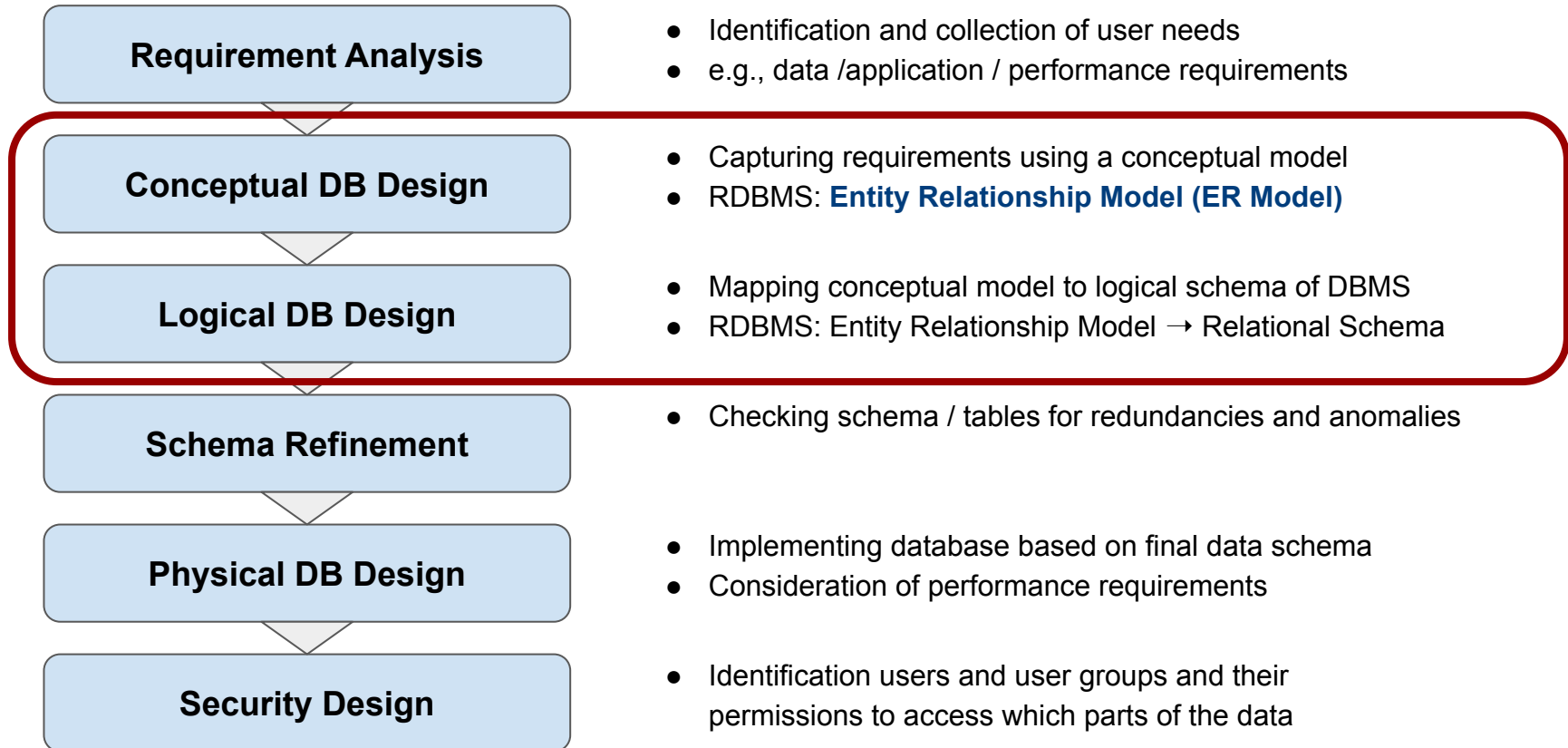


CS2102: Database Systems

Lecture 5 — SQL (Part 2)

Quick Recap: Database Design Process



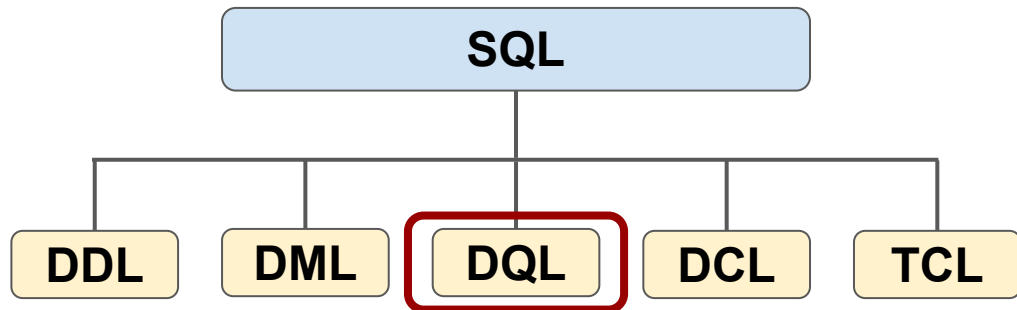
Quick Recap: Where We are Right Now

- Topics covered so far

- Designing a database using conceptual and logical modeling
- Creating a database using DDL (data definition language)
- Inserting, updating and deleting data using DML (data manipulation language)

- Now: Querying a database

- Extracting information using SQL (DQL: data query language)
- Anything with "**SELECT ...**"



Overview

- **SQL vs. Relational Algebra**
- SQL Queries
 - Simple queries
 - Set operations
 - Join queries
 - Subqueries
 - Sorting & rank-based selection
- Summary

SQL vs. Relational Algebra

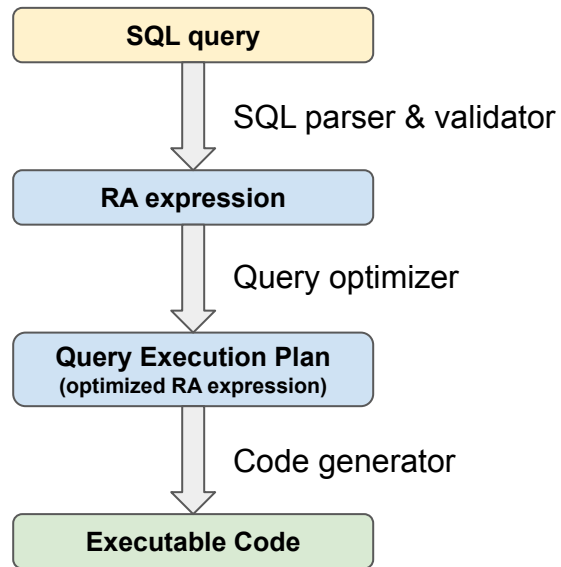
- Relation Algebra (RA)

