1.00 ile 2.00 arasında olan ürünleri seçer. Sonuç olarak, "Banana" ve "Tomato" ürünleri seçilecektir.

- ***SELECT * FROM products WHERE price BETWEEN 1.00 AND 2.00 AND NOT category = 'Fruits';***
  Bu sorgu, "price" sütunu 1.00 ile 2.00 arasında olan ve "category" sütunu 'Fruits' olmayan ürünleri seçer. Sonuç olarak, "Tomato" ürünü seçilecektir.

id	name	category	price
1	Apple	Fruits	2.50
2	Banana	Fruits	1.50
3	Carrot	Vegetables	0.75
4	Tomato	Vegetables	1.20
5	Chocolate	Snacks	3.00

- ***SELECT * FROM products WHERE price NOT BETWEEN 1.00 AND 2.00;***
  Bu sorgu, "price" sütunu 1.00 ile 2.00 aralığına uymayan ürünleri seçer. Sonuç olarak, "Apple", "Carrot" ve "Chocolate" ürünleri seçilecektir.

IN Operatörü

Belirli bir liste içinde yer alan değerleri seçmek için kullanılan bir karşılaştırma operatörüdür. Bu operatör, WHERE ifadesiyle birlikte kullanılarak veritabanında belirli bir sütunda bulunan değerleri belirli bir listedeki değerlerle karşılaştırır.

- ***SELECT * FROM users WHERE city IN ('London', 'Paris', 'Berlin');***
  Bu sorgu, "city" sütunu 'London', 'Paris' veya 'Berlin' olan kullanıcıları seçecektir.

- ***SELECT * FROM users WHERE city NOT IN ('London', 'Paris');***
  Bu sorgu, "city" sütunu 'London' veya 'Paris' olmayan kullanıcıları seçer. Sonuç olarak, "Michael" ve "Sarah" kullanıcıları seçilecektir.

- ***SELECT * FROM users WHERE city IN ('London', 'Paris') AND age > 30;***
  Bu sorgu, "city" sütunu 'London' veya 'Paris' olan ve "age" sütunu 30'dan büyük olan kullanıcıları seçer. Sonuç olarak, yalnızca "Sarah" kullanıcısı seçilecektir.

id	name	age	city
1	John	30	London
2	Emma	28	Paris
3	Michael	35	Berlin
4	Sarah	32	Madrid

Uygulama 2

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan tüm sütunlardaki verileri replacement cost değeri 12.99 dan büyük eşit ve 16.99 dan küçük eşit olma koşuluyla sıralayınız (BETWEEN - AND yapısını kullanınız.)

- **actor** tablosunda bulunan first_name ve last_name sütunlardaki verileri first_name 'Penelope' veya 'Nick' veya 'Ed' değerleri olması koşuluyla sıralayınız. (IN operatörünü kullanınız.)

- **film** tablosunda bulunan tüm sütunlardaki verileri rental_rate 0.99, 2.99, 4.99 VE replacement_cost 12.99, 15.99, 28.99 olma koşullarıyla sıralayınız. (IN operatörünü kullanınız.)

LIKE ve ILIKE Operatörleri

LIKE operatörü, SQL sorgularında metin veya karakter dizilerini belirli bir desene göre aramak veya eşleştirmek için kullanılan bir karşılaştırma operatörüdür. LIKE operatörü, genellikle metinsel verilerde desen tabanlı aramalar yapmak için kullanılır. Büyük/küçük harf kullanımına duyarlıdır.

LIKE operatörü, iki önemli joker karakteri olan "%" (yüzde) ve "_" (alt çizgi) ile birlikte kullanılır:

- **"%" (yüzde):** Bu joker karakter, herhangi bir karakter veya karakter dizisi yerine kullanılabilir. Sorgulamada "%abc%" deseni, "abc" karakter dizisinin herhangi bir yerde bulunduğu tüm değerleri eşleştirecektir.

- **"_" (alt çizgi):** Bu joker karakter, herhangi bir tek karakter yerine kullanılabilir. Sorgulamada "a_bc" deseni, "a" ile başlayan ve "bc" ile biten herhangi bir üç karakterlik diziyi eşleştirecektir (örneğin, "adbc" veya "a1bc").

SELECT * FROM users WHERE name LIKE 'J%';
Bu sorgu, "name" sütunu "J" harfiyle başlayan kullanıcıları seçecektir.

LIKE ve ILIKE Operatörleri

- ***SELECT * FROM users WHERE name LIKE '%hn';***
  Bu sorgu, "name" sütunu "hn" ile biten kullanıcıları seçecektir.

- ***SELECT * FROM users WHERE name LIKE '_o%n';***
  Bu sorgu, "name" sütunu herhangi bir karakter ile başlayan ve ardından "o" ile devam eden, sonunda da "n" ile biten kullanıcıları seçecektir.

```
id | name     | age | city
---+----------+-----+------
1  | John     | 30  | London
2  | Emma     | 28  | Paris
3  | Michael  | 35  | Berlin
4  | Sarah    | 32  | Madrid
```

- ***SELECT * FROM users WHERE name LIKE '100\%%';***
  Bu sorgu, "name" sütunu "100%" ile başlayan kullanıcıları seçecektir. "\" işareti kaçış işareti olarak kullanılır. "%" ve "_" karakterlerini içeren bir metni aramak istediğimizde öncesinde kaçış işareti kullanmamız gerekir.

ILIKE operatörü LIKE operatörünün case - insensitive versiyonudur.

Uygulama 3

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **country** tablosunda bulunan country sütunundaki ülke isimlerinden 'A' karakteri ile başlayıp 'a' karakteri ile sonlananları sıralayınız.

- **country** tablosunda bulunan country sütunundaki ülke isimlerinden en az 6 karakterden oluşan ve sonu 'n' karakteri ile sonlananları sıralayınız.

- **film** tablosunda bulunan title sütunundaki film isimlerinden en az 4 adet büyük ya da küçük harf farketmesizin 'T' karakteri içeren film isimlerini sıralayınız.

- **film** tablosunda bulunan tüm sütunlardaki verilerden title 'C' karakteri ile başlayan ve uzunluğu (length) 90 dan büyük olan ve rental_rate 2.99 olan verileri sıralayınız.

DISTINCT Operatörü

DISTINCT operatörü, SQL sorgularında yinelenen verileri ortadan kaldırmak için kullanılan bir operatördür. DISTINCT operatörü, bir veya daha fazla sütunun değerlerini inceleyerek yalnızca benzersiz (tekrar etmeyen) değerleri döndürür.

- DISTINCT operatörü, özellikle bir sorgunun sonucunda yinelenen değerlerin çıkarılmasını istediğiniz durumlarda kullanılır. Örneğin, bir tablodaki tüm kullanıcıların benzersiz şehirlerini seçmek için DISTINCT operatörünü kullanabilirsiniz:

SELECT DISTINCT city
FROM users;
Bu sorgu, "users" tablosundaki tüm kullanıcıların benzersiz şehirlerini döndürecektir. Her şehir yalnızca bir kez listelenecektir ve tekrar eden şehirler atlanacaktır.

id	name	age	city
1	John	30	London
2	Emma	28	London
3	Michael	35	Berlin
4	Sarah	32	Madrid

COUNT Aggregate Fonksiyonu

COUNT operatörü, SQL sorgularında belirli bir sütundaki değerlerin sayısını döndürmek için kullanılan bir agregat (toplulama) fonksiyonudur. COUNT operatörü, bir tablodaki veya bir sorgunun sonucundaki satır sayısını hesaplamak için kullanılabilir.

