YILDIZ TEKNİK ÜNİVERSİTESİ

Orta-İleri Seviye SQL

Öğr. Gör. Dr. Yasemin Topuz

Yıldız Teknik Üniversitesi

BLM3041 – Veritabanı Yönetimi

31 Ekim 2025 | YTÜ Bilgisayar Mühendisliği

Neler konuşacağız?

- JOIN's İşlemleri
- VIEW Yapısı
- TRANSACTIONS
- DOMAIN
- INDEXES
- YETKİLENDİRME/ROLLER

SQL – NULL

	fname character	lname character	salary integer
1	Franklin	Wong	40000
2	Alex	Freed	89000
3	Ali	Veli	[null]

SELECT AVG (Salary) **FROM** Employee

	avg numeric
1	64500.00

SELECT COUNT (*) **FROM** Employee

	count bigint
1	3

SELECT COUNT (Salary) **FROM** Employee

	count bigint
1	2

```
select fname, lname, salary, salary+1000
as "Zamlı Maaş" from employee
```

	fname character	lname character	salary integer	Zamlı Maaş integer
1	Franklin	Wong	40000	41000
2	Alex	Freed	89000	90000
3	Ali	Veli	[null]	[null]

SQL – Veritabanının Değiştirilmesi – UPDATE

- Tüm çalışanlara %5 zam

```
UPDATE employee SET salary = salary * 1.05;
```

- Maaşı 35.000'den düşük olanlara %5 zam

```
UPDATE employee SET salary = salary * 1.05 WHERE salary < 35000;
```

- Maaşı genel ortalamanın altında olanlara %5 zam

```
UPDATE employee SET salary = salary * 1.05
WHERE salary < (SELECT AVG(salary) FROM employee);
```

SQL – Veritabanının Değiştirilmesi – UPDATE

- Maaşı kendi departman ortalamasının altında olanlara %5 zam

```
UPDATE employee e SET salary = e.salary * 1.05
WHERE e.salary < (SELECT AVG(e2.salary) FROM employee e2
  WHERE e2.dno = e.dno);
```

- Maaşı 40.000 üzerinde olan çalışanların maaşlarını %3, diğerlerini ise %5 artırın.

```
UPDATE employee
SET salary = salary * 1.03
WHERE salary > 40000;

UPDATE employee
SET salary = salary * 1.05
WHERE salary <= 40000;
```

```
UPDATE employee
SET salary = salary * CASE
  WHEN salary > 40000 THEN 1.03
  ELSE 1.05
END;
```

- Sıralama önemlidir.

SQL – Veritabanının Değiştirilmesi – UPDATE

- Her çalışanın projelerde toplam çalışma süresini «sum_proj_hours» sütununa yaz.

```
UPDATE employee e
SET sum_proj_hours = (SELECT SUM(w.hours)
FROM works_on w WHERE w.essn = e.ssn);
```

	ssn [PK] character	fname character v	sum_proj_hours character varying
1	123456789	John	40.0
2	333445555	Franklin	40.0
3	453453453	Joyce	40.0
4	666884444	Ramesh	0
5	888665555	James	13.0
6	987654321	Jennifer	35.0
7	987987987	Ahmad	40.0
8	999887777	Alicia	40.0

- Herhangi bir projede görev almamış personel için «proj_count» değerini null olarak ayarlar.

- sum(w.hours) yerine şunu kullanın:

```
UPDATE employee e
SET sum_proj_hours = (SELECT CASE WHEN SUM(w.hours)
IS NOT NULL THEN SUM(w.hours)ELSE 0 END
FROM works_on w WHERE w.essn = e.ssn);
```

