**1.**

CREATE TYPE securitytype AS ENUM ('weak', 'good', 'very good', 'excellent');

CREATE TABLE bank (

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

NoAccounts int DEFAULT ‘0’ NOT NULL CHECK(NoAccounts>0),

Security securitytype NOT NULL,

CONSTRAINT bank\_PK PRIMARY KEY(BankName, City)

);

CREATE TABLE robberies(

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

Date date NOT NULL,

Amount double precision DEFAULT ‘0’ NOT NULL CHECK(Amount>=0),

CONSTRAINT robberies\_PK PRIMARY KEY(BankName, City,Date),

CONSTRAINT robberies\_FK FOREIGN KEY(BankName,City)

REFERENCES bank(BankName,City)

);

CREATE TABLE plans(

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

PlannedDate date NOT NULL,

NoRobbers int DEFAULT ‘0’ NOT NULL CHECK(NoRobbers>0),

CONSTRAINT plans\_PK PRIMARY KEY(BankName, City, NoRobbers, PlannedDate),

CONSTRAINT plans\_FK FOREIGN KEY(BankName,City)

REFERENCES bank(BankName,City)

);

CREATE TABLE robbers(

RobberID serial NOT NULL,

Nickname varchar(255) NOT NULL,

Age int NOT NULL CHECK(Age>0) CHECK(Age>NoYears),

NoYears int DEFAULT ‘0’ NOT NULL,

CONSTRAINT robbers\_PK PRIMARY KEY(RobberID)

);

CREATE TABLE skills(

SkillID serial NOT NULL UNIQUE,

Description varchar(255) NOT NULL,

CONSTRAINT skill\_PK PRIMARY KEY(SkillID)

);

CREATE TABLE hasskills(

RobberID serial NOT NULL,

SkillID serial NOT NULL,

Preference int NOT NULL CHECK(Preference>0),

Grade varchar(255) NOT NULL,

CONSTRAINT hasskills\_PK PRIMARY KEY(RobberID, SkillID),

CONSTRAINT hasskills\_robber\_FK FOREIGN KEY(RobberID)

REFERENCES robbers(RobberID),

CONSTRAINT hasskills\_skill\_FK FOREIGN KEY(SkillID)

REFERENCES skills(SkillID)

);

CREATE TABLE hasaccounts(

RobberID serial,

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

CONSTRAINT hasaccounts\_PK PRIMARY KEY(RobberID,BankName,City),

CONSTRAINT hasaccounts\_FK FOREIGN KEY(RobberID)

REFERENCES robbers(RobberID)

);

CREATE TABLE accomplices(

RobberID serial,

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

RobberyDate date NOT NULL,

Share double precision DEFAULT ‘0’ NOT NULL CHECK(Share>=0),

CONSTRAINT accomplices\_PK PRIMARY KEY(RobberID, BankName, City, RobberyDate),

CONSTRAINT accomplices\_FK FOREIGN KEY(BankName, City)

REFERENCES bank(BankName, City),

CONSTRAINT accomplices\_robber\_FK FOREIGN KEY(RobberID)

REFERENCES robbers(RobberID)

);

**Primary & Foreign Keys:**

*Banks* – BankName & City are primary keys because when we use them as the candidate key, we are able to uniquely identify each tuple in the Banks. BankName will by itself will not work because there are many banks with the same name, however if we also state the City with the BankName, we will know which bank it would be as each City only contains one Bank of the same name.

*Robberies* – BankName, City and Date is used as the primary key as it is the only way for us to unique determine a tuple in Robberies. The foreign key is Bankname and City so we are able to create a connection to Banks.

*Plans –* BankName, City, NoRobbers and PlannedDate is used here as a Primary Key because it is the only way of successfully identifying each tuple uniquely. The Foreign Key is BankName and City so we can create a connection to Banks.

*Robbers* – RobberID is the primary key because it allows us to identify each tuple uniquely in Robbers. It does not need another attribute to assist it because we know that every RobberID will be unique, therefore it will be sufficient.

*Skills ­*– SkillID is the primary key here because we know each SkillID will be unique, therefore allowing us to identify each tuple successfully.

*HasSkills* – RobberID and SkillID are the primary key because a RobberID by itself may not be sufficient because a Robber may have more than one skill therefore could appear twice in the table. SkillID is also not sufficient enough by itself to determine a tuple because many robbers could have the same skill. However if we combine and use them together, we are able to identify each tuple successfully since no RobberID and SkillID can be repeated. The foreign keys are RobberID and SkillID (separately) as they are used as the primary keys of Robbers and Skills.

