MoreNormalization and Import

Suppose a Database

users				
name	company	company_address	url1	url2
Joe	ABC	1 Work Lane	abc.com	xyz.com
Jill	XYZ	1 Job Street	abc.com	xyz.com

```
create table users1 (
        name varchar(20) NOT NULL,
        company varchar(20) not null,
        company_address varchar(20) not null,
        urls varchar[] not null
);

insert into users1 values ('Joe', 'ABC', '1
work Lane', array['abc.com', 'xyz.com']);

insert into users1 (name, company,
company_address, urls) values ('Jill',
'XYZ', 'i Job Street', array['abc.com',
'xyz.com']);
```

```
create table users1 (
        name varchar(20) NOT NULL,
        company varchar(20) not null,
        company_address varchar(20) not null,
        url1 varchar(20) not null,
        url2 varchar(20) not null
);

insert into users1 (name, company,
company_address, url1, url2) values ('Joe',
'ABC', '1 work Lane', 'abc.com', 'xyz.com');

insert into users1 (name, company,
company_address, url1, url2) values ('Jill',
'XYZ', 'i Job Street', 'abc.com', 'xyz.com');
```

DataBase Normalization

- First Normal Form — There are no multi-valued attributes
  - Eliminate repeating groups in individual tables.
    - e.g. the url1 & url2 fields
  - Create a separate row in table for each set of related data.
  - Identify each set of related data with a primary key.
    - So make a “userID” field to indicate that Joe in rows 1 and 2 is the same Joe

```
create table users1 (
       name varchar(20) NOT NULL,
       company varchar(20) not null,
       company_address varchar(20) not null,
       url varchar(20) not null,
);
insert into users1 (name, company, company_address,
url) values ('Joe', 'ABC', '1 work Lane',
'abc.com');
insert into users1 (name, company, company_address,
url) values ('Jill', 'XYZ', 'i Job Street',
'xyz.com');
insert into users1 (name, company, company_address,
url) values ('Joe', 'ABC', '1 work Lane',
'abc.com');
insert into users1 (name, company, company_address,
url) values ('Jill', 'XYZ', 'i Job Street',
'xyz.com');
```

users				
userId	name	company	company_address	url
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com

DataBase Normalization

- Second Normal Form — Non-key fields must be dependent upon the entire key
  - Create separate tables for sets of values that apply to multiple records.
    - e.g. the URL field for first normal form
  - Relate these tables with a foreign key.
    - reluserid in URLs is a foreign key to userid in users

users			
userId	name	company	company_address
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

2NF SQL

```
\c company

drop table if exists url2;
drop table if exists user2;

create table user2 (
    userid INT GENERATED ALWAYS AS IDENTITY,
    name varchar(20) NOT NULL,
    company varchar(20),
    company_address varchar(20),
    primary key (userid)
);
insert into user2 (name, company, company_address) values ('Joe', 'xyz', '1 work place');
insert into user2 (name, company, company_address) values ('Jill', 'abc', '1 job street');
insert into user2 (name, company, company_address) values ('Abby', 'xyz', '1 work place');
insert into user2 (name, company, company_address) values ('Loe', 'abc', '1 job street');
create table url2 (
    urlid INT GENERATED ALWAYS AS IDENTITY,
    relUserId int,
    url varchar(20),
    FOREIGN KEY(RelUserId) REFERENCES user3(userid),
    primary key(urlid)
);

insert into url2(relUserId, url) values(1, 'abc.com');
insert into url2(relUserId, url) values(2, 'abc.com');
insert into url2(relUserId, url) values(1, 'xyz.com');
insert into url2(relUserId, url) values(2, 'xyz.com');
```

DataBase Normalization

- Third Normal Form
  - Eliminate fields that do not depend on the key.
    - The company affiliation of Joe and Jill is NOT an attribute of those people and may be shared

