Hızlı bir özet yapalım

SQL ifadeleri dört ana kategoriye ayrılır:

* DDL - Data Definition Language Veri Tanımlama Dili
* DML - Data Manipulation Language Veri İşleme Dili ---- select------read işlemi
* DCL - Data Control Language Veri Kontrol Dili
* TCL - Transaction Control Language İşlem Kontrol Dili

DDL, DML, DCL, TCL ayrı diller değildir. SQL olan tek bir veritabanı dilinin parçalarını oluştururlar. Şimdi tanımlarına bakalım.

Sql ifadelerinde en fazla DML kullanılıyor, bunun içinde de Select en fazla kullanılıyor.

Create

Read ----- biz bu zamana kadar query ile okuma işlemi yapmış olduk

Update

Delete

1. DDL – Veri Tanımlama Dili

DDL, veritabanı şemasını belirtir. ----- Data Definition Language

DDL'de kullanılan bazı ifadeler CREATE, ALTER, DROP'tur.

Create --- tablo oluşturacak isek

Alter -----mevcut bir tabloda bir değişiklik yapacaksak

Drop ---- tablomuzu silecek isek

\*\*\*DDL deyimleri genellikle veri eklemeden önce yeni bir veritabanı kurmak ve yapılandırmak için kullanılır.

DDL veya Veri Tanımlama Dili aslında veritabanı şemasını tanımlamak için kullanılabilecek SQL komutlarından oluşur.

Basitçe veritabanı şemasının açıklamalarıyla ilgilenir ve veritabanındaki veritabanı nesnelerinin yapısını oluşturmak ve değiştirmek için kullanılır.

1. DML - Veri İşleme Dili

Data Manipulation Language (DML), kullanıcıların verilere erişmesini veya bunları değiştirmesini sağlar.

INSERT, UPDATE, DELETE, SELECT \* DML'de kullanılan ifadelerdir.

\* Bazı kaynaklarda SELECT deyimi, DQL database query languech (Veri Sorgulama Dili) adı verilen farklı bir kategoride gruplandırılmıştır.

Veritabanında bulunan verilerin işlenmesiyle ilgilenen SQL komutları, DML veya Veri İşleme Dili'ne aittir ve bu, SQL ifadelerinin çoğunu içerir.

INSERT - tabloya veri eklemek için kullanılır.

UPDATE - bir tablodaki mevcut verileri güncellemek için kullanılır.

DELETE – bir veritabanı tablosundan kayıtları silmek için kullanılır.

1. DCL

Veri Kontrol Dili (DCL), erişim kontrolü vermek veya iptal etmek için kullanılır.

Açıklamaları REVOKE ve GRANT şeklindedir.

DCL, temel olarak veritabanı sisteminin hakları, izinleri ve diğer kontrolleriyle ilgilenen GRANT ve REVOKE gibi komutları içerir.

DCL komutlarına örnekler:

GRANT-kullanıcının veritabanına erişim ayrıcalıklarını verir.

REVOKE-kullanıcının GRANT komutu kullanılarak verilen erişim ayrıcalıklarını geri çekme.

\*\*

Benim verilerimi kontrol ediyor. Verilerim nedir, kullanıcının hakları.

1. TCL

İşlem Kontrol Dili (TCL), DML ve DDL komutlarının işlemlerini kontrol eder.

SQL de sorgu cümleleri yazıyoruz, veriyi getirirken hata yapmışsak en fazla veri gelmez, ama bazı durumlarda yazdığımız sql ifadelerinin tamamının çalışmasını isteriz, eğer tamamı çalışmıyorsa aralardaki sorgularda hata oluyorsa hiç biri çalışmasın diyebiliriz. O zaman transleşınlardan faydalanıyoruz.

Eğer tüm sorguların tamamı gerçekleşmediyse tüm sorguları geri almasıyla ilgili anahtar kelimelerimiz.

TCL'deki bazı ifadeler COMMIT, ROLLBACK, SAVEPOINT şeklindedir.

TCL komutları, veritabanı içindeki işlemlerle ilgilenir.

TCL komutlarına örnekler:

COMMIT– bir İşlem gerçekleştirir.

ROLLBACK– herhangi bir hata oluşması durumunda bir işlemi geri alır.

SAVEPOINT–bir işlem içinde bir kayıt noktası ayarlar.

SET TRANSACTION –işlem için özellikleri belirtin.

Şimdi konuştuğumuz ifadelerin çoğuna yer vermeyeceğiz.

Bu dersimizde DDL komutlarını öğreneceğiz. Bir not olarak, kursumuzda diğer dillerin komutlarını ele almayacağız.

DDL komutlarına bakmadan önce önce, veri türleriyle başlayacağız.

DATA TYPES

Bir sütunun veri türü, sütunun tutabileceği değeri tanımlar:

integer, character, date and time, binary, and so on

Bir SQL developer, tablo oluştururken her sütunda ne tür verilerin depolanacağına karar vermelidir. Veri türü, SQL'in her sütunda ne tür verilerin beklendiğini anlaması için bir kılavuzdur ve ayrıca SQL'in depolanan verilerle nasıl etkileşime gireceğini tanımlar.

Üç ana veri türü vardır:

string, numeric, and date and time

Veri türlerinin tüm ayrıntılarını ele almayacağız.