*HasAccounts* – The primary key is RobberID, BankName and City. The foreign key is RobberID, allowing us to create a connection to robbers.

*Accomplices* – The primary key is (RobberID, BankName, City, RobberyDate) and the foreign keys are (BankName, City) and (RobberID). This is because it allows us to uniquely identify each tuple, and since it is the primary key of Robber, then it is also a foreign key.

**Delete & Update Foreign Keys**

*Robberies –* If a tuple was deleted with the same BankName and City, then we do not delete the Robberies tuple with the same BankName and because we may still want to know what the robbers were thinking of targeting, possibly helping with the investigation. If there were any updates, we would also update the Robberies tuple with the same BankName and City.

*Plans* – Like Robberies, if a tuple was deleted with the same BankName and City then we would not want to delete the Plans tuple with the same foreign key. This is because even though it does not exist in Bank we may still want to know what the robbers’ plans were and what banks they were targeting. If there was an update, then we would also update the Plans.

*HasSkills* – If we are removing a robber from robbers, we do not want to remove any tuples with the same robberID because we may want to still keep their data if the robber was to be re-added into the robbers table. This means we can still look up their skills even though they were previously deleted. If we are deleting a skill from Skills, we do not want to delete tuples with the same skillID as we will still need to know about the robber’s skills, but could instead change their skillID to another skillID that they are best at after this skill has been removed. When deleting the skill, we can change the foreign key of skillID to null so that we do not delete that tuple. If we want to update, we can simply update the tuples required.

*HasAccounts* – If the robberID is deleted from Robbers, then we do not want to delete their tuple from Accounts, since it is possible we may still want to look up information about them if the robber was re-added back in Robbers. If we want to update, we can do so and simply update the data required.

*Accomplices* – If the foreign keys (robberID, BankName) or (robberID) were deleted from Robbers, then we do not need to delete them from Accomplices since we may want to still keep information about them if they were re-added. If we want to update, then we will keep them and update the information needed to be updated.

**Attribute Constraints**

* ***Bank –***

*BankName*: I decided to make this NOT NULL because it is important that we are able to identify each tuple, therefore it will not help if it were null. Since it is a primary key with City, we need it to have a value in order to allow us to uniquely identify each tuple.

*City*: Not null, because since it is a primary key with BankName, we need it to not be null in order for us to successfully identify each tuple.

NoAccounts: I decided to make it default 0, because by default the number of accounts will be zero unless more are added.

*Security*: NOT NULL because we do not want any missing information in the tables. This may cause problems when reading from the table.

*CHECK(NoAccounts>0):* This is because the number of accounts can only be above 0.

* ***Robberies –***

*Amount:* The amount is default 0 because we initially do not know how many the robbers have stolen and is only updated after the robbery. It is NOT NULL because we do not want missing information in the table.

*Check:* The amount stolen must be a positive value or 0.

* ***Plans –***

*NoRobbers:* Initially we do not know how many robbers there are therefore it is set at a default value of 0.

*CHECK(NoRobbers>0):*The number of robbers must be a positive value.

* ***Robbers –***

*RobberID:* RobberID cannot be null because it is the primary key of Robbers.

*NoYears:* The default value is 0 because we do not know how many years they have been in jail for. This is updated when the value is known.

*CHECK(Age>0):* Their age must be a positive value.

CHECK(Age>NoYears): Not possible for the number of years in jail to be above age.

* ***Skills –***

*SkillID:* Cannot be null because we require it to uniquely identify each tuple.

* ***HasSkills –***

*RobberID:* Cannot be null because we require it to identify each tuple. It will also not be used for deletion as we want to keep all data about the robbers, so we do not need to make it null.

*CHECK(Preference>0):* The preference rank must be a positive value.

*Check(Grade>0):*The grade must be a positive value.

* ***Accounts –***

*RobberID:* Value must be unique in order for us to uniquely identify each tuple in Accounts.

* ***Accomplices –***

*RobberID:* Must be unique in order to identify each tuple successfully.

*Share:* The share of each robber is default 0 because initially we do not know how much of the share they get. This value is updated later when it is known.

*CHECK(Share>=0):*The share must be 0 or above.

NOTE: I decided to use NOT NULLS for the attributes, because I do not want any missing data which may cause problems if they were null.

**2.**

a.

