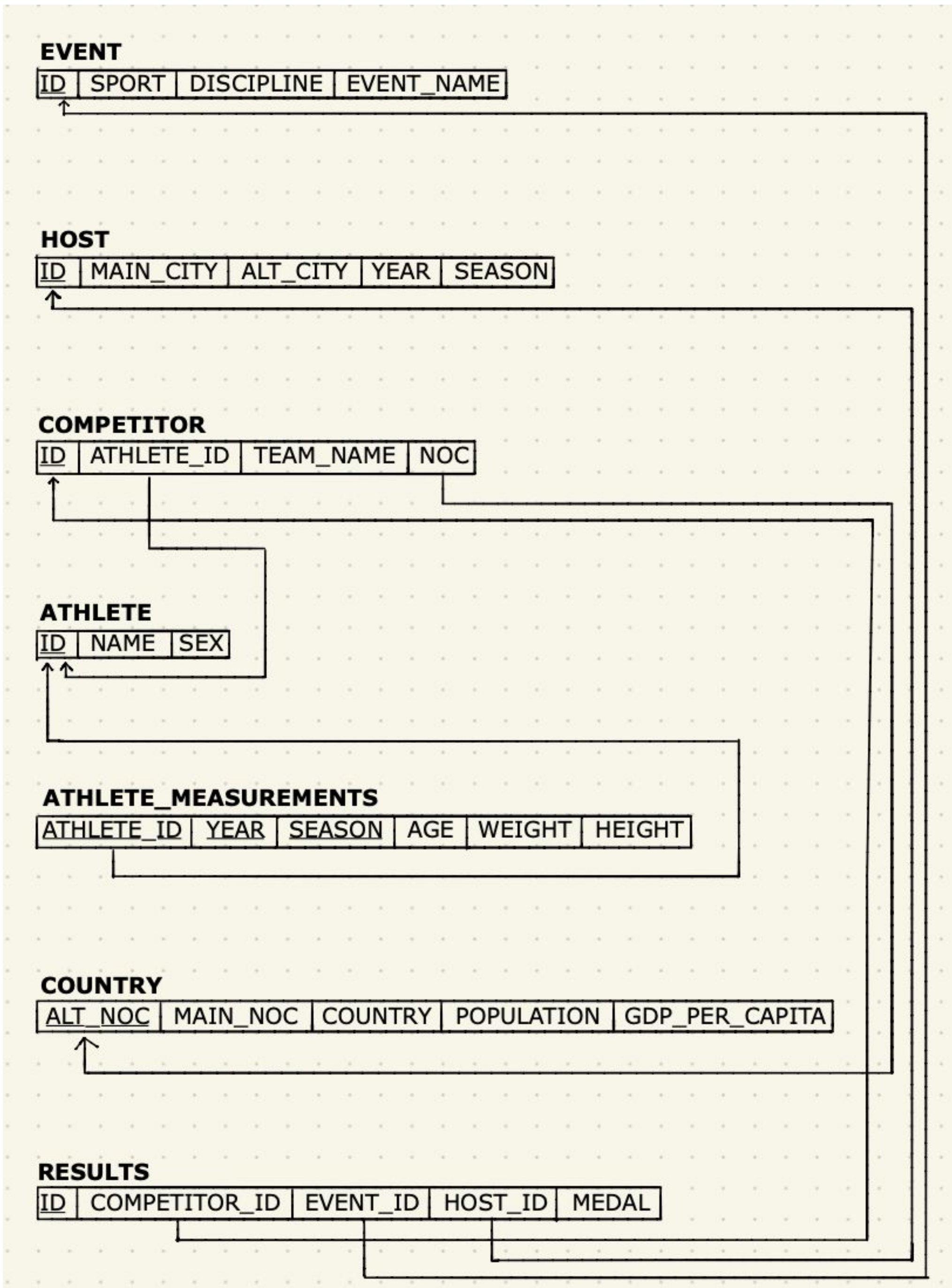
Jacob Moul - A13548393
Joshua Castro - A13575128
DSC100 Final Project

SCHEMA DESIGN

Fig.1: Referential integrity constraints displayed on our OLYMPIC TABLES relational database schema.