- Procedural query language using operators to perform queries
- Query = relational algebra expression (operator tree)

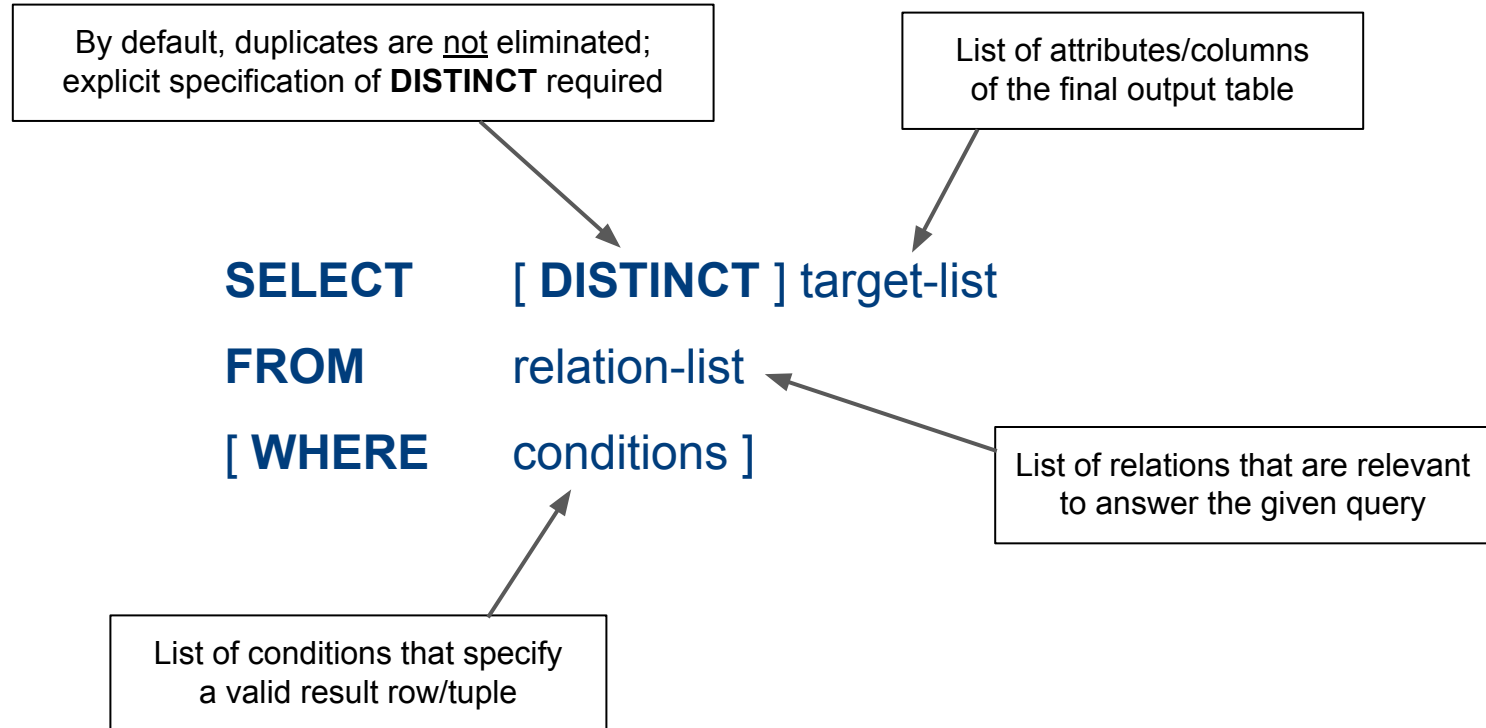
- **SQL** (more precisely: the DQL part of SQL)

- **Declarative** query language built on top of RA
(Focus on *what* to compute, not on *how* to compute)
- Multiset / bag semantics (unlike sets in RA!)
- Query = SELECT statement

| | |
|----------------|---------------------------------|
| SELECT | [DISTINCT] target-list |
| FROM | relation-list |
| [WHERE | conditions] |



SQL Query — Basic Form



SQL Query — Conceptual Evaluation Strategy

SELECT [**DISTINCT**] target-list
FROM relation-list
[**WHERE** conditions]

1. Compute cross product of all tables in **relation-list**
2. Discard all rows/tuples that fail any of the **conditions**
3. Delete attributes/columns not specified in **target-list**
4. If **DISTINCT**, eliminate duplicate rows/tuples