For bank, plans and robberies I simply copied the data from the files to the table as I was not required to do anything. However the rest required inner joining:

**Robbers**

CREATE TABLE temprobbers(

RobberNickname varchar(255) NOT NULL,

Age int NOT NULL CHECK(Age>0) CHECK(Age>NoYears),

NoYears int DEFAULT ‘0’ NOT NULL

);

**Skills**

Creating a temporary hasskills and copy data from hasskills\_14.data to it so I can copy the skills description over to skills table:

CREATE TABLE TEMPhasskills(

RobberNickname varchar(255) NOT NULL,

SkillDescription varchar(255) NOT NULL,

Preference int NOT NULL CHECK(Preference>0),

Grade varchar(255) NOT NULL,

CONSTRAINT TEMPhasskills\_PK PRIMARY KEY(RobberNickname, SkillDescription)

);

Now copy SkillDescription from temphasskills to skills table:

INSERT INTO skills (description)

SELECT skilldescription

FROM temphasskills;

Since SkillID is serial, it will automatically generate an ID for each skill.

**HasAccounts**

Now trying to do hasaccounts:

First off I need to create a temporary hasaccounts and copy data from has accounts\_14.data to it so I can copy the columns I need.

CREATE TABLE temphasaccounts(

RobberNickname varchar(255),

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL

);

Now we can copy bankname and city from temphasaccounts to has accounts and obtain a robberID from robes and put it into HasAccounts.

INSERT INTO HasAccounts

SELECT Robbers.RobberID,TempHasAccounts.BankName,TempHasAccounts.City

FROM Robbers

INNER JOIN TempHasAccounts

ON Robbers.Nickname=TempHasAccounts.robberNickname;

**Accomplices**

Now trying to do accomplices:

First must create a temporary accomplices table and copy data from accomplices\_14.data to it. It is called tempaccomplices.

CREATE TABLE tempaccomplices(

RobberNickname varchar(255),

BankName varchar(255) NOT NULL,

City varchar(255) NOT NULL,

RobberyDate date NOT NULL,

Share double precision DEFAULT ‘0’ NOT NULL CHECK(Share>=0)

);

Now copy from tempaccomplices into accomplices with the robberid from robbers

INSERT INTO accomplices

SELECT Robbers.RobberID,Tempaccomplices.BankName,Tempaccomplices.City, Tempaccomplices.robberydate,Tempaccomplices.share

FROM Robbers

INNER JOIN Tempaccomplices

ON Robbers.Nickname=Tempaccomplices.robberNickname;

**HasSkills**

I will need to get robberid from robber and skillid from skills table.

We have temphasskills which was created earlier to copy preference and grade from.

INSERT INTO hasskills

SELECT Robbers.RobberID,Skills.skillid,temphasskills.preference,temphasskills.grade

FROM Robbers

INNER JOIN Temphasskills

ON Robbers.Nickname=Temphasskills.robberNickname

INNER JOIN skills

ON Skills.description = Temphasskills.skilldescription;

b.

Things that enforced a partial order would be the fact that certain tables had to have certain data exist before other tables could be made. The following is the order I had to create the tables in order for it to function as expected:

**Create bank**

**Copy bank data from banks\_14.data**

This was done first because if I wanted to create robberies and plans table after, I will need to reference a bank.

**Create robberies**

**Copy robberies data from robberies\_14.data**

**Create plans**

**Copy plans data from plans\_14.data**

**Create robbers**

**Copy robbers data from robbers\_14.data**

**Create skills**

Robbers and skills was created before hasskills because hasskills references robberid and skillsid, therefore they are required to be created beforehand.

**Create hasskills**

**Create hasaccounts**

Hasaccounts was created now because we are required to have a robberID to be referenced for this table in order to function as expected.

**Create accomplices**

Accomplices was created now because we needed data from bank and the robberid. Since robberid was just created, we now create the accomplices table with reference to the robbers table.

**Create temphasskills**

**Copy temphasskills data from hasskills\_14.data**

**Insert into skills the description from temphasskills**

Temphasskills is created now because I now want to set up the hasskills table, but I am required to create this before inserting any data into hasskills as hasskills requires a skillid where the file does not contain a skillid, therefore a temporary table must be made in order to to save the data into the table and then copied into hasskills to generate the id or else the data in the file would not be accepted into the hasskills table.

**Create temphasaccounts**

**Copy temphasaccounts data from hasaccounts\_14.data**

**Insert required attributes from temphasaccounts and robbers into has accounts**

We are required to create a temporary hasaccounts table before inserting data into has accounts as we cannot directly copy the data into has accounts due to the table requiring robberid, bankname and city. The file does not contain all these attributes to perfectly match up with the has accounts table, therefore a temporary one is made and is inner joined with other tables in order to obtain the appropriate attributes.

