Java Programming Studio

SQL Continued

Santha Sumanasekara

RMIT UNIVERSITY

Overview of the Studio Class

- Multi-table SQL statements
  - Simple JOINs
  - NATURAL JOIN
  - LEFT OUTER JOINs
  - Self joins
- Sub-queries
  - Within WHERE clause
  - Within FROM clause

SQL JOINS

- Many of the common queries will require data from more than one table.
  - Display names of cities in the world with their countries
  - Display English-speaking countries
  - Display other languages spoken in the countries where Swahili is spoken

SQL JOINS

- Display names of cities in the with their countries

	name	code
1	Aruba	ABW
2	Afghanistan	AFG
3	Angola	AGO
4	Anguilla	AIA
5	Albania	ALB
6	Andorra	AND
7	Netherlands Antilles	ANT
8	United Arab Emirates	ARE
9	Argentina	ARG

countrycode	name
ABW	Oranjestad
AFG	Kabul
AFG	Qandahar
AFG	Herat
AFG	Mazar-e-Sharif
AGO	Luanda
AGO	Huambo
AGO	Lobito
AGO	Benguela

SQL JOIN

- Identify which tables contain required data
- Identify a common attribute that can be used as the JOIN conditi

```
SELECT c1.name, c2.name
    FROM country c1 JOIN city c2
        ON c1.code = c2.countrycode
```

C1 and c2 are alias names given to the tables joined here.

Join condition using CountryCode.

SQL JOINS

- Display names of cities in the with their countries

		name	name:1	name
1	Aruba	Aruba	Oranjestad	
2	Afghanistan	Afghanistan	Kabul	
3	Angola	Afghanistan	Qandahar	
4	Anguilla	Afghanistan	Herat	
5	Albania	Afghanistan	Mazar-e-Sharif	ıarif
6	Andorra	Angola	Luanda	
7	Netherlands Antilles	Angola	Huambo	
8	United Arab Emirate	Angola	Lobito	
9	Argentina	Angola	Benguela	

SQL JOINS – Another Example

- Display the English-speaking countries
  - Language details come from CountryLanguage table
  - Country names are in Country table
  - Country Code is the common attribute
  - So (1) Join these two tables and then
    (2) filter on “English” language

SQL JOINS – Another Example

```
SELECT c.name, cl.language
   FROM country c JOIN countrylanguage cl
          ON c.code = cl.countrycode
   WHERE cl.language = 'English' ;
```

SQL JOINS – Another way

- A quick-and-dirty way of doing a join two tables:
  - Simply add the join condition to WHERE clause.

```
SELECT c.name, cl.language
   FROM country c,  countrylanguage cl
   WHERE c.code = cl.countrycode  AND
            cl.language = 'English' ;
```

Tables to be joined are now a comma-separated list.

Join condition is within WHERE clause

Joining more than two tables

- Some queries may require data from three or more tables.
  - In such situations we nest the joins within each other
  - Say, first join country and city and then the result join with countryLanguage
  - What are the languages spoken in New Delhi?
  - This require all three tables joined: Country, CountryLanguage and City.

Joining more than two tables

First join between country and countryLanguage .

```
SELECT c.name, cy.name, cl.language
   FROM (country c JOIN countrylanguage cl
          ON c.code = cl.countrycode)
                JOIN city cy
                      on c.code = cy.countrycode
   WHERE cy.name = 'New Delhi'
```

Second join between the result of first join and city

Natural Join

- Natural Join is a special case of a join where the two attributes to be used for joining have the same name.
- In that case, we do not need to add an ON clause with the join condition.
- Replace JOIN with NATURAL JOIN in the FROM clause, with no conditions.
- The Join condition is implicit (by their names).
- Consider the following example.

```
Country (Code, name, continent, region, …)
City2 (ID, CityName, code*, district,
       CityPopulation)
```

- In this case, code attribute in City2 is a foreign key referencing code attribute in Country. Make sure no other attributes have common names.
- They both have the same name (unlike in our World DB, where the foreign key is called “CountryCode”.

Natural Join

No explicit condition.

```
SELECT c1.name, c2.name
   FROM country c1 NATURAL JOIN city2 c2;
```

This is essentially the same as:

```
SELECT c1.name, c2.name
     FROM country c1 JOIN city2 c2
          ON c1.code = c2.code;
```

Left Outer Join

- Consider the following query: List all countries in the world, with major cities (if any).
- This query must list all countries, regardless of whether there are any major cities.
- A simple join will list “only” countries that can match with a city, missing out the countries like Antarctica where there are no major cities.
- LEFT OUTER JOIN will be the way to go here.
- The left outer join will display all rows from table on the left side of the join regardless of matching rows from the right table.
- If there are matches, they will show up, and for the rest, NULLs are displayed.

