Veri Tabanı Sistemleri

SELECT INTO, PRIMARY KEY, FOREIGN KEY, Constraints, Data Types

SQL Server SELECT INTO

- Deyim yeni bir tablo oluşturur ve sorgudaki satırları bu tabloya ekler.SELECT INTO
- Aşağıdaki deyim tabloyu oluşturur ve koşulu karşılayan satırları tablodan tabloya kopyalar.

```
SELECT
    select_list
INTO
    destination
FROM
    source
[WHERE condition]
```

- Kaynak tablodaki kısmi verileri kopyalamak istiyorsanız, hangi satırların kopyalanacağını belirtmek için WHERE kullanırsınız
- Benzer şekilde, SELECT listesinde belirterek, hangi sütunların tabloya kopyalanacağını da belirtebilirsiniz.

- SELECT INTO deyiminin birincil anahtar ve indeksler gibi kısıtlamaları kaynak tablodan hedef tabloya kopyalamadığını unutmayın.

A) Aynı veritabanı örneği içinde tabloyu kopyalamak için SQL Server SELECT INTO kullanma

- İlk olarak, yeni tabloyu depolamak için yeni bir şema oluşturun.

```
CREATE SCHEMA marketing;
GO
```

- İkinci olarak, tabloyu tablo gibi oluşturun ve tüm satırları tablodan tabloya kopyalayın:
- **marketing.customers** tablosunu **sales.customers** gibi oluşturun ve **sales.customers** tablosundaki tüm satırları **marketing.customers** tablosuna kopyalayın.

```
SELECT
    *
INTO
    marketing.customers
FROM
    sales.customers;
```

- Üçüncü olarak, kopyayı doğrulamak için tablodaki verileri sorgulayın: `marketing.customers`

```
SELECT
    *
FROM
    marketing.customers;
```

Aşağıdaki resimde kısmi çıktı gösterilmektedir:

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950
11	Deshawn	Mendoza	NULL	deshawn.mendoza@yahoo.com	8790 Cobblestone Street	Monsey	NY	10952
12	Robby	Sykes	(516) 583-7761	robby.sykes@hotmail.com	486 Rock Maple Street	Hempstead	NY	11550
13	Lashawn	Ortiz	NULL	lashawn.ortiz@msn.com	27 Washington Rd.	Longview	TX	75604
14	Garry	Espinoza	NULL	garry.espinoza@hotmail.com	7858 Rockaway Court	Forney	TX	75126

B)Tabloyu veritabanları arasında kopyalamak için SQL Server SELECT INTO deyimini kullanma

- İlk olarak, test için `TestDb` adlı yeni bir veritabanı oluşturun

```
CREATE DATABASE TestDb;
GO
```

- İkinci olarak, mevcut veritabanındaki **BikeStores** veritabanından **TestDb.dbo.customers** tablosuna verileri kopyalayın. Bu sefer, yalnızca **California**'da bulunan müşterilerin müşteri kimliği, ad, soyad ve e-posta bilgilerini kopyalıyoruz: **sales.customers**

```
SELECT
    customer_id,
    first_name,
    last_name,
    email
INTO
    TestDb.dbo.customers
FROM
    sales.customers
WHERE
    state = 'CA';
```

Üçüncü olarak, kopyayı doğrulamak için verileri sorgulayın: `TestDb.dbo.customers`

```
SELECT
    *
FROM
    TestDb.dbo.customers;
```

İşte kısmi sonuç kümesi:

customer_id	first_name	last_name	email
2	Kasha	Todd	kasha.todd@yahoo.com
3	Tameka	Fisher	tameka.fisher@aol.com
5	Charolette	Rice	charolette.rice@msn.com
24	Corene	Wall	corene.wall@msn.com
30	Jamaal	Albert	jamaal.albert@gmail.com
31	Williemae	Holloway	williemae.holloway@msn.com
32	Araceli	Golden	araceli.golden@msn.com
33	Deloris	Burke	deloris.burke@hotmail.com
40	Ronna	Butler	ronna.butler@gmail.com
46	Monika	Berg	monika.berg@gmail.com
47	Bridgette	Guerra	bridgette.guerra@hotmail.com
53	Saturnina	Garner	saturnina.garner@gmail.com
60	Neil	Mccall	neil.mccall@gmail.com
67	Tommie	Melton	tommie.melton@gmail.com

SQL Server PRIMARY KEY

Birincil anahtar, bir tablodaki her satırı benzersiz olarak tanımlayan bir sütun veya sütun grubudur. Kısıtlamayı kullanarak bir tablo için birincil anahtar oluşturursunuz.`PRIMARY KEY`

Birincil anahtar yalnızca bir sütundan oluşuyorsa, kullanım kısıtlamasını sütun kısıtlaması olarak tanımlayabilirsiniz:

```
CREATE TABLE table_name (
    pk_column data_type PRIMARY KEY,
    ...
);
```

Birincil anahtarın iki veya daha fazla sütunu olması durumunda, kısıtlamayı bir tablo kısıtlaması olarak kullanmanız gerekir:

```
CREATE TABLE table_name (
    pk_column_1 data_type,
    pk_column_2 data type,
    ...
    PRIMARY KEY (pk_column_1, pk_column_2)
);
```

- Her tablo yalnızca bir birincil anahtar içerebilir. Birincil anahtara katılan tüm sütunlar tanımlanmalıdır. SQL Server, bu sütunlar için kısıtlama belirtilmemişse, tüm birincil anahtar sütunları için kısıtlamayı otomatik olarak `NOT NULL` ayarlar.

