



Lecture 4

SQL for Data Science

Relational Databases

- In data science, tasks often involve extracting and manipulating data from databases. While files and spreadsheets suit small static datasets, complex and large datasets managed by diverse users across regions require database systems for various reasons:
 - **Performance** - Databases optimize storage for quick queries
 - **Security** - Databases secure and encrypt for breach protection.
 - **Concurrency** - Databases resolve concurrent data conflicts.
 - **Recovery** - Databases recover data post system failures.

Relational Databases

- Major databases are **relational**. They organize data in tables with rows and columns. Tables link using keys and are queried via SQL. E.g., below shows two tables:

Country

CountryCode	CountryName	SurfaceArea	ContinentName
JPN	Japan	377829	Asia
ITA	Italy	301316	Europe
ECU	Ecuador	283561	South America

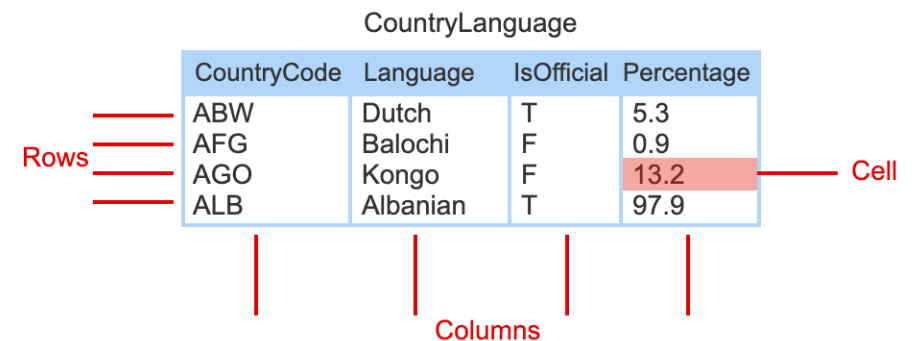
City

CityName	CountryCode	Population
Osaka	JPN	2595674
Kyoto	JPN	1461974
Rome	ITA	2643581
Verona	ITA	255268
Naples	ITA	1002619
Quito	ECU	1573458

Tables

Relational data is structured in tables:

- A **table** has a name, a fixed sequence of columns, and a varying set of rows.
- A **column** has a name and a data type.
- A **row** is an unnamed sequence of values. Each value corresponds to a column and belongs to the column's data type.
- A **cell** is a single column of a single row.



The diagram shows a table titled "CountryLanguage" with four columns: "CountryCode", "Language", "IsOfficial", and "Percentage". The table contains four rows of data. Red horizontal lines to the left of the table are labeled "Rows". Red vertical lines below the table are labeled "Columns". A red line points to the cell containing "13.2", which is labeled "Cell".

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

Tables

A table has at least one column and any number of rows. Tables obey the following rules:

- One value per cell. A cell cannot contain multiple values.
- No duplicate column names. Duplicate column names are allowed in different tables, but not in the same table.
- No duplicate rows. No two rows have identical values in all columns.
- No row order. The organization of rows on storage media, such as a disk drive, never affects query results.

NULL Values

- At times, cells contain unknown or irrelevant data. Relational databases use a special NULL value for such cases.
- For example, in the table below, Italy's surface area is unknown, and Antarctica lacks a capital city.

Country

CountryCode	CountryName	SurfaceArea	CapitalName
JPN	Japan	377829	Tokyo
ITA	Italy	NULL	Rome
ECU	Ecuador	283561	Quito
ATA	Antarctica	13120000	NULL

NULL Values

- NULL differs from zero or an empty string. Ex., a zero balance in a bank account is a definite, relevant value, and a blank survey comment signifies a specific, applicable response.
- When used in arithmetic or comparisons, NULL always yields a NULL result. Ex., the subsequent expressions result in NULL:
 - $4 + \text{NULL}$
 - $11.3 \leq \text{NULL}$
 - $\text{NULL} = \text{NULL}$
- When combined with AND and OR, NULL can yield a NULL outcome, as outlined in the accompanying truth table. For instance, TRUE AND NULL results in NULL, while TRUE OR NULL yields TRUE.
- The result of NOT NULL is NULL.

