ONDOKUZMAYIS ÜNİVERSİTESİ
Bilgisayar Mühendisliği Bölümü

Veri Tabanı Lab. Dersi Deney Föyü-2

Konu: Constraints(Kısıtlar), View (Görünüm) ve Indeks yapılarının araştırılması

Teslim Süresi: 1 Hafta

1. Kısıtlar: Veritabanı üzerinde iş kurallarını zorunlu kılmak ve tablolar arasında ilişki kurarak veri bütünlüğünü sağlamak amacıyla oluşturulur. **(** Postgresql kısıtlamaları; check, not null, unique, primary keys) Diğer VTYS sistemlerinde de benzer kısıtlar vardır.

1) SQL Primary key:

Tablolarda arama yapmamızı kolaylaştırır.Tekrarlanamaz.Aynı primary keyi tekrar kullanmamızı engeller.

Örnek:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

2) SQL Foreign key

Aynı tabloda iki sütun arasında bir ilişki veya farklı tablolar arasında ilişki kurar. Bir veya daha fazla sütun Yabancı anahtar olarak tanımlanabilir.

Örnek :

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT pd_id_fk FOREIGN KEY(product_id) REFERENCES product(product_id)
);
```

3) SQL Not Null Constraint

Girilen değerin boş olmamasını sağlar. Kullanıcı boş bırakamaz..

```
CREATE TABLE employee
( id number(5),
name char(20) CONSTRAINT nm_nn NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

4)SQL Unique Key

Bu sınırlama her satır veya sütunun ayrı bir değere sahip olmasını sağlar. Sütun (lar) değerleri çoğaltılamaz . Ancak boş değer olabilir.(primary keyden farkı budur)

Örneğin, sorgu gibi olacak bir anahtar ile çalışan bir tablo oluşturmak için:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

5) SQL Check Constraint

Bu kısıtlama bir kolon üzerinde bir iş kuralı tanımlar. Tüm satırlar bu kural yerine getirmek zorundadırlar. Kısıtı tek bir sütun veya sütun grubu için uygulanabilir.

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10)
);
```

2.View Kullanımı:

View' lar hazırlanmış sql cümleleri olarak bilinir. Sadece çağırıldıklarında veri kümesini üretirler. Veritabanı programcılığında önemlidir. Sql komutlarından view oluşturma ve faydaları:

2.1. Veri güvenliği

Veri tabanı içinde bulunan tablolardaki bazı sütunlarda bulunan bilgilerin, herkes tarafından görülmesi istenmeyebilir.

Örneğin, personelin maaşlarının herkes tarafından listelenebilir olması mahsurlu olabilir. Bu durumda, Personel adlı temel (base) tablodan, persview adlı bir view oluşturulabilir.

```
CREATE VIEW persview
AS SELECT sicil,sos_g_no,ad,soyad,dog_tar,adres,cinsiyet,bol_no,yon_s_g_n
FROM Personel;
```

persview adlı view, herkesin kullanımına açık, Personel adlı temel (base) tablo ise, yetkili kişiler dışındakilere, erişilemez hale getirilirse, maaşların herkes tarafından erişilebilir bilgi olması önlenmiş olur.

Bir view'den bilgi listelenmesi temel tablodan bilgi listelenmesinden farklı değildir.

```
SELECT *
FROM persview;
```

persview'den maaşlar hariç, tüm personel bilgileri listelenecektir.

Bir temel tablodan bir view oluşturulurken, temel tablodaki aynı sütun (alan) isimlerini kullanmak zorunda değildir. Örneğin, Parça adlı ve par_no, par_ad, pr_no, fiyat ve ağırlık adlı sütun (alan) isimlerini içeren tablo kullanılarak oluşturulan parview içinde, par_no yerine parc_no, fiyat yerine fiy ve ağırlık yerine ağır isimleri kullanılmıştır:

```
CREATE VIEW
Parview(parc_no,fiy,ağır)
AS SELECT par_no,fiyat,ağırlık
FROM Parça;
```

2.2. Sorgulamanın daha basit hale gelmesi

Karmaşık sorgulamalarda, bazı SELECT komutlarının sonuçları diğer SELECT komutlarınca kullanıldığında, sorgulamanın düzenlenmesinde yanlışlıklar yapma olasılığı artar. Karmaşık sorgulamalar, VIEW özelliği kullanılarak daha basit hale getirilebilir. Burada temel fikir şudur: Madem ki bir view, bir sorgulama sonucu elde edilen bilgiyi (tabloyu) isimlendirerek elde edilen bir virtüel tablodur; o halde karmaşık SELECT komutu içinde, sonucu kullanılacak başka bir SELECT komutu kullanmak yerine, bu sonucu bir view olarak isimlendirerek, view adını kullanmak. Bazı durumlarda ise, işletmenin veri tabanı uygulamasında çok sık olarak sorulan karmaşık soruları bir view yapısı içinde saklayarak, daha sonra aynı tip sorgulamalar için bu view yapısını kullanarak daha basit ifadeler kullanmakta olasıdır.

Örnek: Satış bölümünde çalışan personelin herhangi birinden daha düşük maaş alan ve mühendislik bölümünde çalışan kişileri listeleyiniz.

```
SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
FROM Personel
WHERE bol_no=2) AND
bol_no=1;
```

(Satış bölümü kodu 2 ve mühendislik bölümü kodu ise 1 kabul ediliyor.)
Şimdi bu sorunun cevabı olan tablo bir view olarak saklanırsa:

```
CREATE VIEW S1view
AS SELECT *
FROM Personel
WHERE maas<ANY(SELECT maas
FROM Personel
WHERE bol_no=2) AND
bol_no=1;
```

Bundan sonra aynı tip sorgulama için sadece

```
SELECT *
FROM S1view;
```

yazmak yeterli olcaktır.

2.3.Sadece view kullanılarak gerçekleşebilen sorgulamalar

Bir tablodan elde edilecek bilgiler için, iki kademeli işlem gerektiren sorgulamalarda, ilk adımda bir view oluşturup ikinci adımda esas sorgulamayı bu view yardımı ile gerçekleştirmek, çoğu kez kaçınılmaz bir durumdur.

Aşağıdaki soru ve bunun çözümü olan SQL ifadeleri bu konuda bir fikir verecektir:

Örnek: Her bölümde, o bölümdeki ortalama maaştan daha yüksek maaş alanları listeleyiniz. Bu sorunun cevaplandırılması için önce her bölümdeki ortalama maaşların bulunması gereklidir.

```
CREATE VIEW BOL_OR_VIEW(bol_no,ort,maas)
AS SELECT bol_no,AVG(maas)
FROM Personel
GROUP BY bol_no;
```

Daha sonra, yaratılan BOL_OR_VIEW yardımı ile (bu view, bölüm no'ları ve bölüm ortalama maaşlarını saklamaktadır) sorulan sorunun cevabı elde edilebilir:

```
SELECT *
FROM Personel
WHERE bol_no=BOL_OR_VIEW.bol_no
. AND.maas>ort_maas;
```

(Bu sorunun cevabını, şu ana kadar anlatılan diğer bilgilerle bulmaya çalışınız.)

Veri bütünlüğünün sağlanması

View oluşturma esnasında CHECK sözcüğünün kullanılması ile, o view'i oluştururken sağlanması gereken koşulların, daha sonra view içine veri ekleme ya da değişiklik işlemlerinde de ihmal edilmesi engellenmiş olur.

Örneğin aşağıdaki gibi bir VIEW oluşturulsun:

```
CREATE VIEW UST_PER_VIEW
AS SELECT FROM Personel
WHERE maas>25000000,
WITH CHECK OPTION;
```

Burada,maaşı 25000000'un üstünde olan personelden oluşan bir UST_PEVIEW adlı view oluşturulmuştur. Daha sonra bu view içine;

```
INSERT INTO UST_PER_VIEW
VALUES(27521,&#8217;27865427&#8217;,&#8217;Ayşe&#8217;
&#8216;Okan&#8217;,{01/05/62},&#8217;Cumh. Cad. 46-Taksim&#8217;,
.F.,13000000,1,&#8217;27651112&#8217;);
```

komutu ile maaşı 13000000 olan bir personel eklenmek istendiği zaman şu hata mesajı alınacaktır.