Jacob Moul - A13548393
Joshua Castro - A13575128

DESCRIPTION OF CLEANING/PROCESSING

When importing the data, we made sure to update 'NA' values as NULL in *ae* and set the data types correctly in all tables. We also wanted to standardize units by setting instances such as 'kilometre' to 'KM', 'meter' to 'M', and similar instances. We achieved this in bulk by using *regexp_replace* on the whole table, but for more specific cases, we looked at most records in *athlete_events.event* while cross-referencing *(summer|winter).event* for better consistency across tables.

To clean the *events* column of *athlete_events (ae)* and *summer/winter (s|w)* we noticed that *ae.event* looked to be a composite attribute in the formatting equivalent of (*s|w).discipline* + *(s|w)gender* + *(s|w).event*, so we proceeded to work to split *ae.event*. We accounted for instances in which the *gender* equivalent in *ae.event* was 'mixed' or not present, and successfully retrieved *ae.discipline* and *ae.event*. We did not retrieve or format *ae.gender* based on the values of the former *ae.event* because of the inconsistencies and because of the already existing *ae.gender* column.

Once we split *ae.event* into *ae.discipline* and *ae.event*, we took steps to make matching *event* titles consistent across the *athlete_events* and *summer/winter* tables. We accomplished this by using transaction blocks to update records of *ae.event* and *(s|w).event* for each distinct *sport* record. Examples of such updates include standardizing units, standardizing *event* titles, replacing weight ranges into weight classes (for example, changing '56 - 60KG' to 'Lightweight' for *Men's Boxing* records), and removing extraneous text.

In order to match athletes between *ae* and *summer/winter*, we used blocking and array subsetting to attempt to match up athletes. For example, we compared athletes between the two tables who competed for the same country in the same year, and had the same gender, then split up their names into arrays by whitespace and checked for containment both ways to identify potential matches. We then manually inspected one to many matches to try to identify extraneous matches. Undoubtedly, many incorrect matches were not identified because of the extremely tedious nature of the task, resulting in some athletes not being included in the final database, but by our estimation, only a fraction of a percent of the total athletes were lost.

Description of FDs and 3NF Steps

Functional Dependencies

We found the following functional dependencies and attribute closures:

athlete_events(id, name, sex, age, height, weight, team, noc, games, year, season, city, sport, event, medal)

- id -> name, sex
- id, year, season -> age, weight, height
- games -> year, season
- id, event -> sport
  - event -> sport
- id, games -> noc
- id, team -> noc
- id, event, season, year -> year, city
  - id, year, season -> city
- city, year -> games, season
- event, year -> city

attribute closure for **athlete_events:**

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

noc_regions(noc, region, notes)

- noc -> region
- noc -> notes

Attribute closures for **noc_regions:**

- {noc}+ = {noc, region, notes}

summer(year, city, sport, discipline, athlete, country, gender, event)

- event, year, athlete -> gender, medal, country
    - (athlete, year -> gender)
    - (athlete, year -> country)
- year -> city
- athlete, year -> country, gender
- discipline -> sport

attribute closures for **summer:**

- {discipline}+ = {discipline, sport}
- {year}+ = {year, city}
- {athlete, year}+ = {athlete, year, city, country, gender}
- {athlete, year, event}+ = {athlete, year, event, city, country, gender, medal}
- {athlete, year, event, discipline}+ = {athlete, year, event, city, country, gender, medal, discipline, sport}

winter(year, city, sport, discipline, athlete, country, gender, event)

- event, year, athlete -> gender, medal, country
    - (athlete, year -> gender)
    - (athlete, year -> country)
- year -> city
- athlete, year -> country, gender
- discipline -> sport

attribute closures for **winter:**

- {discipline}+ = {discipline, sport}
- {year}+ = {year, city}
- {athlete, year}+ = {athlete, year,city, country, gender}
- {athlete, year, event}+ = {athlete, year, event, city, country, gender, medal}
- {athlete, year, event, discipline}+ = {athlete, year, event, city, country, gender, medal, discipline, sport}

dictionary(country, code, population, gdp_per_capita)