Keys

- **Unique column:** No duplicate values in different rows, each value in one row only despite inserts/updates.
- **Primary key:** Identifies rows, must be unique and not NULL to ensure one value corresponds to one row.
- **Foreign key:** Refers to primary key, can be NULL or match referenced value, preventing references to non-existent rows. Not necessarily unique.
- **Key matching:** Foreign and primary keys share data types, names may differ. Can be in different or same tables.
- **Table diagrams:** Primary key is solid circle (●), leftmost table column. Foreign key is empty circle (○) with arrow to referenced primary key.

Structured Query Language (SQL)

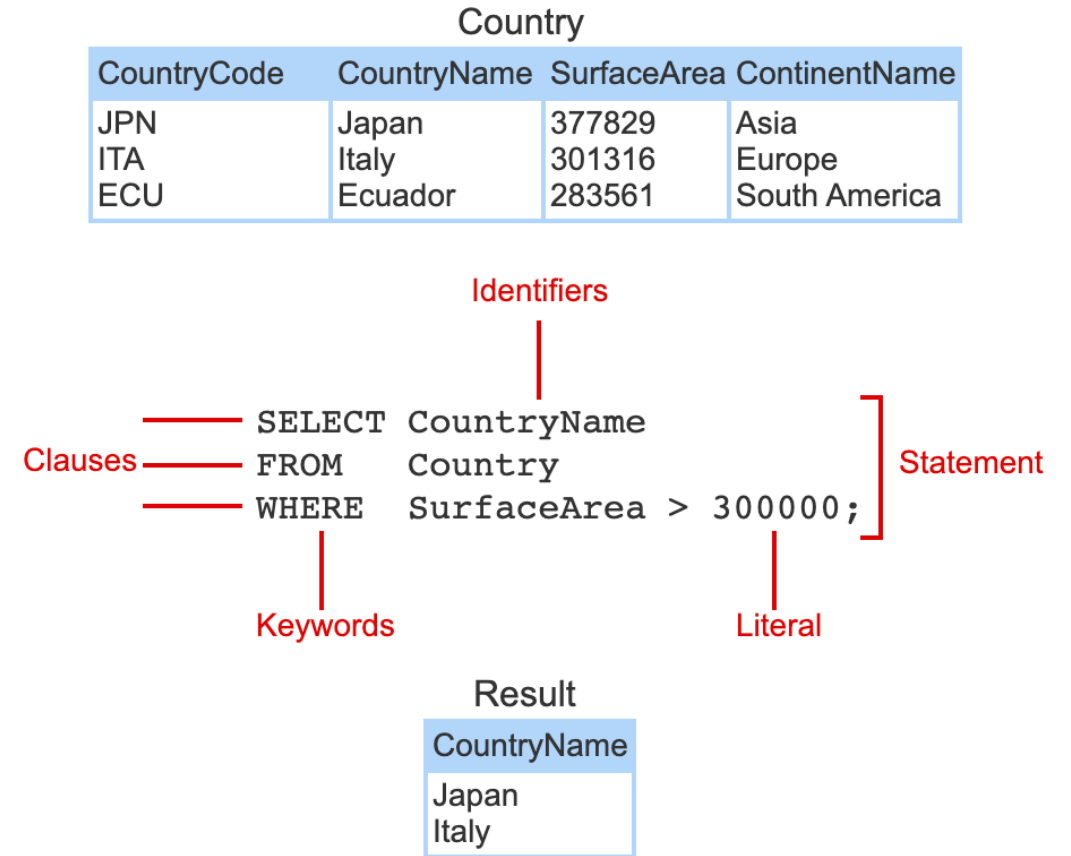
- Structured Query Language (**SQL**) is a computer language for manipulating and retrieving data.
- SQL is the standard language for relational databases and is also supported by many non-relational databases.
- SQL is pronounced either "S-Q-L " or "seekwəl ".