CHAR(size): SABİT uzunlukta bir dize (harfler, sayılar ve özel karakterler içerebilir). size parametresi sütun uzunluğunu karakter cinsinden belirtir - 0 ile 255 arasında olabilir. Varsayılan değer 1'dir

VARCHAR(size): DEĞİŞKEN uzunlukta bir dize (harfler, sayılar ve özel karakterler içerebilir). size parametresi, maksimum sütun uzunluğunu karakter cinsinden belirtir - 0 ile 65535 arasında olabilir.

BINARY(size): CHAR()'a eşittir, ancak ikili bayt dizilerini saklar. size parametresi sütun uzunluğunu bayt cinsinden belirtir. Varsayılan 1'dir.

DATE

format: YYYY-AA-GG. Desteklenen aralık "1000-01-01" ile "9999-12-31" arasındadır.

DATETIME(*fsp*)

Bir tarih ve saat kombinasyonu. Biçim: YYYY-AA-GG ss:dd:ss. Desteklenen aralık "1000-01-01 00:00:00" ile "9999-12-31 23:59:59" arasındadır. Otomatik başlatma ve geçerli tarih ve saate güncelleme için sütun tanımına DEFAULT ve ON UPDATE ekleme

TIMESTAMP(*fsp*)

Bir zaman damgası. TIMESTAMP değerleri, Unix döneminden ('1970-01-01 00:00:00' UTC) bu yana geçen saniye sayısı olarak saklanır. Biçim: YYYY-AA-GG ss:dd:ss. Desteklenen aralık, '1970-01-01 00:00:01' UTC'den '2038-01-09 03:14:07' UTC'ye kadardır. Sütun tanımında DEFAULT CURRENT\_TIMESTAMP ve ON UPDATE CURRENT\_TIMESTAMP kullanılarak otomatik başlatma ve geçerli tarih ve saate güncelleme belirtilebilir

INT

Bir orta tamsayı. İşaretli aralık -2147483648 ila 2147483647 arasındadır. İşaretsiz aralık 0 ila 4294967295 arasındadır. size parametresi maksimum görüntü genişliğini belirtir (255'tir)

FLOAT(boyut, d)

Bir kayan nokta sayısı. Toplam basamak sayısı boyutta belirtilir. Virgülden sonraki basamak sayısı d parametresinde belirtilir. Bu sözdizimi MySQL 8.0.17'de kullanımdan kaldırılmıştır ve gelecekteki MySQL sürümlerinde kaldırılacaktır.

FLOAT(*p*)

Bir kayan nokta sayısı. MySQL, ortaya çıkan veri türü için FLOAT veya DOUBLE kullanılıp kullanılmayacağını belirlemek için p değerini kullanır. p, 0 ile 24 arasındaysa, veri türü FLOAT() olur. p 25 ile 53 arasındaysa, veri türü DOUBLE() olur

DECIMAL(*size*, *d*)

Tam bir sabit noktalı sayı. Toplam basamak sayısı boyutta belirtilir. Virgülden sonraki basamak sayısı d parametresinde belirtilir. Boyut için maksimum sayı 65'tir. d için maksimum sayı 30'dur. Boyut için varsayılan değer 10'dur. d için varsayılan değer 0'dır.

!!!!!!!!!!

Veri türleri, farklı veritabanında farklı adlara sahip olabilir. Ve isim aynı olsa bile, boyut ve diğer detaylar farklı olabilir! Daima belgeleri kontrol edin!

Dökümanlardan, googldan bakıp veri türünü belirlemeliyiz.

Şimdi gidip yeni bir tane tablo oluşturalım

Herhangi bir tabloya tıklayıp modify table deyip oradan veri tiplerine bakalım.

İnteger,

Foregin key ------ delete….., bunlar kayıt silinince buradan da gitsin demek

Primary key

Database structer yapısına gelip sağ tıklayıp veya yukarıda create table diyoruz.

Tablo ismini yazıp enter a basıyoruz.

Add deyip sütun isimlerini yazarak tablo oluşturmaya başlıyoruz.

AU--- otomatik arttırsın

Default --- orada bir şey yazılmazsa benim yazdığım geçerli olsun demek

Slayta bak Create tablo oluştur

Not: VARCHAR sütunlarındaki değerler, değişken uzunluklu dizelerdir. Uzunluk, 0 ila 65.535 arasında bir değer olarak belirtilebilir.

CREATE TABLE "yeni"(id INTEGER, isim TEXT);

Bunu iki kere çalıştırırsam hata verir , zaten bu tablo var der.

CREATE TABLE If NOT EXISTS "yeni"(id INTEGER, isim TEXT);

Bu şekilde dediğim zaman tablo oluştur ama eğer yoksa oluştur.

Slayttaki örnek;

Lütfen mevcut chinook veritabanınıza bir tablo ekleyin:

Tablo adı, çalışanların yıllık veya hastalık izinlerinin kaydını tutmak için kullanacağımız yapraklar olacaktır.

Sütun adları:

* id
* employee\_id
* start\_date
* end\_date

CREATE TABLE If NOT EXISTS "leaves" (id INT, employeeId INT, startDate date, endDate date);

Yanlışlık yaptıysak tabloyu silip tekrar oluşturabiliriz.

DROP TABLE leaves;

DROP TABLE yeni;

Oluşturduğumuz tabloya veri girelim

Lütfen mevcut sql\_course veritabanınıza bir tablo ekleyin:

Tablo adı, bu yaz için çalışanların tatil\_planı **vacation\_plan** olacaktır.