- SQL Server ayrıca, birincil anahtar oluşturduğunuzda otomatik olarak benzersiz bir kümelenmiş dizin (veya bu şekilde belirtilmişse kümelenmemiş bir dizin) oluşturur.

SQL Server PRIMARY KEY Örnekleri

Aşağıdaki örnek, tek bir sütundan oluşan birincil anahtara sahip bir tablo oluşturur:

```
CREATE TABLE sales.activities (
    activity_id INT PRIMARY KEY IDENTITY,
    activity_name VARCHAR (255) NOT NULL,
    activity_date DATE NOT NULL
);
```

Bu **sales.activities** tablosunda, **activity_id** sütunu birincil anahtar sütunudur. Bu, **activity_id** sütununun benzersiz değerler içerdiği anlamına gelir.

IDENTITY özelliği, **activity_id** sütunu için benzersiz tamsayı değerlerini otomatik olarak oluşturmak amacıyla kullanılır

Aşağıdaki ifade, birincil anahtarı iki sütundan oluşan **sales.participants** adlı yeni bir tablo oluşturur:

```
CREATE TABLE sales.participants(
    activity_id int,
    customer_id int,
    PRIMARY KEY(activity_id, customer_id)
);
```

Bu örnekte, **activity_id** veya **customer_id** sütunlarındaki değerler tekrarlanabilir, ancak her iki sütundaki değerlerin birleşimi benzersiz olmalıdır.

Genellikle, bir tablo oluşturulurken her zaman birincil anahtar tanımlanır. Ancak bazen, mevcut bir tabloda birincil anahtar tanımlanmamış olabilir. Bu durumda, **ALTER TABLE** ifadesini kullanarak tabloya birincil anahtar ekleyebilirsiniz. Aşağıdaki örneği inceleyin:

Aşağıdaki deyim, birincil anahtarı olmayan bir tablo oluşturur:

```
CREATE TABLE sales.events(
    event_id INT NOT NULL,
    event_name VARCHAR(255),
    start_date DATE NOT NULL,
    duration DEC(5,2)
);
```

Sütunu birincil anahtar yapmak için aşağıdaki deyimi kullanırsınız: `event_id` `ALTER TABLE`

```
ALTER TABLE sales.events
ADD PRIMARY KEY(event_id);
```

➔ Tabloda zaten veri varsa. sütunu birincil anahtar olarak vükseltmeden önce. tablodaki değerlerin benzersiz olduğundan emin olmanız gerektiğini unutmayın.`sales.events event_id`

SQL Server FOREIGN KEY

Yabancı anahtar. bir tablodaki baska bir tablonun (veya kendi kendine basvuru durumunda avnı tablonun) bir satırını benzersiz bir şekilde tanımlayan bir sütun veya sütun grubudur.

Yabancı anahtar oluşturmak için `FOREIGN KEY` kısıtlaması kullanırsınız.

```
CREATE TABLE procurement.vendor_groups (
    group_id INT IDENTITY PRIMARY KEY,
    group_name VARCHAR (100) NOT NULL
);

CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
);
```

Her tedarikçi bir tedarikçi grubuna aittir ve her tedarikçi grubunun sıfır veya daha fazla tedarikçisi olabilir. **vendor_groups** ve **vendors** tabloları arasındaki ilişki bire-çoktur

```
DROP TABLE vendors;


CREATE TABLE procurement.vendors (
        vendor_id INT IDENTITY PRIMARY KEY,
        vendor_name VARCHAR(100) NOT NULL,
        group_id INT NOT NULL,
        CONSTRAINT fk_group FOREIGN KEY (group_id)
        REFERENCES procurement.vendor_groups(group_id)
);
```

- Yukarıdaki ifadeler tabloyu siler ve bir kısıtlamayla yeniden oluşturur.

- Tablo artık **üst tablo (Parent Table)** olarak adlandırılır ve bu, yabancı anahtar kısıtlamasının başvurduğu tablodur. Tablo, yabancı anahtar kısıtlamasının uygulandığı tablo olan **alt tablo (Chil Table)** olarak adlandırılır. `vendor_groups vendors`

- Yukarıdaki ifadede, aşağıdaki madde tablo içindeki group_id'yi tablo içindeki fk_group'a bağlayan bir kısıtlama oluşturur:

  CONSTRAINT fk_group FOREIGN KEY (group_id) REFERENCES procurement.vendor_groups(group_id)

SQL Server FOREIGN KEY Sözdizimi

Kısıtlama oluşturmak için genel sözdizimi aşağıdaki gibidir: `FOREIGN KEY`

```
CONSTRAINT fk_constraint_name
FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1,column2,..)
```

İlk olarak, anahtar sözcükten sonra kısıtlama adını belirtin. Kısıtlama adı isteğe bağlıdır, bu nedenle aşağıdaki gibi bir kısıtlama tanımlamak mümkündür:`CONSTRAINT FOREIGN KEY`

```
FOREIGN KEY (column_1, column2,...)
REFERENCES parent_table_name(column1,column2,..)
```

Bu durumda, SQL Server kısıtlama için otomatik olarak bir ad oluşturur.

İkinci olarak, anahtar sözcükten sonra parantez içine alınmış virgülle ayrılmış yabancı anahtar sütunlarının bir listesini belirtin.

Üçüncü olarak, yabancı anahtarın başvurduğu üst tablonun adını ve alt tablodaki sütunla bağlantısı olan virgülle ayrılmış sütunların listesini belirtin.

