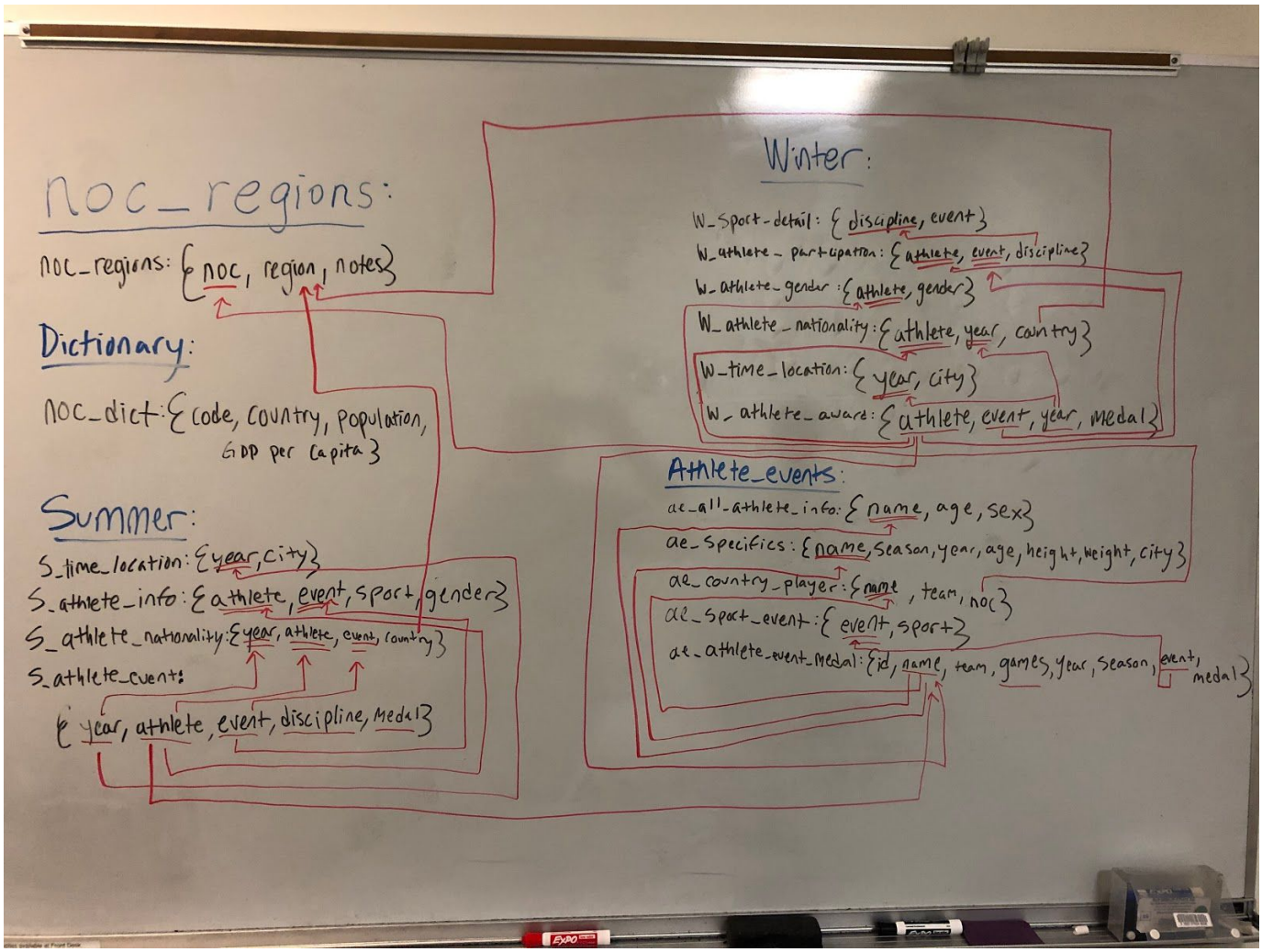
Che-wei Lin
Ryan Cummings
Professor Gupta
DSC 100 Project
March 8, 2020

1. Final Project Schema:

Note: Each relation is a Table with title corresponding to what it is called in our queries. We are also laying out the tables according to which CSV they came from just for ease of understanding how and why they are broken down:



2. Cleaning the Tables:

First we went through and read the queries to understand what we needed. Once we understood we went through and modified the tables according to that.

In dictionary:

Need to clean the star from the country column as well as palestine

```
UPDATE dictionary SET country = replace(country, '*','');
UPDATE dictionary SET country = replace(country, ', Occupied Territories','');
```

In athelete_events table:

No modifications/No changes.

In summer table:

Need to split the first and last name of name column (by comma) and append it back into the athlete name:

```
ALTER TABLE summer ADD "last_name" text;
UPDATE summer SET "last_name" = lower(split_part(athlete, ', ', 1));
ALTER TABLE summer ADD "first_name" text;
UPDATE summer SET "first_name" = lower(split_part(athlete, ', ', 2));

UPDATE summer set athlete = CONCAT(first_name, ' ') || last_name;
ALTER TABLE summer DROP COLUMN "first_name", DROP COLUMN "last_name";
```

In winter table:

Need to split the first and last name of name column (by comma) and append it back into the athlete name:

```
ALTER TABLE winter ADD "last_name" text;
UPDATE winter SET "last_name" = lower(split_part(athlete, ', ', 1));
ALTER TABLE winter ADD "first_name" text;
UPDATE winter SET "first_name" = lower(split_part(athlete, ', ', 2));

UPDATE winter set athlete = CONCAT(first_name, ' ') || last_name;
ALTER TABLE winter DROP COLUMN "first_name", DROP COLUMN "last_name";
```

In noc_regions table:

No modifications/No changes.

Checking for **Null values** within Tables:

In Summer:

Code we used to check for Nulls:

```
SELECT *
From summer
Where year is null
or city is null
or sport is null
or discipline is null
or athlete is null
or country is null
or gender is null
or event is null
or medal is null;
```

Has 4 null values (doping cases):

Through this we discovered that there was some null in summer's country column as well as a Pending placeholder for some athletes:

	year	city	sport	discipline	athlete	cou... 1	gender	event	medal
1	2012	London	Athletics	Athletics	pending	<null>	Women	1500M	Gold
2	2012	London	Weightlifting	Weightlifting	pending	<null>	Women	63KG	Gold
3	2012	London	Weightlifting	Weightlifting	pending	<null>	Men	94KG	Silver
4	2012	London	Wrestling	Wrestling Freestyle	besik kudukhov	<null>	Men	Wf 60 KG	Silver

Through later research we discovered that if an athlete name is pending then that means that they were caught using steroids and if they were caught then their country name was stripped from them, that is why this column became null. This table does not include who later won the medal because the table is not dynamic, there are null values for people who were caught doping but were never replaced with the new people who actually were awarded the medal afterwards. Therefore, we filled the na value with NA but that means that they were "caught doping" therefore later if there is a query that contains these people we can evaluate it knowing that they were stripped of their medal.