```
Error: NOT enough non-NULL VALUES
```

Eğer CHECK opsiyonu kullanılmasaydı hata mesajı alınmadan bu veri view içine yüklenecekti.

View'ler Üzerinde Ekleme, Silme, Değişiklik İşlemleri

VIEW'ler üzerindeki ekleme, silme ve değişiklik işlemleri esas itibarı ile tablolar üzerinde yapılan benzer işlemlerden çok farklı değildir. Fakat VIEW'ler üzerinde bu tip işlemlerin gerçekleştirilmesinde bazı kısıtlamalarda mevcuttur. Aşağıdaki hususların belirtilmesinde fayda vardır: Bir view'in güncellenebilir nitelikte olması için, bir birleştirme (join) işlemi sonucunda üretilmemiş olması gerekir. Başka bir deyişle, CREATE VIEW komutunda FROM sözcüğünü izleyen kısımda sadece tablo adı bulunmalıdır.

View içindeki hiçbir kolon bileşik (aggregate) fonksiyonlarca üretilmiş olmamalıdır. (MAX, SUM v.b) View'in üretildiği SELECT komutunda DISTINCT, GROUP BY ya da HAVING sözcüklerini içeren parçaların yerine getirilmiş olmamalıdır.

Bu koşulları sağlamayan view'ler sadece okunabilir (Readonly) özellikteki view'lerdir ve üzerlerinde herhangi bir değişiklik yapılamaz.

View içine satır ekleme

Daha önceden oluşturulmuş Px adlı view, ad, soyad ve maas alanlarını içermiş olsun. Bu view, güncellenebilir nitelikte ise, aşağıdaki INSERT komutu ile, aynen tablolarda olduğu gibi kendisine bir satır eklemek mümkün olacaktır:

```
INSERT INTO Px
VALUES (&#8216;Ali&#8217;,&#8217;Çakır&#8217;,12000000);
```

Daha önceden, VIEW oluşturulurken, CHECK OPTION alternatifi kullanılmışsa, bu takdirde, ekleme esnasında, VIEW'i oluşturan koşul ihlal ediliyorsa, sistem eklemeye müsaade etmeyecek ve hata mesajı verecektir.

Örnek: Personel adlı tablodan, maaşı 20000000 TL'yi aşan personeli alarak, UST_PER_VIEW adlı bir view oluşturunuz.

```
CREATE VIEW UST_PER_VIEW
AS SELECT FROM Personel
WHERE maas>20000000
WITH CHECK OPTION;
```

Şimdi UST_PER_VIEW içine

```
INSERT INTO UST_PER_VIEW
VALUES (37261,&#8217;34268152&#8217;,&#8217;Beril&#8217;,
&#8217;Caner&#8217;,{01/04/64},&#8217;Kadıköy&#8217;,.F.,
14000000,2,&#8217;37624158&#8217;);
```

komutu ile maaşı 14000000 olan bir kişi eklenmek istendiğinde, bu komut kabul edilmeyecek ve aşağıdaki hata mesajı alınacaktır:

```
Error:NOT enough non-NULL VALUES
```

Eğer CHECK opsiyonu kullanılmasaydı, hata mesajı verilmeksizin bu satır, view içine eklenecektir.

View içinden satır silme

Güncellenebilir bir view içinde satır silme işlemi, tablolardan satır silme işlemi ile aynı şekilde gerçekleştirilir. Örneğin 6.9.2'de oluşturulan UST_PER_VIEW içinden, maaşı 2500000'den az olan kişiler silinmek istenirse;

