NUS | Computing

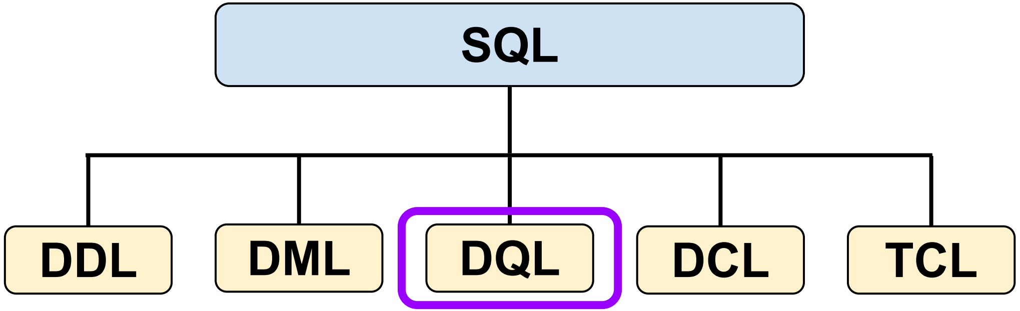
National University of Singapore

CS2102: Database Systems

Lecture 5 — SQL (Part 3)

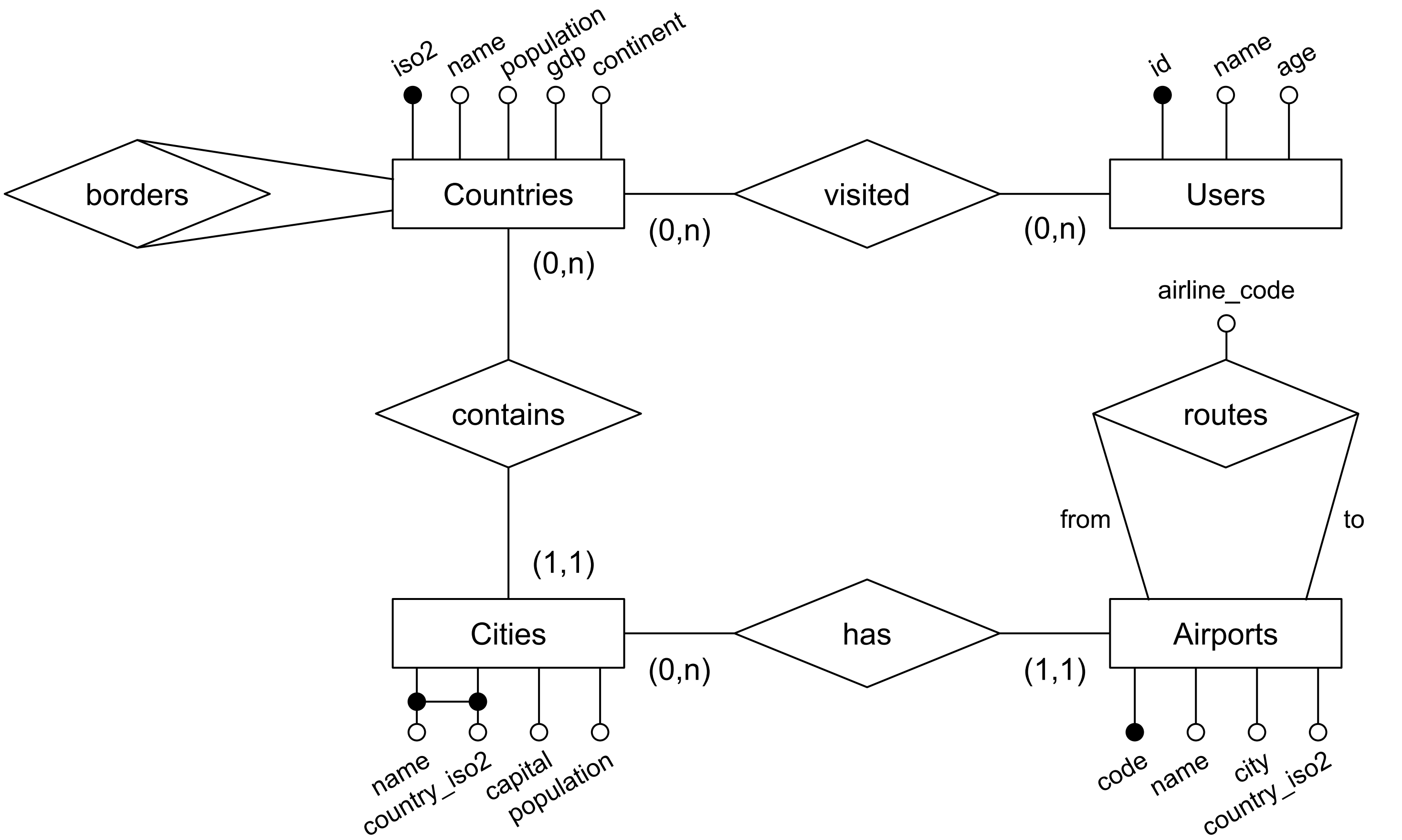
Quick Recap: Where We are Right Now

- Querying a database
  - Extracting information using SQL (DQL: data query language)
  - Anything with "**SELECT …**"