Code:

```
UPDATE summer SET country = 'NA' WHERE country IS NULL;
```

In winter:

Code we used to check for Nulls:

```
SELECT *
From winter
Where year is null
or city is null
or sport is null
or discipline is null
or athlete is null
or country is null
```

```
or gender is null
or event is null
or medal is null;
```
Has 0 null values

In athlete_events:

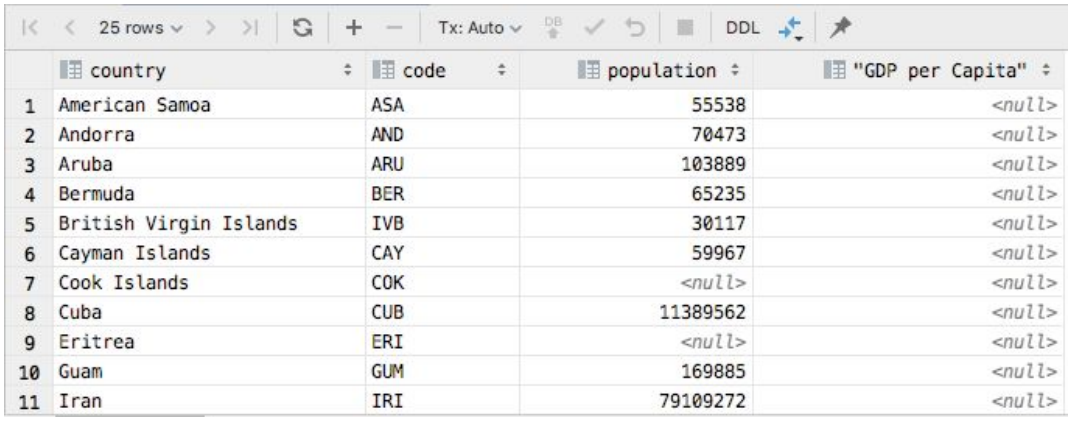
Code we used to check for Nulls:

```
SELECT *
From athlete_events
Where id is null
or name is null
or sex is null
or team is null
or noc is null
or year is null
or season is null
or city is null
or sport is null
or event is null
or medal is null
Or age is null
Or height is null
Or weight is null
```
Has 0 null values

In dictionary:

```
SELECT *
From dictionary
Where country is null
or code is null
or population is null
or "GDP per Capita" is null
```
25 Null rows:

In dictionary there are null values in both the population and in GDP per Capita, and since we have no way of being able to access the population or the GDP perCapita in this schema so we are going to just set it to -1 (Next page):



	country	code	population	"GDP per Capita"
1	American Samoa	ASA	55538	<null>
2	Andorra	AND	70473	<null>
3	Aruba	ARU	103889	<null>
4	Bermuda	BER	65235	<null>
5	British Virgin Islands	IVB	30117	<null>
6	Cayman Islands	CAY	59967	<null>
7	Cook Islands	COK	<null>	<null>
8	Cuba	CUB	11389562	<null>
9	Eritrea	ERI	<null>	<null>
10	Guam	GUM	169885	<null>
11	Iran	IRI	79109272	<null>

```
UPDATE dictionary SET population = -1 WHERE population IS NULL;
UPDATE dictionary SET "GDP per Capita" = -1 WHERE "GDP per Capita" IS NULL;
```

In noc_regions:

```
SELECT *
From noc_regions
where noc is null
or region is null
or notes is null;
```

Has 208 null values:

In this case we are going to impute with NA since the notes column contains more specific information about the country and the teams that represent said country, for example Yemen has three teams representing it. We want to keep these 3 teams separate but also a part of Yemen so we could know more specifically which team in Yemen won. We also want to keep these separate for the queries. Code:

```
UPDATE noc_regions SET notes = 'NA' WHERE notes IS NULL;
```

Checking for **Duplicate Rows** within Tables:

In Summer:

```
SELECT year,city,sport,discipline,athlete,country,gender,event,medal, count(*)

FROM summer

GROUP BY year,city,sport,discipline,athlete,country,gender,event,medal

HAVING count(*) > 1;
```

2 Duplicates:

count	year	city	sport	discipline	athlete	country	gender	event	me
2	2012	London	Badminton	Badminton	yunlei zhao	CHN	Women	Doubles	Gold
2	1980	Moscow	Hockey	Hockey	singh singh	IND	Men	Hockey	Gold

We decided since that in 1NF there cannot be duplicates we are going to drop once occurence of each of these (also professor told us to drop duplicates in the combined LE/DI):

```
with table2 as
 ( select ctid, row_number() over (partition by table1.*) as rn
   from summer as table1
 )
delete from summer as table1
using table2
where table2.rn > 1
 and table2.ctid = table1.ctid
```

In Winter:

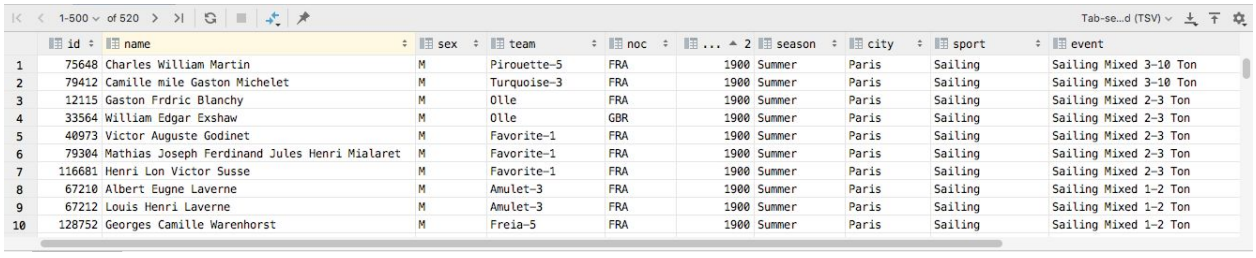
```
SELECT year,city,sport,discipline,athlete,country,gender,event,medal,count(*)
FROM winter
GROUP BY year,city,sport,discipline,athlete,country,gender,event,medal
HAVING count(*) > 1;
```

No duplicates

In athlete_events:

```
SELECT id, name, sex, age, height, weight, team, noc, games, year, season, city, sport, event,
medal,count(*)
FROM athlete_events
GROUP BY  iid, name, sex, age, height, weight, team, noc, games, year, season, city, sport, event,
medal
HAVING count(*) > 1;
```

520 Duplicates



	id	name	sex	team	noc	... 2	season	city	sport	event
1	75648	Charles William Martin	M	Pirouette-5	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 3-10 Ton
2	79412	Camille mile Gaston Michelet	M	Turquoise-3	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 3-10 Ton
3	12115	Gaston Frdric Blanchy	M	Olle	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 2-3 Ton
4	33564	William Edgar Exshaw	M	Olle	GBR	1900	Summer	Paris	Sailing	Sailing Mixed 2-3 Ton
5	40973	Victor Auguste Godinet	M	Favorite-1	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 2-3 Ton
6	79304	Mathias Joseph Ferdinand Jules Henri Mialaret	M	Favorite-1	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 2-3 Ton
7	116681	Henri Lon Victor Susse	M	Favorite-1	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 2-3 Ton
8	67210	Albert Eugne Laverne	M	Amulet-3	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 1-2 Ton
9	67212	Louis Henri Laverne	M	Amulet-3	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 1-2 Ton
10	128752	Georges Camille Warenhorst	M	Freia-5	FRA	1900	Summer	Paris	Sailing	Sailing Mixed 1-2 Ton

In 1NF we cannot have any duplicates and after clarifying on piazza (https://piazza.com/class/k52xs0u2s9pzt?cid=356) we will be dropping them. The code to drop duplicates in athlete_events is here:

```
with table2 as
( select ctid, row_number() over (partition by table1.*) as rn
 from athlete_events as table1
)
delete from athlete_events as table1
using table2
where table2.rn > 1
and table2.ctid = table1.ctid
```

In dictionary:

```
SELECT country,code,count(*)
FROM dictionary
GROUP BY country,code
HAVING count(*) > 1;
```

No duplicates

In noc_regions:

```
SELECT noc,region,notes,count(*)
FROM noc_regions
GROUP BY noc,region,notes
HAVING count(*) > 1;
```

No duplicates

3. Description of FDs and 3NF Steps:

Now that our tables are in the form of 1NF(no nulls, no atomic values, and no duplicates) we need to record the Functional Dependencies:

To determine the functional dependencies we used the PostgreSQL built-in pg_statistic on our tables. In PostgreSQL that code looks like this:

```
CREATE STATISTICS sttx(dependencies) ON insert_column_names_here from insert_table_name_here;

ANALYZE insert_table_name_here;

SELECT stxddependencies

FROM pg_statistics_ext_data
```

For dictionary we ran:

```
CREATE STATISTICS stats2(dependencies) ON country,code,population,"GDP per Capita" FROM project.dictionary;

ANALYZE project.dictionary;

SELECT stxddependencies

FROM pg_statistic_ext_data;
```

Which gave us:

{"1 => 2": 1.000000, "1 => 3": 1.000000, "1 => 4": 1.000000, "2 => 1": 1.000000, "2 => 3": 1.000000, "2 => 4": 1.000000, "3 => 1": 0.975124, "3 => 2": 0.975124, "3 => 4": 1.000000, "4 => 1": 0.875622, "4 => 2": 0.875622, "4 => 3": 0.875622, "1, 2 => 3": 1.000000, "1, 2 => 4": 1.000000, "1, 3 => 2": 1.000000, "1, 3 => 4": 1.000000, "1, 4 => 2": 1.000000, "1, 4 => 3": 1.000000, "2, 3 => 1": 1.000000, "2, 3 => 4": 1.000000, "2, 4 => 1": 1.000000, "2, 4 => 3": 1.000000, "3, 4 => 1": 0.975124, "3, 4 => 2": 0.975124, "1, 2, 3 => 4": 1.000000, "1, 2, 4 => 3": 1.000000, "1, 3, 4 => 2": 1.000000, "2, 3, 4 => 1": 1.000000}

Now this tells us if there is Functional Dependency. A 1.0 means that there is 100% a relation between the columns (which are ordered by their number mapping and not by their name), if it is not a 1.0 then it is **not** a relation.