```
DELETE FROM UST_PER_VIEW
WHERE maas<2500000;
```

komutunu kullanmak yeterli olacaktır.

View satırları üzerinde güncelleme işlemi

Güncellenebilir view'lerde güncelleme işlemi tablolardakinin aynıdır. Örneğin UST_PER_VIEW adlı view'de sicili 27251 olan kişinin maaşının 37000000 olarak değiştirmek için;

```
UPDATE UST_PER_VIEW
SET maas=37000000
WHERE sicil=27251;
```

komutunu kullanmak uygun olacaktır.

Bir view'i silmek

Tabloların silinmesine benzer şekilde, sistemde oluşturulan bir view, DROP VIEW komutu ile silinebilir.

```
DROP VIEW UST_PER_VIEW;
```

Bir view'in silinmesi ile, o view'e bağlı olarak oluşturulmuş diğer bütün view'ler ve bu view ile ilişkili önceliklerin de tümü silinmiş olacaktır.

3. Indexler:

Index Oluşturmanın Amacı:

Bir index, veri tabanı ortamında bir tablo ya da bir view gibi bir nesnedir ve ilişkili olarak kullanıldığı tablo ya da view'deki satırların, indexleme alanı (key field (anahtar alan)) olarak kullanılan kolondaki verilere göre sıralanmış biçimde işleme sokulmasını (listeleme ya da arama işlemi) sağlar.
Bir tablo, indexlenmiş ise, bu tablo içinde gerçekleştirilecek bir arama (search) ya da koşullu listeleme (SELECT komutu ile) işlemi çok daha hızlı biçimde gerçekleştirilebilecektir.

Index Yaratma

SQL'de bir tablo ile ilşkili olarak index yaratmak için gerekli komut CREATE INDEX komutudur. Komutun yazılış biçimi aşağıdaki gibidir:

```
CREATE INDEX index adı

ON tabloadı (kolonadı 1,kolonadı 2,....,kolonadı n );
```

Indexleme artan (ascending) ya da azalan (decending) şeklinde olabilir. Artan, alfabetik olarak A'dan Z'ye nümerik olarak küçükten büyüğe şeklindedir. Azalan ise bunun tersidir. Hiçbir özel sözcük kullanılmazsa indexleme artan sayılır ya da alan adının yanında bir boşluktan sonra ASC sözcüğü kullanılırsa bu alana göre artan sıralama yapılacak demektir.
Herhangi bir alanın adının yanında DESC sözcüğünün kullanılması ise indexlemenin azalan olacağını gösterir. Komutun yazılış biçiminden anlaşılacağı gibi, aynı anda, birden çok alana göre indexleme de yapılabilir.

Tek bir alana göre artan sırada indexleme

İşletmede çalışan personeli maaşlarına göre artan sırada listelemek istersek, maas alanına göre bir index oluşturmalıyız.

```
CREATE INDEX pers_maas

ON Personel (maas);

Index created 127 Rows
```

127 satırlık personel tablosu ile ilişkili olarak maas alanına index anahtarı olarak kullanılan pers_maas adlı index oluşturulmuştur. Bu durumda

```
SELECT *

FROM Personel;
```

şeklindeki listeleme komutu sonucunda, personel tablosundaki tüm personel, maaşlarına göre sıralı olarak listelenecektir.

Tek bir alana göre azalan sırada indexleme
İşletmede çalışan personeli maaşlarına göre azalan sırada (yüksek maaştan düşük maaşa doğru) listelemek istersek, maas alanına göre aşağıdaki şekilde oluşturmak gerekir.