SQL – JOIN İlişkileri

- JOIN (birleştirme) işlemleri iki ilişkiyi alır ve sonuç olarak başka bir ilişki döndürür.
- Bir JOIN işlemi, Kartezyen çarpıma dayanır ve iki ilişkideki tupleların (kayıtların) belirli bir koşula göre eşleşmesini gerektirir. Ayrıca, JOIN sonucunda hangi özniteliklerin (alanların) yer alacağını da belirtir.
- JOIN işlemleri genellikle FROM bölümünde alt sorgu ifadeleri olarak kullanılır.
- Üç tür JOIN vardır:
  - Natural JOIN
  - Inner JOIN
  - Outer JOIN

SQL – JOIN İlişkileri – NATURAL JOIN

- **Doğal birleştirme (Natural join),** iki tabloda **ortak olan tüm sütun adlarında** değerleri eşleşen kayıtları eşleştirir ve sonuçta ortak sütunların sadece **tek kopyasını** tutar.

- Natural join'in doğrudan kullanılabilmesi için, birleştireceğiniz tablolardaki ortak anahtar sütunlarının **aynı isimde** olması gerekir.

```
select name, course_id
from  students, takes
where student.ID = takes.ID;
```

- Aynı sorgu SQL'de "Natural JOIN" yapısıyla

```
select name, course_id
from student natural join takes;
```

SQL – JOIN İlişkileri – NATURAL JOIN Tehlikeleri

- NATURAL JOIN, ortak isimli tüm sütunları otomatik eşitlediği için, alakasız ama adı aynı olan sütunlar (ör. dept_name, dno, dnumber vb.) yanlışlıkla eşitlenip sonucu daraltabilir.

- Natural JOIN kullanacaksan: ortak sütun adlarının gerçekten aynı kavramı temsil ettiğinden emin ol.

- Emin değilsen ya da isim çakışması riski varsa: JOIN ... ON ... ile açık eşitleme yap

SQL – JOIN İlişkileri – OUTER JOIN

- Kayıpsızlığı hedefleyen birleştirme yapısıdır.

- Önce normal bir JOIN yapılır; sonra eşleşmeyen taraf(lar)dan gelen satırlar da sonuca NULL değerlerle eklenir.

- Üç biçimi vardır:
  - left outer join,
  - right outer join,
  - full outer join.

SQL – JOIN İlişkileri – OUTER JOIN

Course

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

Prereq

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- ders bilgisi eksik CS-437
- ön koşul bilgisi eksik CS-315

SQL – JOIN İlişkileri – LEFT OUTER JOIN

```
SELECT c.course_id, c.title, c.dept_name, c. credits, p.prereq_id
FROM COURSE AS c LEFT OUTER JOIN PREREQ AS p
ON c.course_id=p.course_id
```

	course_id character va	title character varyin	dept_name character var	credits integer	prereq_id character v
1	BIO-301	Genetics	Biology	4	BIO-101
2	CS-190	Game Design	Comp. Sci.	4	CS-101
3	CS-315	Robotics	Comp. Sci.	3	[null]

SQL – JOIN İlişkileri – RIGHT OUTER JOIN

```
SELECT p.course_id, p.prereq_id, c.title, c.dept_name, c. credits
FROM COURSE AS c RIGHT OUTER JOIN PREREQ AS p
ON c.course_id=p.course_id
```

	course_id character v	prereq_id character v	title character varyin	dept_name character var	credits integer
1	BIO-301	BIO-101	Genetics	Biology	4
2	CS-190	CS-101	Game Design	Comp. Sci.	4
3	CS-347	CS-101	[null]	[null]	[null]

SQL – JOIN İlişkileri – FULL OUTER JOIN

```
SELECT c.course_id, c.title, c.dept_name, c. credits, p.prereq_id
FROM COURSE AS c FULL OUTER JOIN PREREQ AS p
ON c.course_id=p.course_id
```

	course_id character va	title character varying	dept_name character vary	credits integer	prereq_id character v
1	BIO-301	Genetics	Biology	4	BIO-101
2	CS-190	Game Design	Comp. Sci.	4	CS-101
3	[null]	[null]	[null]	[null]	CS-101
4	CS-315	Robotics	Comp. Sci.	3	[null]

