

## SWEN439 Project 1

### Question 1

Banks:

Primary Key: {BankName, City}

I have chosen this as the primary key because according to the brief, it will always be unique because there are business rules set by the local banking authority to ensure that the combination is unique. It is also minimal and can't be null, so can be a primary key.

Foreign Keys:

None for this table

Attribute constraints:

BankName and City should be NOT NULL as they are part of the primary key

Constraint that means that there has to be 0 or more accounts since you can't have a negative amount of accounts. Constraint called positiveAccountNo

Constraint that means that the security level can only be excellent, very good, good or weak.

Robberies

Primary Key: {BankName, City, Date}

A bank can be robbed more than once so just {BankName, City} wouldn't be unique. Adding Date so would be that it is unique and therefore can be the primary key as it is unlikely that a bank would be robbed twice on the same day.

Foreign Keys:

$\{BankName, City\} \subseteq Banks\{BankName, City\}$

All robberies should occur on a bank that does exist so there should be a foreign key.

On Deletion or Updating:

CASCADE: because if a Bank changes name then the robberies for that bank should also be changed, since changing the name doesn't affect that fact that it has previously been robbed. Also, if a bank is deleted, then the robberies that occurred aren't important anymore

Attribute constraints

BankName, City and Date should be NOT NULL as they are part of the primary key

There should be a constraint to check that the amount stolen is positive. It doesn't make sense if the amount stolen is negative as it would mean that they gave money to the bank. This constraint is called `positiveStolen`

## Plans

Primary Key: {BankName, City, PlannedDate}

Robbers could plan to rob the same bank twice on different dates so by adding `PlannedDate`, it would make it unique, since I am assuming that they can't make plans to rob a bank twice on the same day.

Foreign Keys: {BankName, City}  $\subseteq$  Banks{BankName, City}

All planned robberies should occur on a bank that does exist so there should be a foreign key.

On Deletion or Updating:

CASCADE: because if a Bank changes name then the robberies for that bank should also be changed, since changing the name doesn't affect that fact that someone is planning to rob it. Also, if a bank is deleted, then it can no longer be robbed so the plan is useless

Attribute constraints

BankName, City and PlannedDate should be NOT NULL since they are part of the primary key

There should be a constraint to check that there is at least one robber who plans to rob the banks, as a plan with no robbers doesn't make sense. This constraint is called `positiveRobbers`

## Robbers

Primary Key: {RobberId}

Each robber has a unique id number so it is a good candidate for the primary key.

Foreign Keys: None for this table

Attribute constraints

RobberID should be NOT NULL as it is the primary key

There should be a constraint to check that the robber's age is a positive number since a negative number doesn't make any sense.

There should be a constraint to check that the number of years spent in prison is 0 or more as you can't do negative jail time.

There should be a constraint to check that the number of years spent in prison is less than their age.  
This constraint is called prisonLessAge

Skills

Primary Key: {SkillId}

SkillId is unique for each skill so should be the primary key.

Foreign Keys: None for this table

Attribute constraints:

The description of each skill has to be unique

HasSkills

Primary Key: {RobberId, SkillId}

A robber isn't able to have the same skill twice so it will be unique

Foreign Keys:

$\{\text{RobberId}\} \subseteq \text{Robbers}\{\text{RobberId}\}$

All robbers in the HasSkills table should be in the Robbers table as they need to be robbers that exist

$\{\text{SkillId}\} \subseteq \text{Skills}\{\text{SkillId}\}$

All skills that a robber can have should be listed in the skills table.

On Deletion or Updating

For both foreign keys, I used the same delete and updating policy

ON DELETE CASCADE because if a robber or a skill is deleted then the hasSkill tuple should be deleted as well

ON UPDATE RESTRICT because you shouldn't be able to update either the robberId or skillId from this table

Attribute constraints

RobberID and SkillID should be NOT NULL as they are part of the primary key

## HasAccounts

Primary Key {RobberId, BankName, City}

This table is used to show if a robber has a bank account at a certain bank. This can't be duplicated so it will be unique.