- code -> country, population, gdp_per_capita

Attribute closures for **dictionary:**

- {code}+ = {code, country, population, gdp_per_capita}

Achieving 3NF

We performed the following steps to achieve a 3NF schema:

1. Identified the functional dependencies and candidate keys.
2. Using functional dependencies determined above, identified which were Fully Functional Dependent and which were Transitive Dependent.
    a. examples from athlete_events
        i. *name, sex* are fully functionally dependent on *id*
        ii. *age*, *weight*, *height* are fully functionally dependent on *id, year, season*
    b. examples from summer, winter
        i. *sport* is fully functionally dependent on *discipline*
        ii. *city* is fully functionally dependent on *year*
3. Separated Partially Functional Dependencies and Transitive Functional Dependencies into their own relations.
4. Verified the fully functional nature of prime and non-prime attributes in our resulting relations.
5. Adjusted accordingly.
    a. For example, we initially determined that {athlete_id} -> {name, sex, age, height, weight}, but after creating a relation with those attributes, we determined that in fact *season* and *year* were necessary to determine *age, weight, and height*, and created another relation to reflect that.

Justification of 3NF

After creating these relations, we performed step 4 above, and verified our dependencies in each table. We used the statement structure

```
SELECT attr1[, attr2,...]
FROM relation
GROUP BY attr1[, attr2,...]
HAVING COUNT(DISTINCT attrN)>1;
```

To see if `attrN` was functionally dependent upon `attr1[, attr2,...]`, where the FD was valid if no tuples were returned. We checked subsets of prime attributes in each relation for each non-prime attribute and checked non-prime attributes against each other to verify the Fully Functional and non-Transitive nature of our FDs.

Jacob Moul - A13548393
Joshua Castro - A13575128
DSC100 Final Project

Query 1

SQL

```
WITH in_barca(noc, num_competitors) as
    (
        SELECT noc, count(distinct competitor_id)
        FROM competitor c
        JOIN results r ON c.id = r.competitor_id
        JOIN host h ON r.host_id = h.id
        WHERE (h.main_city ilike '%barcelona%' or h.alt_city ilike
'%barcelona%') and h.year = 1992
        GROUP BY noc
    ),
    not_in_barca(noc, num_competitors) as
    (
        SELECT main_noc, 0
        FROM country ctry
        WHERE ctry.alt_noc NOT IN (SELECT noc from in_barca)
        GROUP BY main_noc
    )

SELECT country, num_competitors
FROM in_barca, country
WHERE noc=alt_noc

UNION

SELECT country, num_competitors
FROM not_in_barca, country
WHERE noc=alt_noc
order by num_competitors DESC;
```

NUMBER OF ROWS

231

SCREENSHOT

	country	num_competitors
1	United States	558
2	Russia	475
3	Germany	471
4	Spain	428
5	United Kingdom	374
6	France	342
7	Italy	309
8	Canada	296
9	Australia	287
10	Japan	256

Jacob Moul - A13548393
Joshua Castro - A13575128

Query 2

SQL

```
SELECT distinct c2.country
FROM competitor c
JOIN results r ON c.id = r.competitor_id
JOIN event e ON e.id = r.event_id
JOIN host h ON h.id = r.host_id
JOIN country c2 on c.noc = c2.alt_noc
WHERE e.sport ilike 'Curling' and (h.main_city ilike 'vancouver' or
                                   h.alt_city ilike 'vancouver') and
      h.year = 2010;
```

NUMBER OF ROWS

12

SCREENSHOT

	country
1	Canada
2	China
3	Denmark
4	France
5	Germany
6	Japan
7	Norway
8	Russia
9	Sweden
10	Switzerland

Query 3

SQL

```
SELECT a.name
FROM athlete a
WHERE a.id IN (SELECT a.id
                FROM athlete a
                JOIN competitor c ON a.id = c.athlete_id
                JOIN results r ON c.id = r.competitor_id
                JOIN host h ON h.id = r.host_id
                WHERE h.year > 1900
                GROUP BY a.id, host_id
                having count(distinct event_id) > 4)
ORDER BY name;
```