SQL Server FOREIGN KEY Örneği

İlk olarak, tabloya birkaç satır ekleyin: `vendor_groups`

```
INSERT INTO procurement.vendor_groups(group_name)
VALUES('Third-Party Vendors'),
      ('Interco Vendors'),
      ('One-time Vendors');
```

İkinci olarak, tabloya bir satıcı grubuna sahip yeni bir satıcı ekleyin: `vendors`

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('ABC Corp',1);
```

Üçüncü olarak, satıcı grubu tabloda bulunmayan yeni bir satıcı eklemeyi deneyin: `vendor_groups`

```
INSERT INTO procurement.vendors(vendor_name, group_id)
VALUES('XYZ Corp',4);
```

SQL Server aşağıdaki hatayı verdi:

```
The INSERT statement conflicted with the FOREIGN KEY constraint "fk_group"
```

Bu örnekte, kısıtlama nedeniyle, SQL Server eklemeyi reddetti ve bir hata verdi. `FOREIGN KEY`

Referans Eylemleri

Yabancı anahtar kısıtlaması bilgi tutarlılığı sağlar. Bu, yalnızca üst tabloda karşılık gelen bir satır varsa alt tabloya bir satır ekleyebileceğiniz anlamına gelir.

Ayrıca, yabancı anahtar kısıtlaması, üst tablodaki satır güncellendiğinde veya silindiğinde başvuru eylemlerini aşağıdaki gibi tanımlamanıza olanak tanır:

```
FOREIGN KEY (foreign_key_columns)
    REFERENCES parent_table(parent_key_columns)
    ON UPDATE action
    ON DELETE action;
```

ve üst tablodaki bir satır güncelleştirildiğinde ve silindiğinde hangi eylemin yürütüleceğini belirtiniz. Aşağıdakiler izin verilen eylemlerdir: `ON UPDATE ON DELETE NO ACTION CASCADE SET NULL SET DEFAULT`

Üst Tablodaki Satırları ve Eylemleri Silme

Üst tablodaki bir veya daha fazla satırı silerseniz, aşağıdaki eylemlerden birini ayarlayabilirsiniz:

- `ON DELETE NO ACTION`: SQL Server bir hata oluşturur ve üst tablodaki satırda silme eylemini geri alır.
- `ON DELETE CASCADE`: SQL Server, üst tablodan silinen satıra karşılık gelen alt tablodaki satırları siler.
- `ON DELETE SET NULL`: SQL Server, alt tablodaki satırları, üst tablodaki karşılık gelen satırların silinmesi durumuna ayarlar. Bu eylemi yürütmek için, yabancı anahtar sütunlarının null atanabilir olması gerekir.NULL
- `ON DELETE SET DEFAULT:` SQL Server, üst tablodaki karşılık gelen satırlar silinirse, alt tablodaki satırları varsayılan değerlerine ayarlar. Bu eylemi yürütmek için, yabancı anahtar sütunlarının varsayılan tanımlara sahip olması gerekir. Null atanabilir bir sütunun varsayılan değerinin if varsayılan değeri belirtilmiş değildir.

Varsayılan olarak, açıkça herhangi bir eylem belirtmezseniz SQL Server, `ON DELETE NO ACTION` uygular.

Üst Tablodaki Satırları Güncelleştirme

Üst tablodaki bir veya daha fazla satırı güncelleştirirseniz, aşağıdaki eylemlerden birini ayarlayabilirsiniz:

- `ON UPDATE NO ACTION`: SQL Server bir hata oluşturur ve üst tablodaki satırda güncelleştirme eylemini geri alır.
- `ON UPDATE CASCADE`: SQL Server, üst tablodaki satırlar güncelleştirildiğinde alt tablodaki karşılık gelen satırları güncelleştirir.
- `ON UPDATE SET NULL`: SQL Server, alt tablodaki satırları, üst tablodaki ilgili satırın güncelleştirildiği zamana ayarlar. Bu eylemin yürütülmesi için yabancı anahtar sütunlarının null atanabilir olması gerektiğini unutmayın.
- `ON UPDATE SET DEFAULT`: SQL Server, üst tablodaki karşılık gelen satırların güncelleştirildiği alt tablodaki satırlar için varsayılan değerleri ayarlar.

SQL Server CHECK

Kısıtlama, bir Boole ifadesini karşılaması gereken bir sütundaki değerleri belirtmenize olanak tanır. `CHECK`

Örneğin, pozitif birim fiyatları zorunlu kılmak için şunları kullanabilirsiniz:

```
CREATE SCHEMA test;
GO

CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2) CHECK(unit_price > 0)
);
```

Gördüğünüz gibi, kısıtlama tanımı veri türünden sonra gelir. Anahtar kelimeden ve ardından parantez içinde mantıksal bir ifadeden oluşur: `CHECK` `CHECK`

```
CHECK(unit_price > 0)
```

Anahtar sözcüğü aşağıdaki gibi kullanarak kısıtlamaya ayrı bir ad da atayabilirsiniz: `CONSTRAINT`

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2) CONSTRAINT positive_price CHECK(unit_price > 0)
);
```

Açık adlar, hata iletilerini sınıflandırmaya yardımcı olur ve bunları değiştirmek istediğinizde kısıtlamalara başvurmanıza olanak tanır.

Bu şekilde bir kısıtlama adı belirtmezseniz, SQL Server sizin için otomatik olarak bir ad oluşturur.

