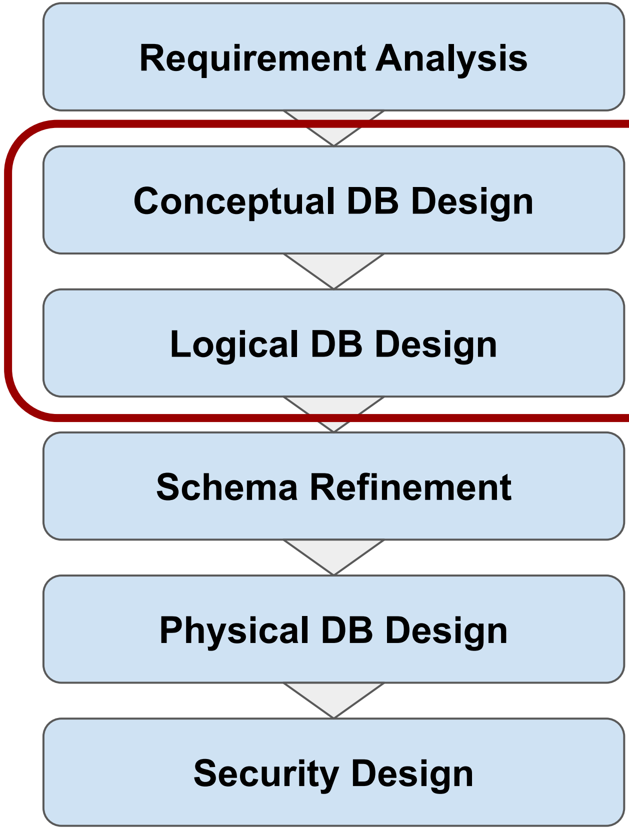
NUS | Computing

National University of Singapore

CS2102: Database Systems

Lecture 5 — SQL (Part 2)