SQL – JOIN İlişkileri – INNER JOIN

```
SELECT c.course_id, c.title, c.dept_name, c. credits, p.prereq_id
FROM COURSE AS c INNER JOIN PREREQ AS p
ON c.course_id=p.course_id
```

	course_id character v	title character varyir	dept_name character var	credits integer	prereq_id character
1	BIO-301	Genetics	Biology	4	BIO-101
2	CS-190	Game Design	Comp. Sci.	4	CS-101

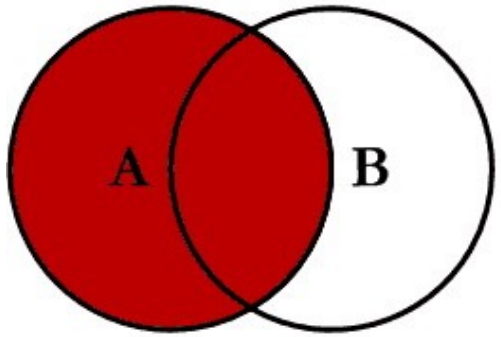
SQL – JOIN İlişkileri

- **JOIN Operations** - iki ilişkiyi alır ve sonuç olarak başka bir ilişki döndürür.
- Bu ek işlemler genellikle **FROM** ifadesinde alt sorgu ifadeleri olarak kullanılır.
- **JOIN Condition** – iki ilişkideki hangi ikililerin eşleştiğini tanımlar.
- **JOIN Türü** – her ilişkideki ikililerin ve diğer ilişkideki hiçbir ikiliyle eşleşmeyen ikililerin (birleştirme koşuluna bağlı olarak) nasıl ele alınacağını tanımlar.

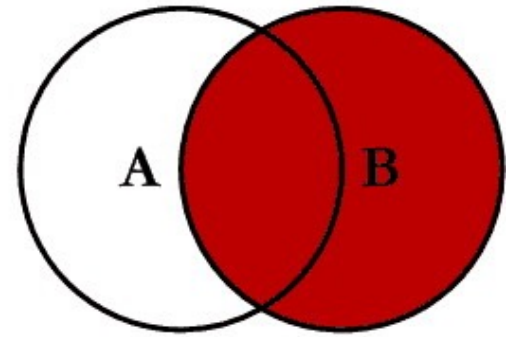
Join types
inner join
left outer join
right outer join
full outer join