- Covered constructs
  - Basic queries: **SELECT** [**DISTINCT**] … **FROM** [**WHERE**]
  - Multirelational queries / join queries: (**INNER**) **JOIN**, **NATURAL JOIN**, **OUTER JOIN**, etc
  - Subquery expressions: (**NOT**) **IN**, (**NOT**) **EXISTS**, **ANY/SOME**, **ALL**
  - Sorting & rank-based selection: **ORDER BY**, **LIMIT**, **OFFSET**

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

- **Common SQL constructs**
  - **Aggregation**
  - Grouping
  - Conditional Expressions
- Structuring Queries
  - Common Table Expressions
  - Views
- Extended concepts
  - Universal Quantification
  - Recursive Queries
- Summary

Aggregation

- Aggregate functions
  - Compute a single value from a set of tuples
  - Examples: **MIN**(), **MAX**(), **AVG**(), **COUNT**(), **SUM**()

Find find the lowest and highest population sizes among all countries, as well as the global population size (= sum over all countries).

```
SELECT MIN(population) AS lowest,
       MAX(population) AS highest,
       SUM(population) AS global
FROM countries;
```

lowest	highest	global
453	1412600000	7712195627

Aggregation — Interpretation of NULL values

...	A	...
...	3	...
...	*null*	...
...	42	...
...	0	...
...	3	...

- Let *R* be a non-empty relation with attribute *A*

Query	Interpretation	Result
SELECT MIN(A) **FROM** R;	Minimum non-null value in A	0
SELECT MAX(A) **FROM** R;	Maximum non-null value in A	42
SELECT AVG(A) **FROM** R;	Average of non-null values in A	12
SELECT SUM(A) **FROM** R;	Sum of non-null values in A	48
SELECT COUNT(A) **FROM** R;	Count of non-null values in A	4
SELECT COUNT(*) **FROM** R;	Count of rows in R	5
SELECT AVG(**DISTINCT** A) **FROM** R;	Average of distinct non-null values in A	15
SELECT SUM(**DISTINCT** A) **FROM** R;	Sum of distinct non-null values in A	45
SELECT COUNT(**DISTINCT** A) **FROM** R;	Count of distinct non-null values in A	3

Aggregation — Interpretation of NULL values

Table R

...	A	...

Table S

...	A	...
...	*null*	...
...	*null*	...
...	*null*	...
...	*null*	...
...	...	...

- Let *R*, *S* be two relations with an attribute *A*
  - Let R be an empty relation
  - Let *S* be a non-empty relation with *n* tuples but only null values for *A*

Query	Result
SELECT MIN(A) **FROM** R;	null
SELECT MAX(A) **FROM** R;	null
SELECT AVG(A) **FROM** R;	null
SELECT SUM(A) **FROM** R;	null
SELECT COUNT(A) **FROM** R;	0
SELECT COUNT(*) **FROM** R;	0

Query	Result
SELECT MIN(A) **FROM** S;	null
SELECT MAX(A) **FROM** S;	null
SELECT AVG(A) **FROM** S;	null
SELECT SUM(A) **FROM** S;	null
SELECT COUNT(A) **FROM** S;	0
SELECT COUNT(*) **FROM** S;	***n***

Aggregation — More Examples

Find the first and last city in the United States with respect to their lexicographic sorting.

```
SELECT MIN(name) AS lexi_first, MAX(name) AS lexi_last
FROM cities
WHERE country_iso2 = 'US';
```

lexi_first	lexi_last
Abbeville	Zuni Pueblo

Find the number countries with at least 10% of the population compared to the country with the largest population size.

```
SELECT COUNT(*) AS num_big_countries
FROM countries
WHERE population >= 0.1 * (SELECT MAX(population)
                           FROM countries);
```

Scalar subquery!

num_big_countries
9

Aggregate Functions — Signatures

- Data type of attribute/column of a table affects:
  - Applicability of aggregate functions
  - Return data type of aggregate functions