Örneğin, "orders" tablosunda "completed" durumuna sahip siparişlerin sayısını hesaplamak için COUNT operatörünü ve WHERE ifadesini kullanabilirsiniz:

- ***SELECT COUNT(*) FROM orders***
  WHERE status = 'completed';
  Bu sorgu, "orders" tablosundaki "status" sütunu 'completed' olan siparişlerin sayısını döndürecektir.

id	customer_name	status	city
1	John	completed	London
2	Emma	completed	London
3	Michael	failed	Berlin
4	Sarah	completed	Madrid

Uygulama 4

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan replacement_cost sütununda bulunan birbirinden farklı değerleri sıralayınız.
- **film** tablosunda bulunan replacement_cost sütununda birbirinden farklı kaç tane veri vardır?
- **film** tablosunda bulunan film isimlerinde (title) kaç tanesini T karakteri ile başlar ve aynı zamanda rating 'G' ye eşittir?
- **country** tablosunda bulunan ülke isimlerinden (country) kaç tanesi 5 karakterden oluşmaktadır?
- **city** tablosundaki şehir isimlerinin kaç tanesi 'R' veya r karakteri ile biter?

IS NULL ve IS NOT NULL Operatörleri

IS NULL ve IS NOT NULL operatörleri, SQL sorgularında NULL değerlerini kontrol etmek için kullanılan karşılaştırma operatörleridir.

- **IS NULL:** Bu operatör, bir sütunun NULL (boş) değerlere sahip olup olmadığını kontrol eder. Bir sütunun değeri NULL ise, IS NULL operatörü doğru sonucunu döndürür.

- **IS NOT NULL:** Bu operatör, bir sütunun NULL değerlere sahip olmadığını kontrol eder. Bir sütunun değeri NULL değilse, IS NOT NULL operatörü doğru sonucunu döndürür.

Bu operatörler aşağıdaki gibi bir formatta kullanılır:

SELECT * FROM orders WHERE status IS NULL;
SELECT * FROM orders WHERE status IS NOT NULL;

id	order_number	customer	status
1	1001	John Smith	Shipped
2	1002	Lisa Brown	Pending
3	1003	David Lee	Shipped
4	1004	Emma Davis	NULL
5	1005	Michael Kim	Pending

PSQL ve Uygulama I

Terminal tabanlı kullanıcı arayüzünü kullanarak temel SQL komutlarını uygulayacağız.

Terminale Erişim

PSQL, PostgreSQL ile birlikte gelen terminal tabanlı bir kullanıcı arayüzüdür. PSQL sayesinde komut satırında sorgular yazıp, sonuçlarını görebiliriz. Yanda temel PSQL komutlarının ilk bölümünü bulabilirsiniz.

- **PSQL ile PostgreSQL'e bağlanmak:**
  psql -U <kullanıcı_adı>
- **Kullanıcıya ait şifreyi girdikten sonra varsayılan veritabanı postgres'e bağlanıyor.**
  postgres=#
- **Bulunan veritabanlarını listelemek için:**
  \l veya \list
- **Bizim örneğimizde dvdrental veritabanına bağlanacağız.**
  \c dvdrental veya \connect dvdrental
- **Bağlanılan dvdrental veritabanında bulunan tabloları listelemek için:**
  \dt
- **Herhangi bir tablonun sütunlarını ve tablo detaylarını görmek için:**
  \d <tablo_adı>
- **PSQL terminal ekranından çıkmak için:**
  \q

SQL Temelleri II

Bu bölümde temel SQL sorgu sözdizimleri ve komutlarını öğrenmeye devam edeceğiz.

ORDER BY Operatörü

ORDER BY operatörü, SQL sorgularında sonuç kümesinin belirli bir sütuna veya sütunlara göre sıralanmasını sağlayan bir operatördür. Sorgu sonucunu belirli bir düzende göstermek için kullanılır.

- ***SELECT * FROM employees ORDER BY last_name;***
  Bu sorgu, employees tablosundaki tüm satırları "last_name" sütununa göre artan (alfabetik) sırada döndürecektir. Eğer ters sıralama isterseniz, ORDER BY ifadesinin sonuna DESC (descending) ekleyebilirsiniz:

- ***SELECT * FROM employees ORDER BY last_name DESC;***
  Bu durumda, sonuçlar "last_name" sütununa göre azalan (ters alfabetik) sırada dönecektir.

id	first_name	last_name	hire_date	salary
1	John	Doe	2020-01-15	50000
2	Jane	Smith	2019-06-10	60000
3	Mike	Johnson	2021-03-05	55000
4	Emily	Davis	2020-08-20	52000

- ***SELECT * FROM employees ORDER BY last_name, first_name;***
  Bu sorgu, önce "last_name" sütununa göre sıralama yapacak ve ardından aynı soyadına sahip çalışanları "first_name" sütununa göre sıralayacaktır.

LIMIT ve OFFSET Operatörleri

LIMIT operatörü, SQL sorgularında sonuç kümesinin belirli bir sayıda satırını almak için kullanılan bir operatördür. Sorgu sonucunda dönen verileri sınırlamak için kullanılır.

OFFSET operatörü, SQL sorgularında sonuç kümesindeki satırları atlamak için kullanılan bir operatördür. Sorgu sonucunda dönen verilerin belirli bir başlangıç noktasından itibaren alınmasını sağlar.

- ***SELECT * FROM employees LIMIT 5;***
  Bu sorgu, employees tablosundaki ilk 5 satırı döndürecektir.

- ***SELECT * FROM employees LIMIT 10 OFFSET 5;***
  Bu sorgu, employees tablosundaki 6. satırdan başlayarak toplamda 10 satırı döndürecektir. Bu şekilde, 6. satırdan 15. satıra kadar olan bir aralığı elde edersiniz.

id	first_name	last_name	hire_date	salary
1	John	Doe	2020-01-15	50000
2	Jane	Smith	2019-06-10	60000
3	Mike	Johnson	2021-03-05	55000
4	Emily	Davis	2020-08-20	52000

- ***SELECT * FROM employees WHERE hire_date >= '2022-01-01' ORDER BY hire_date LIMIT 3;***
  Bu sorgu, employees tablosunda "hire_date" sütunu 2022-01-01'den büyük veya eşit olan çalışanları hire_date'e göre sıralar ve en yeni 3 çalışanın bilgilerini döndürür.

Uygulama 5

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan ve film ismi (title) 'n' karakteri ile biten en uzun (length) 5 filmi sıralayınız.
- **film** tablosunda bulunan ve film ismi (title) 'n' karakteri ile biten en kısa (length) ikinci(6,7,8,9,10) 5 filmi(6,7,8,9,10) sıralayınız.
- **customer** tablosunda bulunan last_name sütununa göre azalan yapılan sıralamada store_id 1 olmak koşuluyla ilk 4 veriyi sıralayınız.

Aggregate Fonksiyonlar

Aggregate fonksiyonlar, SQL sorgularında toplu veri işlemleri gerçekleştirmek için kullanılan fonksiyonlardır. Verileri gruplayarak, bir sütundaki değerler üzerinde hesaplamalar yapar ve tek bir sonuç döndürürler.

- **AVG**: Bir sütundaki sayısal değerlerin ortalamasını döndürür.
  SELECT AVG(price) FROM products WHERE category = 'Electronics';
- **SUM**: Bir sütundaki sayısal değerlerin toplamını döndürür.
  SELECT SUM(price) FROM products WHERE category = 'Clothing';
- **MIN**: Bir sütundaki en küçük değeri döndürür.
  SELECT MIN(price) FROM products;
- **MAX**: Bir sütundaki en büyük değeri döndürür.
  SELECT MAX(price) FROM products WHERE category = 'Electronics';

id	product_name	category	price
1	iPhone	Electronics	1000
2	Galaxy	Electronics	900
3	iPad	Electronics	800
4	MacBook	Electronics	1500
5	T-Shirt	Clothing	20
6	Jeans	Clothing	50
7	Sneakers	Clothing	80
8	Watch	Accessories	200

Uygulama 6

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan rental_rate sütunundaki değerlerin ortalaması nedir?
- **film** tablosunda bulunan filmlerden kaç tanesi 'C' karakteri ile başlar?
- **film** tablosunda bulunan filmlerden rental_rate değeri 0.99 a eşit olan en uzun (length) film kaç dakikadır?
- **film** tablosunda bulunan filmlerin uzunluğu 150 dakikadan büyük olanlarına ait kaç farklı replacement_cost değeri vardır?