Left Outer Join

Order of the tables in the FROM clause is important. Use the table that require all rows displayed must be on the left side of the join

```
SELECT co.name AS "Country Name",
        co.code AS “Country Code”,
        cy.countrycode AS “Country Code”,
        cy.name AS "City Name",
        cy. population AS "City Population”
   FROM country co  LEFT OUTER JOIN city cy
                ON co.code = cy.CountryCode
   WHERE continent ='Oceania'
```

Left Outer Join

There is no country code in City table that match with UMI. So, the corresponding country is padded with null values

	Counry Name	code	countrycode	City Name	y Population
48	French Polynesia	PYF	PYF	Papeete	25553
49	Solomon Islands	SLB	SLB	Honiara	50100
50	Tokelau	TKL	TKL	Fakaofo	300
51	Tonga	TON	TON	Nuku¥alofa	22400
52	Tuvalu	TUV	TUV	Funafuti	4600
53	United States Minor Outlying Islands	UMI	*NULL*	*NULL*	*NULL*
54	Vanuatu	VUT	VUT	Port-Vila	33700
55	Wallis and Futuna	WLF	WLF	Mata-Utu	1137
56	Samoa	WSM	WSM	Apia	35900

Self-Join

- The self-join is special kind of join that allows you to join a table to itself using either inner join or left outer join.
- You use self-join to create a result set that joins the rows with the other rows within the same table.
  - E.g. Find the list of employees supervised by Jennifer S. Wallace. (recall the company database we used in Tute 2)
  - Display other languages spoken in the countries where Swahili is spoken
- You will be required to use two alias names to refer each instance of the table (say: employees emp and employees sup

Self-Join: An Example

```
SELECT sup.fname, sup.minit, sup.lname,
       emp.fname, emp.minit, emp.lname
  FROM employees emp JOIN employees sup
       ON emp.superSSN = sup.SSN
  WHERE sup.fname = 'Jennifer' AND
        sup.minit = 'S' AND
        sup.lname = 'Wallace';
```

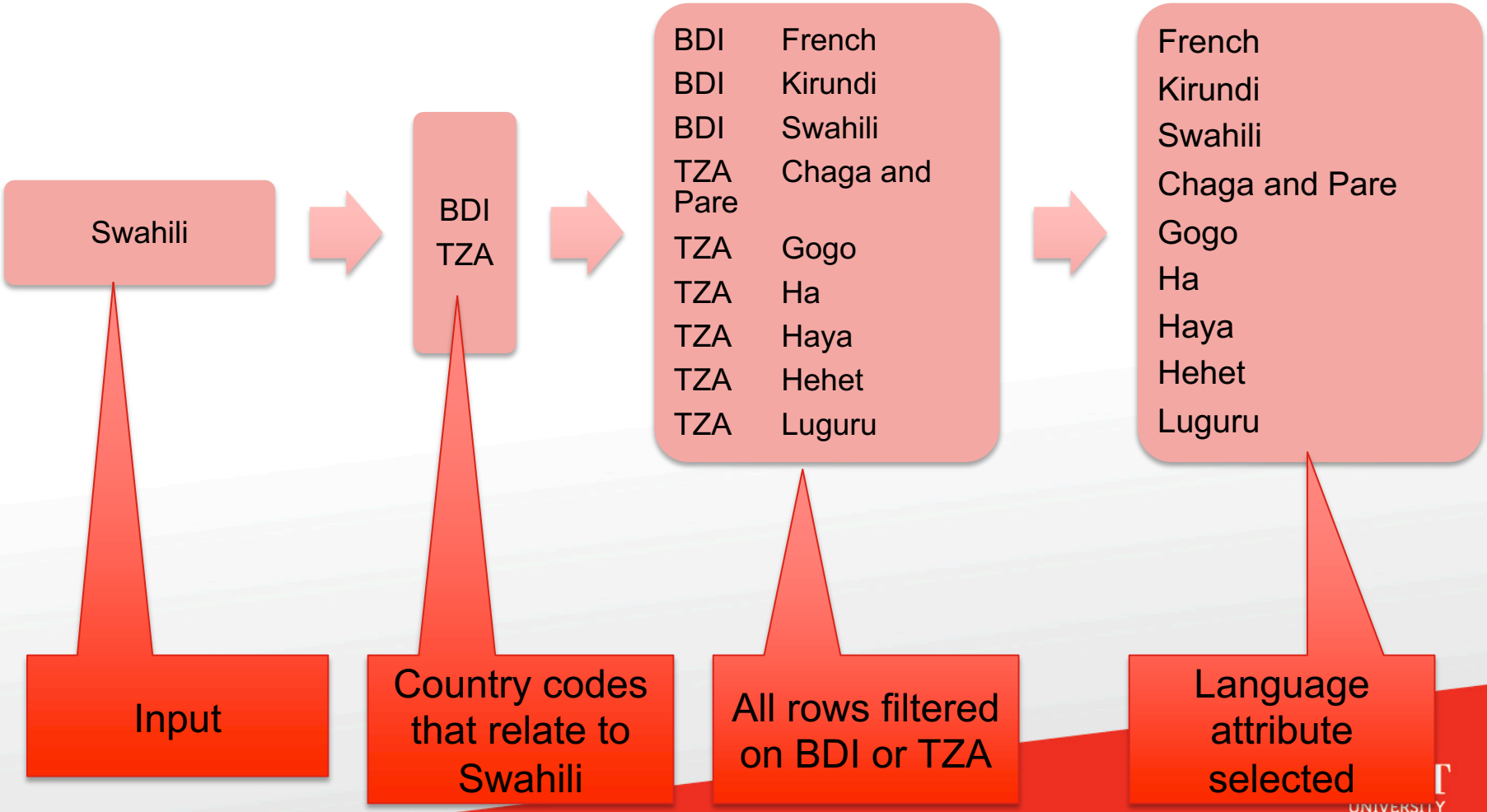
Emplyee table is opened twice, one instance to refer employee superSSN values, and the other to refer SSN values (for supervisors)