- Examples
  - **MIN**(), **MAX**() defined for all data types; return data type same as input data type
  - **SUM**() defined for all numeric data types; **SUM**(INTEGER)→BIGINT, **SUM**(REAL)→REAL, …
  - **COUNT**() defined for all data types; **COUNT**(...)→BIGINT
  - …

Complete list for PostgreSQL: https://www.postgresql.org/docs/current/functions-aggregate.html

Overview

- **Common SQL constructs**
  - Aggregation
  - **Grouping**
  - Conditional Expressions
- Structuring Queries
  - Common Table Expressions
  - Views
- Extended concepts
  - Universal Quantification
  - Recursive Queries
- Summary

Grouping — GROUP BY Clause

- Aggregation so far
  - Application of aggregate functions over <u>all</u> tuples of a relation
  - Result relation has only <u>one</u> tuple

➜ Grouping using **GROUP BY**
  - Logical partition of relation into groups based on values for specified attributes
  - In principle, always applied together with aggregation
    (**GROUP BY** without aggregation valid but typically not meaningful)
  - Application of aggregation functions over each group
  - One result tuple for each group

GROUP BY — Example

For each continent, find the lowest and highest population sizes among all countries, as well as the overall population size for that continent.

Logical partition of "Countries" w.r.t. "continent"

iso2	name	population	area	gdp	gini	continent
DZ	Algeria	44700000	…	…	…	**Africa**
AO	Angola	33086278	…	…	…	**Africa**
…	…	…	…	…	…	…
AF	Afghanistan	40218234	…	…	…	**Asia**
BH	Bahrain	1569446	…	…	…	**Asia**
…	…	…	…	….	…	…
AR	Argentina	45605826	…	..	…	**South America**
BO	Bolivia	11428245	…	…	…	**South America**
…	…	…	…	…	…	…
BS	Bahamas	400516	…	…	…	**North America**
CA	Canada	38526760	…	…	…	**North America**
…	…	…	…	…	…	…
…	…	…	…	…	…	**Europe**

```
SELECT continent,
       MIN(population) AS lowest,
       MAX(population) AS highest,
       SUM(population) AS overall
FROM countries
GROUP BY continent;
```

continent	lowest	highest	overall
Africa	99331	211400708	1354025807
Asia	579330	1412600000	4554731303
South America	575990	212688125	430763036
North America	52441	331893745	585036622
Europe	453	145478097	745055194
Oceania	10834	25997100	42583665

GROUP BY — Example

For each route, find the number of airlines that serve that route.

Logical partition of "Routes" w.r.t. "from_code" and "to_code"

from_code	to_code	airline_code
SIN	**FRA**	SQ
SIN	**FRA**	LH
SIN	**FRA**	US
PEK	**SIN**	CA
PEK	**SIN**	SQ
MNL	**SIN**	3K
MNL	**SIN**	5J
MNL	**SIN**	PR
MNL	**SIN**	SQ
MNL	**SIN**	TR
SIN	**ADL**	ET
SIN	**ADL**	SQ
SIN	**ADL**	VA
SIN	**HEL**	AY
...	...	...

```
SELECT from_code, to_code,
       COUNT(*) AS num_airlines
FROM routes
GROUP BY from_code, to_code;
```

from_code	to_code	num_airlines
SIN	**FRA**	3
PEK	**SIN**	2
MNL	**SIN**	5
SIN	**ADL**	3
SIN	**HEL**	1
MNL	**KLO**	6
ATL	**JFK**	10
KUL	**BKK**	9
...	...	...

20,326 tuples

GROUP BY Clause — Defining Groups

- Given "**GROUP BY** a_1, a_2, …, a_n", 2 tuples t and t' belong to the same group if

  "($t.a_1$ **IS NOT DISTINCT FROM** $t'.a_1$)" and
  "($t.a_2$ **IS NOT DISTINCT FROM** $t'.a_2$)" and
  … and
  "($t.a_n$ **IS NOT DISTINCT FROM** $t'.a_n$)"

  evaluates to "true"

- Example:
  - Table *R* with three attributes *A*, *B*, *C*

A	B	C
null	4	19
6	1	*null*
20	2	10
1	1	2
1	18	2
null	21	19
6	20	*null*

SELECT ...
FROM R
GROUP BY A, C;

→

A	B	C
null	4	**19**
null	21	**19**
6	1	***null***
6	20	***null***
20	2	**10**
1	1	**2**
1	18	**2**

GROUP BY Clause — Restrictions to SELECT Clause

- If column A_i of table *R* appears in the **SELECT** clause, one of the following conditions must hold:
  - A_i appears in the **GROUP BY** clause
  - A_i appears as input of an aggregation function in the **SELECT** clause
  - The primary key ~~or a candidate key~~ of R appears in the **GROUP BY** clause