Sütun adları:

* place\_id
* country
* hotel\_name
* employee\_id
* vacation\_length - ----- tatil\_uzunluğu
* budget ---- bütçe

**Answer: SQLite3**

CREATE TABLE vacation\_plan

(place\_id INT, country TEXT, hotel\_name TEXT, employee\_id INT, vacation\_lenght INT,

budget REAL );

Bu şekilde de yapabiliriz;

CREATE TABLE vacation\_plan

(place\_id INT, country VARCHAR(20), hotel\_name VARCHAR(30), employee\_id INT,

vacation\_lenght INT, budget DECIMAL );

DROP TABLE

Bir tablonun silinmesi, tabloda saklanan tüm bilgilerin kaybolmasına neden olur!

TRUNCATE TABLE deyimi, tablonun kendisini değil, tablo içindeki verileri silmek için kullanılır.

TRUNCATE ile ise tabloyu değil tablonun içindeki tüm verileri silebiliyoruz.

TRUNCATE TABLE leaves;

SQLite desteklemedi. Onun yerine DELETE FROM kullanıyoruz.

DELETE FROM leaves;

DROP TABLE IF EXISTS vacation\_plan; (Daha önce varsa aynı isimde onu siler)

ALTER TABLE

Tabloda değişiklikler yapmak için ALTER kullanmalıyım.

ALTER TABLE deyimi, mevcut bir tablodaki sütunları eklemek, silmek veya değiştirmek için kullanılır.

Ayrıca mevcut bir tabloya çeşitli kısıtlamalar eklemek ve bırakmak için de kullanılır.

Tablo ismini değiştirebilirim.

ALTER TABLE leaves RENAME to izinler;

Sütun adını değiştirmek için

**ALTER TABLE table\_name RENAME COLUMN old\_column\_name TO new\_column\_name;**

ALTER TABLE leaves RENAME COLUMN employeeid to employeid izinler;

Yeni bir sütun değişikliği yapmak için, eklemek için

**ALTER TABLE** **table\_name ADD**  **column\_name** **data\_type;**

Tablodaki bir sütunu silmek için aşağıdaki sözdizimini kullanın

**ALTER TABLE** **table\_name DROP**  **column\_name;**

ALTER TABLE izinler DROP COLUMN aciklama;

Bir tabloya sütun eklemek için aşağıdaki sözdizimini kullanın

**ALTER TABLE** **table\_name ADD**  **column\_name** **data\_type;**

ALTER TABLE table\_name ADD COLUMN new\_column\_definition data\_type;

ALTER TABLE izinler add COLUMN aciklama TEXT;

Tablodaki bir sütunun veri türünü değiştirmek için aşağıdaki sözdizimini kullanın

**ALTER TABLE** **table\_name**

**MODIFY COLUMN**  **column\_name** **data\_type**

Sqlite da aşağıdaki şekilde kullanılır

ALTER TABLE table\_name RENAME COLUMN column\_name TO temp\_column\_name;

ALTER TABLE table\_name ADD COLUMN column\_name new\_data\_type;

UPDATE table\_name SET column\_name = CAST(temp\_column\_name AS new\_data\_type);

ALTER TABLE table\_name DROP COLUMN temp\_column\_name;

**Yeni bir sütun ekleyelim (vacation\_plan “city” )**

ALTER TABLE **vacation\_plan**  add COLUMN **city** TEXT;

\*\*\*

Leaves Tablo adını önce employee\_leaves olarak değiştirin.

Ardından, leaves tablonuza adlı bir sütun ekleyin. “leave\_type”.

"Yıllık izin", "hastalık izni" vb. izin türlerini kullanacağız.

employee\_leaves tablosuna 3 yeni kayıt ekleyin.

İzin türü olarak “annual\_leave”, sick\_leave” vb. kullanabilirsiniz. (yıllık\_izin, hastalık\_izni )

\*\*\*

Şimdi başka bir tablo leave\_types ile ekleyin ( buraya izin tipleri belirlenecek)

id -> PK AUTOINC

leave\_name ->TEXT

create table ‘leavesType’ column(id int primary key not null, izin\_type VARCHAR)

Ve staff\_leaves tablosundaki foregin key olarak yapın

Yani şimdi diğer tabloya sütun eklememiz gerekiyor

Alter table izinler add column izintipi int ;

3 kayıt EKLE left\_types ve staff\_leaves tablosuna

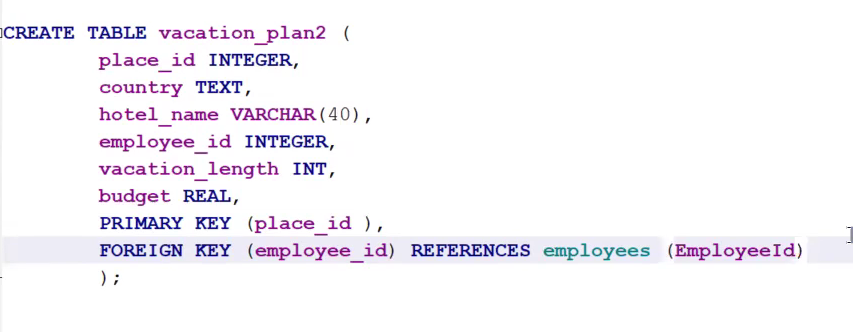
\*\*\*\*

Bu kayıtları eklerken foregin key i farklı yapamayacağımızı gösterelim.

\*\*\*

Biz tablolarımızı silmeden önce exel e kopyalayabiliriz.

Daha sonra exel den database e import edebiliriz.