- Mapping between basic SQL query to corresponding RA expression

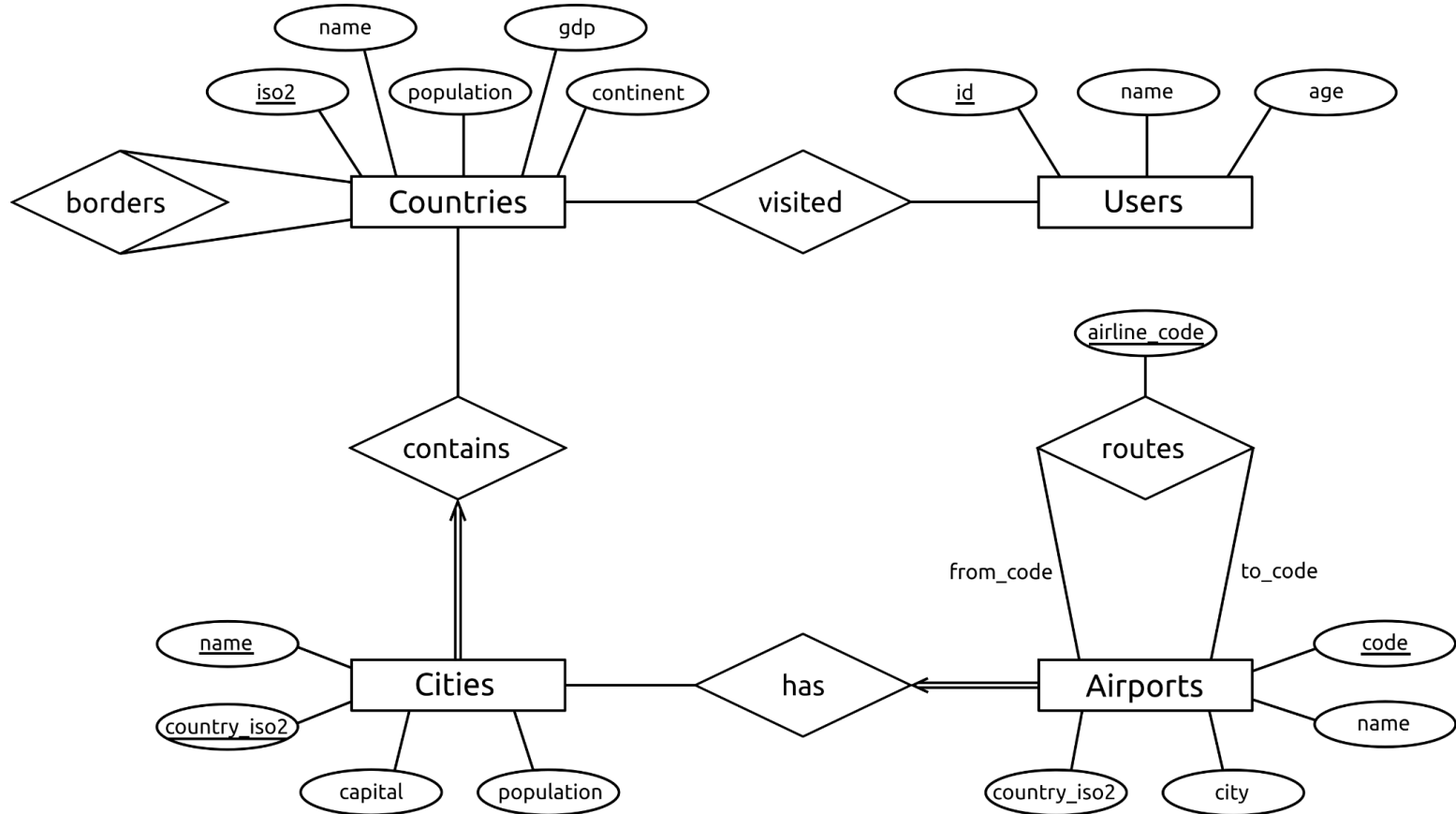
- Conceptual mapping; no consideration of performance

SELECT DISTINCT a_1, a_2, \dots, a_m
FROM r_1, r_2, \dots, r_n
WHERE c



$\pi_{a_1, a_2, \dots, a_m} (\sigma_c(r_1 \times r_2 \times \dots \times r_n))$

Example Database — ER Diagram



Example Database — Data Sample

Countries (225 tuples)

| iso2 | name | population | gdp | continent |
|------|-----------|------------|----------------|-----------|
| SG | Singapore | 5781728 | 488000000000 | Asia |
| AU | Australia | 22992654 | 1190000000000 | Oceania |
| TH | Thailand | 68200824 | 1160000000000 | Asia |
| DE | Germany | 80722792 | 3980000000000 | Europe |
| CN | China | 1373541278 | 21100000000000 | Asia |
| ... | ... | ... | ... | ... |

Borders (699 tuples)

| country1_iso2 | country2_iso2 |
|---------------|---------------|
| SG | <i>null</i> |
| AU | <i>null</i> |
| TH | KH |
| TH | LA |
| TH | MY |
| ... | ... |

Airports (3,372 tuples)

| code | name | city | country_iso2 |
|------|---------------------------|-----------|--------------|
| SIN | Singapore Changi Airport | Singapore | SG |
| XSP | Seletar Airport | Singapore | SG |
| SYD | Sydney Int. Airport | Sydney | AU |
| MEL | Melbourne Int. Airport | Melbourne | AU |
| FRA | Frankfurt am Main Airport | Frankfurt | DE |
| ... | ... | ... | ... |

Cities (24,567 tuples)

| name | country_iso2 | capital | population |
|--------------|--------------|-------------|------------|
| Singapore | SG | primary | 5745000 |
| Kuala Lumpur | MY | primary | 8285000 |
| Nanyang | CN | <i>null</i> | 12010000 |
| Atlanta | US | admin | 5449398 |
| Washington | US | primary | 5379184 |
| ... | ... | ... | ... |

Routes (47,076 tuples)

| from_code | to_code | airline_code |
|-----------|---------|--------------|
| ADD | BKK | SQ |
| ADL | SIN | SQ |
| AKL | SIN | SQ |
| AMS | SIN | SQ |
| BCN | GRU | SQ |
| ... | ... | ... |

Users (9 tuples)

| user_id | name | age |
|---------|-------|-----|
| 101 | Sarah | 25 |
| 102 | Judy | 35 |
| 103 | Max | 52 |
| 104 | Marie | 36 |
| 105 | Sam | 30 |
| ... | ... | ... |

Visited (585 tuples)

| user_id | iso2 |
|---------|------|
| 103 | AU |
| 103 | US |
| 103 | SG |
| 103 | GB |
| 104 | GB |
| ... | ... |

Overview

- SQL vs. Relational Algebra
- **SQL Queries**
 - **Simple queries**
 - Set operations
 - Join queries
 - Subqueries
 - Sorting & rank-based selection
- Summary

Simple Queries (SELECT ... FROM ... WHERE)

Find the name and population of all cities with a population greater than 10 Million.

SELECT name, population
FROM cities
WHERE population > 10000000;

| name | population |
|-------------|------------|
| Mexico City | 20996000 |
| Sao Paulo | 22046000 |
| Delhi | 29617000 |
| Mumbai | 23355000 |
| Manila | 23088000 |
| ... | ... |

Find the name and population of all countries in Asia and Europe with a population between 5 and 6 Million.