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

```
drop table if exists urls3;
drop table if exists user3;
drop table if exists company3;

create table company3 (
    companyID INT GENERATED ALWAYS AS IDENTITY,
    company varchar(20),
    company_address varchar(20),
    primary key(companyid)
);
insert into company3 (company, company_address) values ('ABC', '1 Work Lane');
insert into company3 (company, company_address) values ('XYZ', '1 Job Street');
create table user3 (
    userid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name varchar(20) NOT NULL,
    RelCompanyId int,
    FOREIGN KEY(RelCompanyId) REFERENCES company3(companyID)
);
insert into user3 (name, RelCompanyId) values ('Joe', 1);
insert into user3 (name, RelCompanyId) values ('Jill', 2);
create table url3 (
    urlid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    relUserId int,
    url varchar(20),
    FOREIGN KEY(RelUserId) REFERENCES user3(userid)
);
```

4NF

- The URLs table has info in it that is not an attribute of the URL. Namely the “reluser” column.
- This feels wrong.
  - Also, we have data that repeats and could be updated separately. (The URL)
- Clean up by replacing the URL table with two:
  - URLs and URL_relations
- Fourth Normal Form
  - In a many-to-many relationship, independent entities can not be stored in the same table.

```
create table urls4 (
    urlid int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    url varchar(20)
);
insert into urls4 (url) values ('abc.com');
insert into urls4 (url) values ('xyz.com');
create table url_rel4 (
    relationID int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    RelURLid int,
    RelCompanyID int,
    FOREIGN KEY(RelURLid) REFERENCES urls4(urlid),
    FOREIGN KEY(RelCompanyId) REFERENCES company4(companyID)
);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 2);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 2);
```

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls	
urlId	url
1	abc.com
2	xyz.com

url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2

Querying this database

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls	
urlId	url
1	abc.com
2	xyz.com

url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2

- List all of Joe's urls
- List all employees of company 1
- List all employees of the company that Joe works for
- How many companies are in the database?

Why?

- **To free the collection of relations from undesirable insertion, update and deletion dependencies.**
- To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs.
- To make the relational model more informative to users.
- To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.
  - Codd (1970)

1NF

Remove multi-valued attributes

Why?

id	name	classes
1	Fred	113, 151
2	Ginger	113, 151, 231, 223

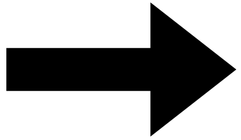
id	name	Class
1	Fred	113
2	Ginger	113
1	Fred	151
2	Ginger	151
2	Ginger	231
2	Ginger	223

2NF

1NF and all items depend only on primary key

Why?

id (Primary)	name	Year	classes
1	Fred	1	113
1	Fred	2	151
2	Ginger	1	113
2	Ginger	1	151
2	Ginger	2	231
2	Ginger	3	223



id (Primary)	name
1	Fred
2	Ginger

Year (Composite)	id (Composite)	classes
1	**1**	113
2	**1**	151
1	**2**	113
1	**2**	151
2	**2**	231
2	**3**	223

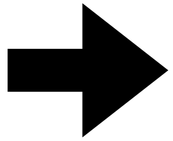
3NF

2NF and No transitive dependencies

Why?

id (Primary)	name
1	Fred
2	Ginger

Year (Com	id (Com	classes	title
1	**1**	113	Intro
2	**1**	151	DS
1	**2**	113	Intro
1	**2**	151	DS
2	**2**	231	DM
2	**3**	223	SP



id (Primary)	name
1	Fred
2	Ginger

Year (Comp	id (Comp	classes_ID
1	**1**	cs113
2	**1**	cs151
1	**2**	cs113
1	**2**	cs151
2	**2**	cs231
2	**3**	cs223

id	Title
cs113	Intro
cs151	DS
cs231	DM
cs223	SP

Databases and Families

```
create table nuclearfamily1 (
    parent1 varchar,
    parent2 varchar,
    child1 varchar,
    child2 varchar,
    child3 varchar,
    child4 varchar,
    child5 varchar
);

insert into nuclearfamily1 (parent1, parent2, child1, child2, child3, child4, child5)
values('a', 'b', 'c', 'd', 'e', 'f', 'g');
insert into nuclearfamily1 values('ab', 'bb', 'cb', 'db', 'eb', 'fb', 'gb');
insert into nuclearfamily1 (parent1, parent2, child1, child2, child4, child5) values('aa',
'ba', 'ca', 'da', 'fa', 'ga');
insert into nuclearfamily1 (parent1,  child1, child2, child4) values('aa', 'cd', 'dd',
'fd');
```