Aşağıdaki insert deyimine bakın:

```
INSERT INTO test.products(product_name, unit_price)
VALUES ('Awesome Free Bike', 0);
```

SQL Server aşağıdaki hatayı verdi:

```
The INSERT statement conflicted with the CHECK constraint "positive_price"
```

Hata, birim fiyatın kısıtlamada belirtildiği gibi sıfırdan büyük olmaması nedeniyle oluştu. `CHECK`

Aşağıdaki deyim düzgün çalışır çünkü kısıtlamada tanımlanan mantıksal ifade şu şekilde değerlendirilir: `CHECK` `TRUE`

```
INSERT INTO test.products(product_name, unit_price)
VALUES ('Awesome Bike', 599);
```

Kısıtlamalar, Boole ifadesi olarak değerlendirilmesine neden olan değerleri reddeder.CHECK FALSE olarak değerlendirildiğinden, bir kısıtlamayı atlamak için NULL UNKNOWN ifadede kullanılabilir.

Örneğin, birim fiyatı aşağıdaki sorguda gösterildiği gibi olan bir ürün ekleyebilirsiniz: `NULL`

```
INSERT INTO test.products(product_name, unit_price)
VALUES ('Another Awesome Bike', NULL);
```

İşte çıktı:

```
(1 row affected)
```

SQL Server sütuna eklendi ve bir hata döndürmedi. `NULL` `unit_price`

Bunu düzeltmek için, sütun için bir kısıtlama kullanmanız gerekir. `NOT NULL` `unit_price`

Birden Çok Sütuna Başvuran CHECK

Bir kısıtlama birden çok sütuna başvurabilir. Örneğin, tabloda normal ve indirimli fiyatlar depoluyorsunuz ve indirimli fiyatın her zaman normal fiyattan daha düşük olduğundan emin olmak istiyorsunuz: `CHECK` `test.products`

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2) CHECK(unit_price > 0),
    discounted_price DEC(10,2) CHECK(discounted_price > 0),
    CHECK(discounted_price < unit_price)
);
```

ve için ilk iki kısıtlama tanıdık gelmelidir. `unit_price` `discounted_price`

Üçüncü kısıtlama, belirli bir sütuna bağlı olmayan yeni bir sözdizimi kullanır. Bunun yerine, virgülle ayrılmış sütun listesinde ayrı bir satır öğesi olarak görünür.

İlk iki sütun kısıtlaması sütun kısıtlamaları, üçüncü sütun kısıtlaması ise tablo kısıtlamasıdır.

Sütun kısıtlamalarını tablo kısıtlamaları olarak yazabileceğinizi unutmayın. Ancak, tablo kısıtlamalarını sütun kısıtlamaları olarak yazamazsınız. Örneğin, yukarıdaki ifadeyi aşağıdaki gibi yeniden yazabilirsiniz:

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2),
    discounted_price DEC(10,2),
    CHECK(unit_price > 0),
    CHECK(discounted_price > 0),
    CHECK(discounted_price > unit_price)
);
```

veya hatta:

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2),
    discounted_price DEC(10,2),
    CHECK(unit_price > 0),
    CHECK(discounted_price > 0 AND discounted_price > unit_price)
);
```

Ayrıca, bir tablo kısıtlamasına, sütun kısıtlamasıyla aynı şekilde bir ad atayabilirsiniz:

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2),
    discounted_price DEC(10,2),
    CHECK(unit_price > 0),
    CHECK(discounted_price > 0),
    CONSTRAINT valid_prices CHECK(discounted_price > unit_price)
);
```

Varolan bir tabloya CHECK Kısıtlaması Ekleme

Varolan bir tabloya kısıtlama eklemek için `CHECK ALTER TABLE ADD CONSTRAINT` deyimini kullanırsınız.

Aşağıdaki tabloya sahip olduğunuzu varsayalım: `test.products`

```
CREATE TABLE test.products(
    product_id INT IDENTITY PRIMARY KEY,
    product_name VARCHAR(255) NOT NULL,
    unit_price DEC(10,2) NOT NULL
);
```

Tabloya bir kısıtlama eklemek için aşağıdaki ifadeyi kullanırsınız: `CHECK` `test.products`

```
ALTER TABLE test.products
ADD CONSTRAINT positive_price CHECK(unit_price > 0);
```

Kısıtlama içeren yeni bir sütun eklemek için aşağıdaki deyimi kullanırsınız: `CHECK`

```
ALTER TABLE test.products
ADD discounted_price DEC(10,2)
CHECK(discounted_price > 0);
```

`valid_price` adında bir kısıtlama eklemek için aşağıdaki deyimi kullanırsınız:
`CHECK valid_price`

```
ALTER TABLE test.products
ADD CONSTRAINT valid_price
CHECK(unit_price > discounted_price);
```

CHECK Kısıtlamalarını Kaldırma

Bir kısıtlamayı kaldırmak için şu ifadeyi kullanırsınız: `CHECK` `ALTER TABLE DROP CONSTRAINT`

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

Bir kısıtlamaya belirli bir ad atarsanız, deyimde bu ada başvurabilirsiniz. `CHECK`

Ancak, kısıtlamaya belirli bir ad atamadıysanız, aşağıdaki ifadeyi kullanarak onu bulmanız gerekir: `CHECK`

```
EXEC sp_help 'table_name';
```

Mesela:

```
EXEC sp_help 'test.products';
```

Bu deyim, kısıtlama adları da dahil olmak üzere birçok bilgi verir:

constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
CHECK on column discounted_price	CK__products__discou__42ACE4D4	(n/a)	(n/a)	Enabled	Is_For_Replication	([discounted_price]>(0))
PRIMARY KEY (clustered)	PK__products__47027DF5162FC24B	(n/a)	(n/a)	(n/a)	(n/a)	product_id
CHECK on column unit_price	positive_price	(n/a)	(n/a)	Enabled	Is_For_Replication	([unit_price]>(0))