Self-Join: Another Example

Display other languages spoken in the countries where Swahili is spoken

- This query can be explained as follows:
- Open CountryLanguage table once. Scan through the table and collate the country codes that relevant to Swahili language.
- Open another instance of the CountryLanguage table, this time filter the rows that match with country codes you got from the previous step.
- display the language attribute of the filtered rows.

Self-Join: Another Example

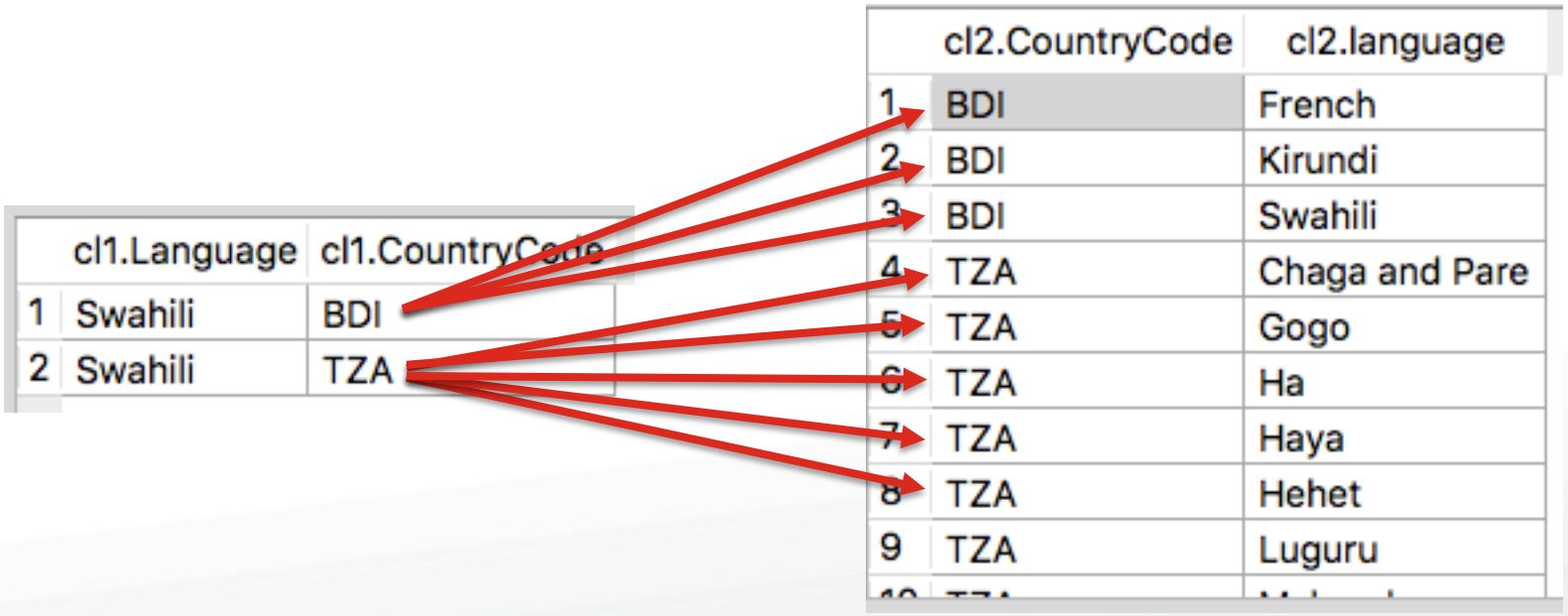


Self-Join: Another Example

```
SELECT cl2.countrycode, cl2.language
   FROM countrylanguage cl1 JOIN countryLanguage cl2
         ON cl1.CountryCode = cl2.countrycode
   WHERE cl1.language = 'Swahili';
```

countryLanguage table is opened twice, one instance to refer country code values related to Swahili, and the other to refer to all reows relevant to those country codes

Self-Join: Another Example



	cl1.Language	cl1.CountryCode
1	Swahili	BDI
2	Swahili	TZA

	cl2.CountryCode	cl2.language
1	BDI	French
2	BDI	Kirundi
3	BDI	Swahili
4	TZA	Chaga and Pare
5	TZA	Gogo
6	TZA	Ha
7	TZA	Haya
8	TZA	Hehet