Structured Query Language (SQL)

- SQL consists of language elements:
 - A **keyword** is a reserved word with special meaning.
 - A **literal** is a fixed numeric or text value. Text literals are enclosed in single or double quotes.
 - An **identifier** is a user-defined name for a table, column, database, etc.
 - A **clause** groups a keyword with identifiers and expressions.
 - A **statement** is a complete database action, consisting of one or more clauses and ending with a semicolon.

Structured Query Language (SQL)

- Keywords and identifiers usually not case-sensitive, like SELECT and select being equivalent.
- SQL ignores whitespace, including line breaks. Clauses are typically on separate lines despite optional one-line formatting.
- SQL statements range from creating/dropping tables/indexes to data retrieval, insertion, updating, and deletion.



Expressions in SQL

- An expression is a string of operators, operands, and parentheses that evaluates to a single value.
- Ex: `Salary > 34000 AND Department = 'Marketing'` is an expression with a logical value.
- Operators in an expression are evaluated in the order of **operator precedence**, shown in the table.

Precedence	Operators
1	- (unary)
2	^
3	* / %
4	+ - (binary)
5	= != < > <= >=
6	NOT
7	AND
8	OR

SELECT Statement in SQL

- The CountryLanguage table contains four rows with columns CountryCode, Language, IsOfficial, and Percentage.
- The SELECT statement selects all columns using *. All rows and columns appear in the result table.
- The SELECT statement selects only columns CountryCode and Language, so only two columns appear in the result table.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT *  
FROM CountryLanguage;
```

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT CountryCode, Language  
FROM CountryLanguage;
```

CountryCode	Language
ABW	Dutch
AFG	Balochi
AGO	Kongo
ALB	Albanian

WHERE Clause in SQL

- The statement selects rows with percentage between 0.0 and 10.0. Two rows are returned.
- The statement selects rows with percentage < 5.0 or percentage > 90.0. Two rows are returned.

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE Percentage > 0.0
AND Percentage < 10.0;
```

CountryCode	Language
ABW	Dutch
AFG	Balochi

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE Percentage < 5.0
OR Percentage > 90.0;
```

CountryCode	Language
AFG	Balochi
ALB	Albanian

IN Operator in SQL

- The IN operator is used in a WHERE clause to determine if a value matches one of several values.
- The SELECT statement in the figure below uses the IN operator to select only rows where the Language column has a Dutch, Kongo, or Albanian value.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
AND	Catalan	T	32.3

```
SELECT *  
FROM CountryLanguage  
WHERE Language IN ('Dutch', 'Kongo', 'Albanian');
```

ABW	Dutch	T	5.3
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9

LIKE Operator in SQL

- The LIKE operator, when used in a WHERE clause, matches text against a pattern using the two wildcard characters % and _.
 - % matches any number of characters. Ex: LIKE 'L%t' matches "Lt", "Lot", "Lift", and "Lol cat".
 - _ matches exactly one character. Ex: LIKE 'L_t' matches "Lot" and "Lit" but not "Lt" and "Loot".
- The LIKE operator performs case-insensitive pattern matching by default or case-sensitive pattern matching if followed by the BINARY keyword. Ex: LIKE BINARY 'L%t' matches 'Left' but not 'left'.
- To search for the wildcard characters % or _, a backslash (\) must precede % or _. Ex: LIKE 'a\%' matches "a%".
- Most relational databases provide other mechanisms to perform more advanced pattern matching with regular expressions. Ex: MySQL uses REGEXP to match text against a regular expression, and PostgreSQL uses SIMILAR TO.