SELECT name, population
FROM countries
WHERE (continent = 'Asia' **OR** continent = 'Europe')
AND (population > 5000000 **AND** population < 6000000);

| name | population |
|----------------------|------------|
| Denmark | 5593785 |
| Finland | 5498211 |
| Kyrgyzstan | 5727553 |
| Norway | 5265158 |
| Singapore | 5781728 |
| Slovakia | 5445802 |
| Turkmenistan | 5291317 |
| United Arab Emirates | 5927482 |

Simple Queries (SELECT ... FROM ... WHERE)

- Additional language constructs
 - Wildcard '*' to include all attributes
 - 'expr **BETWEEN** <lower> **AND** <upper>' for basic value range conditions

Find all countries in Asia and Europe with a population between 5 and 6 Million.

```
SELECT *  
FROM countries  
WHERE (continent = 'Asia' OR continent = 'Europe')  
      AND (population BETWEEN 5000000 AND 6000000);
```

| iso2 | name | population | gdp | continent |
|------|----------------------|------------|-----|-----------|
| DK | Denmark | 5593785 | ... | Europe |
| FI | Finland | 5498211 | ... | Europe |
| KG | Kyrgyzstan | 5727553 | ... | Asia |
| NO | Norway | 5265158 | ... | Europe |
| SG | Singapore | 5781728 | ... | Asia |
| SK | Slovakia | 5445802 | ... | Europe |
| TM | Turkmenistan | 5291317 | ... | Asia |
| AE | United Arab Emirates | 5927482 | ... | Asia |

SELECT Clause — Expressions

- Common use cases for SELECT clause expressions
 - Combine and process attribute values
 - Rename columns

Find the name and the all countries the GDP per capita in SGD rounded to the nearest dollar for all countries.

"||" concatenates strings

"AS" is optional

Convert from USD to SGD

```
SELECT name, 'S$ ' || ROUND((gdp / population)*1.35) AS gdp_per_capita
FROM countries;
```

| name | gdp_per_capita |
|----------------------|----------------|
| Denmark | S\$ 63955 |
| Finland | S\$ 55490 |
| Kyrgyzstan | S\$ 4952 |
| Norway | S\$ 93586 |
| Singapore | S\$ 113944 |
| Slovakia | S\$ 46648 |
| Turkmenistan | S\$ 24166 |
| United Arab Emirates | S\$ 151910 |
| ... | ... |

SELECT Clause — Duplicates

Quick Quiz: Why do you think does SQL not eliminate duplicates by default?

- SQL vs Relational Algebra

- By default, SQL does not eliminate duplicates
- Use keyword **DISTINCT** to enforce duplicate elimination

Solution

- Duplicate elimination is quite expensive operation and you already want to do it if really needed

Find all country codes for which cities are available in the database.

SELECT country_iso2 **AS** code
FROM cities;

24,567 tuples (all cities)

| code |
|------|
| MX |
| ID |
| IN |
| IN |
| PH |
| IN |
| ... |

SELECT **DISTINCT** country_iso2 **AS** code
FROM cities;

OR

SELECT **DISTINCT**(country_iso2) **AS** code
FROM cities;

213 tuples

| code |
|------|
| MX |
| ID |
| IN |
| PH |
| CN |
| TH |
| ... |

SELECT Clause — Duplicates with NULL Values

- Example: two tuples (n_1, c_1) and (n_2, c_2) are considered distinct if

" $(n_1 \text{ IS DISTINCT FROM } n_2)$ " or " $(c_1 \text{ IS DISTINCT FROM } c_2)$ "

evaluates to "true"

SELECT name, capital
FROM cities;

24,567 tuples (all cities)

| name | capital |
|-------------|-------------|
| Mexico City | primary |
| Jakarta | primary |
| Delhi | admin |
| Perth | <i>null</i> |
| Perth | <i>null</i> |
| Shenzhen | minor |
| ... | ... |

SELECT DISTINCT name, capital
FROM cities;

24,047 tuples

| name | capital |
|--------------|--------------------|
| Tokyo | primary |
| Jakarta | primary |
| Delhi | admin |
| Perth | <i>null</i> |
| Shenzhen | minor |
| Manila | primary |
| ... | ... |

WHERE Clause — Conditions for NULL Values

- Finding tuples with NULL or not-NULL as attribute value

- Correct: *"attribute IS NULL"*, *"attribute IS NOT NULL"*

- False: *"attribute = NULL"*, *"attribute <> NULL"*

(**CAREFUL**: the conditions above do not throw an error!)

Find all codes of countries that have no land border with another country.

```
SELECT country1_iso2 AS code
FROM borders
WHERE country2_iso2 IS NULL;
```

68 tuples

| code |
|------|
| AU |
| SG |
| TW |
| PH |
| NZ |
| JP |
| ... |

```
SELECT country1_iso2 AS code
FROM borders
WHERE country2_iso2 = NULL;
```

0 tuples (but no error!)

| code |
|------|
|------|

WHERE Clause — Pattern Matching

- Basic pattern matching with (NOT) LIKE

- "_" matches any single character
- "%" matches any sequence of zero or more characters

Find all cities that start with "Si" and end with "re".

```
SELECT name
FROM cities
WHERE name LIKE 'Si%re';
```

| name |
|--------------|
| Singapore |
| Sierre |
| Sierra Madre |

Examples:

'abc' LIKE 'abc' → true

'abc' LIKE 'a%' → true

'abc' LIKE '_b_' → true

'abc' LIKE '_c' → false

- Advanced pattern matching using Regular Expression

(Out of scope here; check for full details: <https://www.postgresql.org/docs/9.3/functions-matching.html>)

Overview

- SQL vs. Relational Algebra
- **SQL Queries**
 - Simple queries
 - **Set operations**
 - Join queries
 - Subqueries
 - Sorting & rank-based selection
- Summary

Set Operations

- Let Q_1 and Q_2 be two SQL queries that yield **union-compatible** tables:

- $Q_1 \text{ UNION } Q_2 = Q_1 \cup Q_2$
- $Q_1 \text{ INTERSECT } Q_2 = Q_1 \cap Q_2$
- $Q_1 \text{ EXCEPT } Q_2 = Q_1 - Q_2$

- Attention: duplicate elimination**

- **UNION, INTERSECT, EXCEPT**
eliminate duplicate tuples from result
- **UNION ALL, INTERSECT ALL, EXCEPT ALL**
do not eliminate duplicate tuples from result

| R | S |
|-------|-------|
| value | value |
| 1 | 2 |
| 2 | 2 |
| 2 | 3 |

(SELECT value FROM R)
UNION
(SELECT value FROM S);

| value |
|-------|
| 2 |
| 1 |
| 3 |

(SELECT value FROM R)
UNION ALL
(SELECT value FROM S);

| value |
|-------|
| 1 |
| 2 |
| 2 |
| 2 |
| 2 |
| 3 |

Set Operations — Example Queries

Find all names that refer to both a city and a country.