**Create tempaccomplices**

**Copy tempaccomplices data from accomplices\_14.data**

**Insert required attributes from tempaccomplices and robbers into accomplices**

We are required to create a temporary accomplices table as we cannot directly copy data into accomplices table. This is because the accomplices data file does not contain all the attributes so that it perfectly matches the accomplices table’s attributes. Instead we must inner join tempaccomplices and robbers and insert into accomplices in order to create the table.

**Insert required attributes from robbers, skills and temphasskills into hasskills.**

Since we have already read in the data for temphasskills, we now simply insert into hasskills with inner join of robbers, temphasskills and skills in order to obtain the attributes’ data we need for hasskills table.

**3.**

1.

a.

garmandb=> INSERT INTO bank VALUES ('Loanshark Bank', 'Evanston', 100, 'very go

od');

ERROR: duplicate key value violates unique constraint "bank\_pk"

DETAIL: Key (bankname, city)=(Loanshark Bank, Evanston) already exists.

It attempts to add a new entry which contains a bankname and city which already exists in one tuple in the bank table, meaning it would not be unique.

b.

garmandb=> INSERT INTO bank VALUES ('EasyLoan Bank', 'Evanston', -5, 'excellent

');

ERROR: new row for relation "bank" violates check constraint "bank\_noaccounts\_c

heck"

It’s trying to enter a negative value for the number of accounts. However this is not possible as it must be a positive value.

c.

garmandb2=> INSERT INTO bank VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');

ERROR: invalid input value for enum securitytype: "poor"

LINE 1: ...INTO bank VALUES ('EasyLoan Bank', 'Evanston', 100, 'poor');

^

The security type for a bank can only one of four possibilities: weak, good, very good, excellent.

2.

a.

garmandb=> INSERT INTO skills VALUES (20, 'Guarding');

ERROR: duplicate key value violates unique constraint "skill\_pk"

DETAIL: Key (skillid)=(20) already exists.

Trying to add an entry where the key 20 already exists.

3.

a.

garmandb2=> INSERT INTO robbers VALUES (1, 'Shotgun', 70, 0);

ERROR: duplicate key value violates unique constraint "robbers\_pk"

DETAIL: Key (robberid)=(1) already exists.

The robberid of 1 already exists, therefore it cannot add a new entry with the same robberid.

b.

garmandb2=> INSERT INTO robbers VALUES (333, 'Jail Mouse', 25, 35);

ERROR: new row for relation "robbers" violates check constraint "robbers\_check"

The age of the robber cannot be less than the number of years they were in jail.

4.

a.

garmandb2=> INSERT INTO hasskills VALUES (333, 1, 1, 'B-');

ERROR: insert or update on table "hasskills" violates foreign key constraint "hasskills\_robber\_fk"

DETAIL: Key (robberid)=(333) is not present in table "robbers".

It is attempting to refer to a robberid of 333 which does not exist in the robbers table therefore it fails.

b.

garmandb3=> INSERT INTO hasskills VALUES (3, 20, 3, 'B+');

ERROR: insert or update on table "hasskills" violates foreign key constraint "h

asskills\_skill\_fk"

DETAIL: Key (skillid)=(20) is not present in table "skills”.

Attemping to insert into hasskills while referring to a skillid which does not exist, which in this case was 20.

c.

garmandb2=> INSERT INTO hasskills VALUES (1, 7, 1, 'A+');

ERROR: duplicate key value violates unique constraint "hasskills\_pk"

DETAIL: Key (robberid, skillid)=(1, 7) already exists.

It is trying to add a new tuple with a robberid of 1 when it is already in the table.

d.

garmandb2=> INSERT INTO hasskills VALUES (1, 2, 0, 'A');

ERROR: new row for relation "hasskills" violates check constraint "hasskills\_preference\_check"

It is attempting to refer to a preference value of 0 even though a preference rank cannot be 0.

5.

a.

garmandb=> INSERT INTO robberies VALUES ('NXP Bank', 'Chicago', '2009-01-08', 1

000);

ERROR: duplicate key value violates unique constraint "robberies\_pk"

DETAIL: Key (bankname, city, date)=(NXP Bank, Chicago, 2009-01-08) already exists.

It is attempting to add a duplicate primary key when it already exists.

6.

a.

DELETE FROM bank

WHERE bankname=’PickPocket Bank’ AND city=’Evanston’ AND noaccounts=2000 AND security=’very good’;

(‘PickPocket Bank’, ‘Evanston’, 2000, ‘very good’)

ERROR: update or delete on table "bank" violates foreign key constraint "robber

ies\_fk" on table "robberies"

DETAIL: Key (bankname, city)=(PickPocket Bank, Evanston) is still referenced fr

om table "robberies".