Join conditions
natural
on <predicate>
using $(A_1, A_2, \ldots, A_n)$

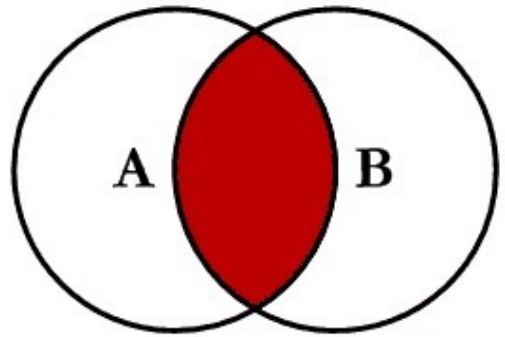
SQL JOINS



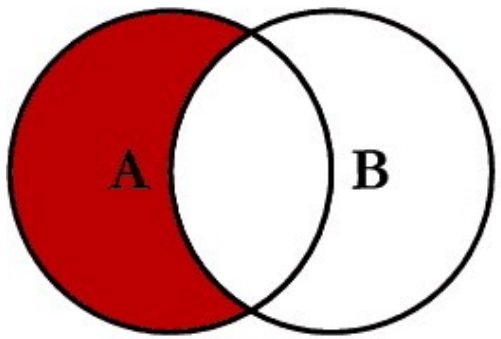

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



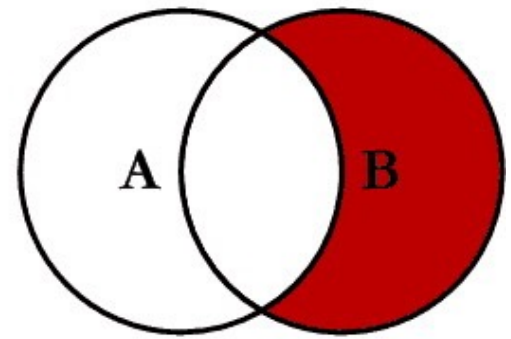

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



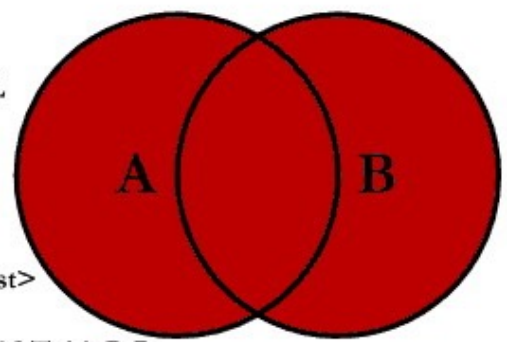

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



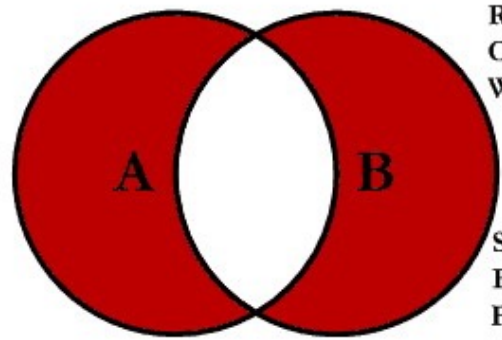

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

SQL – VIEW

- View, bazı verileri belirli kullanıcılardan gizlemek veya veriyi farklı bir bakışla sunmak için kullanılan, fiziksel olarak saklanmayan sanaldır (virtual relation).

- Verileri kendisi depolamayan, bunun yerine bir veya daha fazla temel tablodan elde edilen verilerin dinamik bir temsilini sağlayan sanal bir tablo oluşturmak için kullanılır.

- Kavramsal modele ait olmayan ancak bir kullanıcıya "sanal ilişki" olarak görünen herhangi bir ilişkiye **VIEW** denir.

Neden kullanılır?

- Hassas sütunları gizlemek (ör. salary).
- Karmaşık sorguları basitleştirip yeniden kullanılabilir hâle getirmek.
- Farklı kullanıcı gruplarına farklı veri görünümleri sunmak.

SQL – VIEW

Türler

- **Regular View:** Her sorguda alttaki tablolardan sonuç üretilir. Veri kopyalanmaz.
- **Materialized View**: Sorgu sonucu fiziksel olarak saklanır. Okuma hızlıdır; fakat veriler değişince yenilenmesi gerekir (refresh).
- **Parameterized View:** Bazı sistemlerde (veya şablon/prosedür mantığıyla) değişken/parametre alan görünüm kurguları. Aynı tanımı farklı değerlerle kullanmayı sağlar.
- **Temporary View:** Bir sorgu içinde geçici adlandırılmış alt-sorgular tanımlamak için kullanılır; sadece o sorgunun ömrü boyunca yaşar.

SQL – VIEW

- Bir görünüm, **CREATE VIEW** ifadesi kullanılarak tanımlanır:

```
CREATE VIEW v AS <query_expression>;
```

- v: görünümün (view) adı

- <query_expression>: geçerli herhangi bir SQL sorgusu (SELECT …)

- View tanımlandıktan sonra, sanal ilişki (virtual relation) olarak bu ilişki normal bir tablo gibi kullanılabilir:

```
SELECT * FROM v WHERE ...;
```

SQL – VIEW

- **Önemli nokta:** View ≠ yeni tablo oluşturmak
- Bir view tanımlamak, sorguyu çalıştırıp sonucunu kalıcı bir tabloya yazmak değildir.
- View, ifadenin (sorgunun) tanımını saklar.
- v üzerinden sorgu yazıldığında, veritabanı bu view'in tanımındaki sorguyu araya koyup çalıştırır (mantıksal olarak "yerine koyma/substitution").
- Not: Eğer veri gerçekten diske yazılıp hız için saklanmak istenirse bunun adı materialized view'dır (farklı bir kavram). Normal view'da veri kopyalanmaz.