Quick Recap: Database Design Process



Requirement Analysis

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

Conceptual DB Design

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

Logical DB Design

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

Schema Refinement

- Checking schema / tables for redundancies and anomalies

Physical DB Design

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

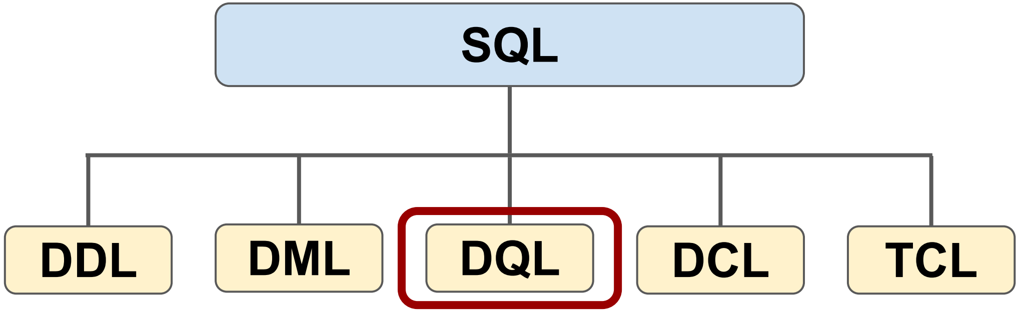
Security Design

- 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 vs. Relational Algebra**

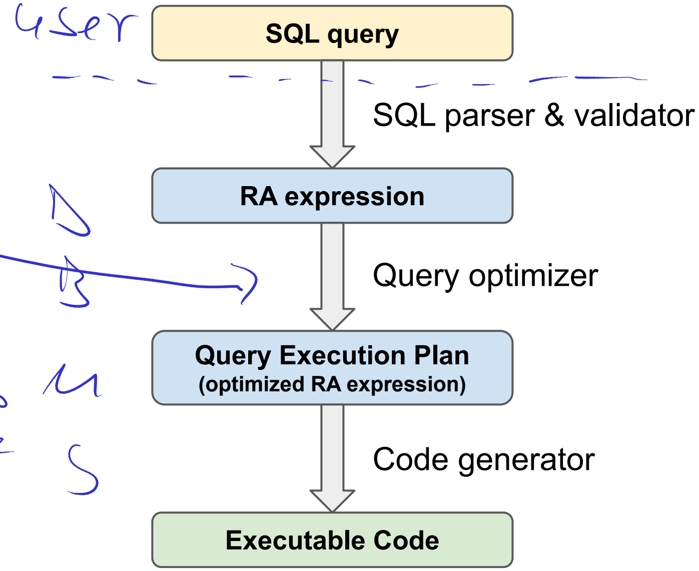
- SQL Queries
  - Simple queries
  - Set operations
  - Join queries
  - Subqueries
  - Sorting & rank-based selection

- 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 ]
```



user

SQL query

SQL parser & validator

RA expression

Query optimizer

Query Execution Plan
(optimized RA expression)

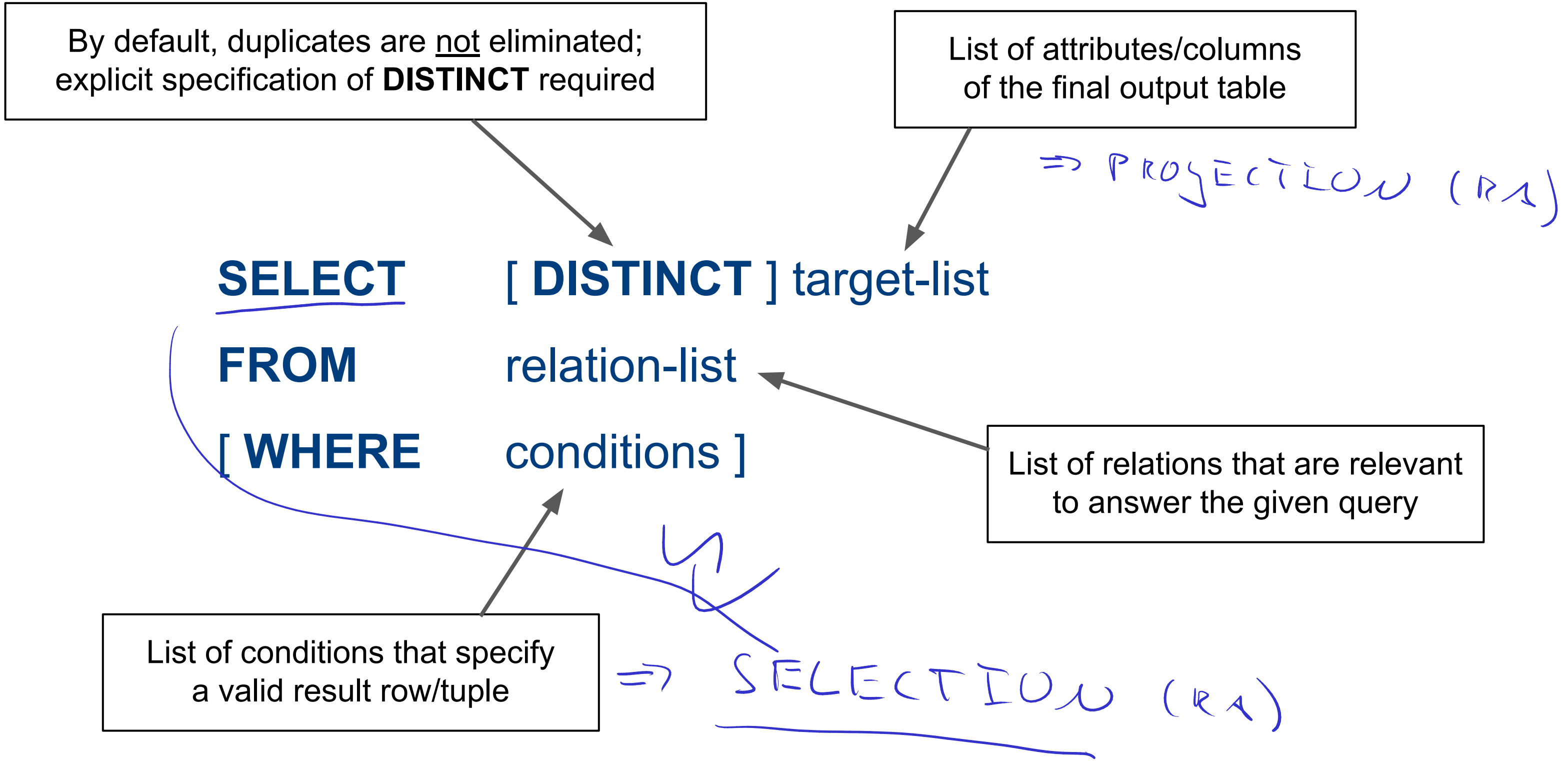
Code generator

Executable Code

D
B
M
S

finding the best op tree

SQL Query — Basic Form



By default, duplicates are not eliminated; explicit specification of **DISTINCT** required

List of attributes/columns of the final output table

⇒ PROJECTION (RA)