Aşağıdaki ifade kısıtlamayı ortadan kaldırır: `positive_price`

```
ALTER TABLE test.products
DROP CONSTRAINT positive_price;
```

Ekleme veya Güncelleme için CHECK Kısıtlamalarını Devre Dışı Bırakmak

Ekleme veya güncelleştirme için bir kısıtlamayı devre dışı bırakmak için aşağıdaki deyimi kullanırsınız: `CHECK`

```
ALTER TABLE table_name
NOCHECK CONSTRAINT constraint_name;
```

Aşağıdaki deyim kısıtlamayı devre dışı bırakır: `valid_price`

```
ALTER TABLE test.products
NO CHECK CONSTRAINT valid_price;
```

SQL Server UNIQUE

SQL Server kısıtlamaları, bir sütunda veya sütun grubunda depolanan verilerin tablodaki satırlar arasında benzersiz olmasını sağlamanıza olanak tanır. `UNIQUE`

Aşağıdaki deyim, sütundaki verileri tablodaki satırlar arasında benzersiz olan bir tablo oluşturur: `email` `hr.persons`

```
CREATE SCHEMA hr;
GO

CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE
);
```

Bu sözdiziminde, **UNIQUE** kısıtlamasını bir sütun kısıtlaması olarak tanımlıyorsunuz. **UNIQUE** kısıtlamasını aşağıdaki gibi bir tablo kısıtlaması olarak da tanımlayabilirsiniz:

```
CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    UNIQUE(email)
);
```

Arka planda, SQL Server, **UNIQUE** kısıtlamasına katılan sütunlarda depolanan verilerin benzersizliğini sağlamak için otomatik olarak bir **UNIQUE** indeks oluşturur. Bu nedenle, bir tekrar eden satırı eklemeye çalışırsanız, SQL Server değişikliği reddeder ve **UNIQUE** kısıtlamasının ihlal edildiğini belirten bir hata mesajı döner.

Aşağıdaki deyim tabloya yeni bir satır ekler: `hr.persons`

```
INSERT INTO hr.persons(first_name, last_name, email)
VALUES('John','Doe','j.doe@bike.stores');
```

Deyim beklendiği gibi çalışır. Ancak, yinelenen e-posta nedeniyle aşağıdaki ifade başarısız olur:

```
INSERT INTO hr.persons(first_name, last_name, email)
VALUES('Jane','Doe','j.doe@bike.stores');
```

SQL Server aşağıdaki hata iletisini verdi:

```
Violation of UNIQUE KEY constraint 'UQ__persons__AB6E616417240E4E'. Cannot
```

Kısıtlama için ayrı bir ad belirtmezseniz, SQL Server bunun için otomatik olarak bir ad oluşturur. Bu örnekte, kısıtlama adı 'dir ve tam olarak okunabilir değildir. `UNIQUE` `UQ__persons__AB6E616417240E4E`

Bir kısıtlamaya belirli bir ad atamak için, anahtar sözcüğü aşağıdaki gibi kullanırsınız: `UNIQUE` `CONSTRAINT`

```
CREATE TABLE hr.persons (
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    CONSTRAINT unique_email UNIQUE(email)
);
```

Bir kısıtlamaya belirli bir ad atamanın avantajları şunlardır: `UNIQUE`

- Hata mesajını sınıflandırmak daha kolaydır.
- Değiştirmek istediğinizde kısıtlama adına başvurabilirsiniz.

UNIQUE vs. PRIMARY KEY

Her iki kısıtlama da verilerin benzersizliğini zorunlu kılsa da, birincil anahtar sütunları olmayan bir sütunun veya bir sütun grubunun benzersizliğini zorunlu kılmak istediğinizde, UNIQUE kısıtlama yerine PRIMARY KEY kısıtlamayı kullanmalısınız.Ayrıca, kısıtlamalar normal bir değer olarak kabul edilir, bu nedenle sütun başına yalnızca bir değere izin verir.

Aşağıdaki deyim, sütundaki değeri şu şekilde olan bir satır ekler: `email NULL`

```
INSERT INTO hr.persons(first_name, last_name)
VALUES('John','Smith');
```

Şimdi, sütuna bir tane daha eklemeye çalışırsanız, bir hata alırsınız: `NULL email`

```
INSERT INTO hr.persons(first_name, last_name)
VALUES('Lily','Bush');
```

İşte çıktı:

```
Violation of UNIQUE KEY constraint 'UQ__persons__AB6E616417240E4E'. Cannot
```

Bir Sütun Grubu için UNIQUE Kısıtlamalar

Bir sütun grubu için bir kısıtlama tanımlamak için, bunu sütun adları virgülle ayrılmış bir tablo kısıtlaması olarak aşağıdaki gibi yazarsınız: `UNIQUE`

```
CREATE TABLE table_name (
    key_column data_type PRIMARY KEY,
    column1 data_type,
    column2 data_type,
    column3 data_type,
    ...,
    UNIQUE (column1,column2)
);
```

Aşağıdaki örnek, iki sütundan oluşan bir kısıtlama oluşturur ve : `UNIQUE` `person_id` `skill_id`

```
CREATE TABLE hr.person_skills (
    id INT IDENTITY PRIMARY KEY,
    person_id int,
    skill_id int,
    updated_at DATETIME,
    UNIQUE (person_id, skill_id)
);
```