However, this function is slightly flawed in the fact that it will try every single combination of relations that exist, and will also spit out “repeats”. A “repeat” is where a relation occurs (has 1.0) then the function just adds another column on top of it and it still

happens to have a relation. So we have gone through and scrapped the most redundant ones and came up with FD's that are:

{country} -> {code,population, GDP per Capita}

{code} -> {country, population, GDP per Capita}

With the key as {country, code}

Something we noticed about this table in particular is that it can be argued that each column determines all the other columns since each value in this whole table is uniue. However, intuitively we decided on these 2 FD since this is what we believe to be the most important for querying.

In noc_regions we ran:

```
CREATE STATISTICS stats3(dependencies) ON noc,region,notes FROM project.noc_regions;

ANALYZE project.noc_regions;

SELECT stxddependencies

FROM pg_statistic_ext_data;
```

With an output of:

{"1 => 2": 1.000000, "1 => 3": 1.000000, "2 => 1": 0.834783, "2 => 3": 0.882609, "3 => 1": 0.091304, "3 => 2": 0.091304, "1, 2 => 3": 1.000000, "1, 3 => 2": 1.000000, "2, 3 => 1": 0.934783}

Which allowed us to come up with the Functional Dependency of:

{noc} -> {region, notes}

With the key as {noc}

In summer we ran:

```
CREATE STATISTICS stats1(dependencies) ON
year,city,sport,discipline,athlete,country,gender,event,medal FROM project.summer;

ANALYZE project.summer;

SELECT stxddependencies

FROM pg_statistic_ext_data;
```

Which had way too large of an output to put here but we determined that the functional dependencies are:

{year}-> {city}

{discipline} -> {sport}

{athlete,event} -> {sport,gender}

{year,athlete,event} -> {gender}

{athlete,discipline} -> {sport}

{year,athlete,event} -> {country}

With the key as {year,discipline,athlete,event,medal}

In winter we ran:

```
CREATE STATISTICS stats4(dependencies) ON
year,city,sport,discipline,athlete,country,gender,event,medal FROM project.winter;

ANALYZE  project.winter;

SELECT stxddependencies

FROM pg_statistic_ext_data;
```

Which once again returned too large of a result to paste here but the Functional Dependencies we determined were:

{year} => {city}
{discipline} => {sport}
{athlete} => {gender}
{year, athlete} => {country}
{athlete, event} => {discipline}
{athlete, event} => {gender}
{city, athlete} => {country}
{year, athlete, event} => {city,sport,discipline,country,gender,medal}
With the key as {year,athlete,event}

In athlete_events we ran:

```
CREATE STATISTICS stats5(dependencies) ON
id,name,sex,age,height,weight,team,noc,games,year,season,city,sport,event,medal FROM
project.athlete_events;

ANALYZE  project.athlete_events;

SELECT stxddependencies

FROM pg_statistic_ext_data;
```

This was the largest output of them all and *once again* it is too large to paste here but the Functional Dependencies we determined were:

{id} => {name}
{name, age} =>{sex}
{name, team} => {noc}
{name, games} => {age,height,weight,city}

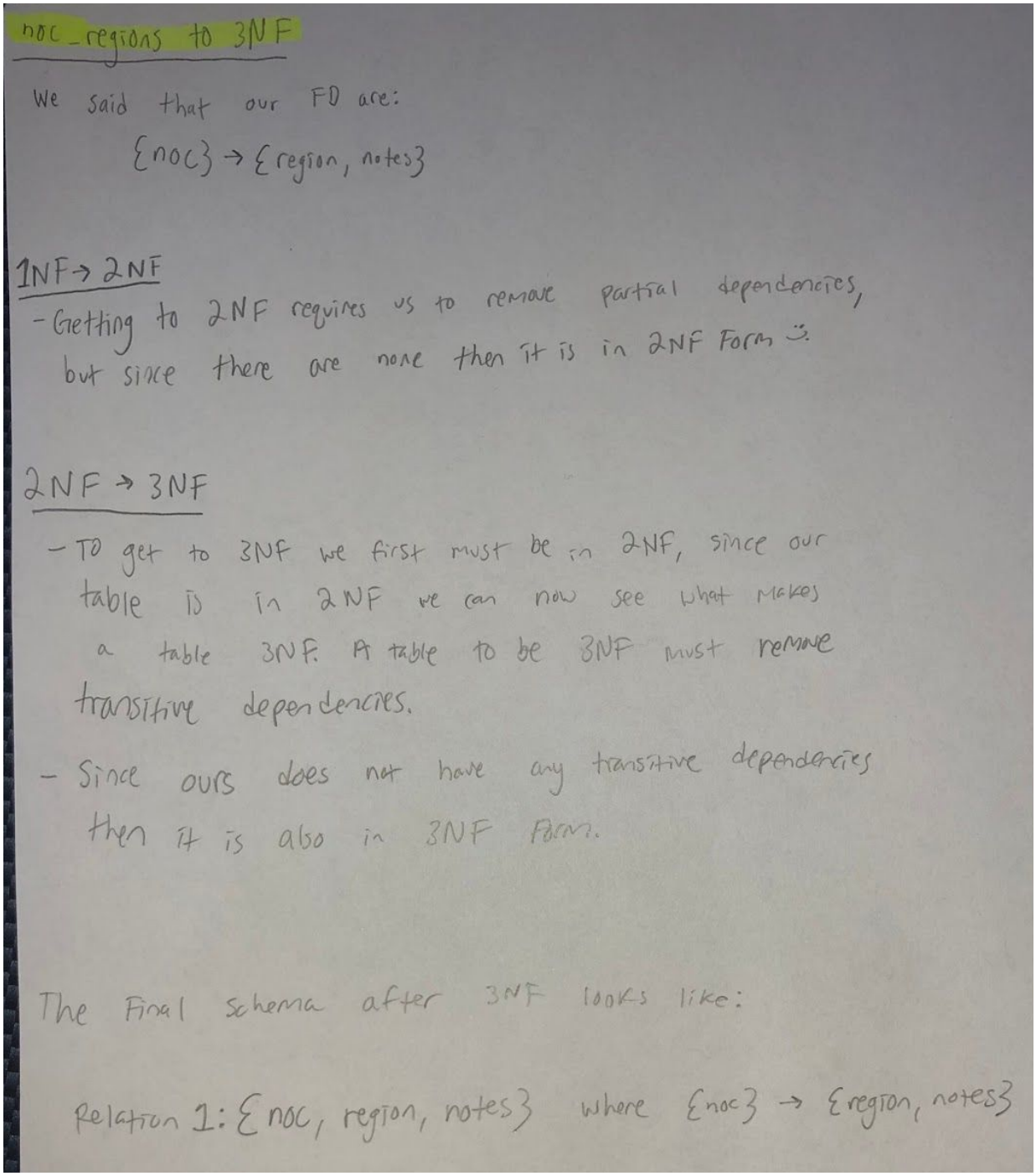
{event} => {sport}
{name, games, event} => {team,medal}
{games} => {year,season}
{year, season} => {games}
{name, games, event} => {id}
{id, games} => {age}