Better Family?

```
create table nuclearfamily2 (
    family_id INT GENERATED ALWAYS AS IDENTITY,
    parent1 varchar,
    parent2 varchar,
    child1 varchar,
    child2 varchar,
    child3 varchar,
    child4 varchar,
    child5 varchar
);

insert into nuclearfamily2 (parent1, parent2, child1, child2, child3, child4, child5) values('a', 'b',
'c', 'd', 'e', 'f', 'g');
-- this does not work!!!
insert into nuclearfamily2 values('ab', 'bb', 'cb', 'db', 'eb', 'fb', 'gb');
-- neither does this
insert into nuclearfamily2 values(2, 'ab', 'bb', 'cb', 'db', 'eb', 'fb', 'gb');

insert into nuclearfamily2 (parent1, parent2, child1, child2, child4, child5) values('aa', 'ba', 'ca',
'da', 'fa', 'ga');
insert into nuclearfamily2 (parent1,  child1, child2, child4) values('aa', 'cd', 'dd', 'fd');
```

Still Better

```
create table nuclearfamily3 (
    family_id INT GENERATED ALWAYS AS IDENTITY,
    parent1 varchar,
    parent2 varchar,
    child1 varchar,
    child2 varchar,
    child3 varchar,
    child4 varchar,
    child5 varchar,
    primary key (family_id)
);
insert into nuclearfamily2 (parent1, parent2, child1, child2, child3, child4, child5)
values('a', 'b', 'c', 'd', 'e', 'f', 'g');
insert into nuclearfamily2 (parent1, parent2, child1, child2, child4, child5) values('aa',
'ba', 'ca', 'da', 'fa', 'ga');
insert into nuclearfamily2 (parent1,  child1, child2, child4) values('aa', 'cd', 'dd',
'fd');
```

Better Still

```
create table people4 (
    person_id INT GENERATED ALWAYS AS IDENTITY,
    name varchar,
    primary key (person_id)
);
create table nuclearfamily4 (
    family_id INT GENERATED ALWAYS AS IDENTITY,
    parent1 int,
    parent2 int,
    child1 int,
    child2 int,
    primary key (family_id),
    foreign key (parent1) references people4(person_id),
    foreign key (parent2) references people4(person_id),
    foreign key (child1) references people4(person_id),
    foreign key (child2) references people4(person_id)
);
insert into people4(name) values('a');
insert into people4(name) values('b');
insert into people4(name) values('c');
insert into people4(name) values('d');
insert into people4(name) values('e');
insert into people4(name) values('f');
insert into people4(name) values('g');
insert into people4(name) values('h');
insert into people4(name) values('i');
insert into people4(name) values('j');
insert into nuclearfamily4(parent1, parent2, child1, child2) values(1,2,3,4);
insert into nuclearfamily4(parent1, parent2, child1, child2) values(5,6,7,8);
insert into nuclearfamily4(parent1, child1) values(9,10);
```

Better Person?

```
create table people6 (
    person_id INT GENERATED ALWAYS AS IDENTITY,
    name varchar,
    reltype varchar,
    primary key (person_id),
    unique (name),
    check (reltype in ('parent', 'child'))
);

insert into people6(name, reltype) values('az', 'parent');
-- this fails as it is not in parent/child
insert into people6(name, reltype) values('ay', 'parentt');
-- this fails as az is already in table.
insert into people6(name, reltype) values('az', 'child');

insert into people6(name, reltype) values(null, 'parent');
insert into people6(name, reltype) values(null, 'parent');

create table people7 (
    person_id INT GENERATED ALWAYS AS IDENTITY,
    name varchar not null,
    reltype varchar not null default 'child',
    primary key (person_id)
);
```