```
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

⇒ SELECTION (RA)

SQL Query — Conceptual Evaluation Strategy

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

1. Compute cross product of all tables in **relation-list**

SELECTION

2. Discard all rows/tuples that fail any of the **conditions**

PROJECTION

3. Delete attributes/columns not specified in **target-list**
4. If **DISTINCT**, eliminate duplicate rows/tuples

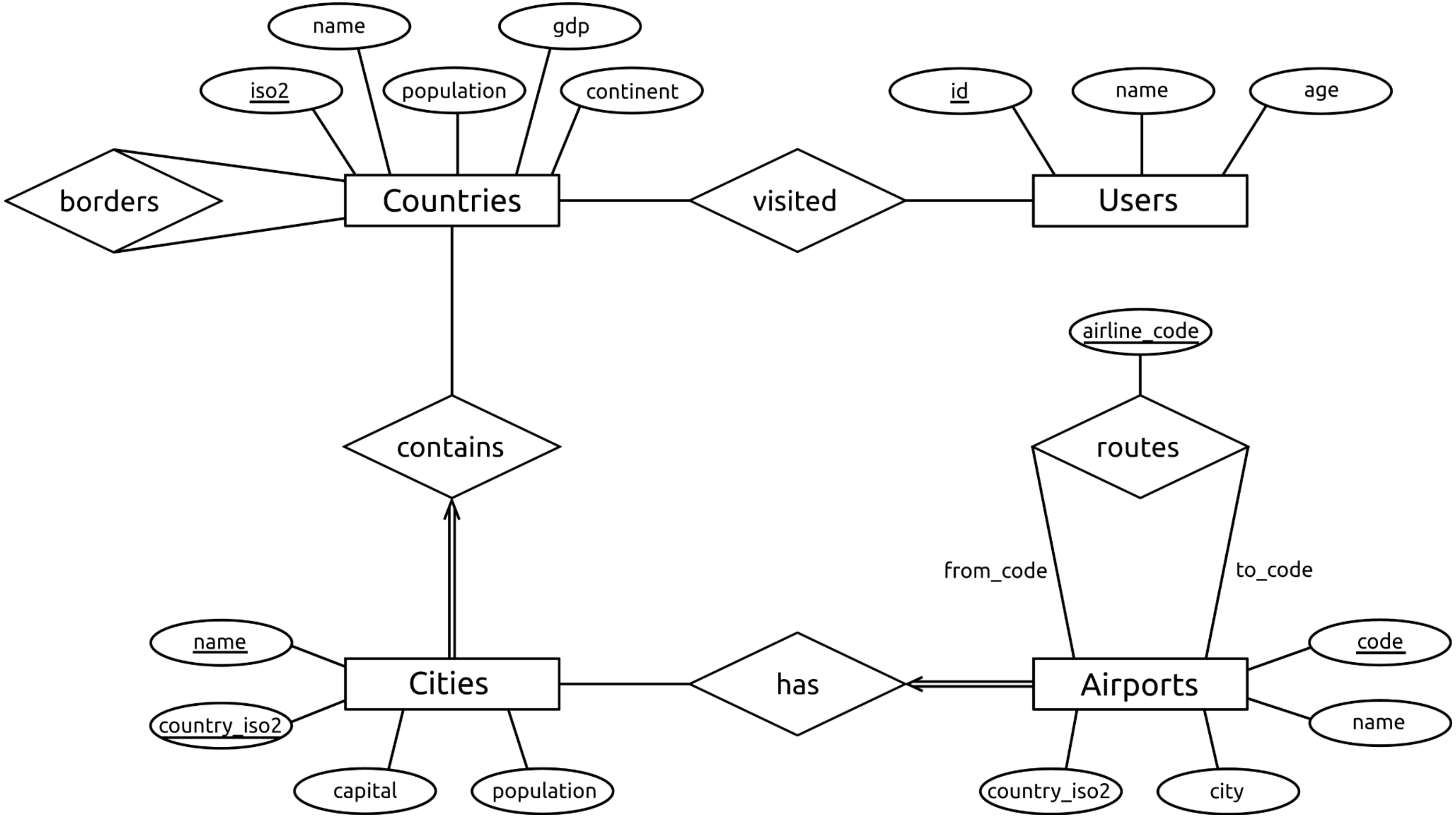
- Mapping between basic SQL query to corresponding RA expression
  - Conceptual mapping; no consideration of performance