Valid in standard SQL but not supported by PostgreSQL.
In this module, we follow PostgreSQL's tighter restriction

Example of an **invalid** query:

SELECT continent, gdp**,** **SUM**(population)
FROM countries
GROUP BY continent;

GROUP BY — Grouping over Primary Key

- Assume table "Countries" was created as shown on the right

```
CREATE TABLE Countries (
        iso2            CHAR(2) PRIMARY KEY,
        name            VARCHAR(255) UNIQUE,
        population      INTEGER,
        gdp             BIGINT,
        continent       VARCHAR(255)
);
```

This query is **valid**!

```
SELECT name, population, COUNT(*)
FROM countries
GROUP BY iso2;
```

Quick Quiz: What is the "problem" with this query?

This query is **valid** SQL standard but **invalid** PostgreSQL!

```
SELECT name, population, COUNT(*)
FROM countries
GROUP BY name;
```

GROUP BY — Grouping over Primary Key

- Assume table "Countries" was created as shown on the right
  - No key constraints on "Cities"

```
CREATE TABLE Countries (
        iso2            CHAR(2) PRIMARY KEY,
        name            VARCHAR(255) UNIQUE,
        population      INTEGER,
        gdp             BIGINT,
        continent       VARCHAR(255)
);
```

This query is **valid**!

```
SELECT n.name, n.population, COUNT(*)
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.iso2;
```

This query is **invalid**!

```
SELECT n.name, c.name, COUNT(*)
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.iso2;
```

This query is **valid**!

```
SELECT n.name, n.population, COUNT(*)
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.iso2;
```

This query is **invalid**!

```
SELECT n.name, c.name, COUNT(*)
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.iso2;
```

n.iso2	n.name	n.population	...	c.name	c.country_iso2	c.population	...
BS	Bahamas	400516	...	Nassau	BS	274400	...
BS	Bahamas	400516	..	Freeport City	BS	45945	...
BS	Bahamas	400516	...	Marsh Harbour	BS	6283	...
SG	Singapore	5453600	...	Singapore	SG	5271000	...
DJ	Djibouti	921804	...	Djibouti	DJ	562000	...
DJ	Djibouti	921804	...	Arta	DJ	null	...
DJ	Djibouti	921804	...	Ali Sabieh	DJ	37939	...
DJ	Djibouti	921804	...	Dikhil	DJ	35000	...
DJ	Djibouti	921804	...	Obock	DJ	21200	...
DJ	Djibouti	921804	...	Tadjourah	DJ	14820	...
AU	Australia	25997100	...	Sydney	AU	4840600	...
AU	Australia	25997100	...	Melbourne	AU	4529500	...
	...	...					...

HAVING Clause — Conditions over Groups

- **HAVING** conditions
  - Conditions check for each group defined by **GROUP BY** clause
  - **HAVING** clause cannot be used without a **GROUP** BY clause
  - Conditions typically involve aggregate functions

Find all routes that are served by more than 12 airlines.

```
SELECT from_code, to_code,
       COUNT(*) AS num_airlines
FROM routes
GROUP BY from_code, to_code
HAVING COUNT(*) > 12;
```

from_code	to_code	num_airlines
ORD	ATL	20
ATL	ORD	19
ORD	MSY	13
HKT	BKK	13

HAVING Clause — Conditions over Groups

Find all countries that have at least one city with a population size larger than the average population size of all European countries

```
SELECT n.name, n.continent
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.name, n.continent
HAVING MAX(c.population) > (SELECT AVG(population)
                           FROM countries
                           WHERE continent = 'Europe');
```

name	continent
Bangladesh	Asia
Japan	Asia
Mexico	North America
India	Asia
Egypt	Africa
Philippines	Asia
Russia	Europe
Thailand	Asia
China	Asia
Brazil	South America
Argentina	South America
South Korea	Asia
Indonesia	Asia
United States	North America

GROUP BY Clause — Restrictions to HAVING Clause

- If column A_i of table *R* appears in the **HAVING** clause, one of the following conditions must hold:
  - A_i appears in the **GROUP BY** clause
  - A_i appears as input of an aggregation function in the **HAVING** clause
  - The primary key ~~or a candidate key~~ of R appears in the **GROUP BY** clause

Valid Queries

```
SELECT continent, COUNT(*)
FROM countries
GROUP BY continent
HAVING AVG(population) > 25000000;
```

```
SELECT continent, COUNT(*)
FROM countries
GROUP BY continent
HAVING continent = 'Asia';
```

```
SELECT continent, COUNT(*)
FROM countries
GROUP BY iso2
HAVING name = 'China';
```

Invalid Query

```
SELECT continent, COUNT(*)
FROM countries
GROUP BY continent
HAVING name = 'China';
```

Quick Quiz: What is the result of this query?