### Foreign Keys

{RobberId}  $\subseteq$  Robbers{RobberId}

Only robbers that exist can have bank accounts

{BankName, City}  $\subseteq$  Banks{BankName, City}

Only banks that exist can have robbers that have bank accounts at that bank

### On Deletion or Updating

For both foreign keys, I used the same delete and updating policy

ON DELETE CASCADE because if a robber or a bank is deleted then the hasAccount tuple should be deleted as well as it doesn't make sense anymore

ON UPDATE RESTRICT because you shouldn't be able to update either the robberId, bankName or City from this table

### Attribute constraints

BankName, City and RobberId shouldn't be null as they are part of the primary key and a null value in any of them doesn't make sense.

A robber can have up to 3 preferred skills but no more.

## Accomplices

Primary Key: {RobberId, BankName, City, RobberyDate}

A robber could help rob the same bank more than once so by adding the RobberyDate, that ensures that it will be unique

### Foreign Keys:

{BankName, City, RobberyDate}  $\subseteq$  Robberies{BankName, Date}

Data in this table should refer to robberies that have happened

{RobberId}  $\subseteq$  Robbers{RobberId}

An accomplice should be a robber that exists

### On Deletion or Updating

For {BankName, City, RobberyDate}  $\subseteq$  Robberies{BankName, City, Date}, it should be ON DELETE RESTRICT ON UPDATE RESTRICT. You should be able to update the bank information from this table. And Robberies has CASCADE on it data so it should be deleted from there.

For {RobberId}  $\subseteq$  Robbers{RobberId}, it should be ON DELETE CASCADE ON UPDATE RESTRICT as if the robber is deleted, then the information about being accomplices is no longer useful. But you shouldn't be able to update the RobberID from this table

### Attribute constraints

RobberId, BankName, City and RobberyDate should all be NOT NULL as they are part of the primary key and it doesn't make sense for any of them to be null

### Question 2

```
copy Banks (BankName, City, NoAccounts, Security) FROM  
'D:\2021\SWEN439\Project1\banks_21.data';
```

```
copy Plans (BankName, City, PlannedDate, NoRobbers) FROM  
'D:\2021\SWEN439\Project1\plans_21.data';
```

```
copy Robberies (BankName, City, Date, Amount) FROM  
'D:\2021\SWEN439\Project1\robberies_21.data';
```

```
CREATE TABLE tempRobbers (
```

```
    Nickname char(20),
```

```
    Age INT NOT NULL,
```

```
    NoYears INT NOT NULL
```

```
);
```

```
copy tempRobbers (Nickname, Age, NoYears) FROM 'D:\2021\SWEN439\Project1\robbers_21.data';
```

```
INSERT INTO Robbers (SELECT nextval('robbers_robberid_seq'), * FROM tempRobbers);
```

```
DROP TABLE tempRobbers;
```

```
CREATE TABLE TempHasSkills (  
    Nickname Char (20),  
    Description Char (20),  
    Preference Integer,  
    Grade Char(3),  
    PRIMARY KEY (Nickname,Description)  
);
```

```
copy tempHasSkills(Nickname,Description,Preference,Grade) FROM  
'D:\2021\SWEN439\Project1\hasskills_21.data';
```

```
CREATE VIEW disSkill AS SELECT DISTINCT Description from tempHasSkills;  
INSERT INTO Skills (SELECT nextval('skills_skillid_seq'), Description from disSkill);
```

```
DROP VIEW disSkill;
```

```
INSERT INTO hasSkills (SELECT r.RobberId, s.SkillId, t.Preference, t.Grade FROM Robbers r, Skills s,  
TempHasSkills t WHERE t.Nickname = r.Nickname AND t.Description = s.description);
```

```
DROP TABLE TempHasSkills;
```

```
CREATE TABLE tempHasAccounts (  
    Nickname char(20),  
    BankName char(20) NOT NULL,  
    City char(20) NOT NULL  
);
```

```
copy tempHasAccounts(Nickname,BankName,City) FROM  
'D:\2021\SWEN439\Project1\hasaccounts_21.data';
```

```
INSERT INTO hasAccounts (SELECT r.RobberId, t.BankName, t.City FROM Robbers r,  
tempHasAccounts t WHERE t.Nickname = r.Nickname);
```

```
DROP TABLE tempHasAccounts;
```

```
CREATE TABLE tempAccomplices(  
    Nickname char(20),  
    BankName char(20) NOT NULL,  
    City char(20) NOT NULL,  
    RobberyDate Date NOT NULL,  
    Share DECIMAL(15,2)  
);
```

```
copy tempAccomplices(Nickname,BankName,City,RobberyDate,Share) FROM  
'D:\2021\SWEN439\Project1\accomplices_21.data';
```

```
INSERT INTO Accomplices (SELECT r.RobberId, t.BankName,t.City,t.RobberyDate, t.Share FROM  
Robbers r, tempAccomplices t WHERE t.Nickname = r.Nickname);
```

```
DROP TABLE tempAccomplices;
```