```
CREATE INDEX pers_maas

ON Personel (maas DESC);
```

Birden fazla alana göre indexleme
İşletmedeki personelin öncelikle adlarına göre, aynı ad da olanların soyadlarına göre, hem adı hem soyadı aynı olanların maaşlarına göre sıralanmış olarak listelenmesi istenirse aşağıdaki komut kullanılmalıdır:

```
CREATE INDEX p_ad_soy_m

ON Personel (ad,soyad,maas);
```

Bu durumda;

```
SELECT *

FROM Personel;
```

komutu sonucunda, aşağıdaki şekilde sıralanmış tablo görüntülenecektir.

sicil	ad	soyad	maas
11117	Ahmet	Caner	15000000 .....
247	Ahmet	Deniz	27000000 .....
645	Ahmet	Zoran	12000000 .....
3871	Ali	Cenker	26000000 .....
15372	Ali	Cenker	34000000 .....
4246	Ali	Cenker	65000000 .....
16656	Ali	Şener	12000000 .....
7216	Beril	Arkan	18000000 .....
.....	.....	.....	.....

Burada, kolayca görüleceği gibi personel öncelikle adı alanına göre sıralanmış (Ahmet, Ali, Beril) aynı ada sahip olanlar soyadlarına göre sıralanmış (Ahmet ismindeki kişilerin soyadları olan Caner, Deniz, Zoran sıralaması gibi), hem ad hem de soyadları aynı olanların sıralanmasında ise maas alanı dikkate alınmıştır.

Index komutu;

Clientno	Name	city	pincode	state	bal.due
0001	Ivan	Bombay	400054	Maharashtra	15000
0002	Vandana	Madras	780001	Tamilnadu	0
0003	Pramada	Bombay	400057	Maharashtra	5000
0004	Basu	Bombay	400056	Maharashtra	0
0005	Ravi	Delhi	100001		2000
0006	Rukmini	Bombay	400050	Maharashtra	0

CREATE INDEX p_ad_soy_m

ON Personel (ad,soyad,maas DESC);

şeklinde yazılsa idi, tablodaki değerler

sicil	ad	soyad	maas
11117	Ahmet	Caner	15000000 .....
247	Ahmet	Deniz	27000000 .....
645	Ahmet	Zoran	12000000 .....
3871	Ali	Cenker	65000000 .....
15372	Ali	Cenker	34000000 .....
4246	Ali	Cenker	26000000 .....
16656	Ali	Şener	12000000 .....
7216	Beril	Arkan	18000000 .....
.....	.....	.....	.....

şeklinde olacaktır.

VeriTabanından Index Kaldırmak

Drop index *index_name*

Deney2: Yapılması gerekenler.

Aşağıdaki tabloları belirtilen kısıtlara göre oluşturan DDL kodlarını yazınız.

Tablo Adı: client_master

client_master

columnname	datatype	size
client_no	varchar2	6
name	varchar2	20
address1	varchar2	30
address2	varchar2	30
city	varchar2	15
state	varchar2	15
pincode	number	6
bal_due	number	10,2

Bu tabloya ilişkin veriler

Clientno	Name	city	pincode	state	bal.due
0001	Ivan	Bombay	400054	Maharashtra	15000
0002	Vandana	Madras	780001	Tamilnadu	0
0003	Pramada	Bombay	400057	Maharashtra	5000
0004	Basu	Bombay	400056	Maharashtra	0
0005	Ravi	Delhi	100001		2000
0006	Rukmini	Bombay	400050	Maharashtra	0

TabloAdı:

Product_master

Columnname	datatype	size
Product_no	varchar2	
Description	varchar2	
Profit_percent	number	
Unit_measure	varchar2	
Qty_on_hand	number	
Reoder_lvlnumber		
Sell_price	number	
Cost_price number		

Veriler:

Data for Product Master:

Product No.	Desciption	Profit %	Unit Percent	Qty measured	Reorder on hand lvl	Sell price	Cost price
P00001	1.44floppies	5	piece	100	20	525	500
P03453	Monitors	6	piece	10	3	12000	11200
P06734	Mouse	5	piece	20	5	1050	500
P07865	1.22 floppies	5	piece	100	20	525	500
P07868	Keyboards	2	piece	10	3	3150	3050