Conceptual Evaluation of Queries

FROM	Compute cross-product of all tables in FROM clause
WHERE	Filter tuples that evaluate to true on the WHERE condition(s)
GROUP BY	Partition table into groups w.r.t. to the grouping attribute(s)
HAVING	Filter groups that evaluate to true on the HAVING condition(s)
SELECT	Remove all attributes not specified in SELECT clause
ORDER BY	Sort tables based on specified attribute(s)
LIMIT/OFFSET	Filter tuples based on their order in the table

Overview

- **Common SQL constructs**
  - Aggregation
  - Grouping
  - **Conditional Expressions**
- Structuring Queries
  - Common Table Expressions
  - Views
- Extended concepts
  - Universal Quantification
  - Recursive Queries
- Summary

CASE — Conditional Expressions

- CASE expression
  - Generic conditional expression
  - Similar to case or if/else statements in programming languages

- Two basic ways for formulating CASE expressions

CASE
    WHEN $condition_1$ **THEN** $result_1$
    WHEN $condition_2$ **THEN** $result_2$
    …
    WHEN $condition_n$ **THEN** $result_n$
    ELSE $result_0$
END

CASE *expression*
    WHEN $value_1$ **THEN** $result_1$
    WHEN $value_2$ **THEN** $result_2$
    …
    WHEN $value_n$ **THEN** $result_n$
    ELSE $result_0$
END

CASE — Conditional Expressions

Find the number of all cities regarding the classification (defined by a cities population size).

City Size	Urban Population (Million)
Super city	>10
Megacity	5–10
Large city	1–5
Medium city	0.5–1
Small city	<0.5

```
SELECT class, COUNT(*) AS city_count
FROM
        (SELECT name, CASE
                WHEN population > 10000000 THEN 'Super City'
                WHEN population > 5000000 THEN 'Mega City'
                WHEN population > 1000000 THEN 'Large City'
                WHEN population > 500000 THEN 'Medium City'
                ELSE 'Small City' END AS class
         FROM cities) t
GROUP BY class;
```

class	city_count
Medium City	576
Large City	546
Small City	38872
Mega City	104
Super City	40

CASE — Conditional Expressions

Find all countries and return the continent in Tamil.

```
SELECT name, CASE continent
        WHEN 'Africa' THEN 'ஆப்பிரிக்கா'
        WHEN 'Asia' THEN 'ஆசியா'
        WHEN 'Europe' THEN 'ஐரோப்பா'
        WHEN 'North America' THEN 'வட அமெரிக்கா'
        WHEN 'South America' THEN 'தென் அமெரிக்கா'
        WHEN 'Oceania' THEN 'ஓசியானியா'
        ELSE NULL END AS continent
FROM countries;
```

name	continent
Afghanistan	ஆசியா
Albania	ஐரோப்பா
Algeria	ஆப்பிரிக்கா
Andorra	ஐரோப்பா
Angola	ஆப்பிரிக்கா
Antigua and Barbuda	வட அமெரிக்கா
Argentina	தென் அமெரிக்கா
...	...

COALESCE — Conditional Expressions for NULL Values

- COALESCE(value1, value2, value3, ...)
  - Returns the first non-NULL value in the list of input arguments
  - Returns NULL if all values in the list of input arguments are NULL
  - Example: **SELECT COALESCE**(*null*, *null*, 1, *null*, 2) ➜

val
1

Find the number of cities for each city type;
consider cities with NULL for column "capital" as "other".

```
SELECT type, COUNT(*) AS city_count
FROM
        (SELECT COALESCE(type, 'other') AS type
         FROM cities) t
GROUP BY type;
```

type	city_count
primary	206
other	30573
admin	5852
minor	3507

NULLIF — Conditional Expressions for NULL Values

- **NULLIF**(*$value_1$*, *$value_2$*)
  - Returns NULL if $value_1$=$value_2$; otherwise returns $value_1$
  - Examples:

SELECT NULLIF(1, 1) **AS** val; ➜

val
null

SELECT NULLIF(1, 2) **AS** val; ➜

val
1

  - Common use case: convert "special" values (zero, empty string) to NULL values

Find the minimum and average Gini Coefficients across all countries (unknown values are represented by 0)

```
SELECT MIN(gini) AS min_gini,
       AVG(gini) AS avg_gini
FROM countries;
```

min_gini	avg_gini
0.0	33.08

```
SELECT MIN(NULLIF(gini, 0)) AS min_gini,
       AVG(NULLIF(gini, 0)) AS avg_gini
FROM countries;
```

min_gini	avg_gini
22.8	37.92

Overview

- Common SQL constructs
  - Aggregation
  - Grouping
  - Conditional Expressions
- **Structuring Queries**
  - **Common Table Expressions**
  - Views
- Extended concepts
  - Universal Quantification
  - Recursive Queries
- Summary

Common Table Expressions (CTEs)

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

- Motivation
  - SQL can quickly become complex and unreadable
  - CTEs allow to structure SQL queries to improve readability

➜ Common Table Expression CTE

- Temporary named query
- One or more CTEs can be used within an SQL statement

Example from last lecture:
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;
```

Common Table Expressions (CTEs)

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

- Same query using a CTE

CTE name

CTE body

```
WITH AirportsInIsolatedEuropeanCountries AS
    (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')
SELECT i.country, i.city, i.airport
FROM AirportsInIsolatedEuropeanCountries i LEFT OUTER JOIN routes r
    ON i.code = r.to_code
WHERE r.to_code IS NULL;
```

CTE usage

Common Table Expressions (CTEs)

- General syntax
  - Each C_i is the name of a temporary table defined by query Q_i
  - Each C_i can reference any other C_j that has been declared before C_i
  - SQL statement S can reference any possible subset of all C_i

WITH
C_1 **AS** (Q_1),
C_2 **AS** (Q_2),
…,
C_n **AS** (Q_n)
SQL statement S;

- Note
  - The goal of using CTEs is <u>not</u> to write less code
  - CTEs help to improve readability, debugging, maintenance

Common Table Expressions (CTEs)

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

- Extended example
  - Multiples CTEs
  - CTE referencing previously declared CTE
  - CTEs are not required to be referenced

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

Overview

- Common SQL constructs
  - Aggregation
  - Grouping
  - Conditional Expressions
- **Structuring Queries**
  - Common Table Expressions
  - **Views**
- Extended concepts
  - Universal Quantification
  - Recursive Queries
- Summary

Views — Virtual Tables

- Common observations when querying databases
  (beyond the case of increasing complexity of SQL queries)
  - Often only parts of a table (rows/columns) are of interest
  - Often not all parts of a table (rows/columns) should be accessible to all users
  - Often the same queries or subqueries are regularly and frequently used

➜ **View**

- Permanently named query (= virtual table)
- Can be used like normal tables
  (with some restrictions; discussed later)
- The result of a query is not permanently stored!
  (query is executed each time the view is used)

```
CREATE VIEW <name> AS
    SELECT ...
    FROM ...
    ...
```

Views — Example

Assumption: Finding all European countries without a land border is a very frequent query.

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

```
CREATE VIEW IsolatedEuropeanCountries AS
        SELECT n.iso2, n.name AS country
        FROM borders b, countries n
        WHERE b.country1_iso2 = n.iso2
            AND b.country2_iso2 IS NULL
            AND n.continent = 'Europe';
```

```
WITH AirportsInIsolatedEuropeanCountries AS (
        SELECT n.country, c.name AS city, a.code, a.name AS airport
        FROM IsolatedEuropeanCountries n, cities c, airports a
        WHERE n.iso2 = c.country_iso2
            AND c.name = a.city)
SELECT i.country, i.city, i.airport
FROM AirportsInIsolatedEuropeanCountries i LEFT OUTER JOIN routes r
    ON i.code = r.to_code
WHERE r.to_code IS NULL;
```

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

Views — Example

```sql
CREATE VIEW CountryUrbanizationStats AS
SELECT
    n.iso2, n.name, n.population AS overall_population, SUM(c.population) AS city_population,
    SUM(c.population) / CAST(n.population AS NUMERIC) AS urbanization_rate
FROM cities c, countries n
WHERE c.country_iso2 = n.iso2
GROUP BY n.iso2, n.name, n.population;
```

Quick Quiz: Why do we need this?

Find all countries with a urbanization rate below 10%.