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	*null*
AU	*null*
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	*null*	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

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

Convert from USD to SGD

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

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 **IS DISTINCT FROM** n_2)" or "(c_1 **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	*null*
Perth	*null*
Shenzhen	minor
...	...

SELECT DISTINCT name, capital
FROM cities;

24,047 tuples

name	capital
Tokyo	primary
Jakarta	primary
Delhi	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;
```

code
AU
SG
TW
PH
NZ
JP
...

68 tuples

UPDATE cities
SET capital = NULL

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

code

evaluates to unknown

0 tuples (but no error!)

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';
```

Sire ✓

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 **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 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.

"**AS**" is optional

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

equivalent queries

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

INNER JOIN ..., ON

implies INNER JOIN

```
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.

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.

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

12 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;
```

attribute selections for join operations

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

subquery ⇒ table

4 tables ⇓ 3 attribute selections

AS optional

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

outer query

```
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);
```

inner query

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

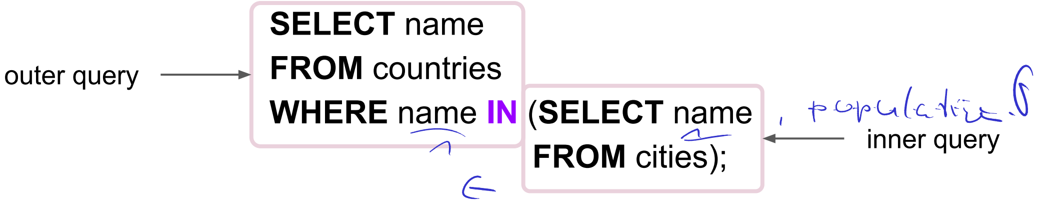
- Subquery expressions
  - **IN** subqueries
  - **EXISTS** subqueries
  - **ANY/SOME** subqueries
  - **ALL** subqueries

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 <u>any</u> subquery row
  - **NOT IN** returns true if *expr* matches with <u>no</u> subquery row

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



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

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 join~~t~~s

↑

(much faster)

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 <u>all</u> subquery rows

Find all countries with a population size smaller than <u>all</u> 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');
```

uncorrelated

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);
```

"Europe"

Singapore

215

GERMANY

France

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 , c2.name
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.

optional

c1. c1. c1.

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

c2.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

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

⇒ Key constraints

- 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
...	...

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!)

