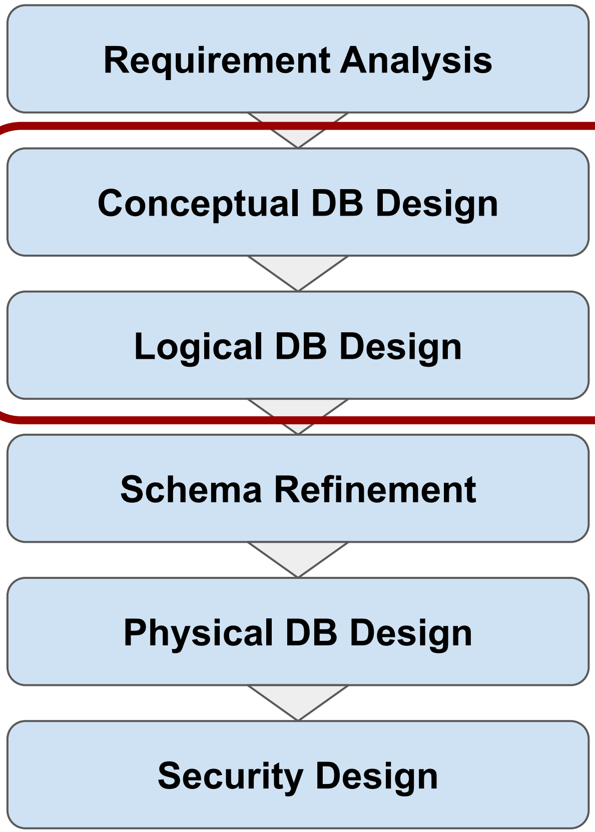
NUS | Computing

National University of Singapore

CS2102: Database Systems

Lecture 4 — SQL (Part 2)

Quick Recap: Database Design Process



- Identification and collection of user needs
- e.g., data /application / performance requirements

- Capturing requirements using a conceptual model
- RDBMS: **Entity Relationship Model (ER Model)**

- Mapping conceptual model to logical schema of DBMS
- RDBMS: Entity Relationship Model → Relational Schema

- Checking schema / tables for redundancies and anomalies

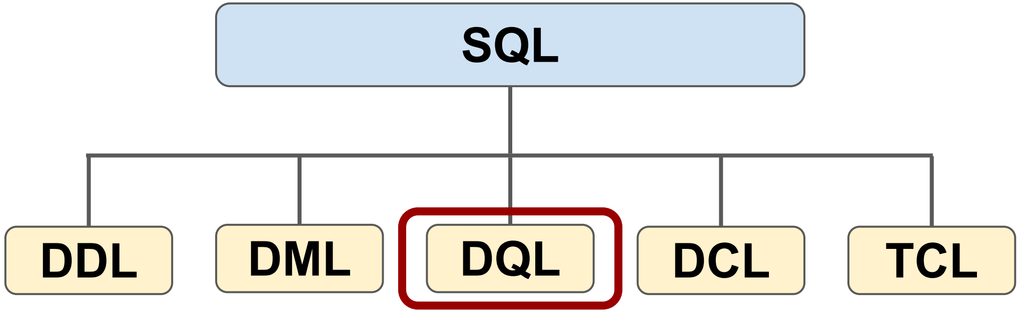
- Implementing database based on final data schema
- Consideration of performance requirements

- Identification users and user groups and their permissions to access which parts of the data

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 …**"