SQL – VIEW – Örnek

```
CREATE VIEW expensive_products AS
SELECT product_id, product_name, price
FROM products WHERE price > 100;
```

```
SELECT * FROM expensive_products;
```

product_id	product_name	price
1	Laptop	800
2	Smartphone	600
3	Tablet	120
5	Monitor	200

product_id	product_name	price
1	Laptop	800
2	smartphone	600
	Tablet	120
	Headphone	80
	Monitor	200

SQL – VIEW – Örnek

- Çalışanların ad-soyad + departman bilgisini göster, maaş/Bdate yok.

```
CREATE VIEW emp_basic AS
SELECT ssn, fname, lname, dno FROM employee;


SELECT * FROM emp_basic
```

SQL – VIEW – View Kullanılarak Tanımlanan View

- Bir view başka bir view'ı kullanabilir.
- v1 tanımında v2 geçiyorsa: v1 doğrudan v2'ye bağlıdır.
- Arada başka view'lar bile olsa, zincir varsa: v1 bağlıdır v2'ye.
- Eğer bir view, doğrudan ya da dolaylı olarak kendi kendisine geri bağlanıyorsa, bu view "recursive"tir.

SQL – VIEW – View Kullanılarak Tanımlanan View

```
CREATE VIEW emp_basic AS
SELECT ssn, fname, lname, dno FROM employee;

CREATE OR REPLACE VIEW emp_res_dep AS
SELECT b.ssn, b. fname, b.lname, b.dno, d.dname FROM
emp_basic b JOIN department d ON d.dnumber = b.dno
WHERE d.dname = 'Research'
```

```
SELECT * FROM emp_res_dep
```

	ssn character (9)	fname character v	lname character v	dno integer	dname character va
1	333445555	Franklin	Wong	5	Research
2	123456789	John	Smith	5	Research
3	666884444	Ramesh	Narayan	5	Research
4	453453453	Joyce	English	5	Research

SQL – VIEW – Genişleme (Expansion)

- Bir view kullanıyorsan, veritabanı o view'ın adını alır ve onu tanımlayan sorgu ile değiştirir.
- Ortada sadece gerçek tablolar kalana kadar bu tekrarlanır.

```
CREATE VIEW emp_res_dep AS
SELECT b.ssn, b.fname, b.lname, b.dno, d.dname
FROM (SELECT ssn, fname, lname, dno FROM employee) AS b
JOIN department d ON d.dnumber = b.dno
WHERE d.dname = 'Research';
```

SQL – VIEW

Materialized Views

- Bazı veritabanı sistemleri, görünüm ilişkilerinin fiziksel olarak depolanmasına izin verir.

- **Materialized View** tanımlandığında sorgunun sonucu gerçekten bir tablo gibi diskte tutulur.

  - Sorguda kullanılan ilişkiler güncellenirse, Materialized View sonucu güncelliğini yitirir.
  - Temel ilişkiler güncellendiğinde View'i güncelleyerek görünümü korumanız gerekir.

SQL – VIEW

YTÜ YILDIZ TEKNİK ÜNİVERSİTESİ

Materialized Views

```
CREATE MATERIALIZED VIEW mv_project_total_hours AS
SELECT p.pnumber AS project_no, p.pname   AS project_name,
SUM(w.hours) AS total_hours FROM project p
LEFT JOIN works_on w ON w.pno = p.pnumber
GROUP BY p.pnumber, p.pname;
```

- Sorguda kullanılan ilişkiler güncellenirse, Materialized View sonucu güncelliğini yitirir.
- Temel ilişkiler güncellendiğinde View'i güncelleyerek görünümü korumanız gerekir.

```
REFRESH MATERIALIZED VIEW mv_project_total_hours;
```