It cannot delete it because it is used as a foreign key in another table, which in this case is in robberies table.

b.

DELETE FROM bank

WHERE bankname=’Outside Bank’ AND city=’Chicago’ AND noaccounts=5000 AND security=’good’;

*I do not see a problem with this delete, there is nothing referring to it from what I can see in the tables.*

7.

a.

garmandb3=> 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 "has

skills\_robber\_fk" on table "hasskills"

DETAIL: Key (robberid)=(1) is still referenced from table "hasskills".

It is attempting to delete a robber when it is still being referred in hasskills table.

8.

a.

DELETE FROM skills

WHERE skillid=7 AND description=’Driving’;

*Note: I changed the skillid to 7 because my driving was skillid 7 not skillid1*

garmandb3=> DELETE FROM skills

WHERE skillid=7 AND description='Driving';

ERROR: update or delete on table "skills" violates foreign key constraint "hass

kills\_skill\_fk" on table "hasskills"

DETAIL: Key (skillid)=(7) is still referenced from table "hasskills".

It is attemping to delete the skill driving when it is still being referred to in hasskills.

**4.**

1.

SELECT bankname, security FROM bank

WHERE noaccounts>9000;

garmandb2=> SELECT bankname, security FROM bank

garmandb2-> WHERE noaccounts>9000;

bankname | security

-----------------+-----------

NXP Bank | very good

Bankrupt Bank | weak

Loanshark Bank | excellent

Loanshark Bank | very good

Loanshark Bank | excellent

Inter-Gang Bank | excellent

Inter-Gang Bank | excellent

NXP Bank | excellent

Penny Pinchers | weak

Dollar Grabbers | very good

Penny Pinchers | excellent

Dollar Grabbers | good

Gun Chase Bank | excellent

PickPocket Bank | weak

Hidden Treasure | excellent

(15 rows)

2.

SELECT DISTINCT bankname FROM hasaccounts

INNER JOIN robbers

ON robbers.nickname = ‘Calamity Jane’ AND robbers.robberid=hasaccounts.robberid;

garmandb2=> SELECT DISTINCT bankname FROM hasaccounts

INNER JOIN robbers

ON robbers.nickname = 'Calamity Jane' AND robbers.robberid=hasaccounts.robberid

;

bankname

-----------------

Bad Bank

Dollar Grabbers

PickPocket Bank

(3 rows)

3.

SELECT bankname, city FROM bank WHERE city!=’Chicago’ ORDER BY noaccounts;

garmandb2=> SELECT bankname, city FROM bank WHERE city!='Chicago' ORDER BY noac

counts;

bankname | city

-----------------+-----------

Gun Chase Bank | Deerfield

PickPocket Bank | Evanston

PickPocket Bank | Deerfield

Penny Pinchers | Evanston

Bankrupt Bank | Evanston

Inter-Gang Bank | Evanston

Gun Chase Bank | Evanston

NXP Bank | Evanston

Dollar Grabbers | Evanston

Loanshark Bank | Deerfield

Loanshark Bank | Evanston

(11 rows)

4.

SELECT bankname, city FROM robberies

ORDER BY date

LIMIT 1;

garmandb2=> SELECT bankname, city FROM robberies

ORDER BY date

LIMIT 1;

bankname | city

----------------+----------

Loanshark Bank | Evanston

(1 row)

5.

SELECT DISTINCT robbers.robberid, robbers.nickname, SUM(share) FROM robbers

INNER JOIN accomplices

ON robbers.robberid=accomplices.robberid

GROUP BY robbers.robberid, robbers.nickname

ORDER BY SUM(share) DESC;

garmandb2=> SELECT DISTINCT robbers.robberid, robbers.nickname, SUM(share) FROM

robbers

garmandb2-> INNER JOIN accomplices

garmandb2-> ON robbers.robberid=accomplices.robberid

garmandb2-> GROUP BY robbers.robberid, robbers.nickname

garmandb2-> ORDER BY SUM(share) DESC;

robberid | nickname | sum

----------+-------------------+----------

5 | Mimmy The Mau Mau | 70000

15 | Boo Boo Hoff | 61447.61

16 | King Solomon | 59725.8

17 | Bugsy Siegel | 52601.1

3 | Lucky Luchiano | 42667

10 | Bonnie | 40085

1 | Al Capone | 39486

4 | Anastazia | 39169.62

8 | Clyde | 31800

21 | Waxey Gordon | 16447.1

7 | Dutch Schulz | 15250

20 | Longy Zwillman | 14648.99

24 | Sonny Genovese | 13664

23 | Lepke Buchalter | 7085

18 | Vito Genovese | 6800

22 | Greasy Guzik | 6549.1

11 | Meyer Lansky | 3000

2 | Bugsy Malone | 2300

13 | Mickey Cohen | 2000

14 | Kid Cann | 1790

12 | Moe Dalitz | 31.99

(21 rows)

6.