GROUP BY Operatörü

GROUP BY operatörü, SQL sorgularında verileri belirli bir sütuna göre gruplamak için kullanılan bir operatördür. Verileri gruplara ayırarak, her bir gruptaki satırlar üzerinde toplu işlemler yapılmasını sağlar. SELECT anahtar kelimesinde bulunan sütunların GROUP BY anahtar kelimesi içerisinde bulunması gerekir.

- Ürün bazında toplam satış miktarını bulmak için:
  SELECT product_id, SUM(quantity) as total_sales FROM orders GROUP BY product_id;

- Müşteri bazında toplam sipariş sayısını bulmak için:
  SELECT customer_id, COUNT(*) as total_orders FROM orders GROUP BY customer_id;

id	customer_id	product_id	quantity	order_date
1	1	1	5	2022-01-01
2	2	2	3	2022-02-01
3	1	3	2	2022-02-15
4	3	1	4	2022-03-01
5	2	3	6	2022-03-15

- Müşteri ve ürün bazında toplam satış miktarını bulmak için:
  SELECT customer_id, product_id, SUM(quantity) as total_sales FROM orders GROUP BY customer_id, product_id;

- Ürün bazında ortalama satış miktarını bulmak için:
  SELECT product_id, AVG(quantity) as average_sales FROM orders GROUP BY product_id;

HAVING Operatörü

HAVING operatörü, SQL sorgularında GROUP BY operatörü ile gruplanmış veriler üzerinde filtreleme yapmak için kullanılan bir operatördür. HAVING, GROUP BY 'dan sonra kullanılır ve WHERE ile benzerlik gösterir. Ancak, WHERE sorgusu, veritabanı tablosunun satırlarını filtrelemek için kullanılırken, HAVING sorgusu gruplanmış sonuçları filtrelemek için kullanılır.

- Toplam satış miktarı 10'dan büyük olan ürün gruplarını bulmak için:
  SELECT product_id, SUM(quantity) as total_sales
  FROM orders
  GROUP BY product_id HAVING SUM(quantity) > 10;

id	customer_id	product_id	quantity	order_date
1	1	1	5	2022-01-01
2	2	2	3	2022-02-01
3	1	3	2	2022-02-15
4	3	1	4	2022-03-01
5	2	3	6	2022-03-15

- Müşteriye ait toplam sipariş sayısı 2'den fazla olan müşteri gruplarını bulmak için:
  SELECT customer_id, COUNT(*) as total_orders FROM orders GROUP BY customer_id HAVING COUNT(*) > 2;

- Müşteri ve ürün bazında toplam satış miktarı 10'dan büyük olan kombinasyonları bulmak için:
  SELECT customer_id, product_id, SUM(quantity) as total_sales FROM orders GROUP BY customer_id, product_id HAVING SUM(quantity) > 10;

Uygulama 7

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda bulunan filmleri rating değerlerine göre gruplayınız.
- **film** tablosunda bulunan filmleri replacement_cost sütununa göre grupladığımızda film sayısı 50 den fazla olan replacement_cost değerini ve karşılık gelen film sayısını sıralayınız.
- **customer** tablosunda bulunan store_id değerlerine karşılık gelen müşteri sayılarını nelerdir?
- **city** tablosunda bulunan şehir verilerini country_id sütununa göre gruplandırdıktan sonra en fazla şehir sayısı barındıran country_id bilgisini ve şehir sayısını paylaşınız.

ALIAS Operatörü

Alias (Takma ad) operatörü, SQL sorgularında sütunlar, tablolar veya aggregate fonksiyonlar için alternatif bir isim veya takma ad belirlemek için kullanılan bir özelliktir. Alias operatörü, sorgunun sonucunda dönen verilerin okunabilirliğini artırır ve sorguyu daha anlaşılır hale getirir.

- Sütunlar için takma adlar belirlemek:
  SELECT first_name AS isim, last_name AS soyisim, age AS yas FROM customers;

- Tablolar için takma adlar belirlemek:
  SELECT c.first_name, o.order_date FROM customers AS c
  JOIN orders AS o ON c.id = o.customer_id;

- Sütunlar için takma adlar belirleyerek aggregate fonksiyonlar kullanmak:
  SELECT COUNT(*) AS toplam_musteri_sayisi, AVG(age) AS ortalama_yas FROM customers;

id	first_name	last_name	age	city
1	John	Doe	30	London
2	Jane	Smith	25	Paris
3	Mike	Johnson	35	Berlin

Tablolarla Çalışmak

Bu bölümde tablo oluşturma, güncelleme ve silme işlemlerini, temel veri tiplerini inceleyeceğiz.

Tablo Oluşturmak

SQL'de tablo oluşturmak için CREATE TABLE ifadesi kullanılır. CREATE TABLE ifadesiyle yeni bir tablo tanımlanır ve tablo için sütunlar ve sütun özellikleri belirlenir.

```
CREATE TABLE  (IF NOT EXISTS) customers (
  id SERIAL PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  age INTEGER,
  city VARCHAR(50)
);
```

Bu örnekte "customers" adında bir tablo oluşturulur. Tablonun sütunları "id", "first_name", "last_name", "age" ve "city" olarak belirlenir. Sütunların veri tipleri ve özellikleri belirtilir.

Tablo Silmek

SQL’de bir tabloyu silmek için DROP TABLE ifadesi kullanılır. DROP TABLE ifadesi, belirtilen tabloyu veritabanından tamamen kaldırır.

Örneğin, “customers” adında bir tabloyu silmek için aşağıdaki SQL ifadesini kullanabilirsiniz:

- ***DROP TABLE customers;***
  veya
- ***DROP TABLE IF EXISTS customers;***

Tabloya Veri Eklemek

Veritabanına veri eklemek için SQL'de kullanılan anahtar kelime INSERT INTO'dur. INSERT INTO ifadesi, bir tabloya yeni bir satır veya kayıt eklemek için kullanılır. İşlemin genel formatı şu şekildedir:
INSERT INTO tablo_adı (sütun1, sütun2, ...)
VALUES (değer1, değer2, ...);

- "customers" tablosuna yeni bir müşteri ekleme:
  INSERT INTO customers (first_name, last_name, age) VALUES ('John', 'Doe', 30);
  Bu örnekte, "customers" tablosuna "first_name", "last_name" ve "age" sütunlarına sahip bir müşteri ekliyoruz. Müşterinin adı 'John', soyadı 'Doe' ve yaşı 30 olarak belirtiliyor.

- "products" tablosuna yeni bir ürün ekleme:
  INSERT INTO products VALUES (1, 'Laptop', 'Electronics', 1500.00, 'In stock');
  Bu örnekte, "products" tablosuna tüm sütunlara değerler vererek bir ürün ekliyoruz. İlk sütun "id", ikinci sütun "name", üçüncü sütun "category", dördüncü sütun "price" ve beşinci sütun "status" değerlerini belirliyoruz.

Tablodaki Veriyi Güncellemek

Tablodaki verileri güncellemek için SQL'de kullanılan anahtar kelime UPDATE'dir. UPDATE ifadesi, belirli bir tablodaki belirli sütunlardaki verileri değiştirmek için kullanılır.

- "customers" tablosunda belirli bir müşterinin adını güncelleme:
  UPDATE customers SET first_name = 'Jane'
  WHERE id = 1;
  Bu örnekte, "customers" tablosunda "id" değeri 1 olan müşterinin "first_name" sütununu "Jane" olarak güncelliyoruz.

- "products" tablosunda stok miktarını artırma:
  UPDATE products
  SET stock = stock + 10
  WHERE category = 'Electronics';
  Bu örnekte, "products" tablosunda "category" değeri "Electronics" olan ürünlerin "stock" sütununu 10 birim artırıyoruz.