İNSERT İNTO

Tablonun içerisine bilgi gireceksek bu komut ile yazıyoruz.

INSERT INTO ifadesi, bir tabloya yeni kayıtlar eklemek için kullanılır.

INSERT INTO deyimini iki şekilde yazmak mümkündür.

İlk yol, hem sütun adlarını hem de eklenecek değerleri belirtir:

Tablonun tüm sütunları için değer ekliyorsanız, SQL sorgusunda sütun adlarını belirtmeniz gerekmez. Ancak, değerlerin sırasının tablodaki sütunlarla aynı sırada olduğundan emin olun.

Yalnızca Belirtilen Sütunlara Veri Ekle

Yalnızca belirli sütunlara veri eklemek de mümkündür.

İzinler tablosuna veri ekleyelim --- İstediğim sütuna veri ekleyebilirim

INSERT INTO izinler(id, employeeId) VALUES(3, 5);

4 sütun 4 değer var ise izinler deyip sütun başlıklarını vermeme gerek yok.

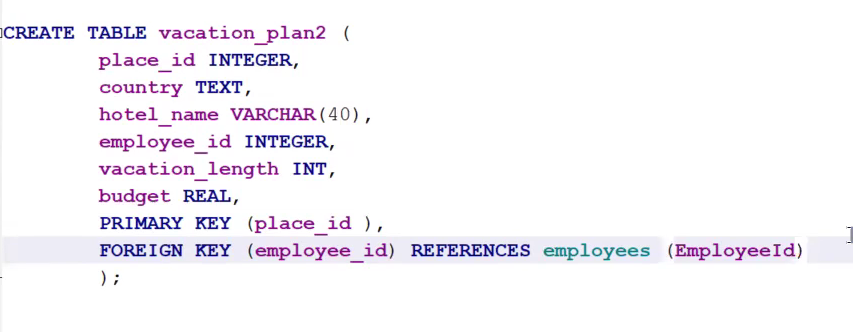
INSERT INTO VALUES(3, 5, “2023-01-01”, “2023-02-02”);

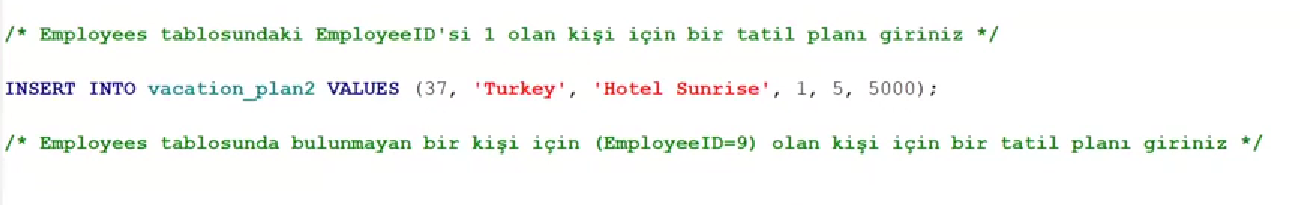
Dediğimiz zaman değerleri kendisi hepsine sırayla yerleştirir.

Eğer otomatik sayı vermiş isek oraya değer vermeme gerek yok, kendisi otomatik olarak atar.

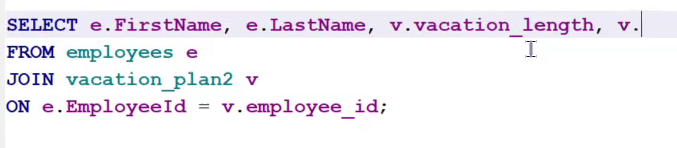
Gidip arayüzden PK ve AI (otomatik arttır) tıklayalım.

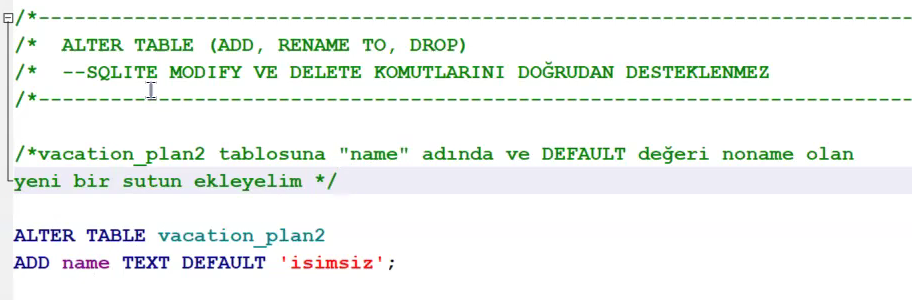
INSERT INTO izinler(employeeId) VALUES(5);