(**SELECT** name **FROM** cities)
INTERSECT ALL
(**SELECT** name **FROM** countries);

27 tuples

| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

Find the codes of all the countries for which there is not city in the database.

(**SELECT** iso2 **FROM** countries)
EXCEPT
(**SELECT DISTINCT**(country_iso2)
FROM cities);

12 tuples

| iso2 |
|------|
| VG |
| AI |
| VI |
| MS |
| TK |
| PN |
| NF |
| EH |
| ... |

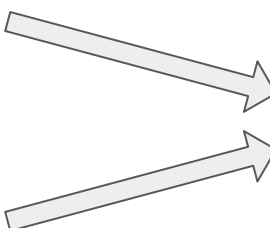
Flexibility of SQL

- Very common: Multiple ways to answer the same query
 - Note: The performance between the queries might differ significantly

Find all airports located in cities named "Singapore" or "Perth".

```
SELECT *  
FROM airports  
WHERE city = 'Singapore'  
      OR city = 'Perth';
```

```
(SELECT * FROM airports  
 WHERE city = 'Singapore')  
UNION  
(SELECT * FROM airports  
 WHERE city = 'Perth');
```



| code | name | city | country_iso2 |
|------|--------------------------|-----------|--------------|
| SIN | Singapore Changi Airport | Singapore | SG |
| XSP | Seletar Airport | Singapore | SG |
| PER | Perth Int. Airport | Perth | AU |
| JAD | Perth Jandakot Airport | Perth | AU |
| PSL | Perth/Scone Airport | Perth | GB |

Overview

- SQL vs. Relational Algebra
- **SQL Queries**
 - Simple queries
 - Set operations
 - **Join queries**
 - Subqueries
 - Sorting & rank-based selection
- Summary

Multi-Relation Queries (Join Queries)

- Multi-relational queries

- Most common in practice: cross product + attribute selection → **join**

For all cities, find their names together with the names of the countries they are located in.

equivalent queries

SELECT c.name, n.name
FROM cities **AS** c, countries **AS** n
WHERE c.country_iso2 = n.iso2;

"AS" is optional

SELECT c.name, n.name
FROM cities c **INNER JOIN** countries n
ON c.country_iso2 = n.iso2;

SELECT c.name, n.name
FROM cities c **JOIN** countries n
ON c.country_iso2 = n.iso2;

| name | name |
|-------------|-------------|
| Mexico City | Mexico |
| Jakarta | Indonesia |
| Delhi | India |
| Mumbai | India |
| Singapore | Singapore |
| Manila | Philippines |
| Mexico City | Mexico |
| Seoul | South Korea |
| ... | ... |

Multi-Relation Queries (Join Queries)

- Natural Joins

- Identical attribute names can be enforced with renaming (but typically not meaningful)

Find all names that refer to both a city and a country.

Solution

- "countries" and "cities" share two attributes: "name" and "population"
- There's no country-city pair where both name and population match

Why?
↓
SELECT DISTINCT(name)
FROM (**SELECT** name **FROM** cities) t1
NATURAL JOIN
(**SELECT** name **FROM** countries) t2;

27 tuples

| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

Quick Quiz: Why is the result of the query below empty?

SELECT * FROM countries **NATURAL JOIN** cities;

Multi-Relation Queries (Join Queries)

- Outer Joins

- Recall: sometimes the "dangling tuples" are of interest

Find the all the countries for which there is not city in the database.

```
SELECT n.name
FROM countries n optional LEFT OUTER JOIN cities c
ON n.iso2 = c.country_iso2
WHERE c.country_iso2 IS NULL;
```

12 tuples

keep only the
dangling tuples

| name |
|----------------|
| Namibia |
| Nauru |
| Palestine |
| Anguilla |
| Montserrat |
| Pitcairn |
| Tokelau |
| Western Sahara |
| ... |

Complex Join Queries

Find all airports in **European countries** **without a land border** which **cannot be reached** by plane given the existing routes in the database.

```
SELECT t1.country, t1.city, t1.airport
FROM
  (SELECT n.name AS country, c.name AS city,
         a.name AS airport, a.code
   FROM borders b, countries n, cities c, airports a
  WHERE b.country1_iso2 = n.iso2
        AND n.iso2 = c.country_iso2
        AND c.name = a.city
        AND b.country2_iso2 IS NULL
        AND n.continent = 'Europe') t1
LEFT OUTER JOIN
  routes r
ON t1.code = r.to_code
WHERE r.to_code IS NULL;
```

All airports in European countries
without a land border (13 tuples)

| country | city | airport |
|---------|----------|----------------------|
| Iceland | Hofn | Hornafjörður Airport |
| Malta | Pembroke | Pembroke Airport |
| Malta | Victoria | Victoria Airport |

Overview

- SQL vs. Relational Algebra
- **SQL Queries**
 - Simple queries
 - Set operations
 - Join queries
 - **Subqueries**
 - Sorting & rank-based selection
- Summary

Subqueries / Nested Queries

- Subqueries in FROM clause

- Consequence of closure property
- Must be enclosed in parentheses
- Table alias mandatory
- Column aliases optional

```
SELECT *  
FROM (  
    SELECT n.iso2, n.name  
    FROM countries n, borders b  
    WHERE n.iso2 = b.country1_iso2  
    AND country2_iso2 IS NULL  
) AS LandborderfreeCountries(code, name);
```

| code | name |
|------|-----------|
| AU | Australia |
| BS | Bahamas |
| SG | Singapore |
| CU | CUBA |
| JP | Japan |
| MV | Maldives |
| ... | ... |

- Subquery expressions

- IN subqueries
- EXISTS subqueries
- ANY/SOME subqueries
- ALL subqueries

IN Subqueries

Quick Quiz: In the example query below, could we simply switch "countries" and "cities"?