9	TZA	Luguru

CountryLanguage opened as cl1

CountryLanguage opened as cl2

Self-Join: Another Example

A small improvement. Remove Swahili from the result set.

```
SELECT cl2.countrycode, cl2.language
   FROM countrylanguage cl1 JOIN countryLanguage cl2
         ON cl1.CountryCode = cl2.countrycode
   WHERE cl1.language = 'Swahili' AND
         cl2.language <> 'Swahili'
```

There is no point display Swahili in the result set, as the original query is to list "other" languages.

Self-Join: Another (harder) Example

Display the countries that share same languages as spoken by South Africans.

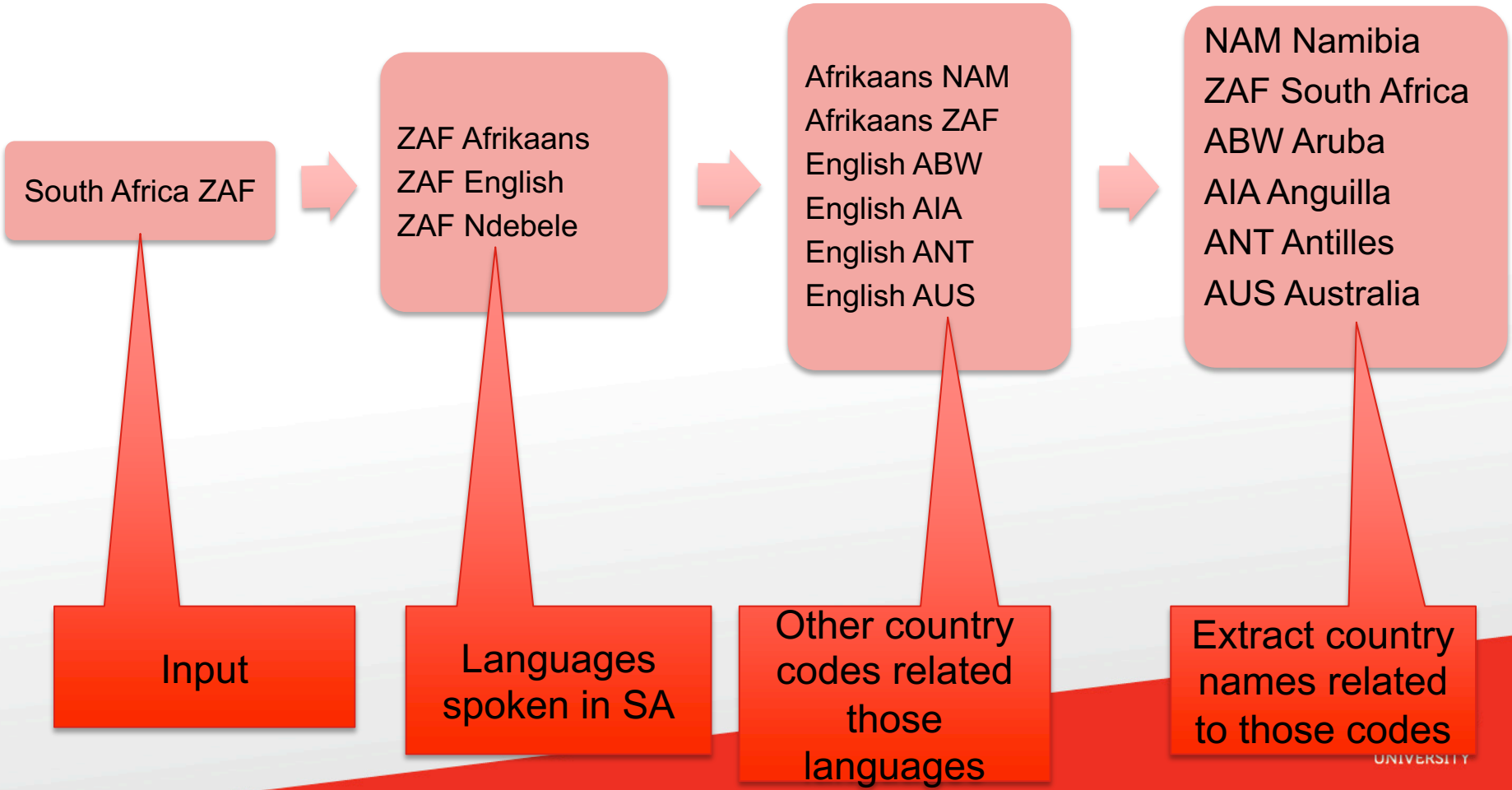
- This query can be explained as follows:
- Open Country and CountryLanguage tables joined together. Scan through the joined table and collate the languages that relevant to South Africa.
- Join that result set with another instance of CountryLanguage table. From this second instance, filter all rows relevant to the languages list you obtained in the previous step.
- Join this again with another instance of Country table to find the maching country names for the rows you obtained in the previous step.