Tablodaki Veriyi Silmek

Tablodaki verileri silmek için SQL'de kullanılan anahtar kelime DELETE'dir. DELETE ifadesi, belirli bir tablodan belirli satırları silmek için kullanılır.

- "customers" tablosunda belirli bir müşteriyi silme:
  DELETE FROM customers WHERE id = 1;
  Bu örnekte, "customers" tablosunda "id" değeri 1 olan müşteriyi sileriz.

- "products" tablosunda belirli bir kategoriye ait ürünleri silme:
  DELETE FROM products
  WHERE category = 'Electronics';
  Bu örnekte, "products" tablosunda "category" değeri "Electronics" olan ürünleri sileriz.

Uygulama 8

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **test** veritabanınızda employee isimli sütun bilgileri id(INTEGER), name VARCHAR(50), birthday DATE, email VARCHAR(100) olan bir tablo oluşturalım.
- Oluşturduğumuz employee tablosuna **'Mockaroo'** servisini kullanarak 50 adet veri ekleyelim.
- Sütunların her birine göre diğer sütunları güncelleyecek 5 adet UPDATE işlemi yapalım.
- Sütunların her birine göre ilgili satırı silecek 5 adet DELETE işlemi yapalım.

PRIMARY KEY

PRIMARY KEY bir tabloda bulunan veri sıralarını birbirinden ayırmamızı sağlayan bir kısıtlama (constraint) yapısıdır. O tabloda bulunan veri sıralarına ait bir "benzersiz tanımlayıcıdır".

- PRIMARY KEY, bir tablodaki bir veya birden fazla sütunu benzersiz bir şekilde tanımlayan bir kısıtlamadır.
- PRIMARY KEY, birincil anahtar olarak da adlandırılır ve tablodaki her bir satırın benzersiz bir şekilde tanımlanmasını sağlar.
- PRIMARY KEY, bir tablodaki sütunlardan oluşur ve birincil anahtar olarak belirlenen sütun veya sütunların kombinasyonu olabilir.
- PRIMARY KEY kısıtlaması sayesinde, birincil anahtar sütununda tekrarlayan veya boş değerlerin bulunmasına izin verilmez.

```
CREATE TABLE customers (
  id SERIAL PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  email VARCHAR(100)
);
```

FOREIGN KEY

FOREIGN KEY bir tabloda bulunan herhangi bir sütundaki verilerin genelde başka bir tablo sütununa referans vermesi durumudur, tablolar arası ilişki kurulmasını sağlar.

- FOREIGN KEY, bir tablodaki bir veya birden fazla sütunu, başka bir tablonun birincil anahtar sütunuyla ilişkilendiren bir kısıtlamadır.
- FOREIGN KEY, tablolar arasında ilişki kurmaya ve referanslar oluşturmaya olanak sağlar.
- FOREIGN KEY kısıtlaması sayesinde, ilişkili tablolar arasında veri bütünlüğü sağlanır ve referanslanan tablodaki değerlerin değiştirilmesi veya silinmesi durumunda uygun önlemler alınır.
- FOREIGN KEY, ilişkili tablonun PRIMARY KEY veya UNIQUE kısıtlamalarına referans alır.

"orders" tablosunda "customer_id" sütunu "customers" tablosundaki "id" sütunu ile FOREIGN KEY kısıtlaması ile ilişkilendirilmiştir:

```
CREATE TABLE orders (
  order_id SERIAL PRIMARY KEY,
  order_date DATE,
  customer_id INTEGER,
  FOREIGN KEY (customer_id) REFERENCES customers (id)
);
```

Temel Veri Tipleri

- Sayısal Veri Tipleri
- Karakter Veri Tipleri
- Boolean Veri Tipleri
- Date / Time Veri Tipleri

İsim	Range
smallint	-32768 to +32767
integer	-2147483648 to +2147483647
bigint	-9223372036854775808 to +9223372036854775807
decimal	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	6 decimal digits precision
double precision	15 decimal digits precision
smallserial	1 to 32767
serial	1 to 2147483647
bigserial	1 to 9223372036854775807

Sayısal Veri Tipleri

SQL' de temel sayısal veri tipleri, sayısal değerleri saklamak ve işlemek için kullanılan veri tipleridir. Bu veri tipleri, farklı boyutlar, hassasiyetler ve aralıklarla birlikte gelir.

- **INTEGER (INT):** Tam sayıları temsil etmek için kullanılır. Pozitif ve negatif tam sayı değerlerini içerebilir. Örnek: -10, 0, 5, 100.
- **SMALLINT**: Daha küçük bir tam sayı veri tipidir. INTEGER'dan daha küçük bir aralıkta değerler alır.
- **BIGINT**: Daha büyük bir tam sayı veri tipidir. INTEGER'dan daha geniş bir aralıkta değerler alır.
- **DECIMAL(p, s):** Hassas ondalık sayıları temsil etmek için kullanılır. p parametresi, toplam basamak sayısını belirtirken s parametresi, ondalık basamak sayısını belirtir. Örnek: DECIMAL(8, 2) - 12345.67.
- **NUMERIC(p, s):** DECIMAL ile aynıdır, ancak bazı veritabanlarında farklı bir isim olarak kullanılır.
- **FLOAT(p):** Yaklaşık değerleri temsil etmek için kullanılır. p parametresi, kayan nokta sayılarının hassasiyetini belirtir. Örnek: FLOAT(4) - 3.14.
- **REAL:** Tek hassasiyetli kayan nokta sayıları için kullanılır.
- **DOUBLE PRECISION:** Çift hassasiyetli kayan nokta sayıları için kullanılır.

Veri tiplerini doğru bir şekilde seçmek, verilerin doğru bir şekilde saklanmasını ve işlenmesini sağlar.

Karakter Veri Tipleri

SQL'de temel karakter veri tipleri, metinsel veya karakter tabanlı verileri saklamak ve işlemek için kullanılan veri tipleridir. Bu veri tipleri, farklı uzunluklar, kapasiteler ve kullanımlarla birlikte gelir.

- **CHAR(n):** Sabit bir karakter dizisini temsil etmek için kullanılır. n parametresi, karakter dizisinin maksimum uzunluğunu belirtir. Eğer veri belirtilen uzunluktan daha kısa ise, geri kalan boşluklarla doldurulur. Örnek: CHAR(10) - "Hello".
- **VARCHAR(n):** Değişken bir karakter dizisini temsil etmek için kullanılır. n parametresi, karakter dizisinin maksimum uzunluğunu belirtir. Eğer veri belirtilen uzunluktan daha kısa ise, boşluklarla doldurulmaz. Örnek: VARCHAR(20) - "Hello World".
- **TEXT:** Değişken uzunluklu karakter dizilerini temsil etmek için kullanılır. Uzun metinlerin saklanması için kullanılır ve genellikle uzun karakter dizileri için tercih edilir.

Bu karakter veri tipleri, SQL 'de metinsel verileri temsil etmek ve işlemek için kullanılır. Her bir veri tipi, belirli bir kullanım senaryosuna ve veri uzunluğuna uygun olarak seçilir. Veri tiplerini doğru bir şekilde belirlemek, metin tabanlı verilerin doğru bir şekilde saklanmasını ve işlenmesini sağlar.

Mantıksal (Boolean) Veri Tipleri

SQL'de temel boolean veri tipleri, mantıksal değerleri saklamak ve işlemek için kullanılan veri tipleridir. Bu veri tipleri, yalnızca iki değeri (true veya false) alabilir.

- **BOOLEAN**: true veya false değerini temsil eder. Bu veri tipi, bir sütunda yalnızca iki olası değeri saklamak için kullanılır.

Örnek:

```
CREATE TABLE employees (
  id INTEGER,
  name VARCHAR(50),
  is_active BOOLEAN
);
```

Yukarıdaki örnekte, "employees" tablosunda "is_active" adında bir BOOLEAN türünde bir sütun oluşturulmuştur. Bu sütun, çalışanın aktif olup olmadığını temsil eder.