I started with the Banks, Plans and Robberies tables because they were created by just reading in the data from the files provided.

The next table I added was the Robbers table. This is because a number of the later tables rely on data from the Robbers table, so it was important that this table was done before them.

The next table I added was the skills table. Just like the Robbers table, tables in the future such as hasSkills relies on data from the skills table so I did that one next.

The next table I added was the hasSkills table. It needed to be done after the Robbers and Skills tables as it requires the Id from both the previous table.

The next table I added was the hasAccounts table. It also requires the RobberId so it was important that that table was created after the Robbers table.

The final table added was the accomplices table as it also requires the RobberID so it needed to be created after the Robbers table.

### Question 3

1a)

Query: INSERT INTO Banks (BANKNAME, CITY, NOACCOUNTS, SECURITY) VALUES ('Loanshark Bank', 'Evanston', '100', 'very good');

ERROR: duplicate key value violates unique constraint "banks\_pkey" Detail: Key (bankname, city)=(Loanshark Bank , Evanston ) already exists.

There is already an entry in the Banks table that has the primary key so this tuple can't be added as it would mean that the primary key isn't unique

1b)

Query: INSERT INTO Banks (BANKNAME, CITY, NOACCOUNTS, SECURITY) VALUES ('EasyLoan Bank', 'Evanston', '-5', 'excellent');

ERROR: new row for relation "banks" violates check constraint "positiveaccountno" Detail: Failing row contains (EasyLoan Bank , Evanston , -5, excellent).

This violates the constraint that the number of accounts has to be 0 or more, as it doesn't make sense to have a negative number of accounts

1c)

Query: INSERT INTO Banks (BANKNAME, CITY, NOACCOUNTS, SECURITY) VALUES ('EasyLoan Bank', 'Evanston', '100', 'poor');

ERROR: new row for relation "banks" violates check constraint "banks\_security\_check" Detail: Failing row contains (EasyLoan Bank , Evanston , 100, poor).



The security for the plan has to be one of ('excellent', 'very good', 'good', 'weak') and poor isn't in that set so it is an error

2)

Query: INSERT INTO SKILLS(skillid, description) VALUES ('21', 'Driving')

ERROR: duplicate key value violates unique constraint "skills\_description\_key" Detail: Key (description)=(Driving ) already exists.

Each skill in the Skills table has to have a unique description and Driving is already in the table so it can't be added again.

3)

Query: INSERT INTO Robberies(BANKNAME, CITY, DATE, AMOUNT) VALUES ('NXP Bank', 'Chicago', '2019-01-08',1000);

ERROR: duplicate key value violates unique constraint "robberies\_pkey" Detail: Key (bankname, city, date)=(NXP Bank , Chicago , 2019-01-08) already exists

There is already an entry in the Robberies table that has those values to create the primary key, so it can't be added.

4)

Query: DELETE FROM Banks

WHERE BankName = 'PickPocket Bank'

AND City = 'Evanston'

AND NoAccounts = 2000

AND Security = 'very good';

ERROR: update or delete on table "robberies" violates foreign key constraint "accomplices\_bankname\_city\_robberydate\_fkey" on table "accomplices" Detail: Key (bankname, city, date)=(PickPocket Bank , Evanston , 2016-03-30) is still referenced from table "accomplices".

This can't be deleted because another table relies in the information can therefore it can't be deleted.