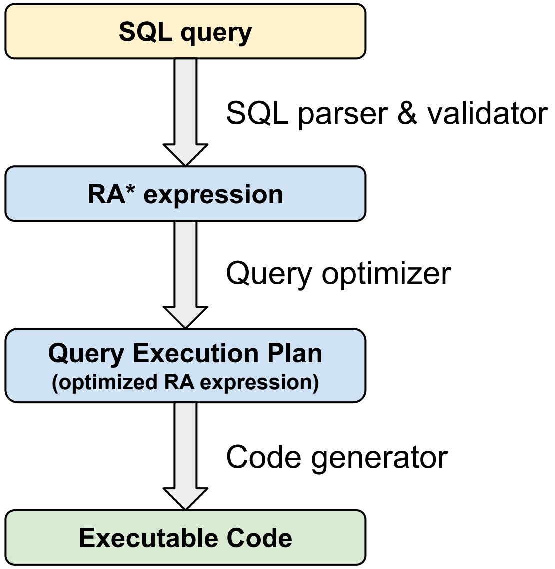
Overview

- **SQL – DQL**
- SQL Queries
  - Simple queries
  - Set operations
  - Join queries
  - Subqueries
  - Sorting & rank-based selection
- Summary

SQL – DQL

- **SQL** (more precisely: the DQL part of SQL)
  - **Declarative** query language for RDBMS
    (Focus on *what* to compute, not on *how* to compute)
  - Multiset / bag semantics
  - Query = SELECT statement

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



* RA = Relational Algebra (Lecture 6)

SQL Query — Basic Form

By default, duplicates are <u>not</u> eliminated; explicit specification of **DISTINCT** required

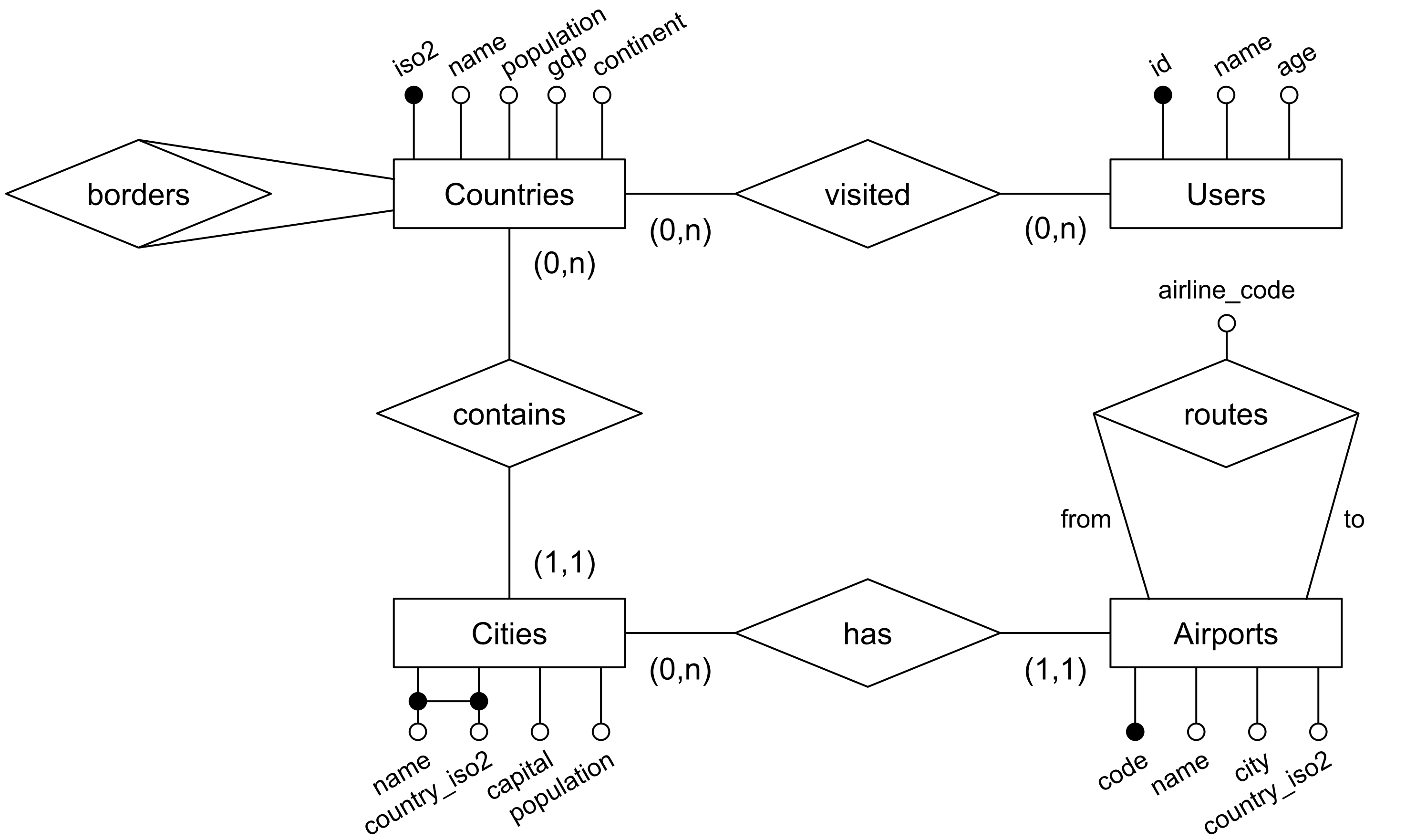
List of attributes/columns of the final output table