Date / Time Veri Tipleri

SQL' de temel date/time veri tipleri, tarih ve saat bilgisini saklamak ve işlemek için kullanılan veri tipleridir. Bu veri tipleri, farklı tarih ve saat değerlerini temsil etmek için kullanılır.

- **DATE**: Yalnızca tarihi temsil etmek için kullanılır. Örnek: '2023-05-19'.
- **TIME**: Yalnızca saati temsil etmek için kullanılır. Örnek: '10:30:00'.
- **DATETIME** veya **TIMESTAMP**: Hem tarihi hem de saati temsil etmek için kullanılır. Örnek: '2023-05-19 10:30:00'.
- **TIMESTAMP WITH TIME ZONE:** Tarih ve saati temsil ederken sunucunun saat dilimini de ekler. Örnek: ‘2023-05-19 10:30:00+03:00’
- **TIME WITH TIME ZONE:** Saati temsil ederken sunucunun saat dilimini de ekler. Örnek: ‘10:30:00+03:00’

Bu date / time veri tipleri, tarih ve saat değerlerini saklamak ve işlemek için kullanılır. İşlemler, tarih ve saat değerlerini karşılaştırmak, aralıkları hesaplamak, sıralamak ve diğer tarih/saat manipülasyonlarını gerçekleştirmek için kullanılabilir.

NOT NULL Kısıtlaması

NOT NULL, SQL'de bir sütuna değer girilmesini zorunlu kılan bir kısıtlamadır. Bir sütunun NOT NULL olarak belirtilmesi, o sütuna her zaman bir değer girmeniz gerektiğini ifade eder. Eğer bir sütunun NOT NULL olarak tanımlanmasına rağmen bir değer girilmezse veya NULL olarak belirtilirse, bir hata oluşur ve veritabanı bu durumu reddeder.

NOT NULL kısıtlaması, veri bütünlüğünü sağlamak için önemlidir. Bu kısıtlama, veritabanında eksik veya boş değerlerin oluşmasını engeller ve sütunlara her zaman geçerli bir değer girilmesini sağlar.

```
CREATE TABLE employees (
  id INTEGER NOT NULL,
  first_name VARCHAR(50) NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  age INTEGER
);
```

Yukarıdaki örnekte, "employees" tablosunda "id", "first_name" ve "last_name" sütunları NOT NULL olarak tanımlanmıştır. Bu sütunlara her zaman değer girilmesi gerekmektedir.

ALTER

ALTER, SQL'de mevcut bir tablo veya sütunu değiştirmek için kullanılan bir ifadedir. ALTER, tablonun veya sütunun yapısal özelliklerini değiştirmenize olanak sağlar. ALTER ifadesi, tablo veya sütun ekleme, silme, yeniden adlandırma, sütun türünü veya boyutunu değiştirme gibi işlemleri gerçekleştirmenizi sağlar.

- Bir sütuna NOT NULL kısıtlaması eklemek:
  ALTER TABLE employees ALTER COLUMN first_name SET NOT NULL;
  Bu örnekte, "employees" tablosundaki "first_name" sütununa NOT NULL kısıtlaması ekliyoruz. Bu, sütuna her zaman bir değer girilmesini zorunlu kılar.

- Bir sütunu silmek:
  ALTER TABLE employees DROP COLUMN age;
  Bu örnekte, "employees" tablosundaki "age" sütununu siliyoruz.

- Bir sütunu yeniden adlandırmak:
  ALTER TABLE employees RENAME COLUMN first_name TO employee_name;
  Bu örnekte, "employees" tablosundaki "first_name" sütununu "employee_name" olarak yeniden adlandırıyoruz.

ALTER

- Bir tabloya yeni bir sütun eklemek:
  ALTER TABLE employees ADD COLUMN email VARCHAR(100);
  Bu örnekte, "employees" tablosuna "email" adında bir VARCHAR sütunu ekliyoruz.

- Bir sütunun veri türünü değiştirmek:
  ALTER TABLE employees ALTER COLUMN age TYPE INTEGER;
  Bu örnekte, "employees" tablosundaki "age" sütununun veri türünü INTEGER olarak değiştiriyoruz.

- Bir tabloya PRIMARY KEY kısıtlaması eklemek:
  ALTER TABLE employees ADD PRIMARY KEY (id);
  Bu örnekte, "employees" tablosuna "id" sütunu üzerinde PRIMARY KEY kısıtlaması ekliyoruz.

- Bir tabloya FOREIGN KEY kısıtlaması eklemek:
  ALTER TABLE orders ADD FOREIGN KEY (customer_id) REFERENCES customers (id);
  Bu örnekte, "orders" tablosuna "customer_id" sütunu üzerinde "customers" tablosuna FOREIGN KEY kısıtlaması ekliyoruz.

ALTER

- Bir tablonun adını değiştirmek:
  ALTER TABLE employees
  RENAME TO staff;
  Bu örnekte, "employees" tablosunun adını "staff" olarak değiştiriyoruz.

- Bir sütunun adını değiştirmek:
  ALTER TABLE employees
  RENAME COLUMN first_name TO employee_name;
  Bu örnekte, "employees" tablosundaki "first_name" sütununun adını "employee_name" olarak değiştiriyoruz.

ALTER ifadesi, tablo veya sütun yapısında değişiklik yapmanıza olanak sağlar. Bu sayede mevcut veritabanı şemasını güncelleyebilir veya tablo/sütun yapılarını değiştirebilirsiniz.

UNIQUE Kısıtlaması

UNIQUE, SQL'de bir sütunda benzersiz (unique) değerlerin kabul edildiğini belirten bir kısıtlamadır. UNIQUE kısıtlaması, bir sütunda tekrarlayan değerlerin bulunmasını engeller. Bir sütunun UNIQUE olarak tanımlanmasıyla, o sütunda her bir değerin yalnızca bir kez bulunmasına izin verilir.

UNIQUE kısıtlaması, veri bütünlüğünü sağlamak ve tekrarlayan değerlerin ortaya çıkmasını önlemek için kullanılır. Bu kısıtlama, genellikle benzersiz kimlik, e-posta adresi, kullanıcı adı gibi alanlarda kullanılır.

Örneğin, aşağıdaki örnekte "users" tablosunda "username" sütunu UNIQUE olarak tanımlanmıştır:

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE,
  email VARCHAR(100)
);
```

Bu tabloda "username" sütunu UNIQUE olarak belirtilmiştir, bu nedenle her bir kullanıcıya benzersiz bir kullanıcı adı atanmalıdır.

UNIQUE Kısıtlaması

ALTER komutu ile UNIQUE kısıtlaması mevcut bir tabloya sonradan eklenmek istendiğinde, aşağıdaki örnek kullanılabilir:

ALTER TABLE users
ADD CONSTRAINT unique_username UNIQUE (username);
veya
ALTER TABLE users
ADD UNIQUE (username);

Bu komutla "users" tablosuna "username" sütunu üzerinde UNIQUE kısıtlaması eklenir. Böylece, tablodaki mevcut ve gelecekteki kayıtların "username" sütunu için benzersiz değerlere sahip olmaları zorunlu hale gelir.

ALTER komutu ile UNIQUE kısıtlaması, tablodaki verilerin bütünlüğünü korumak ve tekrarlayan değerlerin önüne geçmek için kullanışlıdır. Ancak, UNIQUE kısıtlamasını uygularken, mevcut verilere uygunluğun kontrol edilmesi önemlidir. Eğer UNIQUE kısıtlaması eklenmek istenen sütunda mevcut tekrarlayan değerler varsa, bu kısıtlama eklendikten sonra hata alabilirsiniz veya mevcut kayıtları düzeltmeniz gerekebilir.

CHECK Kısıtlaması

CHECK, SQL'de bir tablo sütununa uygulanan bir kısıtlamadır ve belirli bir koşulu sağlayan değerlerin kabul edildiğini belirtir. CHECK kısıtlaması, sütun değerlerinin belirli bir mantıksal veya koşullu ifadeye uygun olmasını sağlar.

CHECK kısıtlaması, veri bütünlüğünü korumak ve geçersiz veya uygunsuz değerlerin sütuna eklenmesini engellemek için kullanılır. Bu kısıtlama, sütun değerlerinin belirli bir şartı karşılaması gerektiğinde kullanışlıdır.

Örneğin, aşağıdaki örnekte "students" tablosunda "age" sütunu için bir CHECK kısıtlaması tanımlanmıştır:

```
CREATE TABLE students (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  age INTEGER CHECK (age >= 18)
);
```

Bu tabloda "age" sütunu için CHECK kısıtlaması, sadece 18 veya daha büyük değerlere izin verir.

CHECK Kısıtlaması

ALTER komutu ile CHECK kısıtlaması mevcut bir tabloya sonradan eklenmek istendiğinde, aşağıdaki örnek kullanılabilir:

ALTER TABLE students
ADD CONSTRAINT check_age CHECK (age >= 18);
veya
ALTER TABLE students
ADD CHECK (age >= 18);

Bu komutla "students" tablosuna "age" sütunu üzerinde CHECK kısıtlaması eklenir. Böylece, tabloya eklenecek veya güncellenecek kayıtların "age" değerinin belirtilen koşulu karşılaması zorunludur.

ALTER komutu ile CHECK kısıtlaması, tablodaki verilerin uygunluğunu ve bütünlüğünü sağlamak için kullanışlıdır. Ancak, mevcut verilere uygunluk kontrol edilmelidir. Eğer mevcut kayıtlardan CHECK kısıtlamasına uymayanlar varsa, bu kısıtlamayı ekledikten sonra hata alabilirsiniz veya mevcut kayıtları düzeltmeniz gerekebilir.

JOIN Yapıları

Bu bölümde tablolar arasında birleştirme işlemi yapmayı sağlayan JOIN yapılarını öğreneceğiz.

(INNER) JOIN

INNER JOIN, SQL'de iki veya daha fazla tablo arasında ortak bir sütun değeri kullanarak verileri birleştirmek için kullanılan bir JOIN türüdür. INNER JOIN, ortak sütun değerlerine sahip olan satırları birleştirir ve yalnızca eşleşen satırları döndürür.

INNER JOIN, ilişkisel veritabanlarında verileri bağlama (join) işlemi yaparken en sık kullanılan JOIN türüdür. Bu JOIN türü, tablolar arasındaki ilişkileri kullanarak veri setlerini birleştirme işlemini gerçekleştirir. INNER JOIN ile birleştirme işlemi, ilişkilendirilen tablolardaki ortak sütunlara göre gerçekleştirilir.

Örneğin, aşağıdaki örnekte "orders" ve "customers" tablolarını INNER JOIN kullanarak birleştirelim:

```
SELECT orders.order_id, orders.order_date, customers.customer_name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

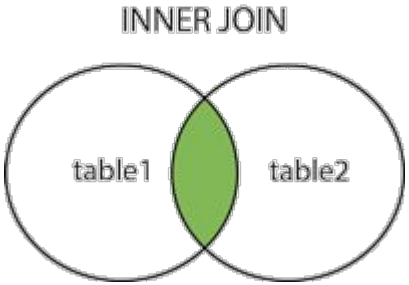
Bu örnekte, "orders" ve "customers" tablolarını "customer_id" sütunu üzerinden INNER JOIN ile birleştiriyoruz. Bu, orders tablosundaki her bir siparişin ilgili müşteriye ait müşteri adını içeren bir sonuç setini döndürür.

(INNER) JOIN

INNER JOIN ile birleştirme işlemi, ortak sütun değerlerine sahip olan satırları birleştirir ve bu şekilde ilişkilendirilmiş verileri elde eder. INNER JOIN ile birden fazla tabloyu birleştirebilir ve daha karmaşık ilişkileri yönetebilirsiniz.

INNER JOIN kullanırken dikkat edilmesi gereken bazı noktalar şunlardır:

- İlişkilendirilecek tablolar arasında ortak sütunlar olmalıdır.
- INNER JOIN kullanıldığında, eşleşmeyen kayıtlar sonuç setine dahil edilmez.
- INNER JOIN, eşleşen satırları döndürürken, birden fazla eşleşme durumu varsa her bir eşleşme kombinasyonunu döndürür.



Uygulama 9

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **city** tablosu ile **country** tablosunda bulunan şehir (city) ve ülke (country) isimlerini birlikte görebileceğimiz INNER JOIN sorgusunu yazınız.

- **customer** tablosu ile **payment** tablosunda bulunan payment_id ile customer tablosundaki first_name ve last_name isimlerini birlikte görebileceğimiz INNER JOIN sorgusunu yazınız.

- **customer** tablosu ile **rental** tablosunda bulunan rental_id ile customer tablosundaki first_name ve last_name isimlerini birlikte görebileceğimiz INNER JOIN sorgusunu yazınız.

LEFT JOIN

- LEFT JOIN, SQL'de iki veya daha fazla tabloyu birleştirirken sol tablodaki tüm satırları ve eşleşen sağ tablo satırlarını döndüren bir JOIN türüdür. Sol tablodaki tüm satırları korurken, sağ tabloda eşleşen satırlar varsa birleştirme işlemi yapar. Eğer eşleşme yoksa, sağ tablo için NULL değerleri döndürür.

- LEFT JOIN, soldaki (sol tablo) verileri korurken, eşleşme sağlayan sağdaki (sağ tablo) verileri birleştirme işlemini gerçekleştirir. Bu JOIN türü, sol tablodaki verilerin tamamını döndürmek istediğinizde ve sağ tablo ile ilişkili verilere ihtiyaç duyduğunuzda kullanışlıdır.

Örneğin, aşağıdaki örnekte "customers" tablosunu "orders" tablosuna LEFT JOIN kullanarak birleştirelim:
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date
FROM customers
LEFT JOIN orders ON customers.customer_id = orders.customer_id;

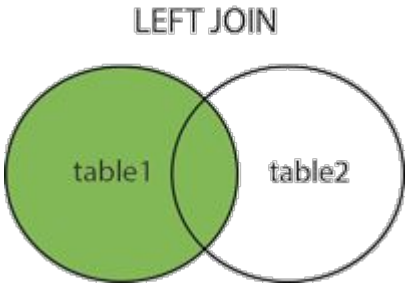
Bu örnekte, "customers" tablosunu "customer_id" sütunu üzerinden "orders" tablosuna LEFT JOIN ile birleştiriyoruz. Bu, customers tablosundaki her bir müşterinin ilgili sipariş bilgilerini içeren bir sonuç setini döndürür. Eğer bir müşteriye ait sipariş yoksa, orders tablosundaki alanlar NULL değeri alır.

LEFT JOIN

LEFT JOIN kullanırken dikkat edilmesi gereken bazı noktalar şunlardır:

- LEFT JOIN kullandığınızda, sol tablodaki tüm satırları döndürdüğünüzden emin olun. Sağ tabloda eşleşmeyen satırlar için NULL değerler alınacaktır.
- Sol ve sağ tablolar arasında ortak bir sütun olmalıdır.
- LEFT JOIN, soldaki tabloya daha fazla önem veren ve sağ tabloda eksik verileri kabul eden bir birleştirme türüdür.

LEFT JOIN, tablolar arasındaki ilişkileri kullanarak veri birleştirme işlemini gerçekleştirmek için sıkça kullanılan bir SQL ifadesidir. Bu sayede sol tablodaki verileri korurken, sağ tablo ile eşleşen verilere erişebilir ve ilişkili bilgileri tek bir sonuç setinde alabilirsiniz.