5)

Query: DELETE FROM robberies

WHERE BankName = 'Loanshark Bank'

AND City = 'Chicago'

AND Date = ''

AND Amount = '';

ERROR: invalid input syntax for type date: ''

This can't be deleted because Date is of the Date format and an empty string isn't a Date so it can't delete it.

6a)

Query: INSERT INTO Robbers(robberid, nickname, age, noyears) Values (1, 'Shotgun', 70, 0)

ERROR: duplicate key value violates unique constraint "robbers\_pkey" Detail: Key (robberid)=(1) already exists.

This can't be inserted into the table because there is already a robber with the ID of 1 and since that is the primary key, another can't be added

6b)

Query: INSERT INTO Robbers(robberid, nickname, age, noyears) Values (666, 'Jail Mouse', 25, 35);

ERROR: new row for relation "robbers" violates check constraint "prisonlessage" Detail: Failing row contains (666, Jail Mouse , 25, 35).

One of the constraints of the Robbers is that they can't have spent more years in prison then they have been alive and since  $35 > 25$ , this violates that constraints

7a)

Query: INSERT INTO HasSkills(robberid, skillid, preference, grade) VALUES (1,7,1,'A+');

ERROR: duplicate key value violates unique constraint "hasskills\_pkey" Detail: Key (robberid, skillid)=(1, 7) already exists.

This can't be added because there is already a tuple with ID of 1 that has the skill 7 so it can't be added again as {robberId, skillId} is the primary key

7b)

Query: INSERT INTO HasSkills(robberid, skillid, preference, grade) VALUES (1,7,1,'A+');

ERROR: new row for relation "hasskills" violates check constraint "preferencelimit" Detail: Failing row contains (1, 2, 0, A ).

The skills a robber has are ranked in preference and you can't rank some 0 as that doesn't mean anything.

7c)

Query: INSERT INTO hasskills(robberid, skillid, preference, grade) Values('666',1,1,'B-');

ERROR: insert or update on table "hasskills" violates foreign key constraint "hasskills\_robberid\_fkey" Detail: Key (robberid)=(666) is not present in table "robbers".

There is no robber with the ID 666 so it can't be added into the hasSkill table as there is no robber to match with that skill.

7d)

Query: INSERT INTO HasSkills(robberid, skillid, preference, grade) VALUES (3,20,3,'B+');

ERROR: insert or update on table "hasskills" violates foreign key constraint "hasskills\_skillid\_fkey" Detail: Key (skillid)=(20) is not present in table "skills"

There is no skillID in the Skills table that has the value of 20, so you can't have a skill that doesn't exist, so it can't be added

8)

Query: DELETE FROM skills

WHERE skillid = 7

AND description = 'Planning';

DELETE 0

Nothing could be deleted since there isn't a tuple in the database that matches. So nothing was deleted

9)

Query: DELETE FROM Robbers

WHERE RobberId = 1

AND NickName = 'Al Capone'

AND Age = 31

AND NoYears = 2;

ERROR: update or delete on table "robbers" violates foreign key constraint "accomplices\_robberid\_fkey" on table "accomplices" Detail: Key (robberid)=(1) is still referenced from table "accomplices".

This can't be deleted because there are rows in other tables that rely on this row, so it can't be deleted.

#### Question 4

1)

SQL Query: SELECT robberid, nickname, age FROM Robbers WHERE Age > 20 AND AGE < 40

Output:

robberid	nickname	age
1	Al Capone	31
6	Tony Genovese	28
11	Meyer Lansky	34
13	Mickey Cohen	24
19	Mike Genovese	35
20	Longy Zwillman	35
22	Greasy Guzik	25
23	Lepke Buchalter	25
24	Sonny Genovese	39

2)

SQL Query: SELECT bankname, city FROM Banks b WHERE NOT EXISTS(SELECT 1 FROM robberies r WHERE r.bankname = b.bankname AND r.city = b.city );

Output:

bankname	city
Bankrupt Bank	Evanston
Loanshark Bank	Deerfield
Inter-Gang Bank	Chicago
NXP Bank	Evanston
Dollar Grabbers	Chicago
Gun Chase Bank	Burbank
PickPocket Bank	Deerfield
Hidden Treasure	Chicago
Outside Bank	Chicago

3)