LIKE Operator in SQL

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
ARW	Arawak	F	4.9

```
SELECT *  
FROM CountryLanguage  
WHERE CountryCode LIKE 'A_W';
```

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ARW	Arawak	F	4.9

```
SELECT *  
FROM CountryLanguage  
WHERE Language LIKE 'A%n';
```

CountryCode	Language	IsOfficial	Percentage
ALB	Albanian	T	97.9

```
SELECT *  
FROM CountryLanguage  
WHERE Language LIKE 'A%';
```

CountryCode	Language	IsOfficial	Percentage
ALB	Albanian	T	97.9
ARW	Arawak	F	4.9

DISTINCT Clause in SQL

- The DISTINCT clause is used with a SELECT statement to return only unique values. Ex: The first SELECT statement in the figure below results in two 'Spanish' rows, but the second SELECT statement returns only unique languages, resulting in only one 'Spanish' row.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Spanish	F	7.4
AFG	Balochi	F	0.9
ARG	Spanish	T	96.8
BLZ	Spanish	F	31.6
BRA	Portuguese	T	97.5

```
SELECT Language
FROM CountryLanguage
WHERE IsOfficial = 'F';
```

Spanish
Balochi
Spanish

```
SELECT DISTINCT Language
FROM CountryLanguage
WHERE IsOfficial = 'F';
```

Spanish
Balochi

ORDER BY Clause in SQL

- A SELECT statement retrieves table rows without assured order. ORDER BY arranges rows by column(s) in ascending order. DESC with ORDER BY yields descending order.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
FSM	Woleai	F	3.7
FSM	Yap	F	5.8
GAB	Fang	F	35.8
GAB	Mbete	F	13.8

```
-- Order by Language (ascending)
SELECT *
FROM CountryLanguage
ORDER BY Language;
```

GAB	Fang	F	35.8
GAB	Mbete	F	13.8
FSM	Woleai	F	3.7
FSM	Yap	F	5.8

```
-- Order by Language (descending)
SELECT *
FROM CountryLanguage
ORDER BY Language DESC;
```

FSM	Yap	F	5.8
FSM	Woleai	F	3.7
GAB	Mbete	F	13.8
GAB	Fang	F	35.8

```
-- Order by CountryCode, then Language
SELECT *
FROM CountryLanguage
ORDER BY CountryCode, Language;
```

FSM	Woleai	F	3.7
FSM	Yap	F	5.8
GAB	Fang	F	35.8
GAB	Mbete	F	13.8

Aggregate Functions in SQL

- Aggregate function summarizes values from rows. It's in SELECT, processes WHERE rows. Without WHERE, processes all rows in SELECT. Common aggregate functions are:
 - **COUNT()** counts the number of selected values.
 - **MIN()** finds the minimum of selected values.
 - **MAX()** finds the maximum of selected values.
 - **SUM()** sums selected values.
 - **AVG()** computes the arithmetic mean of selected values.
 - **VARIANCE()** computes the standard variance of selected values.
- Aggregate functions have a single argument. The argument may be any expression but is usually a column name. Aggregate functions ignore rows for which the expression evaluates to NULL.

GROUP BY Clause in SQL

- Aggregate functions are commonly used with the GROUP BY clause. The GROUP BY clause groups rows with identical values into a set of summary rows. Some important points about the GROUP BY clause:
 - One or more columns are listed after GROUP BY, separated by commas.
 - GROUP BY clause returns one row for each group.
 - Each group may be ordered with the ORDER BY clause.
 - GROUP BY clause must appear before the ORDER BY clause and after the WHERE clause (if present).

GROUP BY Clause in SQL

City

ID	Name	CountryCode	District	Population
3162	Lusaka	ZMB	1	1317000
3163	Ndola	ZMB	2	329200
3164	Kitwe	ZMB	2	288600
3165	Kabwe	ZMB	3	154300
3166	Chingola	ZMB	2	142400
4068	Harare	ZWE	1	1410000
4069	Bulawayo	ZWE	2	621742
4070	Chitungwiza	ZWE	1	274912

```
SELECT CountryCode, SUM(Population)
FROM City
GROUP BY CountryCode;
```

CountryCode	SUM(Population)
ZMB	2231500
ZWE	2306654

```
SELECT CountryCode, District, COUNT(*)
FROM City
GROUP BY CountryCode, District;
```

CountryCode	District	COUNT(*)
ZMB	1	1
ZMB	2	3
ZMB	3	1
ZWE	1	2
ZWE	2	1

HAVING Clause in SQL

- HAVING refines groups via GROUP BY. It follows GROUP BY and comes before optional ORDER BY.
- The HAVING clause must include the same aggregate function that appears in the SELECT clause.