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

19 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	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	21100000000000
Thailand	68200824	1160000000000
Nigeria	186053386	1090000000000
Vietnam	95261021	595000000000
United States	323995528	18600000000000
...	...	...

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')
              );
```

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 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.

```sql
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

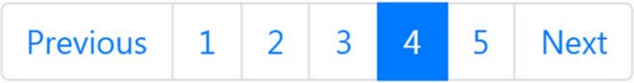
LIMIT & OFFSET — Selection Based on Ranking

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

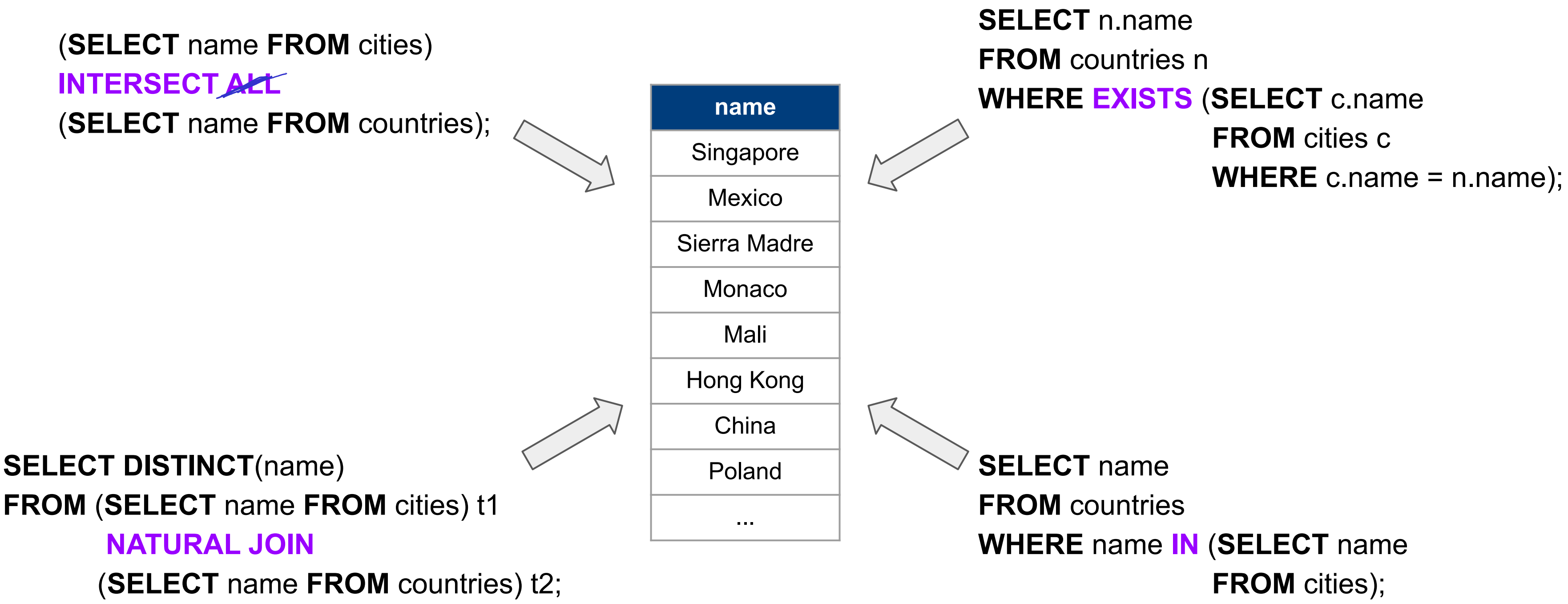
- Typical use case: Pagination on websites



Items per Page: 3

Summary

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



```
(SELECT name FROM cities)
INTERSECT ~~ALL~~
(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
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 queried

    Direct mapping to RA expression (apart from set vs multiset)
  - Subqueries
  - Sorting, rank-based selection

- Next lecture
  - Aggregation, grouping, conditional expressions, extended concepts