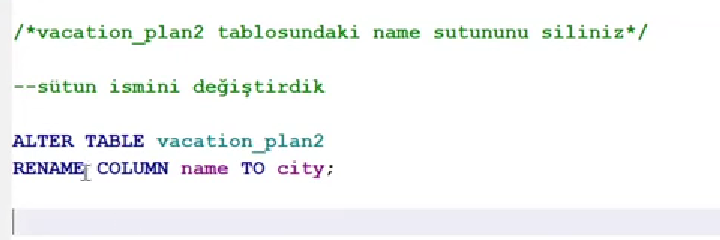


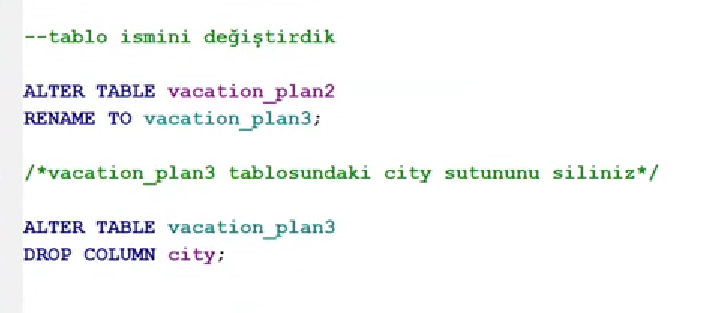






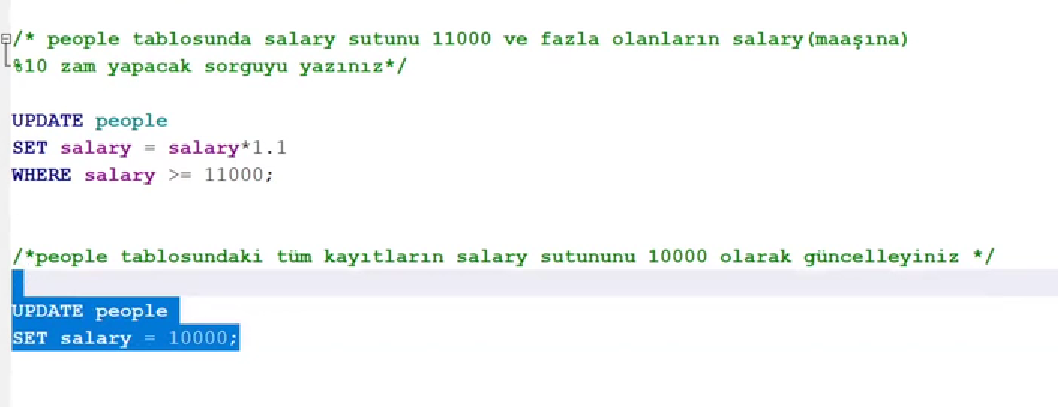
Default yazmasak da olur





Yeni tablo oluştur, people veya workers diye olabilir

İd, name, surname, salary



kendimiz örnek yapalım

**CREATE TABLE** students (Student\_id INT PRIMARY KEY, FirstName TEXT, LastName TEXT, Grade INT);

**INSERT INTO** students (Student\_id, FirstName, LastName, Grade)

**VALUES** (1, 'julia', 'Saelens', 99), (2, 'Hanan', 'Qattan', 89), (3, 'Anu', 'Rowe', 87), (4, 'Nicholas', 'Perros', 94), (5, 'Alexandra', 'Barraclough', 90), (6, 'Luis', 'Vasquez', 96);

**CREATE TABLE** grades (Grade INT PRIMARY KEY, Student\_id INT, FOREIGN KEY (Student\_id)

REFERENCES students (Student\_id)

slayttaki query

INSERT INTO leaves (id, employee\_id,start\_date, end\_date) VALUES (1,2,'2021-02-01','2021-02-15');

Bu tablo employees tablosuyla ilişkili olduğundan orada mevcut olan bir employee\_id numarası değeri vermeliyiz. Yoksa hatalara sebep olabilir.

Constraints

Kısıtlamalar, CREATE TABLE deyimiyle tablo oluşturulduğunda veya ALTER TABLE deyimiyle tablo oluşturulduktan sonra belirtilebilir.

SQL kısıtlamaları, bir tablodaki veriler için kurallar belirlemek için kullanılır.

Kısıtlamalar, bir tabloya girebilecek veri türünü sınırlamak için kullanılır. Bu, tablodaki verilerin doğruluğunu ve güvenilirliğini sağlar. Kısıtlama ile veri eylemi arasında herhangi bir ihlal varsa, işlem iptal edilir.

Kısıtlamalar sütun düzeyinde veya tablo düzeyinde olabilir. Sütun düzeyi kısıtlamaları bir sütuna, tablo düzeyi kısıtlamaları ise tüm tabloya uygulanır.

Aşağıdaki kısıtlamalar SQL'de yaygın olarak kullanılır:

NOT NULL - Bir sütunun NULL değerine sahip olmamasını sağlar

UNIQUE - Bir sütundaki tüm değerlerin farklı olmasını sağlar (e-posta, kullanıcı adı, cep telefonu)

PRIMARY KEY - BOŞ DEĞİL ve BENZERSİZ kombinasyonu. Bir tablodaki her satırı benzersiz şekilde tanımlar

FOREIGN KEY - Başka bir tablodaki bir satırı/kaydı benzersiz şekilde tanımlar

CHECK - Bir sütundaki tüm değerlerin belirli bir koşulu karşılamasını sağlar

DEFAULT - Hiçbir değer belirtilmediğinde bir sütun için varsayılan bir değer ayarlar

INDEX - Veritabanından çok hızlı bir şekilde veri oluşturmak ve almak için kullanılır

Biz tabloda sütun ismi belirlediğimiz zaman, varlığa ait bir özellik belirlediğimiz zaman bununla ilgili kriterler verebiliyorduk bunu arayüzde gördük. Peki komut satırında yazmak istersem nasıl yazacağım

PRIMARY KEY

PRIMARY KEY , bir tablodaki her kaydı benzersiz şekilde tanımlar.

Birincil anahtarlar unique değerler içermelidir ve NULL değerler içeremez.

Bir tablo yalnızca BİR birincil anahtara sahip olabilir; ve tabloda, bu birincil anahtar, tek veya birden çok sütundan (alanlardan) oluşabilir.