SELECT hasskills.robberid, robbers.nickname, skills.description

FROM skills

INNER JOIN hasskills

ON hasskills.skillid = skills.skillid

INNER JOIN robbers

ON robbers.robberid = hasskills.robberid

GROUP BY skills.description;**<- Ignored.**

garmandb3=> SELECT hasskills.robberid, robbers.nickname, skills.description

FROM skills

INNER JOIN hasskills

ON hasskills.skillid = skills.skillid

INNER JOIN robbers

ON robbers.robberid = hasskills.robberid;

robberid | nickname | description

----------+-------------------+----------------

24 | Sonny Genovese | Safe-Cracking

12 | Moe Dalitz | Safe-Cracking

11 | Meyer Lansky | Safe-Cracking

1 | Al Capone | Safe-Cracking

24 | Sonny Genovese | Explosives

2 | Bugsy Malone | Explosives

23 | Lepke Buchalter | Guarding

17 | Bugsy Siegel | Guarding

4 | Anastazia | Guarding

21 | Waxey Gordon | Gun-Shooting

9 | Calamity Jane | Gun-Shooting

18 | Vito Genovese | Cooking

18 | Vito Genovese | Scouting

8 | Clyde | Scouting

23 | Lepke Buchalter | Driving

20 | Longy Zwillman | Driving

17 | Bugsy Siegel | Driving

7 | Dutch Schulz | Driving

5 | Mimmy The Mau Mau | Driving

3 | Lucky Luchiano | Driving

19 | Mike Genovese | Money Counting

14 | Kid Cann | Money Counting

13 | Mickey Cohen | Money Counting

22 | Greasy Guzik | Preaching

10 | Bonnie | Preaching

1 | Al Capone | Preaching

24 | Sonny Genovese | Lock-Picking

22 | Greasy Guzik | Lock-Picking

8 | Clyde | Lock-Picking

7 | Dutch Schulz | Lock-Picking

3 | Lucky Luchiano | Lock-Picking

16 | King Solomon | Planning

15 | Boo Boo Hoff | Planning

8 | Clyde | Planning

5 | Mimmy The Mau Mau | Planning

1 | Al Capone | Planning

18 | Vito Genovese | Eating

6 | Tony Genovese | Eating

(38 rows)

7.

SELECT robberid, nickname, noyears FROM robbers

WHERE noyears>3;

garmandb2=> SELECT robberid, nickname, noyears FROM robbers

garmandb2-> WHERE noyears>3;

robberid | nickname | noyears

----------+----------------+---------

2 | Bugsy Malone | 15

3 | Lucky Luchiano | 15

4 | Anastazia | 15

6 | Tony Genovese | 16

7 | Dutch Schulz | 31

11 | Meyer Lansky | 6

15 | Boo Boo Hoff | 13

16 | King Solomon | 43

17 | Bugsy Siegel | 13

20 | Longy Zwillman | 6

(10 rows)

8.

SELECT robberid, nickname, age-noyears FROM robbers

WHERE noyears>age/2;

garmandb2=> SELECT robberid, nickname, age-noyears FROM robbers

garmandb2-> WHERE noyears>age/2;

robberid | nickname | ?column?

----------+---------------+----------

6 | Tony Genovese | 12

16 | King Solomon | 31

(2 rows)

**5.**

1.

**Stepwise**

*Prints the nicknames of the robbers who participated in* ***more than the average amount****.*

CREATE VIEW nicknames

AS SELECT nickname FROM robbers

INNER JOIN numrobberies

ON robbers.robberid = numrobberies.robberid AND robbers.noyears=0

INNER JOIN average

ON numrobberies.robcount>average.avgcalculated;

*Calculates the number of robberies each robber participated in with 0 years in prison.*

CREATE VIEW numrobberies

AS SELECT accomplices.robberid, count(accomplices.robberid) as robcount FROM accomplices

GROUP BY accomplices.robberid

ORDER BY robcount DESC;

*Calculates the average number of robberies.*

CREATE VIEW average

AS SELECT SUM(robcount)/Count(robberid) as avgcalculated FROM numrobberies;

**Nested**

SELECT robbers.nickname FROM

  (SELECT robberid, count(robberid), sum(share) as total FROM accomplices

    GROUP BY robberid

    HAVING count(robberid)>

(SELECT AVG(num\_of\_robberies) as averagerobberies FROM

    (SELECT robberid, count(robberid) as num\_of\_robberies FROM accomplices

      GROUP BY robberid) AS accompdata

)

  ) AS count\_table

  NATURAL JOIN robbers

  WHERE count\_table.robberid = robbers.robberid AND robbers.noyears=0

  ORDER BY total DESC;

nickname

----------------

Bonnie

Clyde

Sonny Genovese

(3 rows)

2.