With the key as: {name,games,event}

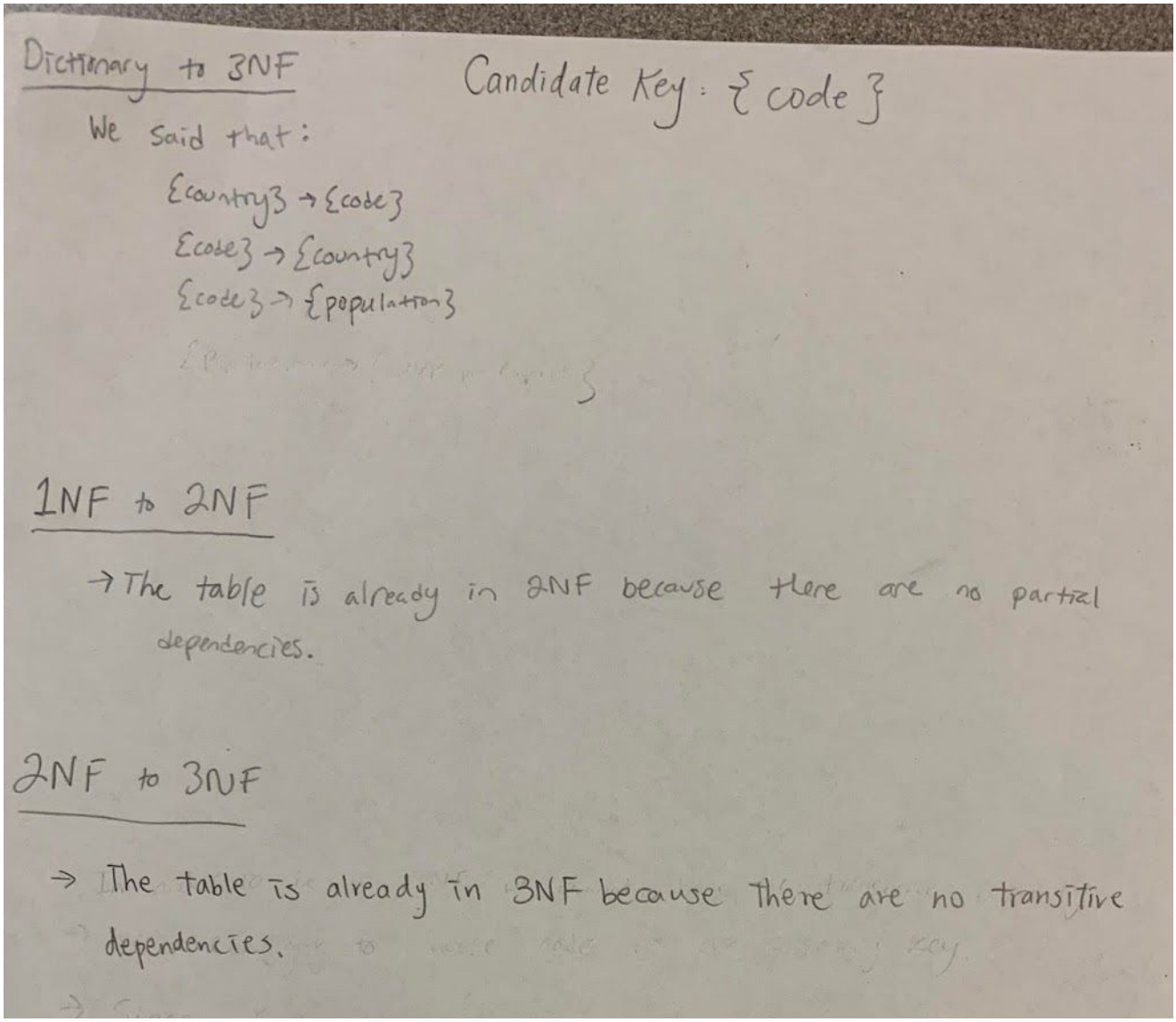
Converting to 3NF on next page:

Now that the original Tables are in First Normal Form (1NF) and we have our FD we can now convert them to 3NF:

Converting noc_regions to 3NF:



Converting Dictionary to 3NF:



Converting Summer to 3NF

Summer to 3NF

Candidate Keys = { year, athlete, event, discipline, medal }

Below is the minimum covers of FDs,

{ year } → { city }

{ athlete, event } → { sport, gender }

{ discipline } → { sport }

{ year, athlete, event } → { country }

1NF to 2NF

To transform the table into 2NF, we need to check if every FD is a partial dependency

- { year } → { city } is a partial dependency, so we break these columns down into

  Relation 1 = { year, city }

- { athlete, event } → { sport, gender } is a partial dependency.

  Relation 2 = { athlete, event, sport, gender }

- { year, athlete, event } → { country } is a partial dependency.

  Relation 3 = { year, athlete, event, country }

  Relation 4 = { year, athlete, event, discipline, medal }

- Now each relation is in 2NF

2NF to 3NF

- To transform the table into 3NF, we need to remove any transitive dependencies.
- Since there's no transitive dependencies, our final tables in 3NF are :