Veya artist\_id, chinook db'de bir PK'dir

Yazılışına bakalım;

FOREIGN KEY

Yabancı anahtar, bir tablodaki, başka bir tablonun her satırını benzersiz şekilde tanımlayan bir sütundur. Bu sütun, başka bir tablonun birincil anahtarını ifade eder. Bu, tablolar arasında bir tür bağlantı oluşturur.

YABANCI ANAHTAR, iki tabloyu birbirine bağlamak için kullanılan bir anahtardır.

YABANCI ANAHTAR, bir tablodaki başka bir tablodaki BİRİNCİL ANAHTAR'a atıfta bulunan bir alandır (veya alanlar topluluğudur).

Yabancı anahtarı içeren tabloya alt tablo, aday anahtarı içeren tabloya ise başvurulan veya ana tablo adı verilir.

Diyagramdan ve veri tabanından albums ve artist tablosunu gösterin:

Albümler tablosundaki ArtistId bir yabancı anahtardır

Foreign key ile ilgili bir örnek yapalım

Mağaza veri tabanı içerisinde müşteri ve sipariş olarak iki tablo oluşturalım.

**CREATE TABLE** clients (Client\_id INT PRIMARY KEY, FirstName TEXT NOT NULL, LastName TEXT);

**INSERT INTO** clients (Client\_id, FirstName, LastName)

**VALUES**  (1, 'John', 'King'), (2, 'Mehmet', 'Silver'), (3, 'Sevde', 'Gold');

-- (4, 'Sevda', 'Gold2');

**CREATE TABLE** orders (Order\_id INT PRIMARY KEY, Order\_Nu INTEGER, Client\_id INTEGER,

FOREIGN KEY (Client\_id) REFERENCES clients (Client\_id) );

**INSERT INTO** orders (Order\_id, Order\_Nu , Client\_id )

**VALUES**  (1, 101, 1), (2, 102, 2), (3, 103, 3);

--(4, 104, 4);

Slaytt 30 daki örnek;

Try to insert a record in albums table with an ArtistID=10000 and AlbumID=347

kayıt eklemeyi deneyin

INSERT INTO albums (AlbumID, Title, ArtistID) VALUES (347,'My new album',1000)

347 zaten veritabanında olduğundan primary key bir kısıtlama hatası alırsınız

Bunu deneyin ve ArtistID 1000 olmadığı için FOREIGN KEY hatası alırsınız.

Not NULL

Bir sütun NULL değerleri içerebilir. NULL değeri, değerin bilinmediği veya mevcut olmadığı anlamına gelen özel bir değerdir.

Açıkça NOT NULL kısıtlamaları belirtmediğimiz sürece, bir tablodaki tüm sütunlar (birincil anahtarın sütunu hariç), NULL değerlerini tutabilir.

Slayt 33 deki örnek;

Title değeri olmayan albümler tablosuna bir kayıt eklemeye çalışın (title not null mu bakalım)

INSERT INTO albums (ArtistID) VALUES (100)

Burada title yi boş bırakamayız, hata verir.

INSERT INTO albums (Title,ArtistID) VALUES ('',100)

vacation\_plan tablosunda düzeltme yapalım;

**CREATE TABLE** vacation\_plan (place\_id INT, country TEXT, hotel\_name TEXT, employee\_id INT,

vacation\_lenght INT, budget REAL );

**DROP TABLE** vacation\_plan;

**CREATE TABLE** new\_vacation\_plan

(place\_id INT PRIMARY KEY, country TEXT, hotel\_name TEXT NOT NULL, vacation\_lenght INT,

budget REAL, employee\_id INT, FOREIGN KEY (employee\_id) REFERENCES employees EmployeeId) );

DROP TABLE leaves;

Ardından, aşağıdaki kısıtlamaları ekleyerek leaves tablosunu yeniden oluşturun:

* id -> PRIMARY KEY , AUTOINC
* employee\_id -> FOREING KEY
* start\_date -> NOT NULL
* end\_date -> NOT NULL

CREATE TABLE "leaves" ( id INTEGER NOT NULL, employee\_id INT, start\_date DATE NOT NULL, end\_date DATE NOT NULL, PRIMARY KEY("id" AUTOINCREMENT), FOREIGN KEY("employee\_id") REFERENCES "employees"("EmployeeId");

INSERT INTO leaves (id, employee\_id,start\_date, end\_date) VALUES (1,22,'2021-02-01','2021-02-15')

INSERT INTO leaves (id, employee\_id,start\_date, end\_date) VALUES (1,22,'2021-02-01','2021-02-15')

\*\*\*

UPDATE;

Bu söz diziminde:

Öncelikle UPDATE yan tümcesinden sonra güncellemek istediğiniz tabloyu belirtin.

İkinci olarak, SET yan tümcesinde tablonun her sütunu için yeni bir değer belirleyin.

Üçüncüsü, WHERE yan tümcesindeki bir koşulu kullanarak güncellenecek satırları belirtin. WHERE deyimi isteğe bağlıdır. Bunu atlarsanız, UPDATE deyimi tablonun tüm satırlarındaki verileri güncelleyecektir.

Update izintipleri SET izintipi=”yil başi”; ------ bunu yapmak tehlikelidir, hepsini değiştirir.

Update izintipleri SET izintipi=”yil başi” where id=3;

Update employees set salary=2000 where dep=”Manager”; ----- olan diyebiliriz.