**Stepwise**

CREATE VIEW security

AS SELECT bank.security, norobberies.robcount, avgamount.average FROM bank

INNER JOIN norobberies

ON bank.security = norobberies.security

INNER JOIN avgamount

ON bank.security = avgamount.security

GROUP BY bank.security,norobberies.robcount, avgamount.average;

CREATE VIEW norobberies

AS SELECT bank.security as security, COUNT(robberies.bankname) as robcount FROM bank

INNER JOIN robberies

ON bank.bankname = robberies.bankname AND bank.city = robberies.city

GROUP BY bank.security;

CREATE VIEW avgamount

AS SELECT bank.security as security, AVG(robberies.amount) as average FROM bank

INNER JOIN robberies

ON bank.bankname = robberies.bankname AND bank.city = robberies.city

GROUP BY bank.security;

**Nested**

SELECT bank.security as banksecurity, COUNT(robberies.bankname) as no\_times\_robbed, AVG(amount) as avg\_amount FROM robberies

INNER JOIN bank

ON robberies.bankname = bank.bankname AND robberies.city = bank.city

GROUP BY bank.security;

banksecurity | no\_times\_robbed | avg\_amount

--------------+-----------------+------------------

good | 2 | 3980

very good | 3 | 12292.4266666667

weak | 4 | 2299.5

excellent | 12 | 39238.0833333333

(4 rows)

3.

**Stepwise**

CREATE VIEW securityskills

AS SELECT bank.security as security, robberinfo.skillid, robberinfo.nickname FROM bank

INNER JOIN accomplices

ON accomplices.bankname = bank.bankname AND accomplices.city = bank.city

INNER JOIN robberinfo

ON accomplices.robberid = robberinfo.robberid

GROUP BY bank.security, robberinfo.skillid, robberinfo.nickname

ORDER BY bank.security;

CREATE VIEW robberinfo

AS SELECT robbers.robberid, robbers.nickname as nickname, hasskills.skillid as skillid FROM robbers

INNER JOIN hasskills

ON robbers.robberid = hasskills.robberid;

**Nested**

SELECT DISTINCT bank.security, skillid, robbers.nickname FROM

(SELECT hasskills.skillid as skillid,robbers.robberid as robberid FROM hasskills

INNER JOIN robbers

ON robbers.robberid = hasskills.robberid) as skill

INNER JOIN robbers

ON robbers.robberid = skill.robberid

INNER JOIN accomplices

ON robbers.robberid = accomplices.robberid

INNER JOIN bank

ON bank.bankname = accomplices.bankname AND bank.city = accomplices.city

ORDER BY bank.security;