P07885	CD Drive	2.5	piece	10	3	5250	5100
P07965	540 HDD	4	piece	10	3	8400	8000
P07975	1.44 Drive	5	piece	10	3	1050	1000
P08865	1.22 Drive	5	piece	2	3	1050	1000

Tablo Adı: Salesman_Master

Columnname	**Datatype**	**Size**	**Attributes**
Salesman_no	varchar2	6	Primary key/first letter must start with 's'
Sal_name	varchar2	20	Not null
Address	varchar2		Not null
City	varchar2	20	
State	varchar2	20	
Pincode	Number	6	
Sal_amt	Number	8,2	Not null, cannot be 0
Tgt_to_get	Number	6,2	Not null, cannot be 0
Ytd_sales	Number	6,2	Not null, cannot be 0
Remarks	Varchar2	30	

Tablo adı: Sales_Order

Columnname	Datatype	Size	Attributes
S_order_no	varchar2	6	Primary/first letter must be 0
S_order_date	Date	6	Primary key reference clientno of client_master table
Client_no	Varchar2	25	
Dely_add	Varchar2	6	
Salesman_no	Varchar2	6	Foreign key references salesman_no of salesman_master table
Dely_type	Char	1	Delivery part(p)/full(f),default f
Billed_yn	Char	1	
Dely_date	Date		Can not be lessthan s_order_date
Order_status	Varchar2	10	Values ('in process';'fulfilled';back order';'canceled

Tablo Adı: Sales_Order_Details

Column	Datatype	Size	Attributes
S_order_no	Varchar2	6	Primary key/foreign key references s_order_no of sales_order
Product_no	Varchar2	6	Primary key/foreign key references product_no of product_master
Qty_order	Number	8	
Qty_disp	Number	8	
Product_rate	Number	10,2	

Tablolara girilecek veriler :

Data for sales_man master table

Salesman_ no	Salesman name	Address	City	Pin code	State	Salamt	Tgt_to_get	Ytd Sales	Rem
500001	Kiran	A/14 worli	Bom bay	400002	Mah	3000	100	50	Goo
500002	Manish	65,narim an	Bom bay	400001	Mah	3000	200	100	Goo
500003	Ravi	P-7 Bandra	Bom bay	400032	Mah	3000	200	100	Goo
500004	Ashish	A/5 Juhu	Bom bay	400044	Mah	3500	200	150	Goo

Data for salesorder table:

S_orderno	S_orderdate	Client no	Dely type	Bill yn	Salesman no	Delay date	Orderstatus
019001	12-jan-96	0001	F	N	50001	20-jan-96	Ip
019002	25-jan-96	0002	P	N	50002	27-jan-96	C
016865	18-feb-96	0003	F	Y	500003	20-feb-96	F
019003	03-apr-96	0001	F	Y	500001	07-apr-96	F
046866	20-may-96	0004	P	N	500002	22-may-96	C
010008	24-may-96	0005	F	N	500004	26-may-96	Ip

Data for sales_order_details table:

S_order no	Product no	Qty ordered	Qty disp	Product_rate
019001	P00001	4	4	525
019001	P07965	2	1	8400
019001	P07885	2	1	5250
019002	P00001	10	0	525
046865	P07868	3	3	3150
046865	P07885	10	10	5250
019003	P00001	4	4	1050
019003	P03453	2	2	1050
046866	P06734	1	1	12000
046866	P07965	1	0	8400
010008	P07975	1	0	1050
010008	P00001	10	5	525

Yazılacak Sorgular:

1. client_master tablosunda client_no alanını için indeks oluşturunuz

2. sales_order tablosu üzerinde s_order_no alanı üzerinde indeks oluşturunuz
3. sales_order tablosu üzerinde s_order_no ve product_no alanları üzerinde iki alanı birlikte kullanarak composite indeks oluşturunuz
4. bir üst seçenekte oluşturduğunuz indeksi geri siliniz (drop index)
5. sales_master tablosu üzerinde sal_amt değeri 3500 den büyük olanlar için bir view oluşturunuz
6. client_master tablosu üzerinde client_view isminde bir view oluşturunuz ve sütun isimlerini add1, add2, city, pcode, state olacak şekilde sırasıyla değiştiriniz.