NUMBER OF ROWS

3469

SCREENSHOT

	name
1	Aage Albert Leidersdorff
2	Aage Berntsen
3	Aagje Vanwalleghem
4	Aarne Alexander Roine
5	Abdel Vares Sharraf
6	Abderrahman Sebti
7	Abdesselem Regragui
8	Abebe Hailou
9	Abel Driggs Santos
10	Abraham Israel "Abie" Grossfeld

Query 4

SQL

```
WITH id_ath(id, athlete) as (
    SELECT distinct h.id, a.id
    from results r
    JOIN host h ON h.id = r.host_id
    JOIN competitor c ON r.competitor_id = c.id
    JOIN athlete a ON c.athlete_id = a.id
    WHERE h.year > 1940
    GROUP BY a.id, h.id
    HAVING count(distinct event_id) > 3
    ORDER BY h.id
)

SELECT h.year, count(athlete)
FROM id_ath, host h
WHERE h.id=id_ath.id
GROUP BY h.year
ORDER BY h.year;
```

NUMBER OF ROWS

23

SCREENSHOT

	year	count
1	1948	164
2	1952	364
3	1956	157
4	1960	313
5	1964	289
6	1968	373
7	1972	346
8	1976	275
9	1980	214
10	1984	300

Query 5

SQL

```
WITH india_at_games(season, year, num_comp) AS
        (
            SELECT h.season, h.year, count(distinct a.id)
            from results r
            JOIN host h ON h.id = r.host_id
            JOIN competitor c ON c.id = r.competitor_id
            JOIN athlete a ON c.athlete_id = a.id
            JOIN country ct ON c.noc = ct.alt_noc
            WHERE year > 1947 AND alt_noc ilike 'ind'
            GROUP BY h.season, h.year
        ),
    india_not_at_games(season, year, num_comp) AS
        (
            SELECT h.season, h.year, 0
            FROM host h, india_at_games i
            WHERE h.main_city NOT IN (
                SELECT main_city
                FROM host h1, india_at_games i1
                WHERE h1.season=i1.season AND
                      h1.year=i1.year) AND h.year>=1947
        )

SELECT *
FROM india_at_games
UNION
SELECT *
FROM india_not_at_games
ORDER BY year, season;
```

NUMBER OF ROWS

34

SCREENSHOT

	season	year	num_comp
1	Summer	1948	80
2	Winter	1948	0
3	Summer	1952	65
4	Winter	1952	0
5	Summer	1956	61
6	Winter	1956	0
7	Summer	1960	45
8	Winter	1960	0
9	Summer	1964	53
10	Winter	1964	1

Query 6

SQL

```
SELECT e.discipline, e.event_name, a.sex, a.name, r.medal
FROM results r
JOIN event e on r.event_id = e.id
JOIN competitor c on r.competitor_id = c.id
JOIN athlete a on c.athlete_id = a.id
JOIN host h on r.host_id = h.id
WHERE e.discipline ILIKE '%swim%' AND
      h.year=2004 AND
      h.season='Summer' AND
      r.medal IS NOT NULL
GROUP BY e.discipline, e.event_name, a.sex, a.name, r.medal, c.noc
ORDER BY e.discipline,
         e.event_name,
         a.sex,
         (CASE r.medal
             WHEN 'Gold' THEN 1
             WHEN 'Silver' THEN 2
             WHEN 'Bronze' THEN 3
             END);
```

NUMBER OF ROWS

221

SCREENSHOT

	discipline	event_name	sex	name	medal
1	Swimming	100M Backstroke	F	Natalie Anne Coughlin (-Hall)	Gold
2	Swimming	100M Backstroke	F	Kirsty Leigh Coventry (-Seward)	Silver
3	Swimming	100M Backstroke	F	Laure Manaudou	Bronze
4	Swimming	100M Backstroke	M	Aaron Wells Peirsol	Gold
5	Swimming	100M Backstroke	M	Markus Antonius Rogan	Silver
6	Swimming	100M Backstroke	M	Tomomi Morita	Bronze
7	Swimming	100M Breaststroke	F	Luo Xuejuan	Gold
8	Swimming	100M Breaststroke	F	Brooke Louise Hanson (-Clarke)	Silver
9	Swimming	100M Breaststroke	F	Leisel Marie Jones	Bronze
10	Swimming	100M Breaststroke	M	Kosuke Kitajima	Gold