City

ID	Name	CountryCode	District	Population
3162	Lusaka	ZMB	1	1317000
3163	Ndola	ZMB	2	329200
3164	Kitwe	ZMB	2	288600
3165	Kabwe	ZMB	3	154300
3166	Chingola	ZMB	2	142400
4068	Harare	ZWE	1	1410000
4069	Bulawayo	ZWE	2	621742
4070	Chitungwiza	ZWE	1	274912

```
SELECT CountryCode, SUM(Population)
FROM City
GROUP BY CountryCode
HAVING SUM(Population) > 2300000;
```

CountryCode	SUM(Population)
ZWE	2306654

```
SELECT CountryCode, District, COUNT(*)
FROM City
GROUP BY CountryCode, District
HAVING COUNT(*) >= 2;
```

CountryCode	District	COUNT(*)
ZMB	2	3
ZWE	1	2

Prefixes and Aliases in SQL

- Duplicate column names: Join tables may have identical column names. To differentiate, use prefix: table name + period.
- Simplifying via alias: Alias replaces column names for simpler queries/results. Alias follows column name, with optional AS keyword.
- Ex: Figure below shows join with prefix for Name, aliases Group and Supervisor simplify result.

Department

• Code	Name	○ Manager
44	Engineering	2538
82	Sales	6381
12	Marketing	6381
99	Technical support	NULL

Employee

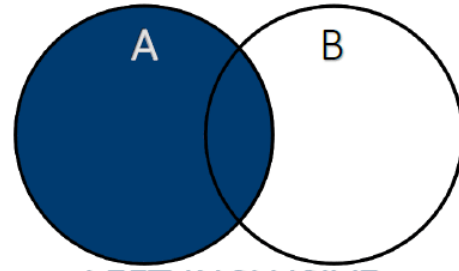
• ID	Name	Salary
2538	Lisa Ellison	45000
5284	Sam Snead	30500
6381	Maria Rodriguez	92300

```
SELECT Department.Name AS Group,  
       Employee.Name AS Supervisor  
FROM Department, Employee  
WHERE Manager = ID;
```

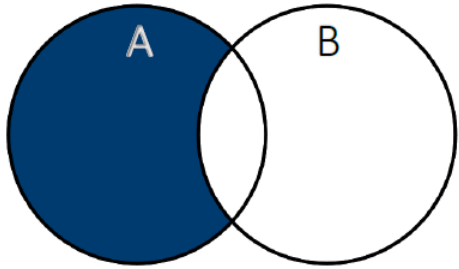
Result

Group	Supervisor
Engineering	Lisa Ellison
Sales	Maria Rodriguez
Marketing	Maria Rodriguez