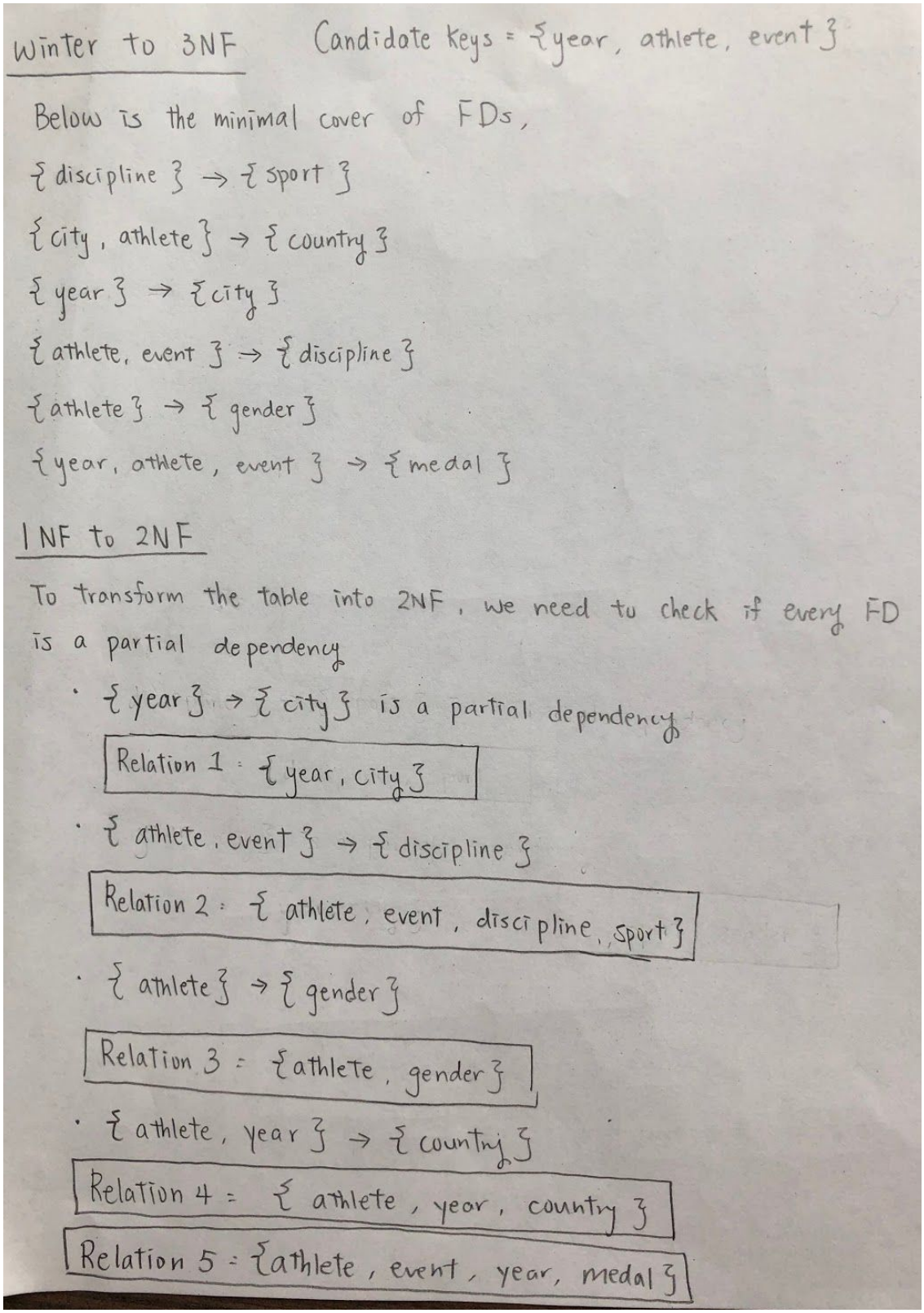
olympic
Relation 1 = { year, city }

athlete
Relation 2 = { athlete, event, sport, gender }

where they come from
Relation 3 = { year, athlete, event, country }

medal
Relation 4 = { year, athlete, event, discipline, medal }

Converting Winter to 3NF:



Winter to 3NF Candidate keys = {year, athlete, event}

Below is the minimal cover of FDs,

{discipline} → {sport}

{city, athlete} → {country}

{year} → {city}

{athlete, event} → {discipline}

{athlete} → {gender}

{year, athlete, event} → {medal}

1NF to 2NF

To transform the table into 2NF, we need to check if every FD is a partial dependency

- {year} → {city} is a partial dependency

  Relation 1 = {year, city}

- {athlete, event} → {discipline}

  Relation 2 = {athlete, event, discipline, sport}

- {athlete} → {gender}

  Relation 3 = {athlete, gender}

- {athlete, year} → {country}

  Relation 4 = {athlete, year, country}

  Relation 5 = {athlete, event, year, medal}

2NF to 3NF

To transform the table into 3NF, we need to remove any transitive dependencies

- Since {athlete, event} → {discipline} → {sport}, we can break {athlete, event, discipline, sport} into {athlete, event, discipline} and {discipline sport}
- So the final schema is:

Relation 1 = {discipline, sport}

Relation 2 = {athlete, event, discipline}

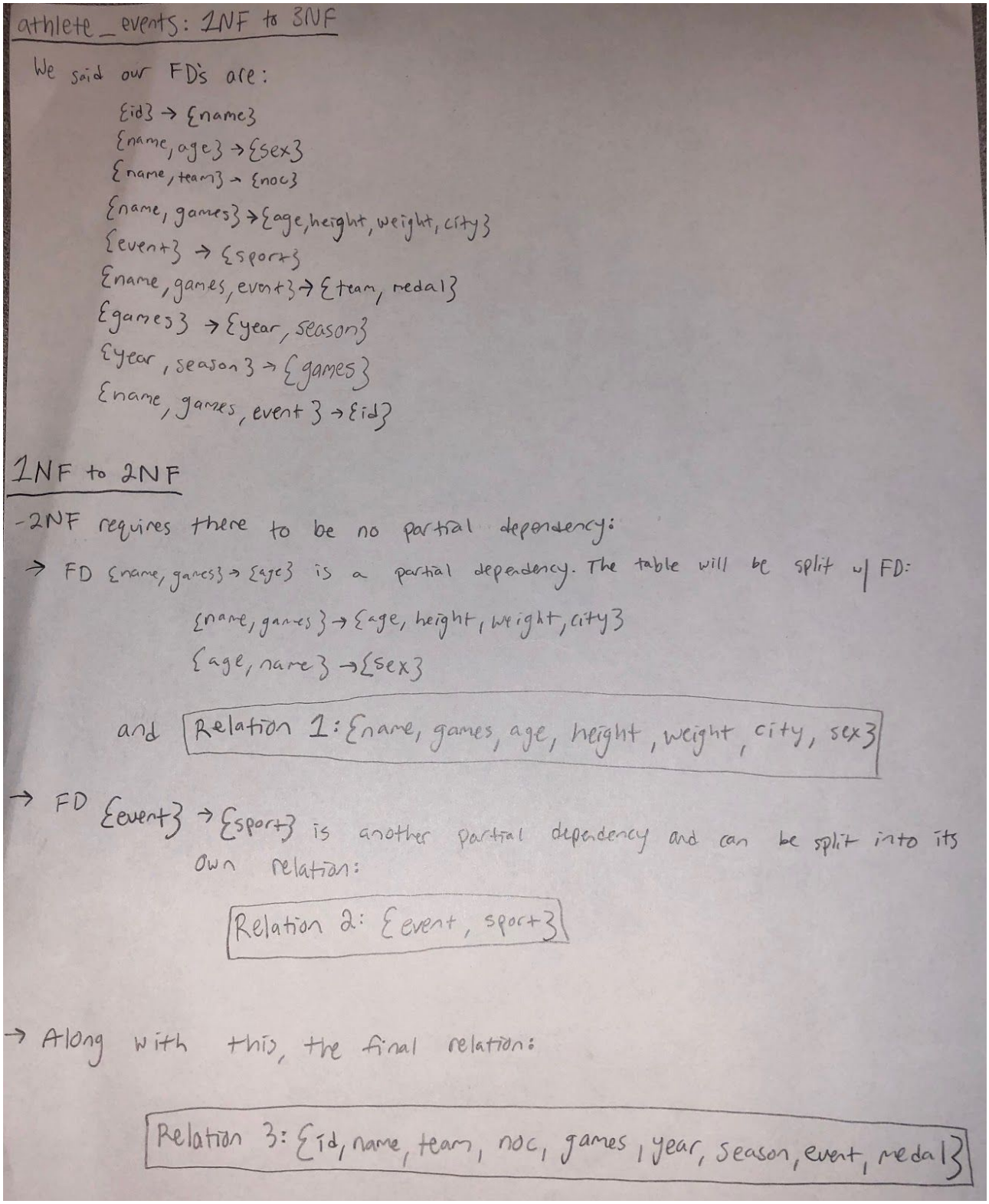
Relation 3 = {athlete, gender}

Relation 4 = {athlete, year, county}

Relation 5 = {year, city}

Relation 6 = {athlete, event, year, medal}

Converting athlete_evets to 3NF:



athlete_events: 1NF to 3NF

We said our FD's are:

{id} → {name}
{name, age} → {sex}
{name, team} → {noc}
{name, games} → {age, height, weight, city}
{event} → {sport}
{name, games, event} → {team, medal}
{games} → {year, season}
{year, season} → {games}
{name, games, event} → {id}

1NF to 2NF

-2NF requires there to be no partial dependency:

→ FD {name, games} → {age} is a partial dependency. The table will be split w/ FD:

{name, games} → {age, height, weight, city}
{age, name} → {sex}

and Relation 1: {name, games, age, height, weight, city, sex}

→ FD {event} → {sport} is another partial dependency and can be split into its own relation:

Relation 2: {event, sport}

→ Along with this, the final relation:

Relation 3: {id, name, team, noc, games, year, season, event, medal}

2NF to 3NF

- 3NF requires no transitive property within the relations

→ Since {name, games} → {age} → {sex} it is a transitive dependency. Therefore,

Relation 1: {name, age, sex}

→ Since {games} → {year, season}, and {name, games} → {age, height, weight, city}
then we can form a new relation which further breaks down R1 from 2NF:

Relation 2: {name, season, year, age, height, weight, city}

→ Since {event, name, games} → {team, medal, id} and {name, team} → {NOC} it is a transitive dependency from R.3 from 2NF, Therefore:

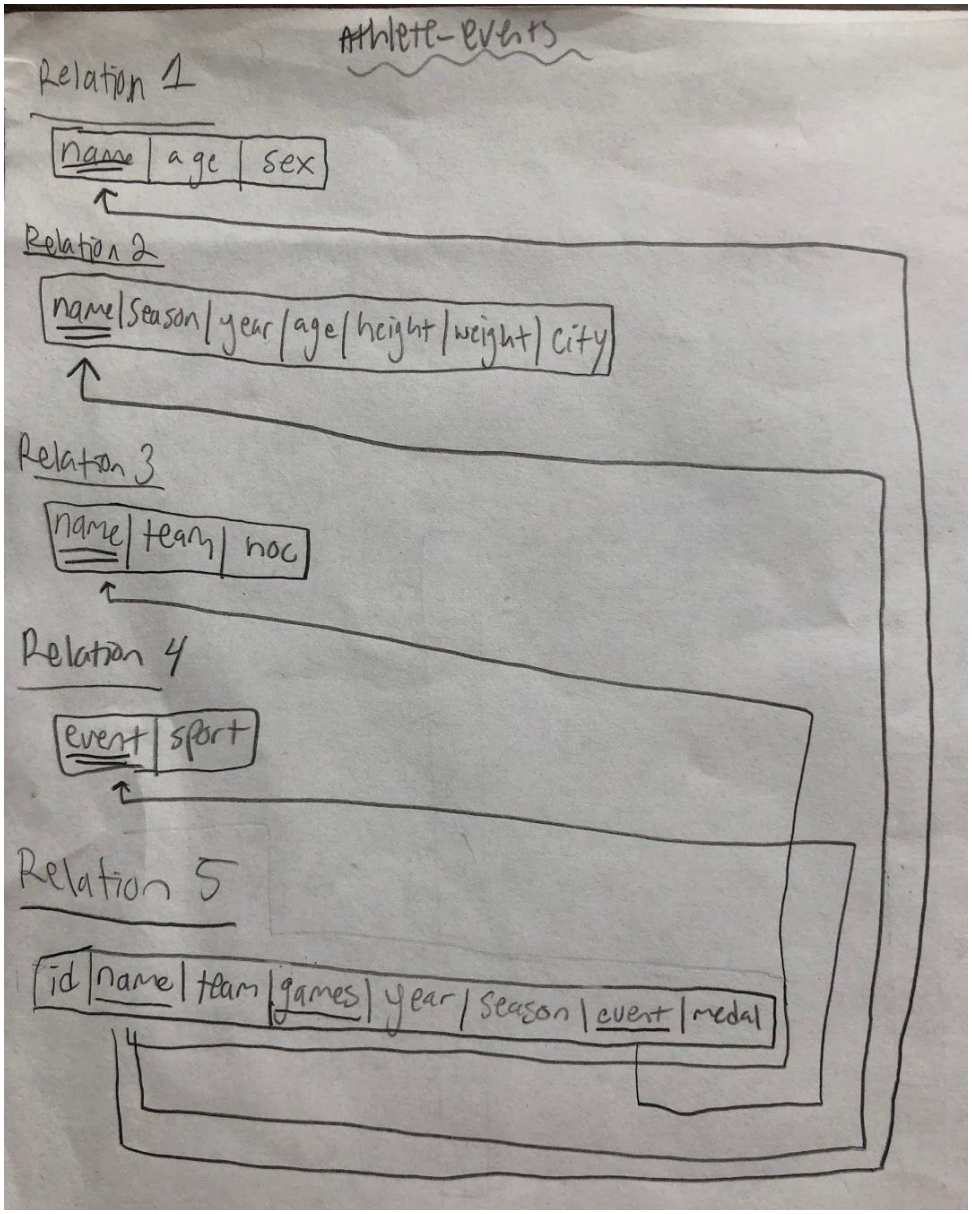
Relation 3: {name, team, noc}

→ {event} → {sport} remains a relation:

Relation 4: {event, sport}

→ And, the final relation is:

Relation 5: {id, name, team, games, year, season, event, medal}


Athlete-events
Relation 1
name | age | sex
Relation 2
name | season | year | age | height | weight | city
Relation 3
name | team | noc
Relation 4
event | sport
Relation 5
id | name | team | games | year | season | event | medal

Now that we got all the tables into the 3NF form we need to repopulate the new Tables with them, the basic code layout to do this is:

```
INSERT INTO new_table
SELECT DISTINCT col,col2,...coln
FROM old_table
```

The tables that are made here are the tables that are in our Final Project Schema as seen in the top of this project:

In dictionary we ran:

```
CREATE TABLE project.noc_dict(country varchar(500), code char(3), population int,"GDP Per Capita" numeric);
INSERT INTO project.noc_dict
SELECT country,code,population
FROM project.dictionary;
```

In noc_regions we kept it the same since it is in 3NF form:

```
CREATE TABLE noc_region(noc char(3), region varchar(500), notes varchar(500));
INSERT INTO project.noc_region
SELECT noc,region,notes
FROM project.noc_regions;
```

In summer we ran:

```
create table project.S_time_location
(
  year int,
  city varchar(500)
);
INSERT INTO project.S_time_location
SELECT year,city
FROM project.summer;

create table project.S_athlete_info
(
  athlete varchar(500),
  event varchar(500),
  sport varchar(500),
  gender varchar(10)
);
```

```
INSERT INTO project.s_athlete_info
SELECT athlete,event,sport,gender
FROM project.summer;

create table project.S_athlete_nationality
(
  year int,
  athlete varchar(500),
  event varchar(500),
  country varchar(300)
);
INSERT INTO project.s_athlete_info
SELECT athlete,event,sport,gender
FROM project.summer;


create table project.S_athlete_event
(
  year int,
  athlete varchar(500),
  event varchar(500),
  discipline  varchar(300),
  medal varchar(10)
);
INSERT INTO project.S_athlete_event
SELECT year,athlete,event,discipline,medal
FROM project.summer;
```

In winter we ran:

```
create table project.W_sport_detail
(
  discipline varchar(500),
  event varchar(500)
);
INSERT INTO project.w_sport_detail
SELECT discipline,sport
FROM project.winter;

create table project.W_athlete_participation
(
  athlete varchar(500),
  event varchar(500),
  discipline varchar(500)
);
INSERT INTO project.w_athlete_participation
SELECT athlete,event,discipline
FROM project.winter;
```

```
create table project.W_athlete_gender
(
  athlete varchar(500),
  gender varchar(5)
);
INSERT INTO project.w_athlete_gender
SELECT athlete,gender
FROM project.winter;

create table project.W_athlete_nationality
(
  athlete varchar(500),
  year int,
  country char(3)
);
INSERT INTO project.w_athlete_nationality
SELECT athlete,year,country
FROM project.winter;


create table project.W_time_location
(
  year int,
  city varchar(500)
);

INSERT INTO project.w_time_location
SELECT year,city
FROM project.winter;


create table project.W_athlete_award
(
  athlete varchar(500),
  event varchar(500),
  year int,
  medal varchar(10)
);
INSERT INTO project.w_athlete_award
SELECT athlete,event,year,medal
FROM project.winter;
```

In athlete_events we ran:

```
create table project.AE_all_athlete_info
(
  name varchar(300),
  age varchar(30),
  sex char
```

```
);
INSERT INTO project.ae_all_athlete_info
SELECT name,age,sex
FROM project.athlete_events;


create table project.ae_specifics
(
  name varchar(300),
  season varchar(300),
  year int,
  age varchar(30),
  height varchar(30),
  weight varchar(30),
  city varchar(50)
);
INSERT INTO project.ae_specifics
SELECT name,season,year,age,height,weight,city
FROM project.athlete_events;


create table project.ae_country_player
(
  name varchar(300),
  team varchar(50),
  noc char(3)
);
INSERT INTO project.ae_country_player
SELECT name, team, noc
FROM project.athlete_events;


create table project.ae_sport_event
(
  event varchar(300),
  sport varchar(50)
);
INSERT INTO project.ae_sport_event
SELECT event,sport
FROM project.athlete_events;


create table project.ae_athlete_event_medal
(
  id varchar(10),
  name varchar(100),
  team varchar(50),
  games varchar(30),
  year int,
  season varchar(10),
```

```
  event varchar(300),
  medal varchar(10)
);
INSERT INTO project.ae_athlete_event_medal
SELECT id,name,team,games,year,season,event,medal
FROM project.athlete_events;
```