Self-Join: Another (harder) Example

```
SELECT c2.name, cl2.language
   FROM ((country c1 JOIN countrylanguage cl1
          ON c1.code = cl1.countrycode)
          JOIN countrylanguage cl2
                ON cl1.language = cl2.language)
                JOIN country c2
                      ON cl2.countrycode = c2.code
   WHERE c1.name ='South Africa' AND
          c2.name <>'South Africa'
```

It is easy to understand the query by decomposing it into three separate steps.

Self-Join: Another (harder) Example



Nested Queries

sub-selects

IN

EXISTS

Sub-Selects

- The result set of a SELECT statement is no different to a table already in your database
- So, you may use a result set from one SQL statement within another SQL statement where a table is accepted.
- For example, the table name in the FROM clause can be another SQL statement.
- Can be used to simplify the logic: e.g. complex JOIN conditions and WHERE clauses.

Sub-Selects

```
SELECT <something>
    FROM (SELECT <something> FROM <somewhere>)
    WHERE <condition>
```

- When you execute this, the inner query is executed first, and the result set is stored temporarily (as a temporary table)
- This temp table is used as the source for outer query, which executed next.

Sub-Selects: An Example

- List the cities in the English-speaking world.
  - This can be decomposed into two problems:
    1. Countries in the English-speaking world
    2. Cities in those countries.
  - The result of the first part itself can be considered as a table (say, ESC)
  - Then, second problem is to find the cities in ESC.
  - Write the first part as a sub-select.
  - If the first part occurs frequently in your queries, you may even be able to reuse that SELECT in many other queries.

Sub-Selects: An Example

```
SELECT esc.name AS "Country",
       cy.name AS "City"
FROM city cy Join <EnglishSpeakingCountries> esc
ON cy.CountryCode = esc.code
```

Outer Query

This is the table we "generate" using inner query

Sub-Selects: An Example

```
SELECT esc.name AS "Country",
       cy.name AS "City"
  FROM city cy Join <EnglishSpeakingCountries> esc
  ON cy.CountryCode = esc.code
```

```
SELECT co.code, co.name
FROM country co JOIN countryLanguage cl
ON co.code = cl.countryCode
WHERE cl.language = 'English'
```