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

List of relations that are relevant to answer the given query

List of conditions that specify a valid result row/tuple

Example Database — ER Diagram



Example Database — Data Sample

Countries (196 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 (657 tuples)

country1_iso2	country2_iso2
SG	*null*
AU	*null*
TH	KH
TH	LA
TH	MY
...	...

Airports (3,920 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 (40,138 tuples)

name	country_iso2	capital	population
Singapore	SG	primary	5745000
Kuala Lumpur	MY	primary	8285000
Nanyang	CN	*null*	12010000
Atlanta	US	admin	5449398
Washington	US	primary	5379184
...	...	...	...

Routes (38,588 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 (527 tuples)

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

Overview

- SQL – DQL
- **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
Tokyo	39105000
Jakarta	35362000
Delhi	31870000
Manila	23971000
Sao Paulo	22495000
...	...

40 tuples

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	5873420
Finland	5536146
Ireland	5011500
Norway	5425270
Palestine	5159076
Singapore	5453600
Slovakia	5449270

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	area	gdp	gini	continent
DK	Denmark	5873420	…	…	…	Europe
FI	Finland	5536146	…	…	…	Europe
IE	Ireland	5011500	…	…	…	Europe
NO	Norway	5425270	…	…	…	Europe
PS	Palestine	5159076	…	…	…	Asia
SG	Singapore	5453600	…	…	…	Asia
SK	Slovakia	5449270	…	…	…	Europe

SELECT Clause — Expressions

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

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

"||" concatenates strings

"**AS**" is optional

SELECT name, 'S$ ' || **ROUND**((gdp / population)*1.28) **AS** gdp_per_capita
FROM countries;

Convert from USD to SGD
(as of August 2025)

name	gdp_per_capita
Denmark	S$ 90342
Germany	S$ 66452
Kyrgyzstan	S$ 1715
Norway	S$ 86351
Singapore	S$ 87872
Slovakia	S$ 29912
Turkmenistan	S$ 9075
United Arab Emirates	S$ 69134
...	...

196 (all countries)

SELECT Clause — Duplicates

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

- Multiset / bad nature of SQL
  - By default, SQL does not eliminate duplicates
  - Use keyword **DISTINCT** to enforce duplicate elimination

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

```
SELECT country_iso2 AS code
FROM cities;
```

40,138 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;
```

193 tuples

code
MX
ID
IN
PH
CN
TH
...

SELECT Clause — Duplicates with NULL Values

x	y	x <> y	x IS DISTINCT FROM y
1	1	**FALSE**	**FALSE**
1	2	**TRUE**	**TRUE**
null	1	**null**	**TRUE**
null	*null*	**null**	**FALSE**

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

  "(n_1 **IS DISTINCT FROM** n_2)" or "(c_1 **IS DISTINCT FROM** c_2)"

  evaluates to TRUE

SELECT name, type
FROM cities;

40,138 tuples (all cities)

name	type
Mexico City	primary
Jakarta	primary
Perth	admin
Perth	*null*
Perth	*null*
Shenzhen	minor
...	...

SELECT DISTINCT name, type
FROM cities;

39,466 tuples

name	type
Tokyo	primary
Jakarta	primary
Perth	admin
Perth	***null***
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;
```