SQL – VIEW – Güncelleme

- Bir view'a INSERT yaptığında aslında fiziksel olarak bir yere veri yazmak istiyoruz.
- View'un kendi fiziksel depolaması olmadığından (materialized view olmadığı sürece), gerçek tabloya yazılmak istenir.
- Eğer view, gerçek tablonun tüm zorunlu kolonlarını göstermiyorsa (ör. maaş/salary yok), o zaman veritabanı bunu otomatik çözmek zorunda kalır — ve bu her zaman mümkün veya güvenli değildir.
- Bu yüzden bazı view'lar güncellenebilir (updatable view) sayılır, bazıları sayılmaz.
  - Tek tabloyu direkt yansıtıyorsa ve kritik kolonları atmadıysa → genelde updatable.
  - JOIN içeriyorsa, agregasyon varsa, ya da zorunlu kolonları gizliyorsa → genelde updatable değildir / DB reddeder.

SQL – VIEW – Güncelleme

```sql
CREATE VIEW expensive_products AS
SELECT product_id, product_name, price
FROM products WHERE price > 100;
```

- Yeni bir ürün var, adı Webcam HD, fiyatı 80.
  Bunu pahalı ürünlere ekle."

  `(6, 'Webcam HD', 80)`

- “ekledim ama neden listede yok?”
- "Bu view yalnızca price > 100 ürünler içindir. Sen 80’lik ürün eklemeye çalışıyorsun. Bu mantıksal olarak tutarsız"

```sql
WITH CHECK OPTION
```

product_id	product_name	price
1	Laptop	800
2	smartphone	600
3	Tablet	120
4	Headphone	80
5	Monitor	200

SQL – CTE(WITH) vs VIEW vs MATERALIZED VIEW

Kriter	CTE (WITH)	VIEW	MATERIALIZED VIEW
Tanım	Tek sorgu içinde adlandırılmış alt sorgu	Şemada kalıcı sorgu tanımı	Sonucu **fiziksel olarak** saklanan view
Ömür	Sadece o **statement** boyunca	Kalıcı nesne	Kalıcı nesne + **içeriği** saklı
Yeniden kullanım	Aynı statement içinde	Her yerde, her sorguda	Her yerde; okuması hızlı
Güncellik	Her çalıştırmada canlı veri	Her çağrıda canlı veri	**REFRESH** edilene kadar eski kalabilir
Depolama	Yok	Yalnızca tanım	**Var** (sonuçlar diskte)
Performans	Okunabilirlik; PG12+'da inline olur	Her çağrıda yeniden hesap	Okuma çok hızlı; maliyet **REFRESH** anında
Güncellenebilirlik	—	Basit tek-tablo view'lar genelde mümkün; `WITH CHECK OPTION` ile kural zorlanır	Doğrudan güncellenmez; base tablolar güncellenir, sonra **REFRESH**

SQL – TRANSACTIONS

- Transaction, bir ya da daha fazla SQL komutunu tek bir iş birimi (unit of work) olarak çalıştırmaktır.
- Birkaç INSERT, UPDATE, DELETE, hatta bazı SELECT'ler yapıldığında, bunların hepsini "ya hep beraber başarılı olsun" ya da "hiçbiri olmamış gibi geri al" diyebiliriz.
- Transaction ne zaman başlar?

```
BEGIN;   -- veya START TRANSACTION;
-- buradan itibaren yapılanlar aynı transaction
```

- Transaction nasıl biter?
  - "Tamam, her şeyi kaydet."

    COMMIT;
  - Bu transaction içindeki tüm değişiklikler iptal edilir, sanki hiç yapmamışsın gibi olur.

    ROLLBACK;
- Atomik ⇒ ya hep ya hiç.
- Isolation ⇒ aynı anda çalışan işlemler birbirini bozamaz.