Maybe actually Good?

```
create table nf10 (
    family_id INT GENERATED ALWAYS AS IDENTITY,
    description varchar,
    primary key (family_id)
);

create table people10 (
    person_id INT GENERATED ALWAYS AS IDENTITY,
    family_id int,
    name varchar,
    reltype varchar,
    primary key (person_id),
    foreign key (family_id) references nf10(family_id),
    check (reltype in ('parent', 'child'))
);

insert into nf10(description) values('small family');
insert into nf10(description) values('medium family');
insert into nf10(description) values('big family');
insert into people10(family_id, name, reltype) values(1, 'a', 'parent');
insert into people10(family_id, name, reltype) values(1, 'b', 'child');
insert into people10(family_id, name, reltype) values(2, 'c', 'parent');
insert into people10(family_id, name, reltype) values(2, 'd', 'child');
insert into people10(family_id, name, reltype) values(2, 'e', 'parent');
insert into people10(family_id, name, reltype) values(2, 'f', 'child');
insert into people10(family_id, name, reltype) values(3, 'g', 'parent');
insert into people10(family_id, name, reltype) values(3, 'h', 'child');
insert into people10(family_id, name, reltype) values(3, 'i', 'parent');
insert into people10(family_id, name, reltype) values(3, 'k', 'child');
insert into people10(family_id, name, reltype) values(3, 'm', 'child');
insert into people10(family_id, name, reltype) values(3, 'o', 'child');
```

Importing data into PSQL

- Sometimes, rarely, you get lucky,
  - have a CSV file that exactly aligns with table structure
    - or you can adjust your table structure to match
  - 1 line of SQL!

- All other times
  - read data file into a real PL
  - use ODBC to send inserts to PSQL

Columns in table are in same order as CSV

FULL absolute path name!

Lucky

```
\c volleyb
drop table
drop table
drop table matches;
drop table points;
drop table users;

create table actionnames (
    actionid varchar(10),
    aname varchar(50),
    primary key(actionid)
);
copy actionnames from '/home/gtowell/Private/
383/Volleyball/aactionnames.csv' delimiter ';';
select count(*) from actionnames;

create table actions (
    matchid varchar(20),
    pointid varchar(10),
    player varchar(5),
    actionid varchar(4),
    value integer,
    setno integer,
    foreign key(actionid) references
actionnames(actionid)
);

copy actions from '/home/gtowell/Private/383/
Volleyball/aactions.csv' delimiter ';';
select count(*) from actions;
```

0;unusedo
1;unused1
2;Own Point
3;Opponent Point
4;Kill
5;Hitting Error
6;Attack
7;Dig
8;Assist
9;Ace Serve
10;Serve Error
11;Serve Made
12;Serve Received
13;Serve Receive Score
14;Serve Receive Error
15;Block
16;Block Error

"yA9zuWIMSCq2GFpIn45Y";"8KL5RHOraJ";"23";"3";"1";"1"
"yA9zuWIMSCq2GFpIn45Y";"8KL5RHOraJ";"2";"6";"1";"1"
"yA9zuWIMSCq2GFpIn45Y";"8KL5RHOraJ";"23";"11";"1";"1"
"yA9zuWIMSCq2GFpIn45Y";"fmQNXsSKUf";"23";"9";"1";"1"
"yA9zuWIMSCq2GFpIn45Y";"fmQNXsSKUf";"23";"22";"0";"1"
"yA9zuWIMSCq2GFpIn45Y";"fmQNXsSKUf";"2";"2";"1";"1"

Above is legal CSV but postgreSQL will not import

yA9zuWIMSCq2GFpIn45Y;8KL5RHOraJ;23;3;1;1
yA9zuWIMSCq2GFpIn45Y;8KL5RHOraJ;2;6;1;1
yA9zuWIMSCq2GFpIn45Y;8KL5RHOraJ;23;11;1;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;23;9;1;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;23;22;0;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;2;2;1;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;12;2;1;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;7;2;1;1
yA9zuWIMSCq2GFpIn45Y;fmQNXsSKUf;20;2;1;1

type is alternative to the "campus" table of Tuesday
Problem: 'type' is not in SQL standard

Unlucky

```
\c hurricane;
drop table if exists observation;
drop table if exists hurricane;
drop type if exists stype;

create type stype as enum ('', 'TY', 'PT', 'ET', 'ST', 'TD', 'TS', 'HU', 'EX', 'SD', 'SS', 'LO', 'WV', 'DB');

create table hurricane (
       HID char(8),
       Name varchar(25),
       constraint pp_con
       primary key(HID)
       );

create table observation (
       HID char(8),
       date date,
       time time,
       type stype,
       latitude float,
       latitudehemi char(1),
       longitude float,
       longitudehemi char(1),
       maxSustained integer,
       constraint PK_CON
       foreign Key(HID) references hurricane(HID)
       );
```