security | skillid | nickname

-----------+---------+-------------------

weak | 1 | Al Capone

weak | 1 | Sonny Genovese

weak | 2 | Sonny Genovese

weak | 3 | Bugsy Siegel

weak | 3 | Lepke Buchalter

weak | 5 | Vito Genovese

weak | 6 | Clyde

weak | 6 | Vito Genovese

weak | 7 | Bugsy Siegel

weak | 7 | Dutch Schulz

weak | 7 | Lepke Buchalter

weak | 9 | Al Capone

weak | 9 | Greasy Guzik

weak | 10 | Clyde

weak | 10 | Dutch Schulz

weak | 10 | Greasy Guzik

weak | 10 | Sonny Genovese

weak | 11 | Al Capone

weak | 11 | Boo Boo Hoff

weak | 11 | Clyde

weak | 12 | Vito Genovese

good | 5 | Vito Genovese

good | 6 | Vito Genovese

good | 8 | Kid Cann

good | 8 | Mickey Cohen

good | 12 | Vito Genovese

very good | 1 | Al Capone

very good | 1 | Moe Dalitz

very good | 1 | Sonny Genovese

very good | 2 | Bugsy Malone

very good | 2 | Sonny Genovese

very good | 3 | Anastazia

very good | 3 | Lepke Buchalter

very good | 7 | Lepke Buchalter

very good | 7 | Longy Zwillman

very good | 9 | Al Capone

very good | 10 | Sonny Genovese

very good | 11 | Al Capone

very good | 11 | King Solomon

excellent | 1 | Al Capone

excellent | 1 | Meyer Lansky

excellent | 1 | Sonny Genovese

excellent | 2 | Sonny Genovese

excellent | 3 | Anastazia

excellent | 3 | Bugsy Siegel

excellent | 4 | Waxey Gordon

excellent | 6 | Clyde

excellent | 7 | Bugsy Siegel

excellent | 7 | Dutch Schulz

excellent | 7 | Longy Zwillman

excellent | 7 | Lucky Luchiano

excellent | 7 | Mimmy The Mau Mau

excellent | 9 | Al Capone

excellent | 9 | Bonnie

excellent | 9 | Greasy Guzik

excellent | 10 | Clyde

excellent | 10 | Dutch Schulz

excellent | 10 | Greasy Guzik

excellent | 10 | Lucky Luchiano

excellent | 10 | Sonny Genovese

excellent | 11 | Al Capone

excellent | 11 | Boo Boo Hoff

excellent | 11 | Clyde

excellent | 11 | King Solomon

excellent | 11 | Mimmy The Mau Mau

(65 rows)

4.

**Stepwise**

CREATE VIEW bankdata

AS SELECT bank.bankname, bank.city, bank.security, robberydates.date FROM bank

INNER JOIN robberydates

ON bank.bankname = robberydates.bankname AND robberydates.city = bank.city

GROUP BY bank.bankname, bank.city, bank.security, robberydates.date

ORDER BY max(robberydates.date) DESC LIMIT 1;

CREATE VIEW robberydates

AS SELECT bank.bankname, bank.city, robberies.date as date FROM bank

INNER JOIN robberies

ON bank.bankname = robberies.bankname AND bank.city = robberies.city

INNER JOIN plans

ON bank.bankname = plans.bankname AND bank.city = plans.city

WHERE EXTRACT(year FROM robberies.date) !=2013

AND EXTRACT(year FROM plans.planneddate)=2015

GROUP BY bank.bankname, bank.city, robberies.date;

**Nested**

SELECT bank.bankname, bank.city, bank.security, robberies.date FROM robberies

INNER JOIN bank

ON robberies.bankname = bank.bankname AND robberies.city = bank.city

INNER JOIN plans

ON robberies.bankname = plans.bankname AND robberies.city = plans.city

INNER JOIN hasaccounts

ON robberies.bankname = hasaccounts.bankname AND robberies.city = hasaccounts.city

WHERE EXTRACT(year FROM robberies.date) != 2013

AND EXTRACT(year FROM plans.planneddate) = 2015

GROUP BY bank.bankname, bank.city, bank.security, robberies.date

ORDER BY count(hasaccounts.robberid), max(robberies.date) DESC LIMIT 1;

bankname | city | security | date

-----------------+---------+----------+------------

PickPocket Bank | Chicago | weak | 2011-09-21

(1 row)

5.

**Stepwise**

CREATE VIEW norobbers

AS SELECT robberies.bankname as bankname, robberies.city as city, robberies.date as date,robberies.amount as amount,count(robberies.bankname||robberies.city) as count FROM robberies

INNER JOIN accomplices

ON robberies.bankname = accomplices.bankname AND robberies.city = accomplices.city AND robberies.date = accomplices.robberydate

GROUP BY robberies.bankname, robberies.city, robberies.date, robberies.amount

ORDER BY robberies.bankname ASC;

CREATE VIEW bankaverages

AS SELECT norobbers.bankname, norobbers.city, norobbers.date, norobbers.amount/ norobbers.count as average FROM norobbers;

CREATE VIEW cityaverages

AS SELECT bankaverages.city, SUM(bankaverages.average)/COUNT(bankaverages.bankname) as averageshare FROM bankaverages

GROUP BY bankaverages.city;

**Nested**

SELECT bankaverages.city, SUM(bankaverages.average)/COUNT(bankaverages.bankname) as averageshare FROM

(SELECT norobbers.bankname, norobbers.city, norobbers.date, norobbers.amount/ norobbers.count as average FROM

(SELECT robberies.bankname as bankname, robberies.city as city, robberies.date as date,robberies.amount as amount,count(robberies.bankname||robberies.city) as count FROM robberies

INNER JOIN accomplices

ON robberies.bankname = accomplices.bankname AND robberies.city = accomplices.city AND robberies.date = accomplices.robberydate

GROUP BY robberies.bankname, robberies.city, robberies.date, robberies.amount

ORDER BY robberies.bankname ASC) as norobbers) as bankaverages

GROUP BY bankaverages.city;

city | averageshare

----------+------------------

Evanston | 7366.26025641026

Chicago | 3196.74523809524

(2 rows)