SQL Query: SELECT bankname, city FROM hasaccounts WHERE robberid = (SELECT robberid FROM robbers WHERE nickname = 'Al Capone');

Output:

bankname	city
Bankrupt Bank	Evanston
Loanshark Bank	Deerfield
Inter-Gang Bank	Chicago
NXP Bank	Evanston
Dollar Grabbers	Chicago
Gun Chase Bank	Burbank
PickPocket Bank	Deerfield
Hidden Treasure	Chicago
Outside Bank	Chicago

4)

SQL Query: SELECT bankname, city, noaccounts FROM banks b WHERE bankname NOT IN (SELECT d.bankname FROM Banks d WHERE d.city = 'Deerfield') ORDER BY noaccounts

Output:

bankname	city	noaccounts
Gun Chase Bank	Burbank	1999
Outside Bank	Chicago	5000
Bad Bank	Chicago	6000
Dollar Grabbers	Chicago	56005
Inter-Gang Bank	Chicago	100000
Penny Pinchers	Evanston	130013
Penny Pinchers	Chicago	156165
Bankrupt Bank	Evanston	444000
Inter-Gang Bank	Evanston	555555
Gun Chase Bank	Evanston	656565
NXP Bank	Evanston	656565
Dollar Grabbers	Evanston	909090
Hidden Treasure	Chicago	999999
NXP Bank	Chicago	1593311

5)

SQL Query: SELECT a.robberid, r.nickname, SUM(a.Share) AS Earnings FROM accomplices a, robbers r WHERE r.robberid = a.robberid GROUP BY a.robberid, r.nickname HAVING SUM(a.share) > 30000 ORDER BY SUM(a.Share) DESC;

Output:

robberid	nickname	earnings
5	Mimmy The Mau Mau	70000
15	Boo Boo Hoff	61448
16	King Solomon	59726
17	Bugsy Siegel	52601
3	Lucky Luchiano	42667
10	Bonnie	40085
1	Al Capone	39486
4	Anastazia	39169
8	Clyde	31800

6)

SQL Query: SELECT robberid, nickname, noyears from robbers WHERE noyears > 10

Output:

robberid	nickname	noyears
2	Bugsy Malone	15
3	Lucky Luchiano	15
4	Anastazia	15
6	Tony Genovese	16
7	Dutch Schulz	31
15	Boo Boo Hoff	13
16	King Solomon	43
17	Bugsy Siegel	13

7)

SQL Query: SELECT robberid, nickname, (age-noyears) as NumberOfYearsNotInPrison from robbers  
WHERE noyears > age/2

Output:

robberid	nickname	numberofyearsnotinprison
6	Tony Genovese	12
16	King Solomon	31

8)

SQL Query: SELECT s.Description, h.RobberId, r.Nickname FROM hasskills h, skills s, robbers r WHERE  
h.skillid = s.skillid AND h.robberid = r.RobberId ORDER BY s.description

Output:

description ↕	robberid ↕	nickname ↕
Cooking	18	Vito Genovese
Driving	17	Bugsy Siegel
Driving	3	Lucky Luchiano
Driving	5	Mimmy The Mau Mau
Driving	23	Lepke Buchalter
Driving	7	Dutch Schulz
Driving	20	Longy Zwillman
Eating	6	Tony Genovese
Eating	18	Vito Genovese
Explosives	24	Sonny Genovese
Explosives	2	Bugsy Malone
Guarding	4	Anastazia
Guarding	17	Bugsy Siegel
Guarding	23	Lepke Buchalter
Gun-Shooting	9	Calamity Jane
Gun-Shooting	21	Waxey Gordon
Lock-Picking	8	Clyde
Lock-Picking	3	Lucky Luchiano
Lock-Picking	7	Dutch Schulz
Lock-Picking	22	Greasy Guzik
Lock-Picking	24	Sonny Genovese
Money Counting	13	Mickey Cohen
Money Counting	14	Kid Cann
Money Counting	19	Mike Genovese
Planning	15	Boo Boo Hoff
Planning	8	Clyde
Planning	5	Mimmy The Mau Mau
Planning	1	Al Capone
Planning	16	King Solomon
Preaching	22	Greasy Guzik
Preaching	10	Bonnie
Preaching	1	Al Capone
Safe-Cracking	1	Al Capone
Safe-Cracking	24	Sonny Genovese
Safe-Cracking	12	Moe Dalitz
Safe-Cracking	11	Meyer Lansky
Scouting	8	Clyde
Scouting	18	Vito Genovese