```sql
SELECT name, urbanization_rate
FROM CountryUrbanizationStats
WHERE urbanization_rate < 0.1
ORDER BY urbanization_rate ASC;
```

name	urbanization_rate
Grenada	0.039
Micronesia	0.061
Ethiopia	0.073
Burundi	0.087
Uganda	0.099

Views — Usability

- No restriction when used in SQL queries (**SELECT** statements)
    - But what about **INSERT**, **UPDATE**, **DELETE** statements?

➜ Updatable View — requirements

- Only one entry in **FROM** clause (table or updatable view)
- No **WITH**, **DISTINCT**, **GROUP BY**, **HAVING**, **LIMIT**, or **OFFSET**
- No **UNION**, **INTERSECT** or **EXCEPT**
- No aggregate functions
- etc. (incl. no constraint violations)

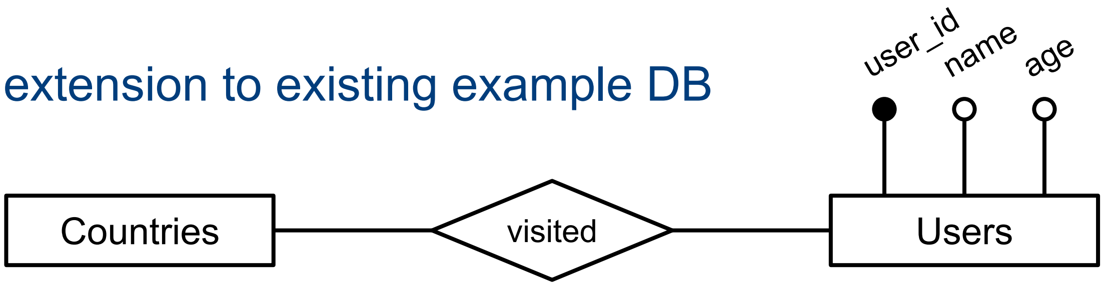
For more details: https://www.postgresql.org/docs/current/sql-createview.html

Overview

- Common SQL constructs
  - Aggregation
  - Grouping
  - Conditional Expressions
- Structuring Queries
  - Common Table Expressions
  - Views
- **Extended concepts**
  - **Universal Quantification**
  - Recursive Queries
- Summary

Universal Quantification

- Small extension to existing example DB



- Query with universal quantification
  - *"Find the names of all users that have visited all countries."*

Visited

user_id	iso2
101	SG
101	DE
103	SG
103	CN
103	FR
...	...

Users

user_id	name	age
101	Sarah	25
102	Judy	35
103	Max	52
...	...	...

➜ Problem: SQL directly supports only existential quantification (**EXISTS**)

Universal Quantification

- "Transformation" of query using logical equivalences
  - *"user who visited all countries"* ➜ *"there does <u>not exists</u> a country the user has <u>not</u> visited"*

- Useful subquery
  - *All countries a user with user_id = x has not visited*

```
SELECT n.iso2
FROM countries n
WHERE NOT EXISTS (SELECT 1
                  FROM visited v
                  WHERE v.iso2 = n.iso2
                    AND v.user_id = x);
```

TRUE only for countries that do <u>not</u> have a match in "Visited" for all tuples where the user_id = x

Universal Quantification

"Find the names of all users that have visited <u>all</u> countries."

```
SELECT user_id, name
FROM users u
WHERE NOT EXISTS (SELECT n.iso2
                  FROM countries n
                  WHERE NOT EXISTS (SELECT 1
                                    FROM visited v
                                    WHERE v.iso2 = n.iso2
                                    AND v.user_id = u.user_id)
                 );
```

user_id	name
103	Max
107	Emma

➜ While not overly common, SQL queries requiring universal quantification can get "ugly".

Universal Quantification

- Alternative interpretation
  - *"user who visited all countries"* ➜ *"the number of tuples in "Visited" for that user must match the total number of countries"*

"Find the names of all users that have visited <u>all</u> countries."

```
SELECT u.user_id, u.name
FROM users u, visited v
WHERE u.user_id = v.user_id
GROUP BY u.user_id
HAVING COUNT(*) = (SELECT COUNT(*) FROM countries);
```

user_id	name
103	Max
107	Emma

Overview

- Common SQL constructs
  - Aggregation
  - Grouping
  - Conditional Expressions
- Structuring Queries
  - Common Table Expressions
  - Views
- **Extended concepts**
  - Universal Quantification
  - **Recursive Queries**
- Summary

Recursive Queries

```
CREATE TABLE connections AS
  SELECT DISTINCT(from_code, to_code)
  FROM routes;
```

- Small extension to existing example DB
  - Create table "Connections" as shown
  - Eliminates duplicate routes served by multiple airlines

- Interesting queries
  - *"Find all airports that can be reached from SIN non-stop."*

```
SELECT to_code
FROM connections
WHERE from_code = 'SIN';
```

90 tuples

to_code
PEK
BKK
FRA
KUA
...

  - *"Find all airports that can be reached from SIN with 1/2/3/… stops."* ➜ **???**

Recursive Queries

*Find all airports that can be reached from SIN with **1** stop.*

825 tuples