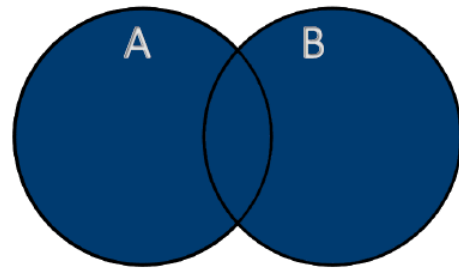
JOINS in SQL



LEFT INCLUSIVE

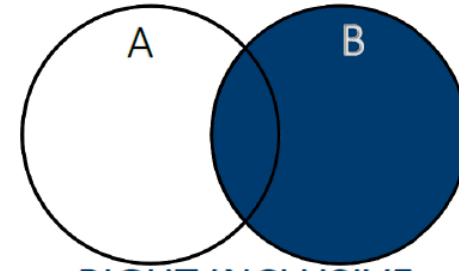


LEFT EXCLUSIVE

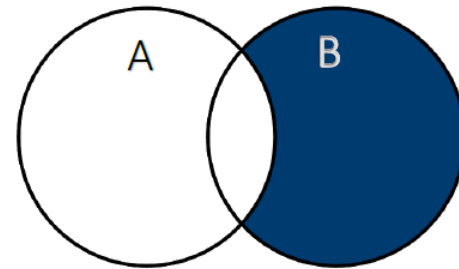


FULL OUTER INCLUSIVE

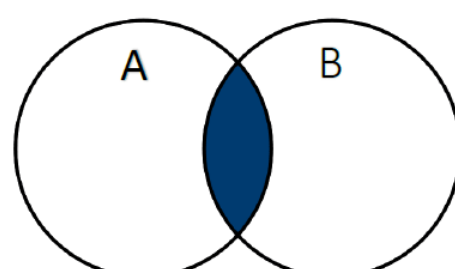
SQL JOINS	
LEFT INCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key	RIGHT INCLUSIVE SELECT [Select List] FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key= B.Key
LEFT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE B.Key IS NULL	RIGHT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key= B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	FULL OUTER EXCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL
INNER JOIN SELECT [Select List] FROM TableA A INNER JOIN TableB B ON A.Key = B.Key	



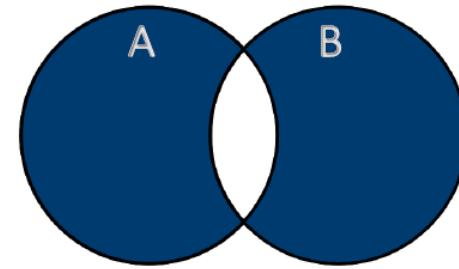
RIGHT INCLUSIVE



RIGHT EXCLUSIVE



INNER JOIN



FULL OUTER EXCLUSIVE

Subquery in SQL

- Subquery, also known as **nested query**, is query within SQL query. Often in WHERE of SELECT, it limits and feeds data to outer query. Enclosed in ().

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albanian	T	97.9
AND	Catalan	T	32.3

Country

Code	Name	Continent
ABW	Aruba	North America
AFG	Afghanistan	Asia
AGO	Angola	Africa
ALB	Albania	Europe
AND	Andorra	Europe

```
SELECT Language, Percentage
FROM CountryLanguage
WHERE Percentage > 5.3
```

Subquery {

```
(SELECT Percentage
FROM CountryLanguage
WHERE CountryCode = 'ABW'
AND IsOfficial = 'T'));
```

Language	Percentage
Kongo	13.2
Albanian	97.9
Catalan	32.3

```
SELECT CountryCode, Language
FROM CountryLanguage
WHERE CountryCode IN (ALB,AND)
(SELECT Code
FROM Country
WHERE Continent = 'Europe');
```

} Subquery

CountryCode	Language
ALB	Albanian
AND	Catalan

Correlated Subquery in SQL

- Correlated subquery: Subquery's WHERE refers outer query's column. Rows selected based on outer query's current row.
- Differentiating columns: Correlated subquery uses `TableName.ColumnName`. Ex., `City.CountryCode` points to City table's CountryCode column.
- Alias for distinction: Alias is temp name for column/table. AS keyword creates alias. Ex: `SELECT Name AS N FROM Country AS C` for N (Name) and C (Country). AS is optional, e.g., `SELECT Name N FROM Country C`.

Correlated Subquery in SQL

1. The outer query and correlated subquery both select from the City table. The outer query uses an alias C for the City table, so C.CountryCode refers to the outer query's CountryCode column.
2. The outer query executes first to process rows in the City table. As each City row is processed, the subquery finds the average population for the city's country.
3. Then the outer query executes using the average population returned from the subquery. Buenos Aires has a population 2982146 > 2124303.5.
4. The outer query processes the next row, and the average population for ARG is calculated again. La Matanza is not selected because La Matanza's population is not > 2124303.5.
5. The outer query finds São Paulo also has a population > BRA's average population.
6. Rio de Janeiro is not selected because Rio de Janeiro's population 5598953 is not > 7783719.