38 tuples

code
AU
SG
BH
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 – DQL
- **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 **UNION** Q_2 = $Q_1 \cup Q_2$
  - Q_1 **INTERSECT** Q_2 = $Q_1 \cap Q_2$
  - Q_1 **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

value
1
2
2

S

value
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
(SELECT name FROM countries);
```

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

Find the codes of all the countries for which they have no city in the database.

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

iso2
NA
EH
PS

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
PER	Perth Int. Airport	Perth	AU
JAD	Perth Jandakot Airport	Perth	AU
PSL	Perth/Scone Airport	Perth	GB

Overview

- SQL – DQL
- **SQL Queries**
  - Simple queries
  - Set operations
  - **Join queries**
  - Subqueries
  - Sorting & rank-based selection
- Summary

Multi-Table Queries (Join Queries)

- So far: only single-table queries
  - Each SQL statement only had one table in the FROM clause
  - Some set queries contain multiple tables but in each in a different FROM clause

- Often: required information across multiple table ➜ multi-query queries
  - Example: *"Find all countries where at least one neighboring country has a larger population."*

Countries (196 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 (657 tuples)

country1_iso2	country2_iso2
SG	*null*
AU	*null*
TH	KH
TH	LA
TH	MY
...	...

Multi-Table Queries (Join Queries)

- Basic SQL syntax
  - Multiple table names in the same FROM clause
  - Very common and always recommended: use of aliases
- Cross product / Cartesian product
  - Multi-table query with WHERE clause
  - Computes all possible pairs of tuples

SELECT c.name, n.name
FROM cities **AS** c, countries **AS** n;

Returns all combinations of city and country names

name	name
Tokyo	Albania
Tokyo	Algeria
Tokyo	Andorra
Tokyo	Angola
Tokyo	Antigua and Barbuda
Tokyo	Argentina
Tokyo	Armenia
Tokyo	Australia
Tokyo	Austria
Tokyo	Azerbaijan
Tokyo	Bangladesh
Tokyo	Belize
...	...

7,867,048 tuples

Multi-Table Queries (Join Queries)

comparison between 2 attributes ("=", "<>", "<", "<=", ">=", ">")

- Multi-table queries
  - Most common in practice: cross product + attribute selection ➜ **join**

"**AS**" is optional

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;
```

```
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
...	...

40,138 tuples (all cities)

Multi-Table Queries (Join Queries)

Find the names and the population of all countries with directly neighboring countries that have a larger population. Include the neighbors and their population as well.

```
SELECT c1.name, c1.population, c2.name, c2.population
FROM countries c1, borders b, countries c2
WHERE c1.iso2 = b.country1_iso2
AND c2.iso2 = b.country2_iso2
AND c1.population < c2.population;
```

2 join conditions needed here!

name	population	name	population
Andorra	79535	Spain	47450795
Andorra	79535	France	67413000
United Arab Emirates	9282410	Saudi Arabia	34218000
Afghanistan	40218234	People's Republic of China	1412600000
Afghanistan	40218234	Iran	83183741
...	...	…	…

309 tuples

Multi-Table Queries (Join Queries)

- Natural Joins
  - Join condition (attribute selection) implied by attribute names
  - Natural joins only defined for equality comparisons ("=")
  - Result does contain only one instance of matching attributes

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

Why?

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

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

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

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

Multi-Table Queries (Join Queries)

- Outer Joins
  - Sometimes we are interest in tuples that do not have a match in another table
  - Important: this is not the same as using "<>" for the join condition

- 3 basic types:
  - LEFT OUTER JOIN = INNER JOIN + all remaining tuples from the left table
  - RIGHT OUTER JOIN = INNER JOIN + all remaining tuples from the right table
  - FULL OUTER JOIN = INNER JOIN + all remaining tuples from both tables

Missing values get filled with NULL values

Multi-Table Queries (Join Queries)

- Outer Joins – basic examples

R

x
1
2
3
4
5

S

y
3
4
5
6
6

SELECT x, y
FROM R **LEFT OUTER JOIN** S
ON x = y;

x	y
1	null
2	null
3	1
4	2
5	3

SELECT x, y
FROM R **RIGHT OUTER JOIN** S
ON x = y;

x	y
3	3
4	4
5	5
null	6
null	7

SELECT x, y
FROM R **FULL OUTER JOIN** S
ON x = y;

x	y
1	null
2	null
3	3
4	4
5	5
null	6
null	7

INNER JOIN tuples

Multi-Table Queries (Join Queries)

- Outer Joins – practical example
  - Note: LEFT OUTER JOIN and RIGHT OUTER JOIN just mirrored version
    (strictly speaking, we do not need both, but having both is consistent and flexible)

Find the all the countries for which there is no city in the database.

optional

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

keep only the dangling tuples

name
Namibia
Palestine
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 c.country_iso2 = a.country_iso2
         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;
```