\*\*\*

Leave\_types tablosunda Yıllık iznin adını Evlilik İzni olarak değiştirin

\*\*\*

employee\_leaves tablosundaki bir kaydın başlangıç ​​ve bitiş tarihi değerlerini değiştirme

DELETE;

Bu söz diziminde:

Öncelikle, DELETE FROM anahtar kelimesinden sonra satırları kaldırmak istediğiniz tablonun adını belirtin.

İkinci olarak, kaldırılacak satırları belirlemek için WHERE yan tümcesine bir arama koşulu ekleyin. WHERE deyimi, DELETE deyiminin isteğe bağlı bir parçasıdır. WHERE deyimini atlarsanız, DELETE deyimi tablodaki tüm satırları siler.

İzin türleri tablosundan bir kaydı silme

DELETE FROM employee\_leaves where id = 1

Çalışan izin tablosundan bir kaydı silme

Delete from izintipleri where id=4;

4 numaralı id nin kaydını silecektir.

\*\*\*Böyle şeylerle uğraşıyorsanız önce veri tabanınızın yedeğini alın daha sonra bu işlemleri yapın.

GRANT ve REVOKE

DCL, temel olarak veritabanı sisteminin hakları, izinleri ve diğer kontrolleriyle ilgilenen GRANT ve REVOKE gibi komutları içerir.

DCL komutlarına örnekler:

**GRANT** kullanıcının veritabanına erişim ayrıcalıklarını verir.

yetki ver, hangi tablo , hangi kullanıcı için yetki verilecek

Sistem ya da nesne haklarının kullanıcılara veya rollere atanması için kullanılan komuttur.

**GRANT DELETE ON personel TO ogrenci**

Yukarıdaki örnekte ogrenci kullanıcısına personel tablosunda silme yapma yetkisi verilir.

**REVOKE** kullanıcının GRANT komutu kullanılarak verilen erişim ayrıcalıklarını geri çekme.

Sistem ya da nesne haklarının kullanıcılardan veya rollerden geri alınnması için kullanılan komuttur.

REVOKE DELETE ON personel FROM ogrenci

Yukarıdaki örnekte ogrenci kullanıcısından personel tablosunda silme yapma yetkisi geri alınır

\*\*\*Bunlarla siz uğraşmayacaksınız, size zaten yetki verilmiş şekilde kullanırsınız. Database admin ler bunları ayarlıyor, herkes her şey de yetkili değil.

GRANT ALL ON \*.\* TO ‘user\_name’@’localhost’;

--- tüm yetkileri (SELECT,INSERT,UPDATE,DELETE)tüm veri tabanlarındaki tüm tablolarda şu kullanıcıya yetki ver,

Bu komut satırıyla da deneme user’ımıza full yetki vermiş oluruz.

(TO 'user\_name'@'localhost': Verilen yetkilerin hangi kullanıcıya ve hangi bağlantı noktasından (localhost) geldiğini belirtir.)

DROP USER user\_name; ----- kullanıcıyı silmiş

İNDEX

İndex diye bir verimiz var

Veri arttıkça arama işleminde bize veri o kadar geç dönüyor, aradığımıza ulaşmak zor oluyor. Bunları kolaylaştırmak için index denen yapılar kullanılmiş. Yaptığı şey veri tabanının kendisinde ( artık hangi algoritmayı kullanıyorsa) bir algoritma kullanarak, sizin belirttiğiniz sütunlarla ilgili bir index oluşturuyor. Bu index vasıtasıyla veriye hızlı erişiliyor.

Mesela, buradaki yazı olan sütuna bakalım, içerdiği harflere göre bir sayısal değerle tutsam bir yerde daha hızlı erişebilirim ) ASCII kodlarına göre mesela bu bir yöntem olabilir) Benim bildiğim bilmediğim herhangi bir algoritmayı kullanarak istediğimiz sütunda index oluşturuyorlar. İndex ler vasıtasıyla ben verilere daha kısa sürede erişebiliyorum.

Bu index oluşturmak işlem gerektirdiği için veri tabanımda şişmeye sebep oluyor, extra bir yük getiriyor, hacim artıyor. O yüzden istediğimiz sütuna index atayalım mevzusu pek tavsiye edilmiyor. Hangi sütunlarda çok arama yapılıyorsa o sütunlara index oluşturuluyor. Genelde veri tabanları PK olarak belirlediğiniz sütunlara index oluşturur. Diğerlerini ben belirtebilirim.

Nasıl belirtebilirim, slaytta belirtildiği gibi yapabilirim.

Create index diyoruz sonra hangi tablonun hangi sütununa göre index oluşturacaksam. Hem adına göre oluşturayım hem de soyadına göre oluşturayım, o zaman ne yapacağım ikisini de belirteceğim.

Database Strakçır a geldiğimizde altta bakın Indices diye bir şey oluşturmuş bakın.

Create index nnameicinindex on tracks(name);

Artık ne olacak ben name e göre arama yaptığım zaman bana daha hızlı getirecek ama bu veri tabanımda daha fazla yer kaplamasına sebep oldu. Veri tabanının hacmi artmış oldu ve bu da bana extra maaliyete sebep olmuş oldu. Çok aranan bir sütun varsa bunu yapmalı yoksa yapılmamalı.

Bir indeks oluşturabilmek için “CREATE ANY INDEX” sistem hakkına sahip olmak gerekir.

İndeks bir tablonun bir alanı üzerinde tanımlanabileceği gibi birden fazla alan üzerinde de tanımlanabilir.