```
SELECT DISTINCT(c2.to_code) AS to_code
FROM
   connections c1,
   connections c2
WHERE c1.to_code = c2.from_code
     AND c1.from_code = 'SIN';
```

to_code
DUB
PEK
SIN
MME
...

*Find all airports that can be reached from SIN with **2** stop.*

1,561 tuples

```
SELECT DISTINCT(c3.to_code) AS to_code
FROM
   connections c1,
   connections c2,
   connections c3
WHERE c1.to_code = c2.from_code
     AND c2.to_code = c3.from_code
     AND c1.from_code = 'SIN';
```

to_code
DUB
PEK
SIN
MME
...

Recursive Queries

- Observation: *X* stops requires query with *X* joins
  - Requires to write a separate query for each X

➜ Recursive Queries using CTEs

Non-recursive term;
can not reference "cte_name"!

Recursive term
referencing "cte_name"

WITH RECURSIVE cte_name AS (
    Q_1
    UNION [**ALL**]
    Q_2(cte_name)
)
SELECT * **FROM** cte_name;

Recursive Queries

*Find all airports that can be reached from SIN with **0..2** stops.*
(limitation to max. 2 stops purely for performance reasons)

```
WITH RECURSIVE flight_path AS (
        SELECT from_code, to_code, 0 AS stops
        FROM connections
        WHERE from_code = 'SIN'
        UNION ALL
        SELECT c.from_code, c.to_code, p.stops+1
        FROM flight_path p, connections c
        WHERE p.to_code = c.from_code
        AND p.stops < 2
)
SELECT DISTINCT to_code, stops
FROM flight_path
ORDER BY stops ASC;
```

to_code	stops	
PEK	0	90 tuples
BKK	0	
FRA	0	
...	...	
KUA	0	
DUB	1	825 tuples
PEK	1	
SIN	1	
...	...	
MME	1	
AMS	2	1,561 tuples
BKK	2	
PER	2	
...	....	
ZYL	2	

*Find all airports that can be reached from SIN with **0..2** stops, including the exact paths.*

(limitation to max. 2 stops purely for performance reasons)

```
WITH RECURSIVE flight_path (airport_codes, stops, is_visited) AS (
        SELECT
                ARRAY[from_code, to_code],
                0 AS stops,
                from_code = to_code
        FROM connections
        WHERE from_code = 'SIN'
        UNION ALL
        SELECT
                (airport_codes || to_code)::char(3)[],
                p.stops + 1,
                c.to_code = ANY(p.airport_codes)
        FROM
                connections c,
                flight_path p
        WHERE p.airport_codes[ARRAY_LENGTH(airport_codes, 1)] = c.from_code
            AND NOT p.is_visited
            AND p.stops < 2
)
SELECT DISTINCT airport_codes, stops
FROM flight_path
ORDER BY stops;
```

airport_codes	stops	
{SIN, PEK}	0	90 tuples
{SIN, BKK}	0	
{SIN, FRA}	0	
...	...	
{SIN, KUA}	0	
{SIN, BKK, PEK}	1	4,058 tuples
{SIN, FRA, PEK}	1	
{SIN, DOH, PEK}	1	
...	...	
{SIN, MFM, DMK}	1	
{SIN, ADL, HKG, PEK}	2	184,059 tuples
{SIN, ADL, KUL, PEK}	2	
{SIN, ADL, SYD, PEK}	2	
...	....	
{SIN, TPE, FRA, CSS}	2	

Dealing with the Limitations of (Basic) SQL

- Other types of queries poorly or not support by basic SQL
  - *"Sorted by GDP, are there somewhere in the ranking 5 Asian countries listed in a row."*
  - Queries/tasks common for time series: moving average, sliding window, etc.

- Common approaches
  - Keep or move logic into the application
  - Use features that make SQL turing-complete*
    (e.g. using SQL/PSM — Persistent Stored Modules)

    ➜ Covered in next lectures
  - Use a different data model / DBMS
    (e.g., a graph database for recursive queries, or time series databases)

*strictly speaking the support of Recursive CTEs already made SQL turing-complete

Summary

- Covered: SQL (DQL)
  - Most common vocabulary for writing queries
  - Basic means to "organize" complex queries (CTEs, Views)

- Limitations of SQL (more general: Relational Model)
  - Universal quantification
  - Recursive queries
  - Sequential data
  - Graph data
  - ...

RDBMS & SQL not the solution for everything

Quick Quiz Solutions

Quick Quiz (Slide 17)

Quick Quiz (Slide 22)

Quick Quiz (Slide 38)