City

Id	Name	CountryCode	Population
69	Buenos Aires	ARG	2982146
70	La Matanza	ARG	1266461
206	São Paulo	BRA	9968485
207	Rio de Janeiro	BRA	5598953

```
SELECT Name, CountryCode, Population
FROM City C
WHERE Population >
      (SELECT AVG(Population)
       FROM City
       WHERE CountryCode = C.CountryCode);
```

Name	CountryCode	Population
Buenos Aires	ARG	2982146
São Paulo	BRA	9968485

Flattening Subqueries in SQL

- Subqueries often become joins for better performance. Joins optimize together, making them faster and preferable for performance.
- Substituting subquery with JOIN is query flattening. Flattening criteria complex, SQL implementation dependent. IN, EXISTS, single-value returning subqueries often flattenable. NOT EXISTS, GROUP BY-containing subqueries usually not flattenable.
- The following steps are a first pass at flattening a query:
 - Retain the outer query SELECT, FROM, GROUP BY, HAVING, and ORDER BY clauses.
 - Add INNER JOIN clauses for each subquery table,
 - Move comparisons between subquery and outer query columns to ON clauses.
 - Add a WHERE clause with the remaining expressions in the subquery and outer query WHERE clauses.
 - If necessary, remove duplicate rows with SELECT DISTINCT.
 - After the first pass, test the flattened query and adjust to achieve the correct result. Verify that the original and flattened queries return the same result for a variety of data.

Flattening Subqueries in SQL

1. The subquery selects country codes for cities with population > 1000000.
2. The outer query selects the country names.
3. To flatten the query, replace the subquery with an INNER JOIN clause.
4. The join query selects the one country name for each city with population > 1000000.
5. The DISTINCT clause eliminates duplicate rows. The subquery and join query are equivalent.

Country		
Code	Name	Continent
AUS	Australia	Australia
SAF	South Africa	Africa
SPA	Spain	Europe
USA	United States	North America

```
SELECT Name
FROM Country
WHERE Code IN
  (SELECT CountryCode
   FROM City
   WHERE Population > 1000000);
```

Name
South Africa
Spain

City			
ID	Name	CountryCode	Population
144	Salzburg	AUS	152367
384	Cape Town	SAF	4618000
471	Durban	SAF	3442361
650	Barcelona	SPA	1620000
938	Madrid	SPA	3233000
942	Denver	USA	705576

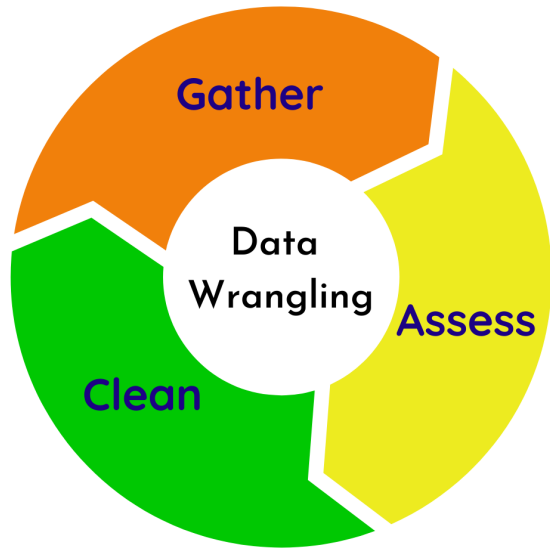
```
SELECT DISTINCT Name
FROM Country
INNER JOIN City ON Code = CountryCode
WHERE Population > 1000000;
```

Name
South Africa
Spain



Case Study

Queries on the Movies Dataset



Next Lecture

Data Wrangling