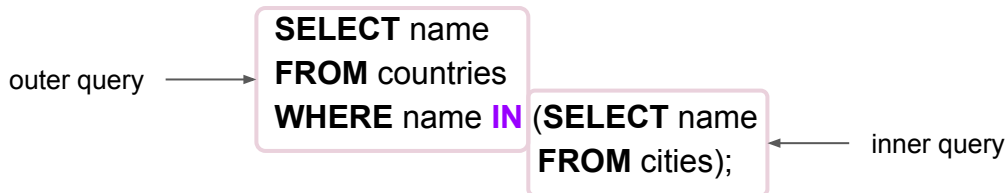
- **(NOT) IN** subquery expressions

- Basic syntax: "*expr* **IN** (*subquery*)", "*expr* **NOT IN** (*subquery*)"
- The subquery must return exactly one column
- **IN** returns true if *expr* matches with any subquery row
- **NOT IN** returns true if *expr* matches with no subquery row

Solution

- Yes we can, but we then need a "SELECT DISTINCT name ..." for the outer query
- Reason: there are 2 duplicate city names that are also a country ("Jordan" and "San Marino")

Find all names that refer to both a city and a country.



27 tuples

| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

IN Subqueries

Find the codes of all the countries for which there is not city in the database.

```
SELECT iso2
FROM countries
WHERE iso2 NOT IN (SELECT country_iso2
                   FROM cities);
```

12 tuples

| iso2 |
|------|
| VG |
| AI |
| VI |
| MS |
| TK |
| PN |
| NF |
| EH |
| ... |

- **Rule of thumb** (can have significant impact on query performance)
 - **IN** subqueries can typically be replaced with (inner) joins
 - **NOT IN** subqueries can typically be replaced with outer joins

IN Subquery

- Special syntax: "manual" specification of subquery result
 - Syntax: "*expression* (**NOT**) **IN** (*value*₁, *value*₂, ..., *value*_{*n*})"

Find all countries in Asia and Europe with a population between 5 and 6 Million.

```
SELECT *  
FROM countries  
WHERE continent IN ('Asia', 'Europe')  
      AND population BETWEEN 5000000 AND 6000000;
```

| iso2 | name | population | gdp | continent |
|------|----------------------|------------|-----|-----------|
| DK | Denmark | 5593785 | ... | Europe |
| FI | Finland | 5498211 | ... | Europe |
| KG | Kyrgyzstan | 5727553 | ... | Asia |
| NO | Norway | 5265158 | ... | Europe |
| SG | Singapore | 5781728 | ... | Asia |
| SK | Slovakia | 5445802 | ... | Europe |
| TM | Turkmenistan | 5291317 | ... | Asia |
| AE | United Arab Emirates | 5927482 | ... | Asia |

ANY/SOME Subqueries (ANY and SOME are synonymous)

All Londons

| name | country | population |
|--------|---------|------------|
| London | GB | 10979000 |
| London | CA | 383822 |
| London | US | 37714 |
| London | US | 14870 |

- **ANY** subquery expressions

- Basic syntax: "*expr op ANY (subquery)*"
- The subquery must return exactly one column
- Expression *expr* is compared to each subquery row using operator *op*
- **ANY** returns true if comparison evaluates to true for at least one subquery row

Find all countries with a population size smaller than any city called "London" (there are actually 4 cities called "London" on the database).

```
SELECT name, population
FROM countries
WHERE population < ANY (SELECT population
                        FROM cities
                        WHERE name = 'London');
```

145 tuples

| name | population |
|-----------|------------|
| Singapore | 5781728 |
| Portugal | 10833816 |
| Sweden | 9880604 |
| Brunei | 436620 |
| Bhutan | 750125 |
| Austria | 8711770 |
| Jamaica | 2970340 |
| Maldives | 392960 |
| ... | |

ALL Subqueries

- **ALL** subquery expressions

- Basic syntax: "*expr op ALL (subquery)*"
- The subquery must return exactly one column
- Expression *expr* is compared to each subquery row using operator *op*
- **ALL** returns true if comparison evaluates to true for all subquery rows

Find all countries with a population size smaller than all cities called "London" (there are actually 4 cities called "London" on the database).

```
SELECT name, population
FROM countries
WHERE population < ALL (SELECT population
                        FROM cities
                        WHERE name = 'London');
```

17 tuples

All Londons

| name | country | population |
|--------|---------|------------|
| London | GB | 10979000 |
| London | CA | 383822 |
| London | US | 37714 |
| London | US | 14870 |

| name | population |
|------------------|------------|
| Nauru | 9591 |
| Tuvalu | 10959 |
| Vatican | 1000 |
| Cook Island | 9556 |
| Falkland Islands | 2931 |
| Montserrat | 5267 |
| Niue | 1190 |
| Norfolk Island | 2210 |
| ... | ... |

Correlated Subqueries

- Correlated subquery

- Subquery uses value from outer query
- Result of subquery depends on value of outer query → potentially slow performance

For each continent, find the country with the highest GDP.

```
SELECT name, continent, gdp
FROM countries c1
WHERE gdp >= ALL (SELECT gdp
                   FROM countries c2
                   WHERE c2.continent = c1.continent);
```

| name | continent | gdp |
|---------------|---------------|----------------|
| Australia | Oceania | 1190000000000 |
| Brazil | South America | 3080000000000 |
| China | Asia | 21100000000000 |
| Egypt | Africa | 1110000000000 |
| Germany | Europe | 3980000000000 |
| United States | North America | 18600000000000 |

Correlated Subqueries — Scoping Rules

- Potential pitfall: naming ambiguities

- Same attribute names in inner and outer queries (here: "continent")
- Best approach: resolve ambiguities using table aliases (here: c1, c2)
- Otherwise: application of scoping rules

```
SELECT name, continent, gdp
FROM countries c1
WHERE gdp >= ALL (SELECT gdp
                  FROM countries c2
                  WHERE c2.continent = c1.continent);
```

- Scoping Rules

- A table alias declared in a (sub-)query Q can only be used in Q or subqueries nested within Q (In example above: "SELECT c1.name, c1.continent, c1.gdp ..." OK, but "SELECT c2.name, c2.continent, c2.gdp ..." fails)
- If the same table alias is declared both in a subquery Q and in an outer query (or not at all) the declaration in Q is applied (general rule: "from inner to outer queries" in case of multiple nestings)

Scoping Rules Gone Wrong

For each continent, find the country with the highest GDP.