7. client_view isimli view'den şehir ismi 'Bombay' olan client isimlerini getiren bir sorgu yazınız
8. client_view isimli view' ı yok ediniz.
9. sales_order tablosunu kullanarak günlük siparişleri listelemeye yarayan bir view oluşturunuz. Bu view her çalıştırıldığında sistem tarihini alarak o güne ilişkin siparişleri listeler
10. sipariş tarihi 10 gün geçen siparişleri müşteri isimleri ve ürün isimleri olarak liteleyen bir sql sorgusu yazınız

Ek-A

Postgresql'in Özellikleri

Veritabanı büyüklüğü:Sınırsız

PostgreSQL bir veritabanının toplam büyüklüğü için herhangi bir sınır koymaz.PostgreSQL'in veriyi düzenleme yönteminden dolayı çok fazla tablo içeren veritabanlarında başarım gittikçe düşer. PostgreSQL veriyi saklamak için çok sayıda dosya kullanacaktır, ve işletim sistemi tek bir dizinde bu kadar çok dosyayı yönetemezse, başarım düşecektir.

Tablo büyüklüğü: 16Tb-64Tb

PostgreSQL normalde tablo verilerini 8k'lık parçalarda tutar. Bu blokların sayıları 32-bit signed integer kadar sınırlıdır (2 milyarın hemen üstü) ve 16 TeraByte kadar bir tablo büyüklüğü sağlar. Temel blok büyüklüğü PostgreSQL kurulurken 32k ya kadar yükseltilebilir ve bu da teorik olarak 64 TB'lık bir sınır getirir.Bazı işletim sistemleri dosya büyüklükleri için bir sınır koyarlar. Bu nedenden, PostgreSQL tablo verilerini her biri en fazla 1GB büyüklükte olabilecek çoklu dosyalarda tutar. Büyük tablolar için bu bir çok dosya anlamına gelecek ve daha önce de belirttiğim gibi sistem başarımının düşmesine neden olacaktır.Bu büyüklük işletim sisteminden bağımsızdır.

Tablodaki satır sayısı: Sınırsız

PostgreSQL tablodaki satırlarda herhangi bir sınır koymaz. Aslında toplam COUNT fonksiyonu 32-bit tamsayı döndürür, dolayısıyla 2 milyar satırın üzerindeki tablolar için COUNT anlamsız olacaktır.Bu değer sürüm 7.1 ve sonrasında sınırsız olmuştur.

Tablo indexleri: Sınırsız

Tablo üzerinde yaratılabilecek indexlerde PostgreSQL tarafından konan herhangi bir limit yoktur. Ancak unutulmaması gereken, oldukça fazla kolon içeren bir tabloda çok fazla index yaratma çalışırsak başarım gittikçe düşecektir.

Field büyüklüğü: 1Gb

PostgreSQL , sürüm 7.1 ve sonrasında bir tablodaki herhangi bir field için 1 GB'lik bir sınır getirmiştir. Pratikte bu limit sunucunun veriyi işleme ve istemciye transfer etmesi için gerekli hafıza miktarından gelir.

Tablodaki kolon sayısı: 250+

PostgreSQL'de tutulabilecek en fazla kolon sayısı, konfigure edilmiş blok büyüklükleri ve kolon tiplerine bağlıdır. Varsayılan değer olarak blok büyüklüğü olan 8k'da en az 250 kolon saklanabilir, bu sayı eğer fieldlar oldukça basit ise (tamsayı değerleri gibi) 1600 e kadar çıkabilir.Blok büyüklüğünü arttırmak eş zamanlı olarak bu limitleri de arttırır.

Satır büyüklüğü : Sınırsız

Bir satırın büyüklüğü için bir sınır yoktur, ancak kolonlar ve onların büyüklüğü yukarıda anlatıldığı gibi sınırlıdır.Bu sınır, sürüm 7.1'den sonra kaldırılmıştır.

Desteklediği İşletim Sistemleri

Windows , Mac/Os X , Linux , BSD , Unix .