attribute selections for join operations

All airports in European countries without a land border (4 tuples)

country	city	airport
Saint Lucia	Castries	George F. L. Charles Airport

Join Queries — Remarks

- In practice
  - Join condition very often along foreign key relationships
  - Most common comparison for join conditions: "=" (equality)
  - NATURAL JOIN not really needed and may yield unexpected results if you are not careful
    (it is typically "safer" to specify all join conditions explicitly – even if the query gets longer)

Overview

- SQL – DQL
- **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
...	...

38 tuples

- Subquery expressions
  - **IN** subqueries
  - **EXISTS** subqueries
  - **ANY/SOME** subqueries
  - **ALL** subqueries

Quick Quiz: How can we rewrite the query without the column aliases but yielding the same result?

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

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

outer query

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

inner query

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

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);
```

iso2
NA
EH
PS

- 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	area	gdp	gini	continent
DK	Denmark	5873420	…	…	…	Europe
FI	Finland	5536146	…	…	…	Europe
IE	Ireland	5011500	…	…	…	Europe
NO	Norway	5425270	…	…	…	Europe
PS	Palestine	5159076	…	…	…	Asia
SG	Singapore	5453600	…	…	…	Asia
SK	Slovakia	5449270	…	…	…	Europe

ANY/SOME Subqueries (ANY and SOME are synonymous)

All Londons

name	country	population
London	GB	11120000
London	CA	383822
London	US	37714

- **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 3 cities called "London" on the database).

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

name	population
Singapore	5453600
Portugal	10344802
Sweden	10402070
Brunei	460345
Bhutan	754388
...	

113 tuples

ALL Subqueries

All Londons

name	country	population
London	GB	11120000
London	CA	383822
London	US	37714

- **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 <u>all</u> subquery rows

Find all countries with a population size smaller than <u>all</u> cities called "London" (there are actually 3 cities called "London" on the database).

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

name	population
Nauru	10834
Palau	17907
San Marino	33600
Tuvalu	11900
Vatican City	453

Correlated Subqueries

Quick Quiz: Wait a minute, why is "Europe" and "Africa" missing from the result set?

- 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	1748000000000
Brazil	South America	1810000000000
China	Asia	19910000000000
United States	North America	25350000000000

Correlated Subqueries

- Correlated subquery
  - ALL condition must be true for all (duh!) result of the subquery
  - Problematic if subquery contains NULL value ➜ condition never evaluates to TRUE

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
                  AND c2.gdp IS NOT NULL);
```

name	continent	gdp
Australia	Oceania	1748000000000
Brazil	South America	1810000000000
China	Asia	19910000000000
Germany	Europe	4319000000000
Nigeria	Africa	498060000000
United States	North America	25350000000000

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
                  AND c2.gdp IS NOT NULL);
```

- **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
                  AND c2.gdp IS NOT NULL);
```

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
                  AND c2.gdp IS NOT NULL);
```

name	continent	gdp
China	Asia	21100000000000

Scoping Rules Gone Wrong

Quick Quiz: Can you explain the result of the 3rd query?

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
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

(✗)

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

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 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
...

196 tuples (all countries)

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);
```