* **İndeksler** verilere **hızlı erişim** amacıyla kullanılır.
* OLAP (On-Line Analytical Processing) sistemlerde mümkün olduğu kadar çok indeks kullanılır, OLTP’de ise fazlası uygulamaya zarar verebilir.
* **Primary Key’ler otomatik olarak indekslenir**, Foreign Key’ler de genellikle indekslenmesi gereken alanlardır.

VİEW

View tablolar, bir başka tabirle özet tablolar.

Gerçekte olmayan tablolardır. Sadece SELECT ifadesinden ibarettirler.

Select \* from invoices where Billingcountry=’Germany’;

Ben ben her seferinde bu sorguyuçalıştırmak istemiyorum diyelim, sık çalıştırılan bir sorgudur

Onu view olarak kaydederim

Create view almanyafaturalari as Select \* from invoices where Billingcountry=’Germany’;

Deyince artık bu sorgu ne oldu, bu artık özet bir tablo oldu. Benim invoices tablosundan sorgu vasıtasıyla elde ettiğim bir özet tablo. Ne faydası var

Neden view kullanılır?

* Veri erişimini sınırlamak için
* Karmaşık sorguları kolay hale getirmek için
* Veri bağımsızlığını sağlamak için
* Aynı verinin farklı görünümlerini sunmak için

Mesela hergün döviz kuruna bakmak istese, böyle bir yöntemle database i tekrar tekrar yormamış olur.

\*\*Eğer Primary Key View’da yer alıyorsa DML (insert, delete, update) işlemleri o view üzerinde uygulanabilir. (Deneleyim)

Bu özet tebloyu normal tablo gibi kullanıp veri ekleyebilirim

Transaction

* Daha küçük parçalara ayrılamayan en küçük işlem yığınına Transaction denir.
* Geçerli kabul edilmesi bir dizi işlemin tamamının yolunda gitmesine bağlı durumlarda transaction kullanılır.
* Transaction bloğu ya hep ya hiç mantığı ile çalışır. Ya tüm işlemler düzgün olarak gerçekleşir ve geçerli kabul edilir veya bir kısım işlemler yolunda gitse bile, blok sona ermeden bir işlem bile yolunda gitmese hiçbir işlem olmamış kabul edilir.
* Bir transaction COMMIT işlemi ile tamamlanır. Eğer ROLLBACK işlemi uygulanırsa tüm transaction işlemleri geri alınır.

Örneğin; bir havale işleminde, havale yapanın hesap bilgilerinden havale yaptığı miktar düşüldükten sonra, elektrik kesintisi, donanımsal veya yazılımsal bir arıza nedeniyle alıcının hesabına bu miktar eklenemez ise; gönderenin hesabından düşülen paranın iade edilmesi gerekir.

Aksi halde bu paranın sahibinin kimliği kaybedilmiş olur. Bu da sistemin olası haller dışında veri kaybetmeye müsait bir durumda olması demektir. Bu tür aksaklıklar transaction kavramı sayesinde kontrol altına alınır.

Ya da herkesin maaşını güncellemek istiyorsunuz, ard arda sorgular var, gidiyor tek tek bakıyor sırayla bunları yapması gerekiyor. Bütünlüğü bozmadan yapmalı arada bir yerde hata alırsa tüm işlemi iptal etmeli.

Begin ile başlayacak, her şey sağlıklı ise commit yok değilse rolback verir.

mysql\_connect("localhost", "kullanici", "parola");

mysql\_select\_db("veritabani");

*//transaction basliyor* mysql\_query("BEGIN");

*//sorgular hazirlaniyor*

$sorgu1 = mysql\_query("UPDATE akbank SET mevcut\_para = mevcut\_para – 10000 WHERE hesap\_no = '625021'");

$sorgu2 = mysql\_query("UPDATE garanti SET mevcut\_para = mevcut\_para + 10000 WHERE hesap\_no = '124500'"); ,

if (!$sorgu1 or !$sorgu2 ) {

*//hata var, geri aliyoruz* mysql\_query("ROLLBACK");

}

else {

*//hata yok, islem tamamlaniyor* mysql\_query("COMMIT");

Trigger:

Kelime anlamı tetikleyici demektir. Tablolar üzerinde gerçeklerşen bir işlemin başka bir işlemi  
tetiklemesini sağlayan sql ifadeleri topluluğuna trigger denir

Bir veri tabanında belirli bir tablodaki bir satırın değişmesi gibi belirli bir işlem gerçekleştiğinde otomatik olarak bir işlemi başlatan bir dizi SQL ifadesidir.

Bir trigger bir olaydan (insert, delete ya da update ifadelerin belirtilen tabloda oluşması) ve bir hareketten (ilgili prosedür) oluşur.

Mesela basit olarak şirket tablosuna yeni bir kayıt eklendiğinde (insert) git log tablosuna şirket no’sunu ve o anki tarih ve zamanı kaydet (insert).

trigger\_time:Trigger’ın Çalışma zamanıdır. Yani After(sonra,işlemden sonra Çalış)  
veya Before(önce,işlemden önce Çalış).

trigger\_event:Trigger’ın hangi olayda Çalışcağını yazıyoruz. Bu olaylar:  
INSERT, UPDATE veya DELETE olayları olabilir.

trigger\_stmt: BEGIN ……. END blogları arasında tanımladığımız olay ve zamanda yapılmasını istediğimiz işleri yazıyoruz.