Sub-Selects: An Example

```
SELECT esc.name AS "Country",
          cy.name AS "City"
FROM city cy Join
        (
         SELECT co.code, co.name
           FROM country co JOIN countryLanguage cl
                   ON co.code = cl.countryCode
           WHERE cl.language = 'English'
        ) esc
        ON cy.CountryCode = esc.code
```

In WHERE clause

- If the result set of a SELECT statement contain only one attribute, it can be considered as a "set of values or a list" and can be used with IN operator within WHERE clause.
- Recall the SQL statement we discussed last week where we used IN operator to list countries in Asia-Pacific region (i.e countries in Asia or Oceania continents).
- At that time, we hard-coded the continents list – better than that, we can generate it dynamically, using an inner query.

```
SELECT name AS "Country Name"
    FROM country
    WHERE continent IN (AsiaPacificRegion);
```

AsiaPacificRegion is generated using anoher SQL statement

In WHERE clause: An Example

- List the cities in the English-speaking world.
  - This can again be decomposed into two problems:
    1. Country codes in the English-speaking world
    2. Cities in those countries.
  - The result of the first part itself can be considered as a list of codes
  - Then, second problem is to find the cities in those countries.
  - Write the first part as a sub-query producing a list of codes.
  - Embed this list within outer query.

In WHERE clause: An Example

```
SELECT name AS "City"
   FROM city
   WHERE countryCode IN ( <EnglishSpecking CountryCodes>  );
```

Outer Query

This is the list we "generate" using inner query

In WHERE clause: An Example

```
SELECT name AS "City"
   FROM city
   WHERE countryCode IN ( <EnglishSpecking CountryCodes>  );
```

```
SELECT cl.countrycode
   FROM countryLanguage cl

   WHERE cl.language = 'English'
```

In WHERE clause: An Example

```
SELECT name AS "City"
    FROM city
    WHERE countryCode IN (
                                SELECT co.code
                                    FROM country co JOIN
                                            countryLanguage cl
                                            ON co.code = cl.countryCode
                                    WHERE cl.language = 'English'
                                );
```

In WHERE clause: An Example

Similar to IN, we can use NOT IN to list cities in the rest of the world.

```
SELECT name AS "City"
    FROM city
    WHERE countryCode NOT IN (
                                SELECT co.code
                                    FROM country co JOIN
                                            countryLanguage cl
                                            ON co.code = cl.countryCode
                                    WHERE cl.language = 'English'
                                );
```

With EXISTS (and NOT EXISTS)

- The EXISTS operator is different.
- It doesn't compare values vs attributes, etc
- What it does is to check if the associated list with it is empty or not empty.
- If it is empty, it returns a FALSE value; if the list is not empty, it returns a TRUE value.
- This "associated list" can be generated using an inner query.

```
SELECT name AS "Country Name"
   FROM country
   WHERE  EXISTS AsiaPacificRegion;
```

Note that there is no attribute to compare against.

AsiaPacificRegion is generated using another SQL statement

With EXISTS clause: An Example

- List the cities in the English-speaking world.
  - This can again be decomposed into three problems:
    1. Pick a city, identify corresponding country code;
    2. Take that country code and cross check with country codes list of English speaking countries.
    3. If it does exist, show the city you picked up.
  - In this example, the execution starts at the outer query
  - For each city in outer query, take the country code and feed it into the inner query to check if it is there.
  - If it does, display the result, or else go to the next city.

With EXISTS clause: An Example

```
SELECT name AS "City"
   FROM city cy
   WHERE EXISTS ( <cy.countrycode matched against
                   EnglishSpecking CountryCodes> );
```

Outer Query

The inner query will produce a non-empty result if cy.countrycode is an EnglishSpeakingCountry or empty, if it isn't

With EXISTS clause: An Example

```
SELECT name AS "City"
    FROM city cy
    WHERE EXISTS ( <cy.countrycode matched against
                    EnglishSpecking CountryCodes> );
```

```
SELECT *
FROM CountryLanguage cl
WHERE cl.countryCode = cy.CountryCode AND
cl.Language = 'English'
```

With EXISTS clause: An Example

```
SELECT name AS "City"
   FROM city cy
   WHERE EXISTS (
                  SELECT *
                     FROM CountryLanguage cl
                     WHERE cl.countryCode = cy.CountryCode AND
                              cl.Language = 'Engli
                  );
```

Note that, within the inner query we can refer to attributes and tables of the outer query. This is because the order of execution is different with EXISTS.

More on Nested Queries

- There are many other ways we can have nested queries.
- We discuss next week a few other nested queries.