SQL – DOMAIN

- CREATE DOMAIN ile kendi "mini veri tipi"mizi tanımlayabiliriz.
- Bu domain NOT NULL, CHECK, vs. gibi kurallar içerebilir.
- Bu domain tablo kolon tipi gibi kullanılabilir.
- Böylece tekrarlı kurallar merkezi bir yerde toplanır ve veri bütünlüğü standart hale gelir.

```
CREATE DOMAIN degree_level VARCHAR(10)
    CONSTRAINT degree_level_test
    CHECK (VALUE IN ('Bachelors', 'Masters', 'Doctorate'))

CREATE TABLE student (
    sid          INTEGER PRIMARY KEY,
    full_name    person_name,
    degree_type  degree_level   --'Bachelors'/'Masters'/'Doctorate' olabilir);
```

SQL – INDEXES

- Çoğu sorgu aslında tablonun tamamını istemez.
- Genelde “şu ID’li müşteri”, “şu barkodlu ürün”, “şu TC numarası”, “şu sipariş numarası” gibi küçük bir alt kümeyi isteriz.
- Gerçek hayattan düşün:
  - 1 milyar satır ürün hareketi var.
  - Sen aslında sadece barcode = '8690323' olan 1 satırı istiyorsun.
- Eğer indeks yoksa veritabanı bazen tüm tabloyu satır satır taramak zorunda kalır (full table scan).
- Bu çok yavaş ve çok pahalıdır (I/O, CPU).

SQL – INDEXES

- SQL'deki indeksler, veri alma işlemlerinin hızını artıran özel veri yapılarıdır.
- Veritabanı için hızlı bir arama tablosu gibi çalışırlar; tüm tabloyu satır satır taramak yerine, veritabanı indeksi kullanarak gerekli satırları doğrudan bulabilir.
- Veritabanlarında performans ve verimliliğin artırılmasında önemli bir rol oynarlar çünkü:
  - Sorguları hızlandırın (SELECT, JOIN, WHERE, ORDER BY).
  - Disk G/Ç'yi azaltın ve büyük tablolarda verimliliği artırın.
  - Benzersiz indekslerle veri bütünlüğünü sağlayın.
  - Çok fazla indeks INSERT, UPDATE ve DELETE işlemlerini yavaşlatabileceğinden akıllıca kullanılmalıdır.

SQL – INDEXES – Gerçek Senaryo

💊 Anadolu Ajansı'na göre 2016'da Türkiye'de 2 milyar kutu ilaç satılmış.

➡️ Bu da 2 milyar tekil barkod = 2 milyar satır veri anlamına geliyor.

⏱️ Saniyede okutulan barkod sayısı: 65

👤 1 kullanıcı ile test: Index olmayan tabloda 1 barkod okutma süresi ≈ 10 saniye

💡 Okuma miktarı: 32 GB (yaklaşık 4 milyon page)

👥 5 kullanıcı ile test: 60 saniyede cevap, CPU %100

🔥 Zoom dondu, makine restart

Full scan = tabloyu baştan sona gez = ölüm 😅

⚙️ Sonra tabloya index ekledik...

👨‍💻 50 kullanıcı ile test sonucu: Sorgu süresi 12 ms ⚡

🔍 Okuma miktarı: Sadece 8 page (her biri 8 KB)

🧊 CPU kullanımı: sadece %10 (ve çoğu simülasyon ekran yenilemesinden kaynaklıydı!)

🏁 Sonuç: 📈 1 milyar satırlık tabloda index ile 500.000 kat (yarım milyon kat!) performans artışı

Ömer Çolakoğlu'ndan alıntıdır.

SQL – INDEXES

- CREATE INDEX index_adi ON tablo_adi (kolon_adi);

```
CREATE INDEX idx_employee_ssn ON EMPLOYEE (Ssn);
```

Ssn	satır_adresi
123456789	(page 17, offset 3)
333445555	(page 99, offset 12)
987654321	(page 4, offset 1)
...	...