SELECT name, continent, gdp
FROM countries c1
WHERE gdp >= **ALL** (**SELECT** gdp
 FROM countries c2
 WHERE c2.continent = c1.continent);



| name | continent | gdp |
|---------------|---------------|----------------|
| Australia | Oceania | 1190000000000 |
| Brazil | South America | 3080000000000 |
| China | Asia | 21100000000000 |
| Egypt | Africa | 1110000000000 |
| Germany | Europe | 3980000000000 |
| United States | North America | 18600000000000 |

SELECT name, continent, gdp
FROM countries c1
WHERE gdp >= **ALL** (**SELECT** gdp
 FROM countries c2
 WHERE c2.continent = ~~c1~~.continent);



| name | continent | gdp |
|-------|-----------|----------------|
| China | Asia | 21100000000000 |

Scoping Rules Gone Wrong

Find all names that refer to both a city and a country.

SELECT name
FROM countries
WHERE name **IN** (**SELECT** name
 FROM cities);



| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

27 tuples

SELECT c.name
FROM countries c
WHERE name **IN** (**SELECT** c.name
 FROM cities c);



| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

27 tuples

SELECT c1.name
FROM countries c1
WHERE name **IN** (**SELECT** c1.name
 FROM cities c2);



| name |
|-----------|
| Singapore |
| China |
| Germany |
| Japan |
| Brasil |
| Russia |
| Malaysia |
| Vietnam |
| ... |

225 tuples

EXISTS Subqueries

- (NOT) EXISTS subquery expressions
 - Basic syntax: "**EXISTS** (*subquery*)", "**NOT EXISTS** (*subquery*)"
 - **EXISTS** returns true if the subquery returns at least one tuple
 - **NOT EXISTS** returns true if the subquery returns no tuple

Find all names that refer to both a city and a country.

```
SELECT n.name
FROM countries n
WHERE EXISTS (SELECT c.name
              FROM cities c
              WHERE c.name = n.name);
```

27 tuples

| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

EXISTS Subqueries

Find the all the countries for which there is not city in the database.

```
SELECT n.name
FROM countries n
WHERE NOT EXISTS (SELECT *
                  FROM cities c
                  WHERE c.country_iso2 = n.iso2);
```

12 tuples

| name |
|----------------|
| Namibia |
| Nauru |
| Palestine |
| Anguilla |
| Montserrat |
| Pitcairn |
| Tokelau |
| Western Sahara |
| ... |

- Rule of thumb

- (NOT) EXISTS subqueries are generally always correlated
- Uncorrelated (NOT) EXISTS subqueries are either wrong or unnecessary

Scalar Subqueries

- Scalar subquery — definition

- Subquery returns a single value (i.e., table 1 row with 1 column)
- Can be used as an expression in queries

For all cities, find their names together with the names of the countries they are located in.

```
SELECT name AS city,  
       (SELECT name AS country  
        FROM countries n  
        WHERE n.iso2 = c.country_iso2)  
FROM cities c;
```

Quick Quiz: How do we know the subquery will return only a single value?

Solution

- The DBMS doesn't know ahead of time
- An error raised when executing the query and the subquery returns more than 1 row or column

| city | country |
|-------------|-------------|
| Tokyo | Japan |
| Jakarta | Indonesia |
| Delhi | India |
| Mumbai | India |
| Singapore | Singapore |
| Manila | Philippines |
| Mexico City | Mexico |
| Seoul | South Korea |
| ... | ... |

Scalar Subqueries

*For Illustration Purposes Only
— Don't Try This At Home! :)*

Find all cities that are located in a country with a country population smaller than the population of all cities called "London" (there are actually 4 cities called "London" on the database).

```
SELECT c.name AS city, c.country_iso2 AS country, c.population
FROM cities c
WHERE (SELECT population
      FROM countries n
      WHERE n.iso2 = c.country_iso2) < ALL (SELECT population
      FROM cities
      WHERE name = 'London');
```

population of the country for a given city which
is located in that country (single value!)

19 tuples

| city | country | population |
|--------------|---------|------------|
| Funafuti | TV | 6025 |
| Avarua | CK | 5445 |
| Vatican City | VA | 825 |
| Stanley | FK | 2213 |
| ... | ... | ... |

Subqueries — Row Constructors

- So far: Requirement for IN, ANY/SOME, and ALL subqueries
 - Subquery must return exactly one attribute/column

→ Row Constructors

- Allow subqueries to return more than one attribute/column
- The number of attributes/columns in row constructor must match the one of the subquery

Attention: The semantics of comparison using row constructors can be rather unintuitive!

Subqueries — Row Constructors

| name | population | gdp |
|---------|------------|---------------|
| France | 66836154 | 2700000000000 |
| Germany | 80722792 | 3980000000000 |

Find all countries with a higher population or higher gdp than France or Germany

```
SELECT name, population, gdp
FROM countries
WHERE ROW(population, gdp) > ANY (SELECT population, gdp
                                   FROM countries
                                   WHERE name IN ('Germany', 'France'));
```

20 tuples

| name | population | gdp |
|---------------|------------|---------------|
| China | 1373541278 | 2110000000000 |
| Thailand | 68200824 | 1160000000000 |
| Nigeria | 186053386 | 1090000000000 |
| Vietnam | 95261021 | 595000000000 |
| United States | 323995528 | 1860000000000 |
| ... | ... | ... |

Note: Only one of the two attribute comparisons has to evaluate to "true". E.g., Thailand matches since the population is larger than the one for France.
For more details: <https://www.postgresql.org/docs/current/functions-comparisons.html#ROW-WISE-COMPARISON>