4. QUERIES and Results:

Query1:

Code:

```
(with temp_table as
(select noc_region.noc, noc_region.region, count(DISTINCT ae_country_player.name)
from project.ae_country_player, project.ae_athlete_event_medal, project.noc_region
where ae_athlete_event_medal.name = ae_country_player.name and ae_country_player.noc =
noc_region.noc and ae_athlete_event_medal.year = 1992
group by noc_region.noc, noc_region.region)
select noc_region.noc, noc_region.region, coalesce(temp_table.count, 0) as count
from temp_table right outer join project.noc_region on noc_region.noc = temp_table.noc
order by count DESC
)
```

Number of Rows: 230

Screenshot of output:

230 rows

	noc	region	count
1	USA	USA	708
2	EUN	Russia	603
3	GER	Germany	582
4	FRA	France	451
5	ESP	Spain	446
6	GBR	UK	425
7	ITA	Italy	416
8	CAN	Canada	405
9	JPN	Japan	316
10	AUS	Australia	311

Query2:

Code:

```
SELECT DISTINCT team
from project.ae_athlete_event_medal
where year = 2010 and (event LIKE '%curling%' or event LIKE '%Curling%')
```

```
group by team;
```

Number of Rows: 12

Screenshot of Output:

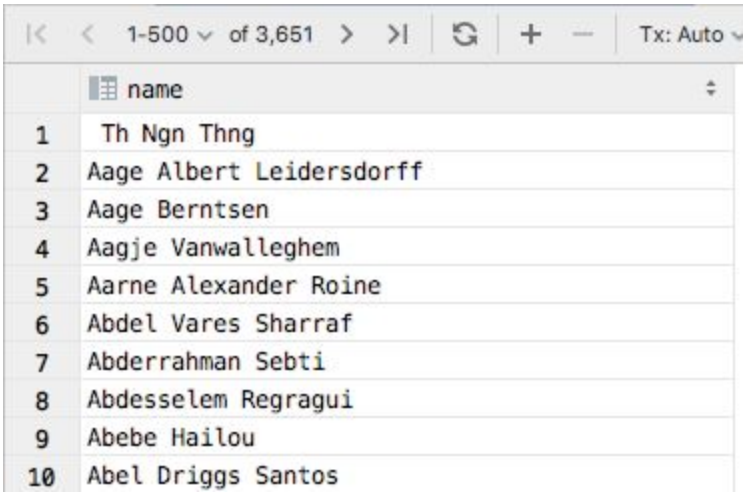
Query3:

Code:

```
select distinct name
from project.ae_athlete_event_medal
where year >= 1900
group by name, games
HAVING count(distinct event) > 4
```

Number of Rows: 3651

Screenshot of Output:



Query4:

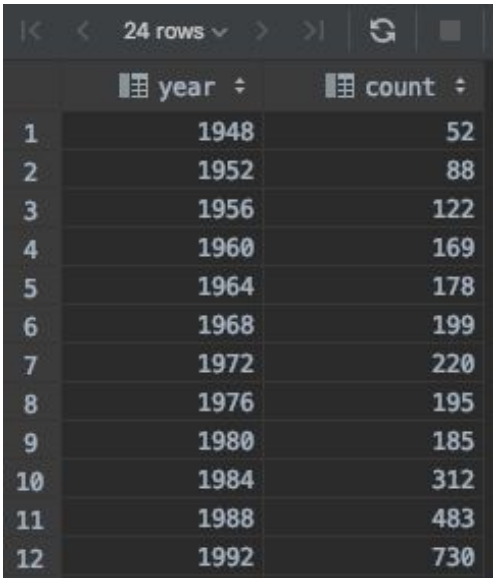
Code:

```
with merged_table as(
        with year_table as (select name,event,year
        from project.ae_athlete_event_medal
        where year > 1940)
          ,
        main_table as(
        select name, event, count(event)
        from project.ae_athlete_event_medal
        where year > 1940 group by name,event having count(event) >3)

        Select main_table.name,main_table.event, year_table.year
        from year_table
        RIGHT JOIN main_table
        ON year_table.name = main_table.name AND year_table.event = main_table.event)
select year, count(distinct name)
from merged_table
group by year;
```

Number of Rows: 24

Screenshot of Output:



24 rows

	year	count
1	1948	52
2	1952	88
3	1956	122
4	1960	169
5	1964	178
6	1968	199
7	1972	220
8	1976	195
9	1980	185
10	1984	312
11	1988	483
12	1992	730

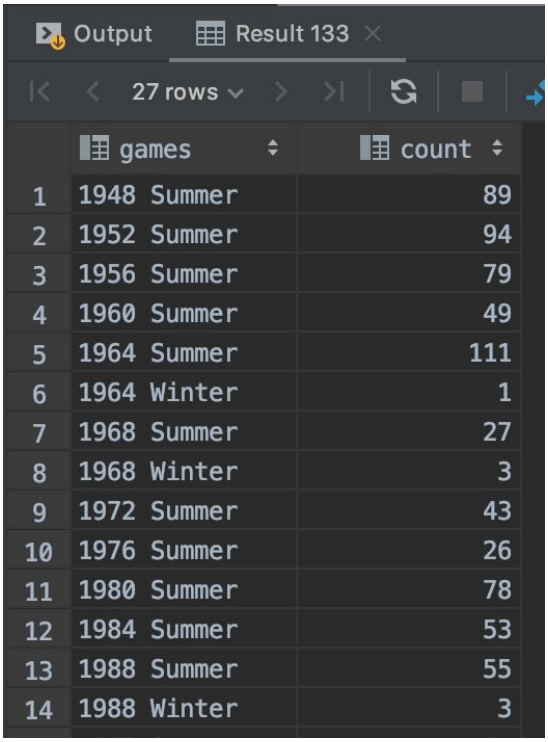
Query5:

Code:

```
select Distinct games,count(name) from project.ae_athlete_event_medal where (team LIKE
'India%' or team LIKE 'India-1%' or team LIKE 'India-2%')
and year >= 1942 group by games;
```

Number of Rows: 27

Screenshot of Output:



Output | Result 133 | 27 rows

	games	count
1	1948 Summer	89
2	1952 Summer	94
3	1956 Summer	79
4	1960 Summer	49
5	1964 Summer	111
6	1964 Winter	1
7	1968 Summer	27
8	1968 Winter	3
9	1972 Summer	43
10	1976 Summer	26
11	1980 Summer	78
12	1984 Summer	53
13	1988 Summer	55
14	1988 Winter	3

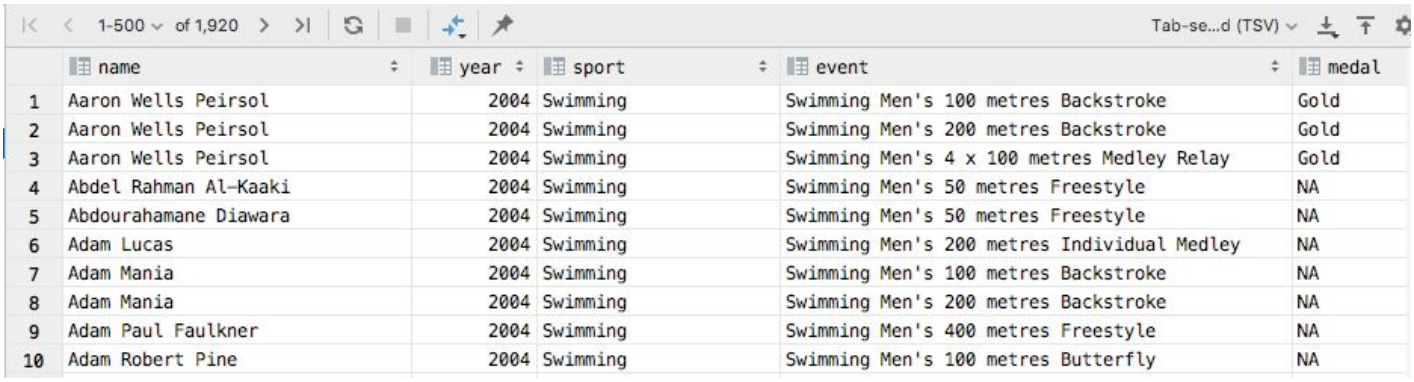
Query6:

Code:

```
(select DISTINCT name, year, sport, ae_sport_event.event, medal
from project.ae_athlete_event_medal LEFT OUTER JOIN project.ae_sport_event ON
(ae_athlete_event_medal.event = ae_sport_event.event)
where year = 2004 AND sport LIKE '%Swimming%')
UNION
(Select athlete as name, year, discipline as sport, event, medal
from project.s_athlete_event
where (discipline like '%Swimming%' or discipline like '%swimming%') and year = 2004)
```

Number of rows in the output: 1,920

Screenshot of output:



1-500 of 1,920 | Tab-se...d (TSV)

	name	year	sport	event	medal
1	Aaron Wells Peirsol	2004	Swimming	Swimming Men's 100 metres Backstroke	Gold
2	Aaron Wells Peirsol	2004	Swimming	Swimming Men's 200 metres Backstroke	Gold
3	Aaron Wells Peirsol	2004	Swimming	Swimming Men's 4 x 100 metres Medley Relay	Gold
4	Abdel Rahman Al-Kaaki	2004	Swimming	Swimming Men's 50 metres Freestyle	NA
5	Abdourahamane Diawara	2004	Swimming	Swimming Men's 50 metres Freestyle	NA
6	Adam Lucas	2004	Swimming	Swimming Men's 200 metres Individual Medley	NA
7	Adam Mania	2004	Swimming	Swimming Men's 100 metres Backstroke	NA
8	Adam Mania	2004	Swimming	Swimming Men's 200 metres Backstroke	NA
9	Adam Paul Faulkner	2004	Swimming	Swimming Men's 400 metres Freestyle	NA
10	Adam Robert Pine	2004	Swimming	Swimming Men's 100 metres Butterfly	NA

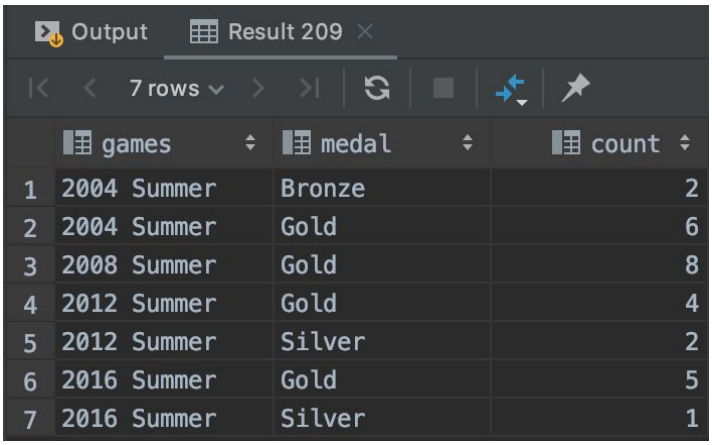
Query7:

Code:

```
select games,medal, count(*)
from project.ae_athlete_event_medal
WHERE name = 'Michael Fred Phelps, II' and medal != 'NA'
group by games,medal
order by games ASC;
```

Number of Rows in Output: 7

Screenshot of Output:



	games	medal	count
1	2004 Summer	Bronze	2
2	2004 Summer	Gold	6
3	2008 Summer	Gold	8
4	2012 Summer	Gold	4
5	2012 Summer	Silver	2
6	2016 Summer	Gold	5
7	2016 Summer	Silver	1

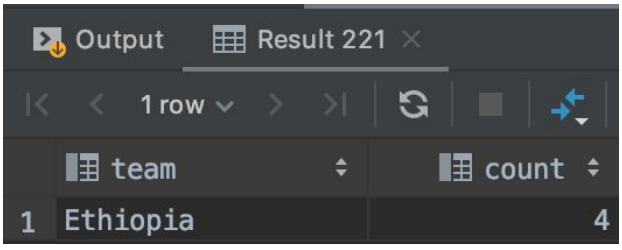
Query8:

Code:

```
select team,count(*)
from project.ae_athlete_event_medal
where (event like '%Men''s Marathon%') and medal = 'Gold'
group by team
order by count DESC
LIMIT 1;
```

Number of Rows: 1

Screenshot of Output:



	team	count
1	Ethiopia	4

Query9:

Code:

```
CREATE TABLE q9_table AS SELECT * FROM project.ae_athlete_event_medal;
```

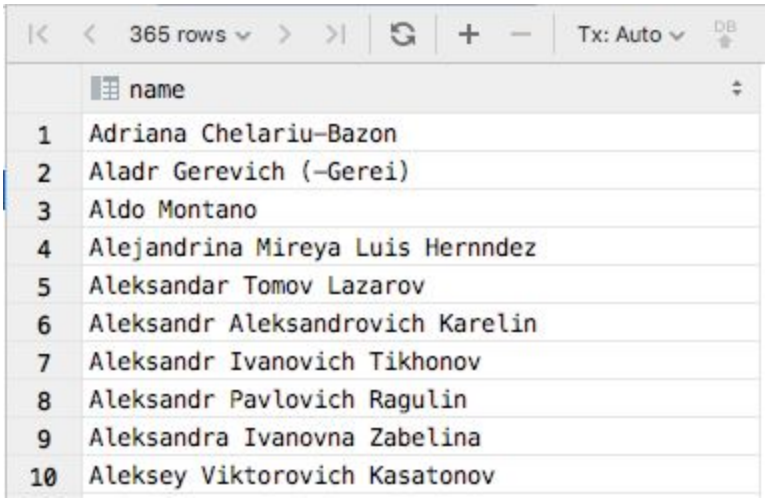
```
UPDATE q9_table
set
  medal = REPLACE(REPLACE(REPLACE(REPLACE(medal,'NA','0'),'Bronze','1'),'Silver','2'),'Gold','3')

CREATE TABLE q9_map1 as (
select name,year,event,medal,lag(medal,1) over (partition by name,event order by year ASC)
previous_event_medal,
    lag(medal,2) over (partition by name,event order by year ASC) previous_2_event_medal
from q9_table); --null means they did not participate

delete from q9_map1
where (q9_map1.medal = '0' and q9_map1.previous_event_medal = '0' and
q9_map1.previous_2_event_medal = '0');
delete from q9_map1
where (medal is null or previous_event_medal is null or previous_2_event_medal is null)

select distinct name
from q9_map1
where
(medal >= previous_event_medal) and (previous_event_medal >= previous_2_event_medal) and
(medal!='0') and (previous_event_medal!='0')
 and (previous_2_event_medal != '0')
 and year > 1940
```

Number of Rows: 365
Screenshot of Output:



365 rows | Tx: Auto

	name
1	Adriana Chelariu-Bazon
2	Aladr Gerevich (-Gerei)
3	Aldo Montano
4	Alejandrina Mireya Luis Hernndez
5	Aleksandar Tomov Lazarov
6	Aleksandr Aleksandrovich Karelin
7	Aleksandr Ivanovich Tikhonov
8	Aleksandr Pavlovich Ragulin
9	Aleksandra Ivanovna Zabelina
10	Aleksey Viktorovich Kasatonov

REFERENCES:

https://stackoverflow.com/questions/12310986/combine-two-columns-and-add-into-one-new-column

https://stackoverflow.com/questions/8584967/split-comma-separated-column-data-into-additional-columns

https://dba.stackexchange.com/questions/27410/transform-all-columns-records-to-lowercase

https://stackoverflow.com/questions/50479880/replace-null-in-my-table-with-some-value-in-postgresql
https://stackoverflow.com/questions/35445673/select-all-duplicates-from-a-table-in-postgres

https://en.wikipedia.org/wiki/Sailing_at_the_1900_Summer_Olympics_%E2%80%93_2_to_3_ton

https://en.wikipedia.org/wiki/Fr%C3%A9d%C3%A9ric_Blanchy