SQL – INDEXES – Dikkat

- Tabloya yeni satır eklendiğinde, sadece tabloya yazılmaz. İndeksin de güncellenmesi gerekir.
  - Yeni SSN geldi → indekse de bu (değer, adres) çifti eklenmeli.
  - Güncelleme olduysa → indekste de güncellenmeli.
  - Silindiyse → indeksten de kaldırılmalı.
- Bu yüzden:
  - Sık okunan ama nadiren değişen kolonlarda indeks şahane.
  - Sürekli güncellenen "çok değişken" kolonlarda gereksiz indeks sistemi yavaşlatabilir.
- İndeksler de depolanır.
  - Yani ek disk/memory maliyeti var.
  - Tablo ne kadar büyürse indeks de büyüyor.
- İyi indeks = çok sık aranan (ve genelde dar filtrelenen) kolonda kurulan indeks. "Her şeye indeks" kötü fikirdir.

SQL – Yetkilendirme

- Veritabanının belirli bölümlerinde bir kullanıcıya çeşitli yetkilendirme biçimleri atayabiliriz.

Veriyi Kullanma

- **READ (Okuma) -** Verilerin okunmasına izin verir, ancak değiştirilmesine izin vermez.
- **INSERT (Ekle) -** Yeni verilerin eklenmesine izin verir, ancak mevcut verilerin değiştirilmesine izin vermez.
- **UPDATE (Güncelle) -** Verilerin değiştirilmesine izin verir, ancak silinmesine izin vermez.
- **DELETE (Sil) -** Verilerin silinmesine izin verir.

- Bu yetkilendirme türlerinin her birine ayrıcalık (privilege) denir.
- Kullanıcıya, ilişki (table) veya görünüm (view) gibi veritabanının belirli bölümlerinde bu ayrıcalık türlerinin tümünü, hiçbirini veya bir kombinasyonunu yetkilendirebiliriz.

SQL – Yetkilendirme

Veritabanı şemasını değiştirme yetkilendirme biçimleri

- **INDEX:** indeks oluşturma/silme izni
- **RESOURCES:** yeni tablo (relation) oluşturma izni
- **ALTERATION:** var olan tabloya sütun ekleme/silme/değiştirme
- **DROP:** tabloyu (relation) silme

SQL – Yetkilendirme

GRANT <privilege list> **ON** <relation or view > **TO** <user list>

- Ayrıcalığı veren kişi, belirtilen öğe üzerinde ayrıcalığa zaten sahip olmalıdır (veya veritabanı yöneticisi olmalıdır).

- Bir görünüm üzerinde ayrıcalık vermek, altta yatan ilişkiler üzerinde herhangi bir ayrıcalık verildiği anlamına gelmez.

SQL – Yetkilendirme

```
GRANT SELECT ON EMPLOYEE TO analyst_user;
      INSERT
      UPDATE
      DELETE
  ALL PRIVILEGES

REVOKE SELECT ON EMPLOYEE TO analyst_user;
       INSERT
       UPDATE
       DELETE
   ALL PRIVILEGES
```

SQL – Roller

- Rol, çeşitli kullanıcıları veritabanında nelere erişebilecekleri/güncelleyebilecekleri açısından ayırt etmenin bir yoludur.

CREATE ROLE <name>

CREATE ROLE hr_role;

- Bir rol oluşturulduktan sonra, şu komutu kullanarak role "kullanıcılar" atayabiliriz:

GRANT <role> **TO** <users>

GRANT hr_role **TO** fatma_hr, mert_hr;

SQL – Roller

- **Yönetilebilirlik:** 100 kişi var diyelim. Tek tek hepsine yetki vermek yorucu
- **Tutarlılık / güvenlik:** Örn. İK'nin ne görebileceği standart olur.

YILDIZ TEKNİK ÜNİVERSİTESİ

Yasemin Topuz

Yıldız Teknik Üniversitesi

ytopuz@yildiz.edu.tr