Query 7

SQL

```
SELECT h.year,
       sum(
           CASE WHEN r.medal = 'Gold' THEN 1
           ELSE 0 END
           ) AS gold,
       sum(
           CASE WHEN r.medal = 'Silver' THEN 1
           ELSE 0 END
           ) AS silver,
       sum(
           CASE WHEN r.medal = 'Bronze' THEN 1
           ELSE 0 END
           ) AS bronze
FROM results r
JOIN event e ON e.id = r.event_id
JOIN competitor c ON c.id = r.competitor_id
JOIN athlete a ON c.athlete_id = a.id
JOIN host h ON r.host_id = h.id
WHERE a.name ilike '%michael%phelps%' or a.name ilike
'%phelps%michael%'
GROUP BY h.year;
```

NUMBER OF ROWS

4

SCREENSHOT

4 rows

	year	gold	silver	bronze
1	2000	0	0	0
2	2004	6	0	2
3	2008	8	0	0
4	2012	4	2	0

Query 8

SQL

```
SELECT C.country, sum(CASE WHEN R.medal='Gold' THEN 1 ELSE 0 END) as
gold_count
FROM country C, results R
JOIN competitor CP ON R.competitor_id=CP.id
JOIN athlete A ON CP.athlete_id = A.id
JOIN event E ON R.event_id = E.id
WHERE A.sex='M' AND E.event_name ILIKE '%marathon%' AND
E.sport='Athletics' AND C.alt_noc=CP.noc
GROUP BY C.country

ORDER BY gold_count DESC
LIMIT 1;
```

NUMBER OF ROWS

1

SCREENSHOT

	country	gold_count
1	Ethiopia	4

Query 9

SQL

```
WITH aey(aid, eid, year, diff_year, diff_med) AS
        (SELECT distinct a.id, e.id, h.year,
                         h.year - lag(h.year) over (PARTITION BY
a.id, e.id ORDER BY a.id, e.id, h.year) as diff_year,
                         (CASE r.medal
                           WHEN 'Gold' THEN 3
                           WHEN 'Silver' THEN 2
                           WHEN 'Bronze' THEN 1
                          END) - lag((CASE r.medal
                           WHEN 'Gold' THEN 3
                           WHEN 'Silver' THEN 2
                           WHEN 'Bronze' THEN 1
                          END)) over (PARTITION BY a.id, e.id ORDER
BY a.id, e.id, h.year) as diff_med
        FROM athlete a
        JOIN competitor c on a.id = c.athlete_id
        JOIN results r on c.id = r.competitor_id
        JOIN host h on r.host_id = h.id
        JOIN event e on r.event_id = e.id),
     gtr3(aid, eid) as
        (SELECT aid, eid
        FROM aey
        where diff_year<>0
        group by aid, eid
        having count(*)>=3 and max(diff_year)<=4 and
min(diff_med)>=0),
     good_athletes(ath_id) as
        (select aey.aid from aey, gtr3 where aey.aid=gtr3.aid and
aey.eid=gtr3.eid and diff_year<>0 and diff_med>=0)

select distinct name
from athlete, good_athletes
where id = ath_id
ORDER BY name;
```

NUMBER OF ROWS

269

SCREENSHOT

	name
1	Abdon Pamich
2	Abdul Hamid
3	Adam Stanisaw Pape
4	Adhemar Ferreira da Silva
5	Agneta Monica Andersson
6	Alberto Tomba
7	Aleksandr Ivanovich Tikhonov
8	Aleksandr Vladimirovich Popov
9	Aleksandr Yuryevich Zubkov
10	Alessandro Fei