Subqueries — Remarks

- Queries can contain multiple nested subqueries

Find all the airports in Denmark.

```
SELECT name, city
FROM airports
WHERE city IN (SELECT name
                FROM cities
                WHERE country_iso2 IN (SELECT iso2
                                       FROM countries
                                       WHERE name = 'Denmark')
                );
```

```
SELECT a.name, a.city
FROM airports a, cities c, countries n
WHERE a.city = c.name
AND c.country_iso2 = n.iso2
AND n.name = 'Denmark';
```

Alternative query
using only joins

| name | city |
|-----------------------------|------------|
| Aarhus Airport | Aarhus |
| Copenhagen Kastrup Airport | Copenhagen |
| Esbjerg Airport | Esbjerg |
| Odense Airport | Odense |
| Copenhagen Roskilde Airport | Copenhagen |
| Aalborg Airport | Aalborg |

Subqueries — Remarks

- Not all constructs are absolutely required
 - "*expr* **IN** (subquery)" is equivalent to "*expr* = **ANY** (subquery)"
 - "*expr1* *op* **ANY** (**SELECT** *expr2* **FROM** ... **WHERE** ...)" is equivalent to "**EXISTS** (**SELECT** * **FROM** ... **WHERE** ... **AND** *expr1* *op* *expr2*)"
 -

Overview

- SQL vs. Relational Algebra
- **SQL Queries**
 - Simple queries
 - Set operations
 - Join queries
 - Subqueries
 - **Sorting & rank-based selection**
- Summary

Sorting — ORDER BY

- Sorting tables

- By default, order of tuples in a table is unpredictable!
- Sorting of tuples with **ORDER BY** in ascending order (**ASC**) or descending order (**DESC**)
- Sorting w.r.t. multiple attributes and different orders supported

*Find the GDP per capita for all countries
sorted from highest to lowest.*

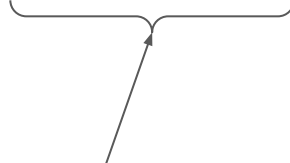
```
SELECT name, (gdp/population) AS gdp_per_capita  
FROM countries  
ORDER BY gdp_per_capita DESC;
```

| name | gdp_per_capita |
|----------------------|----------------|
| Monaco | 250809 |
| Qatar | 148342 |
| Liechtenstein | 131270 |
| United Arab Emirates | 112526 |
| Kuwait | 106256 |
| Macao | 105820 |
| Luxembourg | 100877 |
| Singapore | 84403 |
| ... | ... |

Sorting — ORDER BY

Find all cities sorted by country (ascending from A to Z) and for each country with respect to the cities' population size in descending order.

```
SELECT n.name AS country, c.name AS city, c.population  
FROM cities c, countries n  
WHERE c.country_iso2 = n.iso2  
ORDER BY n.name ASC, c.population DESC;
```



The 2nd sorting criteria only affects result if 1st sorting criteria does not yield an unambiguous order already!

| country | city | population |
|-------------|----------|------------|
| Afghanistan | Kabul | 3678034 |
| Afghanistan | Kandahar | 491500 |
| Afghanistan | Herat | 436300 |
| ... | ... | ... |
| Albania | Tirana | 418495 |
| Albania | Durres | 142432 |
| Albania | Vlore | 130827 |
| ... | ... | ... |
| Zimbabwe | Chivhu | 10263 |
| Zimbabwe | Mazoe | 9966 |
| Zimbabwe | Plumtree | 2148 |

LIMIT & OFFSET — Selection Based on Ranking

- Returning only a portion of the result table
 - **LIMIT** *k*: return the "first" *k* tuples of the result table
 - **OFFSET** *i*: specify the position of the "first" tuple to be considered
 - Typically only meaningful in combination with **ORDER BY**

Find the top-5 countries regarding their GDP per capita for all countries.

```
SELECT name, (gdp/population) AS gdp_per_capita
FROM countries
ORDER BY gdp_per_capita DESC
LIMIT 5;
```

| name | gdp_per_capita |
|----------------------|----------------|
| Monaco | 250809 |
| Qatar | 148342 |
| Liechtenstein | 131270 |
| United Arab Emirates | 112526 |
| Kuwait | 106256 |

LIMIT & OFFSET — Selection Based on Ranking

Find the "second" top-5 countries regarding their GDP per capita for all countries.

```
SELECT name, (gdp/population) AS gdp_per_capita
FROM countries
ORDER BY gdp_per_capita DESC
OFFSET 5
LIMIT 5;
```

| name | gdp_per_capita |
|------------|----------------|
| Macao | 105820 |
| Luxembourg | 100877 |
| Singapore | 84403 |
| Brunei | 77252 |
| Bermuda | 73720 |

- Typical use case: Pagination on websites

Previous 1 2 3 4 5 Next

« 1 2 3 4 5 6 7 »

Items per Page: 3 ▼

< 1 2 3 4 5 ... 11 >

Summary

Find all names that refer to both a city and a country.

(**SELECT** name **FROM** cities)
INTERSECT
(**SELECT** name **FROM** countries);

SELECT n.name
FROM countries n
WHERE EXISTS (**SELECT** c.name
FROM cities c
WHERE c.name = n.name);



A central table with a blue header 'name' and a list of names. Three grey arrows point towards the table from the surrounding SQL queries: one from the top-left (INTERSECT), one from the top-right (EXISTS), and one from the bottom-left (NATURAL JOIN).

| name |
|--------------|
| Singapore |
| Mexico |
| Sierra Madre |
| Monaco |
| Mali |
| Hong Kong |
| China |
| Poland |
| ... |

SELECT DISTINCT(name)
FROM (**SELECT** name **FROM** cities) t1
NATURAL JOIN
(**SELECT** name **FROM** countries) t2;

SELECT name
FROM countries
WHERE name **IN** (**SELECT** name
FROM cities);

Summary

- Querying relational databases with SQL (DQL)
 - Declarative query language
 - Built on top of Relational Algebra
 - This lecture
 - Basic queries (SELECT ... FROM ... WHERE)
 - Set queries and join queries
 - Subqueries
 - Sorting, rank-based selection
 - Next lecture
 - Aggregation, grouping, conditional expressions, extended concepts
- } Direct mapping to RA expression (apart from set vs multiset)