RIGHT JOIN

- RIGHT JOIN, SQL 'de iki veya daha fazla tabloyu birleştirirken sağ tablodaki tüm satırları ve eşleşen sol tablo satırlarını döndüren bir JOIN türüdür. Sağ tablodaki tüm satırları korurken, sol tabloda eşleşen satırlar varsa birleştirme işlemi yapar. Eğer eşleşme yoksa, sol tablo için NULL değerleri döndürür.

- RIGHT JOIN, sağdaki (sağ tablo) verileri korurken, eşleşme sağlayan soldaki (sol tablo) verileri birleştirme işlemini gerçekleştirir. Bu JOIN türü, sağ tablodaki verilerin tamamını döndürmek istediğinizde ve sol tablo ile ilişkili verilere ihtiyaç duyduğunuzda kullanışlıdır.

Örneğin, aşağıdaki örnekte "orders" tablosunu "customers" tablosuna RIGHT JOIN kullanarak birleştirelim:
SELECT orders.order_id, orders.order_date, customers.customer_id, customers.customer_name
FROM orders
RIGHT JOIN customers ON orders.customer_id = customers.customer_id;

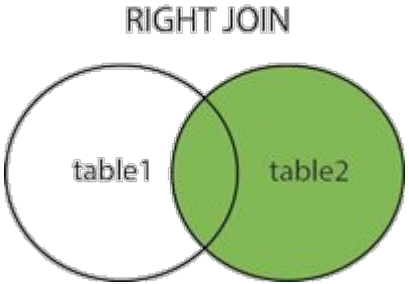
Bu örnekte, "orders" tablosunu "customer_id" sütunu üzerinden "customers" tablosuna RIGHT JOIN ile birleştiriyoruz. Bu, orders tablosundaki her bir siparişin ilgili müşteriye ait müşteri adını içeren bir sonuç setini döndürür. Eğer bir siparişe ait müşteri bilgisi yoksa, customers tablosundaki alanlar NULL değeri alır.

RIGHT JOIN

RIGHT JOIN kullanırken dikkat edilmesi gereken bazı noktalar şunlardır:

- RIGHT JOIN kullandığınızda, sağ tablodaki tüm satırları döndürdüğünüzden emin olun. Sol tabloda eşleşmeyen satırlar için NULL değerler alınacaktır.
- Sol ve sağ tablolar arasında ortak bir sütun olmalıdır.
- RIGHT JOIN, sağdaki tabloya daha fazla önem veren ve sol tabloda eksik verileri kabul eden bir birleştirme türüdür.

RIGHT JOIN, tablolar arasındaki ilişkileri kullanarak veri birleştirme işlemini gerçekleştirmek için sıkça kullanılan bir SQL ifadesidir. Bu sayede sağ tablodaki verileri korurken, sol tablo ile eşleşen verilere erişebilir ve ilişkili bilgileri tek bir sonuç setinde alabilirsiniz.



FULL JOIN

- FULL JOIN, SQL'de iki veya daha fazla tabloyu birleştirirken hem sol hem de sağ tablodaki tüm satırları döndüren bir JOIN türüdür. FULL JOIN, sol ve sağ tablolardaki eşleşen satırları birleştirirken, eşleşmeyen satırları da döndürür. Eğer bir tabloda eşleşme yoksa, diğer tablodaki değerler için NULL değerleri döndürür.

- FULL JOIN, sol ve sağ tablolardaki tüm verilerin birleştirilmesini sağlar. Sol tabloda eşleşmeyen veriler için NULL değerleri, sağ tabloda eşleşmeyen veriler için de NULL değerleri içeren bir sonuç seti döndürür.

Örneğin, aşağıdaki örnekte "customers" ve "orders" tablolarını FULL JOIN kullanarak birleştirelim:

```
SELECT customers.customer_id, customers.customer_name, orders.order_id, orders.order_date
FROM customers
FULL JOIN orders ON customers.customer_id = orders.customer_id;
```

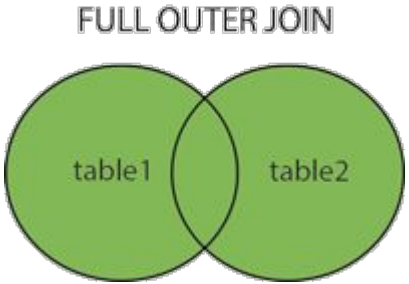
Bu örnekte, "customers" ve "orders" tablolarını "customer_id" sütunu üzerinden FULL JOIN ile birleştiriyoruz. Bu, customers ve orders tablolarındaki her bir müşteri ve sipariş kombinasyonunu içeren bir sonuç setini döndürür. Eğer bir müşteriye ait sipariş yoksa veya bir siparişin müşterisi yoksa, ilgili alanlar NULL değeri alır.

FULL JOIN

FULL JOIN kullanırken dikkat edilmesi gereken bazı noktalar şunlardır:

- FULL JOIN kullandığınızda, hem sol hem de sağ tablodaki tüm satırları döndürdüğünüzden emin olun. Eşleşmeyen satırlar için NULL değerler alınacaktır.
- Sol ve sağ tablolar arasında ortak bir sütun olmalıdır.
- FULL JOIN, sol ve sağ tablolar arasındaki ilişkileri kullanarak birleştirme işlemini gerçekleştirir ve tüm verileri tutar.

FULL JOIN, tablolar arasındaki ilişkileri kullanarak veri birleştirme işlemini gerçekleştirmek için kullanılan bir SQL ifadesidir. Bu sayede sol ve sağ tablolardaki tüm verilere erişebilir ve eksik veya eşleşmeyen verileri tespit edebilirsiniz.



Uygulama 10

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **city** tablosu ile **country** tablosunda bulunan şehir (city) ve ülke (country) isimlerini birlikte görebileceğimiz LEFT JOIN sorgusunu yazınız.

- **customer** tablosu ile **payment** tablosunda bulunan payment_id ile customer tablosundaki first_name ve last_name isimlerini birlikte görebileceğimiz RIGHT JOIN sorgusunu yazınız.

- **customer** tablosu ile **rental** tablosunda bulunan rental_id ile customer tablosundaki first_name ve last_name isimlerini birlikte görebileceğimiz FULL JOIN sorgusunu yazınız.

UNION

UNION, SQL'de kullanılan bir operatördür ve birden fazla SELECT ifadesinin sonuçlarını birleştirir. UNION operatörü, birleştirilen sonuç setleri arasında benzersiz (distinct) değerler döndürür.

Örneğin, iki farklı tablodan verileri birleştirmek için UNION operatörünü kullanabiliriz:

```
SELECT column1, column2 FROM table1
UNION
SELECT column1, column2 FROM table2;
```

Bu örnekte, table1 ve table2 tablolarından column1 ve column2 sütunlarını birleştiriyoruz. UNION operatörü, her iki SELECT ifadesinin sonucunu birleştirerek tek bir sonuç seti döndürür. Dönen sonuç setinde benzersiz (distinct) değerler bulunur.

UNION operatörü ile birleştirme yaparken dikkat edilmesi gereken nokta, birleştirilen SELECT ifadelerinin sütun sayılarının ve veri tiplerinin uyumlu olmasıdır. Ayrıca, her bir SELECT ifadesi için aynı sütun isimleri kullanılmalıdır.

UNION ALL

- UNION ALL operatörü ise UNION operatöründen farklı olarak, birleştirilen sonuç setlerindeki tüm değerleri döndürür. Benzersizlik kontrolü yapmadan tüm değerleri birleştirir. Yani, eşleşmeyen veya tekrar eden değerler de döner.

Örneğin:

SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;

Bu örnekte, table1 ve table2 tablolarından column1 ve column2 sütunlarını UNION ALL operatörü ile birleştiriyoruz. Dönen sonuç setinde, her iki tablodaki tüm değerler bulunur ve tekrar eden değerler de dikkate alınır.
UNION ve UNION ALL operatörleri ile birleştirme işlemi yaparken aşağıdaki noktalara dikkat etmek önemlidir:

Birleştirilen SELECT ifadeleri için aynı sütun sayısı ve uyumlu veri tipleri kullanılmalıdır.
UNION operatörü, benzersiz (distinct) değerleri döndürürken, UNION ALL operatörü ise tüm değerleri döndürür.
Sıralama işlemi yapmak isterseniz, UNION veya UNION ALL operatöründen sonra ORDER BY kullanabilirsiniz.