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

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);
```

name
Namibia
West Sahara
Palestine

- Rule of thumb
  - (**NOT**) **EXISTS** subqueries are generally always correlated
  - Uncorrelated (**NOT**) **EXISTS** subqueries are either wrong or unnecessary

Scalar Subqueries

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

- Scalar subquery — definition
  - Subquery returns a single value (i.e., table 1 row with 1 column)
  - Can be used as a 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;
```

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

40,138 tuples

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 3 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!)

city	country	population
Funafuti	TV	6025
San Marino	SM	4040
Vatican City	VA	825
Yaren	NR	NULL
Ngerulmud	PW	271
...	...	...

15 tuples

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	67413000	3140000000000
Germany	83190556	4319000000000

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'));
```

name	population	gdp
China	1412600000	19910000000000
Turkey	84680273	692000000000
Nigeria	211400708	498060000000
Vietnam	96208984	340602000000
United States	331893745	25350000000000
...	...	...

19 tuples

Note: For the <, <=, > and >= cases, the row elements are compared left-to-right, stopping as soon as an unequal or null pair of elements is found.
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')
               );
```

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

```
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

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 – DQL
- **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
WHERE gdp is NOT NULL
ORDER BY gdp_per_capita DESC;
```

name	gdp_per_capita
Monaco	193838
Liechtenstein	107612
Luxembourg	107612
Ireland	102963
Switzerland	87396
Qatar	84513
Brunei	77235
…	…

194 tuples

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
AND c.population IS NOT NULL
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	4273156
Afghanistan	Kandahar	614254
Afghanistan	Herat	556205
...	...	...
Albania	Tirana	418495
Albania	Vlore	130827
Albania	Kamez	126777
...	...	...
Zimbabwe	Chivhu	10263
Zimbabwe	Mazoe	9966
Zimbabwe	Plumtree	2148

39,493 tuples

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
WHERE gdp IS NOT NULL
ORDER BY gdp_per_capita DESC
LIMIT 5;
```

name	gdp_per_capita
Monaco	193838
Liechtenstein	176676
Luxembourg	107612
Ireland	102963
Switzerland	87396

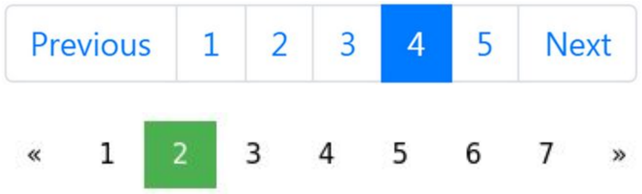
LIMIT & OFFSET — Selection Based on Ranking

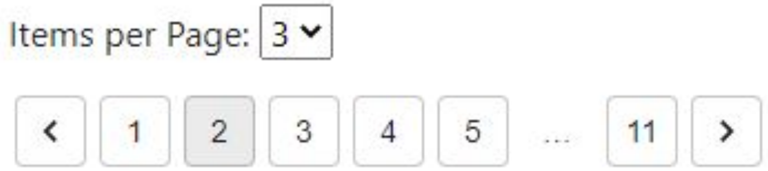
Find the "second" top-5 countries regarding their GDP per capita for all countries.

```
SELECT name, (gdp/population) AS gdp_per_capita
FROM countries
WHERE gdp IS NOT NULL
ORDER BY gdp_per_capita DESC
OFFSET 5
LIMIT 5;
```

name	gdp_per_capita
Qatar	84513
Brunei	77235
United States	76379
Denmark	70580
Singapore	68650

- Typical use case: Pagination on websites





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);
```

name
Singapore
Mexico
Peru
Monaco
Mali
El Salvador
China
Poland
...

29 tuples

```
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 (Lecture 6)
- 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

Quick Quiz Solutions

Quick Quiz (Slide 13)

Quick Quiz (Slide 26)

Quick Quiz (Slide 33)

Quick Quiz (Slide 34)

Quick Quiz (Slide 39)

Quick Quiz (Slide 43)

Quick Quiz (Slide 46)