### Question 5

1)

SQL Query: `SELECT DISTINCT p.bankname, p.city from Plans p LEFT JOIN robberies r ON p.bankname = r.bankname AND r.city=p.city WHERE r.date IS NULL OR extract(YEAR from p.planneddate) != extract(Year from r.date)`

Output:

bankname	city
Bad Bank	Chicago
Dollar Grabbers	Chicago
Gun Chase Bank	Evanston
Hidden Treasure	Chicago
Inter-Gang Bank	Evanston
Loanshark Bank	Deerfield
PickPocket Bank	Chicago
PickPocket Bank	Deerfield

2)

SQL Query: `SELECT DISTINCT r.robberid, r.nickname from Robbers r JOIN accomplices a on r.robberid = a.robberid JOIN banks b on a.bankname = b.bankname WHERE b.bankname NOT IN(SELECT bankname FROM hasaccounts h WHERE h.robberid = r.robberid)`

Output:

robberid	nickname
1	Al Capone
2	Bugsy Malone
3	Lucky Luchiano
4	Anastazia
7	Dutch Schulz
8	Clyde
10	Bonnie
12	Moe Dalitz
13	Mickey Cohen
14	Kid Cann
15	Boo Boo Hoff
16	King Solomon
17	Bugsy Siegel
18	Vito Genovese
20	Longy Zwillman
21	Waxey Gordon
22	Greasy Guzik
23	Lepke Buchalter
24	Sonny Genovese

3)

SQL Query: SELECT r.robberid, r.nickname, s.description FROM Robbers r JOIN hasskills h on r.robberid = h.robberid JOIN skills s on h.skillid = s.skillid WHERE 1 < (SELECT COUNT(h.robberid) from hasskills h WHERE h.robberid = r.robberid ) AND h.preference = 1

Output:

robberid	nickname	description
1	Al Capone	Planning
3	Lucky Luchiano	Lock-Picking
5	Mimmy The Mau Mau	Planning
7	Dutch Schulz	Lock-Picking
8	Clyde	Lock-Picking
17	Bugsy Siegel	Driving
18	Vito Genovese	Scouting
22	Greasy Guzik	Preaching
23	Lepke Buchalter	Driving
24	Sonny Genovese	Explosives

4)

SQL QUERY: With AvgShare AS (

SELECT City, Avg(share) as AvShare, Rank() Over(ORDER BY Avg(share) DESC) as ShareRank from accomplices a GROUP By a.city

)

SELECT r.bankname, r.city,r.date from robberies r WHERE r.city = (SELECT City from AvgShare WHERE ShareRank = 1)

Output:

bankname	city	date
Loanshark Bank	Evanston	2019-02-28
Inter-Gang Bank	Evanston	2018-02-14
Penny Pinchers	Evanston	2016-08-30
Gun Chase Bank	Evanston	2016-04-30
PickPocket Bank	Evanston	2016-03-30
Loanshark Bank	Evanston	2017-04-20
Inter-Gang Bank	Evanston	2016-02-16
Penny Pinchers	Evanston	2017-10-30
PickPocket Bank	Evanston	2018-01-30
Penny Pinchers	Evanston	2019-05-30
Loanshark Bank	Evanston	2016-04-20
Inter-Gang Bank	Evanston	2017-03-13
Dollar Grabbers	Evanston	2017-11-08
Dollar Grabbers	Evanston	2017-06-28

5)

Step-wise query:

CREATE VIEW earnings as (

select robberid, COUNT(RobberId) as total\_robberies, Sum(Share) as total\_earnings

from accomplices

GROUP BY robberid

```
);
```

```
CREATE VIEW activeRobbers as (
```

```
    SELECT * from earnings
```

```
    WHERE total_robberies > (select AVG(total_robberies) as total_robberies from earnings)
```

```
);
```

```
CREATE VIEW nicknames as (
```

```
    SELECT r.RobberId, r.Nickname
```

```
    from activeRobbers a
```

```
    JOIN robbers r on a.robberid = r.robberid
```