INTERSECT

INTERSECT, SQL'de kullanılan bir operatördür ve iki SELECT ifadesinin kesişimini döndürür. INTERSECT operatörü, her iki SELECT ifadesinin sonucunda ortak olan değerleri içeren bir sonuç seti oluşturur.

Örneğin, iki farklı SELECT ifadesinin kesişimini bulmak için INTERSECT operatörünü kullanabiliriz:

```
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```

Bu örnekte, table1 ve table2 tablolarından column1 ve column2 sütunlarının kesişimini bulmak için INTERSECT operatörünü kullanıyoruz. Dönen sonuç setinde, her iki SELECT ifadesinin sonucunda ortak olan değerler bulunur.

EXCEPT

EXCEPT operatörü ise bir SELECT ifadesinin diğer SELECT ifadesinden farkını döndürür. EXCEPT operatörü, sol SELECT ifadesinin sonucunda bulunan ve sağ SELECT ifadesinin sonucunda olmayan değerleri içeren bir sonuç seti oluşturur.

Örneğin, sol SELECT ifadesinin sağ SELECT ifadesinden farkını bulmak için EXCEPT operatörünü kullanabiliriz:
SELECT column1, column2 FROM table1
EXCEPT
SELECT column1, column2 FROM table2;

Bu örnekte, table1 tablosundan table2 tablosunda olmayan column1 ve column2 değerlerini bulmak için EXCEPT operatörünü kullanıyoruz. Dönen sonuç setinde, sol SELECT ifadesinde bulunan ancak sağ SELECT ifadesinde bulunmayan değerler bulunur.

INTERSECT ve EXCEPT operatörlerinin yanı sıra INTERSECT ALL, EXCEPT ALL gibi ek operatörler de mevcuttur. Bu operatörler, benzer şekilde çalışırlar, ancak tekrar eden değerleri de dikkate alır. INTERSECT ALL operatörü, iki SELECT ifadesinin kesişimini döndürürken, EXCEPT ALL operatörü bir SELECT ifadesinin diğer SELECT ifadesinden farkını döndürür ve tekrar eden değerleri de içerir.

Uygulama 11

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **actor** ve **customer** tablolarında bulunan first_name sütunları için tüm verileri sıralayalım.
- **actor** ve **customer** tablolarında bulunan first_name sütunları için kesişen verileri sıralayalım.
- **actor** ve **customer** tablolarında bulunan first_name sütunları için ilk tabloda bulunan ancak ikinci tabloda bulunmayan verileri sıralayalım.
- İlk 3 sorguyu tekrar eden veriler için de yapalım.

Alt Sorgular

Bu bölümde SQL sorgusunun içinde yer alan alt sorguları öğreneceğiz.

Alt Sorgu Nedir?

Alt sorgu (subquery), bir SQL sorgusunun içinde yer alan başka bir sorgudur. Alt sorgu, bir ana sorgunun bir parçası olarak kullanılarak daha karmaşık sorgulama işlemlerini gerçekleştirmek için kullanılır. Alt sorgu, iç içe geçmiş bir sorgu yapısı oluşturarak veritabanından belirli verileri çekmek veya sorgulamak için kullanılır.

Alt sorgu, bir ana sorgunun SELECT, FROM, WHERE, HAVING, INSERT, UPDATE veya DELETE gibi bölümlerinde kullanılabilir. Alt sorgunun sonucu, ana sorgunun bir parçası olarak kullanılır ve sonuç setini etkiler. Alt sorgu, genellikle WHERE veya HAVING bölümlerinde filtreleme veya kısıtlama amacıyla kullanılır.

Örneğin, aşağıdaki örnekte alt sorgu kullanılarak müşteriler tablosundan belirli bir ülkeye ait müşteri sayısını bulabiliriz:

```
SELECT DISTINCT Country, (
        SELECT COUNT(*) FROM Customers WHERE Customers.Country = Main.country
) AS CustomerCount
FROM Customers AS Main;
```

ANY Operatörü

Alt sorgudan gelen herhangi bir değer koşulu sağlaması durumunda TRUE olarak ilgili değerin koşu sağlamasını sağlar. bookstore veritabanında yapmış olduğumuz aşağıdaki sorguyu inceleyelim.

```
SELECT first_name, last_name FROM author
WHERE id = ANY
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

Yukarıda görmüş olduğunuz gibi alt sorgudan gelebilecek potansiyel iki id değeri var, bu id değerinin her ikisi de birbirinden bağımsız olarak ana sorgudaki id sütununda bulunan değerler ile eşleştiği için sorgu sonucunda oluşan sanal tabloda id değeri 4 ve 5 olan yazarlara ait first_name ve last_name değerlerini göreceğiz.

ALL Operatörü

Alt sorgudan gelen tüm değerlerin koşulu sağlaması durumunda TRUE olarak döner. bookstore veritabanındaki yine aynı sorguyu inceleyelim.

```
SELECT first_name, last_name FROM author
WHERE id = ALL
(
  SELECT id
  FROM book
  WHERE title = 'Abe Lincoln in Illinois' OR title = 'Saving Shiloh'
)
```

Burada ne söylemiştik, alt sorgu tarafından 4 ve 5 id leri gelecek burada eşitlik olduğu için aynı anda 4 ve 5 in bu koşulu sağlaması olanaksız olduğu için herhangi bir değer dönmeyecektir.

Uygulama 12

Aşağıdaki sorgu senaryolarını **dvdrental** örnek veritabanı üzerinden gerçekleştiriniz.

- **film** tablosunda film uzunluğu length sütununda gösterilmektedir. Uzunluğu ortalama film uzunluğundan fazla kaç tane film vardır?
- **film** tablosunda en yüksek rental_rate değerine sahip kaç tane film vardır?
- **film** tablosunda en düşük rental_rate ve en düşün replacement_cost değerlerine sahip filmleri sıralayınız.
- **payment** tablosunda en fazla sayıda alışveriş yapan müşterileri(customer) sıralayınız.

Alt Sorgular ve JOIN Yapısı

Alt sorgular ve JOIN kavramları birlikte çok sık kullanılırlar. Aşağıdaki iki senaryoda bu iki yapıyı birlikte kullanacağız.

- İlk senaryomuz: **bookstore** veri tabanında kitap sayfası sayısı ortalama kitap sayfası sayısından fazla olan kitapların isimlerini, bu kitapların yazarlarına ait isim ve soyisim bilgileriyle birlikte sıralayınız.

```
SELECT author.first_name, author.last_name, book.title FROM author
INNER JOIN book ON book.author_id = author.id
WHERE page_number >
(
  SELECT AVG(page_number) FROM book
);
```

Yukarıdaki sorgumuzda kitaplara ait yazar bilgilerini JOIN kullanarak elde ediyoruz. Ortalama sayfa sayısını da alt sorgudan getiriyoruz.

Alt Sorgular ve JOIN Yapısı

- İkinci senaryomuz: **dvdrental** veritabanında en uzun filmlerin isimlerini aktör isim ve soy isimleriyle birlikte sıralayalım.

```
SELECT actor.first_name, actor.last_name, film.title FROM actor
JOIN film_actor ON film_actor.actor_id = actor.actor_id
JOIN film ON film.film_id = film_actor.film_id
WHERE film.length =
(
  SELECT MAX(length)  FROM film
);
```

Burada da görmüş olduğumuz gibi filmlerin aktör bilgilerini ikili JOIN yapısı kullanarak elde ediyoruz. En uzun film süresini de alt sorgudan getiriyoruz.

Teşekkürler! :)