Hurricaine Data

```
AL011851,            UNNAMED,     14,
18510625, 0000,   , HU, 28.0N,  94.8W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 0600,   , HU, 28.0N,  95.4W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 1200,   , HU, 28.0N,  96.0W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 1800,   , HU, 28.1N,  96.5W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510625, 2100, L, HU, 28.2N,  96.8W,  80, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 0000,   , HU, 28.2N,  97.0W,  70, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 0600,   , TS, 28.3N,  97.6W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 1200,   , TS, 28.4N,  98.3W,  60, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510626, 1800,   , TS, 28.6N,  98.9W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 0000,   , TS, 29.0N,  99.4W,  50, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 0600,   , TS, 29.5N,  99.8W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 1200,   , TS, 30.0N, 100.0W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510627, 1800,   , TS, 30.5N, 100.1W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
18510628, 0000,   , TS, 31.0N, 100.2W,  40, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999, -999,
AL092016,            HERMINE,     47,
20160830, 1200,   , TD, 23.9N,  86.8W,  30, 1003,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160830, 1800,   , TD, 24.0N,  87.3W,  30, 1003,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160831, 0000,   , TD, 24.1N,  87.8W,  30, 1003,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160831, 0600,   , TS, 24.4N,  88.0W,  35, 1003,   40,   60,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160831, 1200,   , TS, 24.8N,  87.9W,  40, 1003,   70,   90,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160831, 1800,   , TS, 25.1N,  87.7W,  45, 1002,  100,  110,   60,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
20160901, 0000,   , TS, 25.5N,  87.2W,  50,  999,  120,  120,   90,    0,   20,   20,    0,    0,    0,    0,    0,    0,    0,
20160901, 0600,   , TS, 26.0N,  86.8W,  55,  995,  130,  130,   90,    0,   80,   70,    0,    0,    0,    0,    0,    0,    0,
20160901, 1200,   , TS, 26.9N,  86.2W,  60,  991,  130,  140,   80,   60,  100,  100,   30,   30,    0,    0,    0,    0,    0,
20160901, 1800,   , HU, 27.9N,  85.5W,  65,  988,  130,  150,   60,   60,   90,  110,   30,   40,   40,   40,    0,    0,    0,
20160902, 0000,   , HU, 29.0N,  84.8W,  70,  983,  130,  150,   60,   60,   70,  110,   40,   30,   40,   40,   30,    0,    0,
```

Using Python to import

```
import psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT

db=1

def insertStringH(sline):
    insert = "insert into hurricane (hid, name) values('{}', '{}');"
    return insert.format(sline[0].strip(), sline[1].strip());

def insertStringO(hid, sline):
    aa = {'Jan':'01', 'Feb':'02', 'Mar':'03', 'Apr':'04', 'May':'05', 'Jun':'06', 'Jul':'07', 'Aug':'08', 'Sep':'09', 'Oct':'10', 'Nov':'11', 'Dec':'12'}
    insert = "insert into observation (hid, date, time, type, latitude, latitudehemi, longitude, longitudehemi, maxsustained) values('{}', '{}', '{}', '{}', {}, '{
'{}', {});"
    return insert.format(hid.strip(), sline[0].strip(), sline[1], sline[3].strip(), sline[4][:-1], sline[4][-1], sline[5][:-1], sline[5][-1], sline[6])


if db!=0:
    cnx = psycopg2.connect(user="dbuser", database="hurricane", host="localhost", password="12345678")
    cnx.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT) # <-- ADD THIS LINE
    cursor = cnx.cursor()


cc=0
with open("hurdat2-1851-2019-052520.txt") as fp:
    line = fp.readline()
    hid=''
    while line:
        spl = line.strip().split(",")
        if len(spl)<6:
            hid=spl[0]
            ins = insertStringH(spl)
        else:
            ins = insertStringO(hid, spl)
        if db==1:
            try:
                cursor.execute(ins)
            except:
                print(line)
                print(ins)
                cc=cc+1
        else:
            print(ins)
        line = fp.readline()

print(cc)

if db!=0:
    cnx.commit()
    cursor.close()
    cnx.close()
```