Mevcut Sütunda UNIQUE Kısıtlama Eklemek

Varolan bir sütuna veya tablodaki bir sütun grubuna kısıtlama eklediğinizde, SQL Server tüm değerlerin benzersiz olduğundan emin olmak için önce bu sütunlardaki varolan verileri inceler. SQL Server yinelenen değerleri bulursa, bir hata döndürür ve kısıtlamayı eklemez. `UNIQUE` `UNIQUE`

Aşağıda, bir tabloya kısıtlama eklemenin sözdizimi gösterilmektedir: `UNIQUE`

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE(column1, column2,...);
```

Aşağıdaki tabloya sahip olduğunuzu varsayalım: `hr.persons`

```
CREATE TABLE hr.persons (
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255),
    phone VARCHAR(20),
);
```

Aşağıdaki deyim sütuna bir kısıtlama ekler: `UNIQUE email`

```
ALTER TABLE hr.persons
ADD CONSTRAINT unique_email UNIQUE(email);
```

Benzer şekilde, aşağıdaki deyim telefon sütununa bir kısıtlama ekler: `UNIQUE`

```
ALTER TABLE hr.persons
ADD CONSTRAINT unique_phone UNIQUE(phone);
```

UNIQUE Kısıtlamalarını Silme

Bir kısıtlama tanımlamak için, deyimi aşağıdaki gibi kullanırsınız: `UNIQUE` `ALTER TABLE DROP CONSTRAINT`

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

Aşağıdaki deyim hr.person tablosundan kısıtlamayı kaldırır: `unique_phone`

```
ALTER TABLE hr.persons
DROP CONSTRAINT unique_phone;
```

UNIQUE Kısıtlamaları Değiştirme

SQL Server'ın bir kısıtlamayı değiştirmek için doğrudan bir deyimi yoktur, bu nedenle, kısıtlamayı değiştirmek istiyorsanız önce kısıtlamayı bırakmanız ve yeniden oluşturmanız gerekir.

SQL Server NOT NULL Kısıtlaması

- SQL Server NOT NULL kısıtlamaları, bir sütunun NULL'u varsaymaması gerektiğini belirtir. Aşağıdaki örnek, şu sütunlar için NOT NULL kısıtlamaları olan bir tablo oluşturur: first_name, last_name ve email:

```
CREATE SCHEMA hr;
GO

CREATE TABLE hr.persons(
    person_id INT IDENTITY PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(20)
);
```

- NOT NULL kısıtlamalarının her zaman sütun kısıtlamaları olarak yazıldığını unutmayın.
- Varsayılan olarak, NOT NULL kısıtlamasını belirtmezseniz, SQL Server sütunun NULL'u kabul etmesine izin verir. Bu örnekte, telefon sütunu NULL'u kabul edebilir.

Mevcut bir sütuna NOT NULL kısıtlaması ekleme

- Mevcut bir sütuna NOT NULL kısıtlamasını eklemek için şu adımları izleyin:
  - Öncelikle, sütunda NULL olmayacak şekilde tabloyu güncelleyin:

```
UPDATE table_name
SET column_name = <value>
WHERE column_name IS NULL;
```

  - İkinci olarak, sütunun özelliğini değiştirmek için tabloyu değiştirin:

```
ALTER TABLE table_name
ALTER COLUMN column_name data_type NOT NULL;
```

- Örneğin, hr.persons tablosunun telefon sütununa NOT NULL kısıtlamasını eklemek için aşağıdaki ifadeleri kullanırsınız:

  - İlk olarak, bir kişinin telefon numarası yoksa, telefon numarasını şirket telefon numarasına güncelleyin, örneğin (408) 123 4567:

```
UPDATE hr.persons
SET phone = "(408) 123 4567"
WHER phone IS NULL;
```

  - İkinci olarak, telefon sütununun özelliğini değiştirin:

```
ALTER TABLE hr.persons
ALTER COLUMN phone VARCHAR(20) NOT NULL;
```

NOT NULL kısıtlamasını kaldırma

- Bir sütundan NOT NULL kısıtlamasını kaldırmak için, ALTER TABLE ALTER COLUMN ifadesini aşağıdaki gibi kullanırsınız:

```
ALTER TABLE table_name
ALTER COLUMN column_name data_type NULL;
```

- Örneğin, telefon sütunundan NOT NULL kısıtlamasını kaldırmak için, aşağıdaki ifadeyi kullanırsınız:

```
ALTER TABLE hr.pesons
ALTER COLUMN phone VARCHAR(20) NULL;
```

SQL Server Veri Türleri

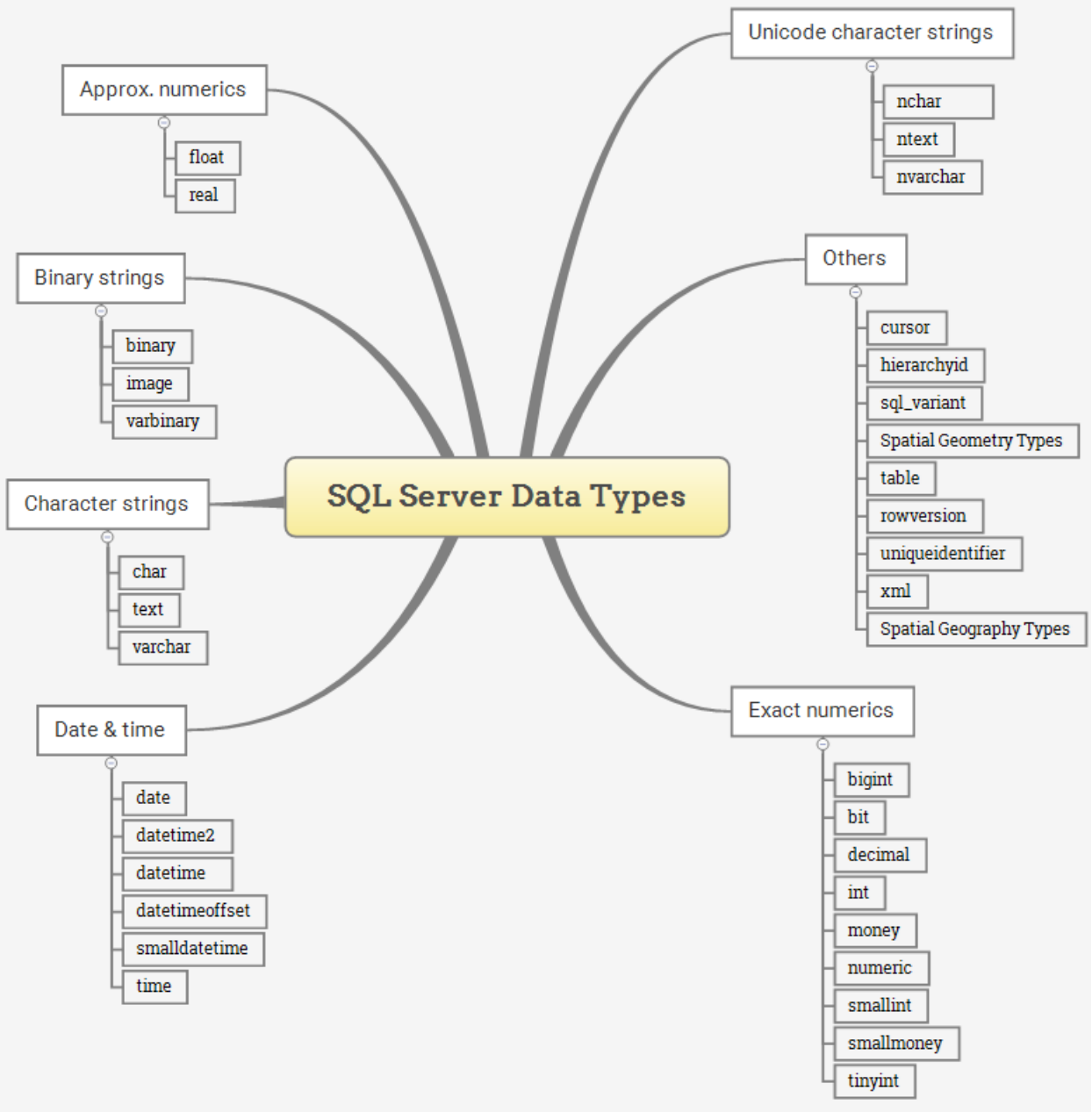
- SQL Server'da bir sütun, değişken ve parametre, bir türle ilişkilendirilen veya veri türü olarak da bilinen bir değeri tutar.

- Veri türü, bu nesnelerin depolayabileceği veri türünü belirten bir özniteliktir.

- Bir tam sayı, karakter dizisi, para birimi, tarih ve saat vb. olabilir.

- SQL Server, örneğin bir sütun tanımlama veya bir değişken bildirme gibi kullanabileceğiniz tüm veri türlerini tanımlayan bir veri türleri listesi sağlar.

- Aşağıdaki resim SQL Server veri türleri sistemini göstermektedir:



- SQL Server'ın gelecekteki sürümünde ntext, text ve image veri türlerini kaldıracağını unutmayın. Bu nedenle, bu veri türlerini kullanmaktan kaçınmalı ve bunun yerine nvarchar(max), varchar(max) ve varbinary(max) veri türlerini kullanmalısınız.

Tam sayısal veri türleri

- Tam sayısal veri türleri, tam sayı, ondalık veya para miktarı gibi tam sayıları depolar.
- Bit, 0, 1 ve NULL olmak üzere üç değerden birini depolarint, bigint, smallint ve tinyint veri türleri tam sayı verilerini depolar.Decimal ve numeric veri türleri, sabit hassasiyet ve ölçeğe sahip sayıları depolar. Decimal ve numeric'in eşanlamlı olduğunu unutmayın.
- Money ve smallmoney veri türleri para birimi değerlerini depolar.
- Yandaki tablo tam sayısal veri türlerinin özelliklerini göstermektedir:

Data Type	Lower limit	Upper limit	Memory
bigint	-2^63 (-9,223,372, 036,854,775,808)	2^63-1 (-9,223,372, 036,854,775,807)	8 bytes
int	-2^31 (-2,147, 483,648)	2^31-1 (-2,147, 483,647)	4 bytes
smallint	-2^15 (-32,767)	2^15 (-32,768)	2 bytes
tinyint	0	255	1 byte
bit	0	1	1 byte/8bit column
decimal	-10^38+1	10^381-1	5 to 17 bytes
numeric	-10^38+1	10^381-1	5 to 17 bytes
money	-922,337, 203, 685,477.5808	+922,337, 203, 685,477.5807	8 bytes
smallmoney	-214,478.3648	+214,478.3647	4 bytes

Yaklaşık sayısal veri türleri

- Yaklaşık sayısal veri türü, kayan nokta sayısal verilerini depolar. Genellikle bilimsel hesaplamalarda kullanılırlar.

Data Type	Lower limit	Upper limit	Memory	Precision
float(n)	-1.79E+308	1.79E+308	Depends on the value of n	7 Digit
real	-3.40E+38	3.40E+38	4 bytes	15 Digit

Tarih ve Saat veri türleri

- Tarih ve saat veri türleri, verileri ve saat verilerini ve tarih saat ofsetini depolar.
- Yeni bir uygulama geliştirirseniz, time, date, datetime2 ve datetimeoffset veri türlerini kullanmalısınız. Çünkü bu türler SQL Standardı ile uyumludur ve daha taşınabilirdir. Ayrıca, time, datetime2 ve datetimeoffset daha fazla saniye hassasiyetine sahiptir ve datetimeoffset time zone'u destekler.

Data Type	Storage size	Accuracy	Lower Range	Upper Range
datetime	8 bytes	Rounded to increments of .000, .003, .007	1753-01-01	9999-12-31
smalldatetime	4 bytes, fixed	1 minute	1900-01-01	2079-06-06
date	3 bytes, fixed	1 day	0001-01-01	9999-12-31
time	5 bytes	100 nanoseconds	00:00:00.0000000	23:59:59.9999
datetimeoffset	10 bytes	100 nanoseconds	0001-01-01	9999-12-31
datetime2	6 bytes	100 nanoseconds	0001-01-01	9999-12-31

Karakter dizeleri veri türleri

- Karakter dizeleri veri türleri, sabit uzunlukta (char) veya değişken uzunlukta verileri (varchar) depolamanıza olanak tanır.
- Metin veri türü, sunucunun kod sayfasında Unicode olmayan verileri depolayabilir.

Data Type	Lower limit	Upper limit	Memory
char	0 chars	8000 chars	n bytes
varchar	0 chars	8000 chars	n bytes + 2 bytes
varchar (max)	0 chars	2^31 chars	n bytes + 2 bytes
text	0 chars	2,147,483,647 chars	n bytes + 4 bytes

Unicode karakter dizisi veri türleri

- Unicode karakter dizisi veri türleri, sabit uzunlukta (nchar) veya değişken uzunlukta (nvarchar) Unicode karakter verilerini depolar.

Data Type	Lower limit	Upper limit	Memory
nchar	0 chars	4000 chars	2 times n bytes
nvarchar	0 chars	4000 chars	2 times n bytes + 2 bytes
ntext	0 chars	1,073,741,823 char	2 times the string length

Binary string veri türleri

- Binary veri türleri sabit ve değişken uzunluktaki binary verileri depolar.

Data Type	Lower limit	Upper limit	Memory
binary	0 bytes	8000 bytes	n bytes
varbinary	0 bytes	8000 bytes	The actual length of data entered + 2 bytes
image	0 bytes	2,147,483,647 bytes	

Diğer veri türleri

Data Type	Description
cursor	for variables or stored procedure OUTPUT parameter that contains a reference to a cursor
rowversion	expose automatically generated, unique binary numbers within a database.
hierarchyid	represent a tree position in a tree hierarchy
uniqueidentifier	16-byte GUID
sql_variant	store values of other data types
XML	store XML data in a column, or a variable of XML type
Spatial Geometry type	represent data in a flat coordinate system.
Spatial Geography type	store ellipsoidal (round-earth) data, such as GPS latitude and longitude coordinates.
table	store a result set temporarily for processing at a later time

SQL Server BIT

- SQL Server BIT veri türü, 0, 1 veya NULL değerini alabilen bir tam sayı veri türüdür.
- Aşağıda BIT veri türünün sözdizimi gösterilmektedir:

```
BIT
```

- SQL Server, BIT sütunlarının depolanmasını optimize eder. Bir tabloda 8 veya daha az bit sütunu varsa, SQL Server bunları 1 bayt olarak depolar. Bir tabloda 9 ila 16 bit sütunu varsa, SQL Server bunları 2 bayt olarak depolar, vb.
- SQL Server, bir dize değerini TRUE'yu 1'e ve FALSE'u 0'a dönüştürür. Ayrıca sıfır olmayan herhangi bir değeri 1'e dönüştürür.

SQL Server BIT örnekleri

- Aşağıdaki ifade, bir BIT sütunu olan yeni bir tablo oluşturur:

```
CREATE TABLE test.sql_server_bit (
    bit_col BIT
);
```

- Bit sütununa bir bit 1 eklemek için aşağıdaki ifadeyi kullanırsınız:

```
INSERT INTO test.sql_server_bit (bit_col)
OUTPUT inserted.bit_col
VALUES(1);
```

- Çıktı şudur:

```
bit_col
-------
1

(1 row affected)
```

- Bit sütununa bir bit 0 eklemek için yandaki ifadeyi kullanırsınız:

```
INSERT INTO test.sql_server_bit (bit_col)
OUTPUT inserted.bit_col
VALUES(0);
```

- Çıktı şudur:

```
bit_col
-------
0

(1 row affected)
```

- Bit sütununa True dize değerini eklerseniz, SQL Server bunu bit 1'e dönüştürür:

```
INSERT INTO test.sql_server_bit (bit_col)
OUTPUT inserted.bit_col
VALUES
    ('True');
```

- Yandaki çıktıyı gösterir:

```
bit_col
-------
1

(1 row affected)
```

- Benzer şekilde, SQL Server, false dize değerini bit 0'a dönüştürür:

```
INSERT INTO test.sql_server_bit (bit_col)
OUTPUT inserted.bit_col
VALUES
    ('False');
```

- Çıktı şudur:

```
bit_col
-------
0

(1 row affected)
```

- SQL Server, sıfır olmayan herhangi bir değeri bit 1'e dönüştürür.

```
INSERT INTO test.sql_server_bit (bit_col)
OUTPUT inserted.bit_col
VALUES
    (0.5);
```

- Çıktı şudur:

```
bit_col
-------
1

(1 row affected)
```