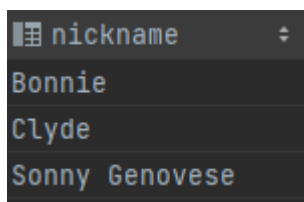
```
    WHERE r.noyears = 0
```

```
    ORDER BY total_earnings DESC
```

```
);
```

```
SELECT nickname from nicknames
```

Step Wise Output:



nickname
Bonnie
Clyde
Sonny Genovese

Single Nested Query:

```
SELECT nickname
```

```
FROM
```

```
(SELECT *
```

```
FROM (SELECT robberid,
```

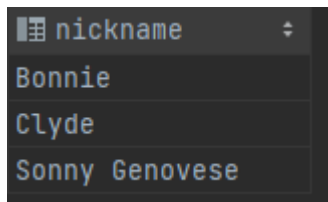
```
    COUNT(robberid) as totalrobberies,
```

```

SUM(share) as totalearnings
FROM accomplices
GROUP BY robberid) as earnings
WHERE totalrobberies > (SELECT AVG (totalrobberies)
FROM (select robberid,
COUNT(robberid) as totalrobberies,
SUM(share) as totalearnings
from accomplices
GROUP BY robberid) as earnings)) as activeRobbers
JOIN robbers r
ON r.robberid = activeRobbers.robberid
WHERE r.noyears = 0
ORDER BY totalearnings DESC;

```

Single Nested Query Output:



nickname
Bonnie
Clyde
Sonny Genovese

6)

Step-wise Query:

```

CREATE VIEW BanksAndSecurity as
(
    SELECT b.bankname, b.city, r.date, b.security
    FROM banks b
    JOIN Robberies r on b.bankname = R.bankname and b.city = R.city
);

```

```

CREATE VIEW AddAccomplices as (

    SELECT r.bankname, r.city, r.date, a.robberid, r.security

    FROM   accomplices a

    JOIN BanksAndSecurity r on r.bankname = a.bankname and r.city = a.city and r.date =
a.robberydate

);

```

```

CREATE VIEW AddSkills as (

    SELECT DISTINCT r.bankname, r.city, r.date, r.security, h.skillID

    FROM hasskills h

    JOIN AddAccomplices r on h.robberid = r.robberid

);

```

```

CREATE VIEW commonSkills as (

    SELECT DISTINCT r.security, r.skillid

    FROM AddSkills r

    WHERE (SELECT COUNT(skillid) from AddSkills a  WHERE a.skillid = r.skillid AND
a.security=r.security) = (SELECT COUNT(security) from BanksAndSecurity b WHERE b.security =
r.security)

);

```

```

CREATE VIEW robbersWithSkills as (

    SELECT r.security, r.skillID, h.robberId

    FROM hasskills h

    JOIN commonSkills r on h.skillid = r.skillid

);

```

```

CREATE VIEW nickname as (

```

```
SELECT r.security, r.skillID, rob.nickname  
FROM robbers rob  
JOIN robberswithskills r on rob.robberid = r.robberid  
);
```

```
CREATE VIEW AddDescription as (  
    SELECT r.security, r.nickname, s.description  
    FROM skills s  
    JOIN nickname r on s.skillid = r.skillid  
);
```

```
SELECT * FROM AddDescription;
```

Step Wise Output:

security	nickname	description
excellent	Al Capone	Planning
excellent	Mimmy The Mau Mau	Planning
excellent	Clyde	Planning
excellent	Boo Boo Hoff	Planning
excellent	King Solomon	Planning
good	Vito Genovese	Cooking
good	Tony Genovese	Eating
good	Vito Genovese	Eating
good	Clyde	Scouting
good	Vito Genovese	Scouting
good	Mickey Cohen	Money Counting
good	Kid Cann	Money Counting
good	Mike Genovese	Money Counting
very good	Bugsy Malone	Explosives
very good	Sonny Genovese	Explosives
very good	Al Capone	Safe-Cracking
very good	Meyer Lansky	Safe-Cracking
very good	Moe Dalitz	Safe-Cracking
very good	Sonny Genovese	Safe-Cracking

