Data management lecture 2

Deleting a database

```
DROP database dbName
```

Active connections to a database will block a deletion attempt!

Creating Tables with Relationships

```
CREATE TABLE account (id serial PRIMARY KEY,
                     username varchar(50) UNIQUE NOT NULL,
                     password varchar(50) NOT NULL,
                     email varchar(255) NOT NULL,
                     created_on timestamp default NOW()
);
```

```
CREATE TABLE blog_entries (id serial PRIMARY KEY,
                          header varchar(255) not null,
                          body TEXT not null,
                          created_by integer NOT NULL REFERENCES account(id)
);
```

Constraint Enforcement

```
INSERT INTO accounts (username, password, email) VALUES ('Tobias', 'Tobias123', 'tob
```

```
INSERT INTO blog_entries (header, body, created_by) VALUES ('My article', 'Hello wor
```

```
INSERT INTO blog_entries (header, body, created_by) VALUES ('My article', 'Hello wor
```

Error: insert or update on table "blog_entires" violates foreign key constraint "blog_entires_created_by_fkey"
Detail: Key (created_by = 6) is not present in the table "account"

Querying Your Result Set - Nested Queries

```
SELECT * FROM account, blog_entries WHERE blog_entires.created_by = account.id;
```

can be written as:

```
select username, email, created_by from (
    select * from account, blog_entries where blog_entires.created_by = accounts.id
) as result_set where created_by = 2
```

Join types

- INNER JOIN→ For each row R1 of T1, the joined table has a row for each row in T2 that satisfies the join condition with R1.
- LEFT OUTER JOIN → First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.
- RIGHT OUTER JOIN → First, an inner join is performed. Then, for each row in T2 that does not satisfy the join condition with any row in T1, a joined row is added with null values in columns of T1. This is the converse of a left join: the result table will always have a row for each row in T2.
- FULL OUTER JOIN → First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Also, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

Source

Inner Join

T1 table:

num	name
1	a
2	b
3	c

T2 table:

num	value

num	value
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 INNER JOIN t2 ON t1.num = t2.num;

SELECT * FROM t1, t2 WHERE t1.num = t2.num;
```

output:

num	name	num	value
1	a	1	xxx
3	c	3	yyy

Left Outer Join

T1 table

num	name
1	a
2	b
3	c

T2 table

num	value
1	xxx
3	yyy
5	zzz

```
    SELECT * FROM t1 LEFT JOIN t2 on t1.num = t2.num;
```

output

num	name	num	value
1	a	1	xxx
2	b	null	null
3	c	3	yyy

Right Outer Join

T1 table

num	name
1	a
2	b
3	c

T2 table

num	value
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;
```

output

num	name	num	value
1	a	1	xxx
3	c	3	yyy
null	null	5	zzz

Full Outer Join

num	name
1	a

num	name
2	b
3	c

T2 table

num	value
1	xxx
3	yyy
5	zzz

```
SELECT * FROM t1 FULL JOIN t2 on t1.num = t2.num;
```

output

num	name	num	value
1	a	1	xxx
2	b	null	null
3	c	3	yyy
null	null	5	zzz

Views – Creating Virtual Tables

```
CREATE VIEW someView AS
    SELECT email, username FROM accounts, blog_entires
    WHERE blog_entires.created_by = accounts.id
```

```
select * from someView
```

Exercises

```
create table if not exists Customers(id serial not null primary key, username varcha

insert into Customers  (username, email, password) values ('John', 'john@acme.com',
create table if not exists Products(id serial not null primary key, name varchar not

alter table Products add column manufacturer varchar not null;


insert into products(name, price, manufacturer) values
    ('Samsung Galaxy S20', 7799.95, 'Samsung'),
    ('Samsung galaxy s20 - leather cover', 799.95, 'Samsung'),
    ('Iphone 11 Pro', 8899, 'Apple'),
    ('Iphone 11 Pro - leather cover', 399.5, 'Apple'),
    ('Huawai P30 lite', 1664.5, 'Google'),
    ('Huawai P30 lite - leather cover', 1664.5, 'Google');


create table if not exists Orders (id serial not null primary key , order_number cha


insert into Orders(order_number, customer_id) values
    ('DA-0001234', 1),
    ('DA-001235', 1),
    ('DE-0001236', 2),
    ('DE-0001237', 2);

create table Order_lines(
    id serial not null primary key,
    order_id int not null references Orders(id),
    product_id int not null references Products(id),
    amount int not null
);

insert into Order_lines(order_id, product_id, amount) values
    (1,1,2),
    (1,2,2),
    (1,5,1),
    (3,3,2),
    (3,4,1),
    (4,1,1);


select c.username, c.email, p.name, p.price, p.manufacturer, ol.amount from Orders o
    inner join Customers c on o.customer_id = c.id
    inner join Order_lines ol on o.id = ol.order_id
    inner join Products p on ol.product_id = p.id;


create view order_info_view as
    select o.order_number, c.username, c.email, p.name, p.price, p.manufacturer, ol.
        inner join Customers c on o.customer_id = c.id
        inner join Order_lines ol on o.id = ol.order_id
        inner join Products p on ol.product_id = p.id;
```

```sql
select * from order_info_view where order_number = 'DA-0001234';
```