Single Nested Query:

```

SELECT j.security, j.nickname, s.description FROM skills s JOIN (
SELECT i.security, i.skillId, r.nickname FROM robbers r JOIN (
SELECT q.security, q.skillID, h.robberID FROM hasskills h JOIN (
SELECT DISTINCT n.security, n.skillId FROM (
SELECT DISTINCT m.bankname, m.city, m.date, m.security, h.skillID FROM hasskills h
JOIN (
SELECT l.bankname, l.city, l.date, a.robberId, l.security FROM accomplices a
JOIN ( SELECT b.bankname, b.city, r.date, b.security FROM banks b JOIN robberies r on
b.bankname = r.bankname and b.city = r.city) l on a.bankname = l.bankname AND a.city = l.city AND
a.robberydate = l.date) m

```



on m.robberid = h.robberid) n WHERE (SELECT COUNT(skillid) from (SELECT DISTINCT m.bankname, m.city, m.date, m.security, h.skillid FROM hasskills h

JOIN (

SELECT l.bankname, l.city, l.date, a.robberid, l.security FROM accomplices a

JOIN ( SELECT b.bankname, b.city, r.date, b.security FROM banks b JOIN robberies r on b.bankname = r.bankname and b.city = r.city) l on a.bankname = l.bankname AND a.city = l.city AND a.robberydate = l.date) m

on m.robberid = h.robberid)o WHERE n.skillid = o.skillid AND n.security = o.security) = (SELECT COUNT(security) FROM (SELECT b.bankname, b.city, r.date, b.security FROM banks b JOIN robberies r on b.bankname = r.bankname and b.city = r.city)p WHERE p.security=n.security))

q on h.skillid = q.skillid) i on r.robberid = i.robberid)j on s.skillid = j.skillid;

Single Nested Query Output:

security	nickname	description
excellent	Al Capone	Planning
excellent	Mimmy The Mau Mau	Planning
excellent	Clyde	Planning
excellent	Boo Boo Hoff	Planning
excellent	King Solomon	Planning
good	Vito Genovese	Cooking
good	Tony Genovese	Eating
good	Vito Genovese	Eating
good	Clyde	Scouting
good	Vito Genovese	Scouting
good	Mickey Cohen	Money Counting
good	Kid Cann	Money Counting
good	Mike Genovese	Money Counting
very good	Bugsy Malone	Explosives
very good	Sonny Genovese	Explosives
very good	Al Capone	Safe-Cracking
very good	Meyer Lansky	Safe-Cracking
very good	Moe Dalitz	Safe-Cracking
very good	Sonny Genovese	Safe-Cracking

7)

Step Wise Query:

```
CREATE VIEW robberiesSecurity as (  
    SELECT b.bankname, b.city, b.security, r.amount  
    FROM banks b  
    JOIN robberies r on b.bankname = r.bankname and b.city = r.city  
    ORDER BY b.security  
);
```

```
CREATE VIEW robberiesByLevel as (  
    SELECT security as SecurityLevel, COUNT(security) as TotalRobberies, AVG(amount) as  
    averageAmountStolen  
    FROM robberiesSecurity  
    GROUP BY security  
    ORDER BY TotalRobberies DESC  
);
```

```
SELECT * FROM robberiesByLevel;
```

Step Wise Output:

securitylevel	totalrobberies	averageamountstolen
excellent	12	39238.083333333333
weak	4	2299.5
very good	3	12292.426666666667
good	2	3980

Single Nested Query:

```
SELECT security as SecurityLevel, COUNT(security) as TotalRobberies, AVG(amount) as  
averageAmountStolen
```

FROM (SELECT b.bankname, b.city, b.security, r.amount FROM Banks b JOIN robberies r on  
b.bankname = r.bankname and b.city = r.city ORDER BY b.security) as RobbbberiesBySecurity

GROUP BY security

ORDER BY TotalRobberies DESC;

Single Nested Query Output:

securitylevel	totalrobberies	averageamountstolen
excellent	12	39238.083333333333
weak	4	2299.5
very good	3	12292.426666666666
good	2	3980