TECH ELEVATOR

Java

# Intro to Databases

# Ordering, Limiting, and Grouping

Module 2 - Week 1

# Housekeeping

- Welcome to Module 2
    - Databases
    - Don't forget about Java, DAOs and JDBC coming up.
    - Java Katas added to Module 1
    - Think about side projects (collaborations are okay!).

# Schedule

| Week | Topics |
|---|---|
| Week 0 | Module 2 Orientation / PostgreSQL |
| Week 1 | Intro to databases / Ordering, limiting, and grouping |
| Week 2 | SQL joins / Insert, update, and delete / Database design |
| Week 3 | Data Access / Data security |
| Week 4 | DAO testing |
| Week 5 | **Mid-module project** |
| Week 6 | Postman / NPM / Networking and HTTP / Consuming RESTful APIs |
| Week 7 | Server-side APIs |
| Week 8 | Securing APIs |
| Week 9 | **End-of-module project** |
| Week 10 | **Assessment** |

© 2023 Tech Elevator, LLC Confidential

# SQL and NoSQL Databases

**Pros of Relational Databases**

- Great for structured data
- Use of an existing query language (SQL)
- Great for complex queries
- Easy data navigation
- High level of data integration, due to relationships and constraints among tables
- Transactions are secure

**Pros of Non-Relational Databases**

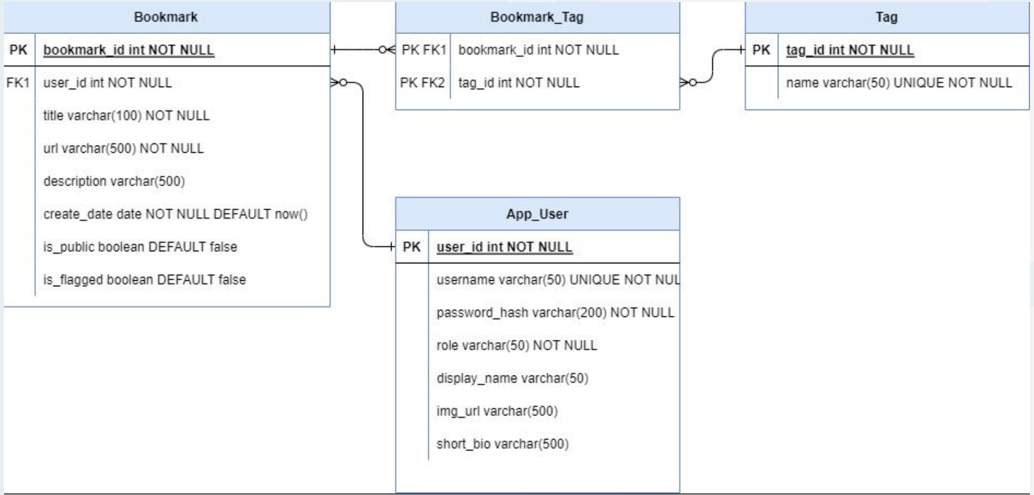
- Flexible data model
- Rapid adaptation to changing requirements: dynamic changes to a item do not affect the other items
- Storage of huge amount of data with little structure
- High performance

Source: towardsdatascience.com

© 2023 Tech Elevator, LLC Confidential

# Bookmark Manager Sample Application

- Log in as a user
- View a list of public bookmarks
- Review any flagged bookmarks
- Manage a list of tags that users associate with a bookmark
- Review ERD
  - users can have many bookmarks that they're associated with
  - users may assign a bookmark many different tags
  - users may use the same tag for many bookmarks



**Bookmark**

| | |
|---|---|
| PK | bookmark_id int NOT NULL |
| FK1 | user_id int NOT NULL |
| | title varchar(100) NOT NULL |
| | url varchar(500) NOT NULL |
| | description varchar(500) |
| | create_date date NOT NULL DEFAULT now() |
| | is_public boolean DEFAULT false |
| | is_flagged boolean DEFAULT false |

**Bookmark_Tag**

| | |
|---|---|
| PK FK1 | bookmark_id int NOT NULL |
| PK FK2 | tag_id int NOT NULL |

**Tag**

| | |
|---|---|
| PK | tag_id int NOT NULL |
| | name varchar(50) UNIQUE NOT NULL |

**App_User**

| | |
|---|---|
| PK | user_id int NOT NULL |
| | username varchar(50) UNIQUE NOT NUL |
| | password_hash varchar(200) NOT NULL |
| | role varchar(50) NOT NULL |
| | display_name varchar(50) |
| | img_url varchar(500) |
| | short_bio varchar(500) |

© 2023 Tech Elevator, LLC Confidential

# (R)DBMS

A Relational Database Management System ( (R)DBMS) is a software application designed to manage a database.  It has four basic functions

1. Data Definition
2. Data Storage
3. Data Retrieval
4. Administration

RDBMSs include databases like Oracle, Microsoft SQL Server, PostgreSQL, MySQL, are relational, and are commonly called ***SQL Databases***.

***NoSQL*** Databases are those that do not use a relational structure, instead they structure data specific to the problem they are designed to solve.  NoSQL databases include MongoDB, Cassandra, Google BigTable, HBase, DynamoDB, and Firebase.

DB-Engines has a ranking measuring popularity of current DBMS platforms.

# Structured Query Language (SQL)

SQL is a  declarative programming language used to manage a database and its data.

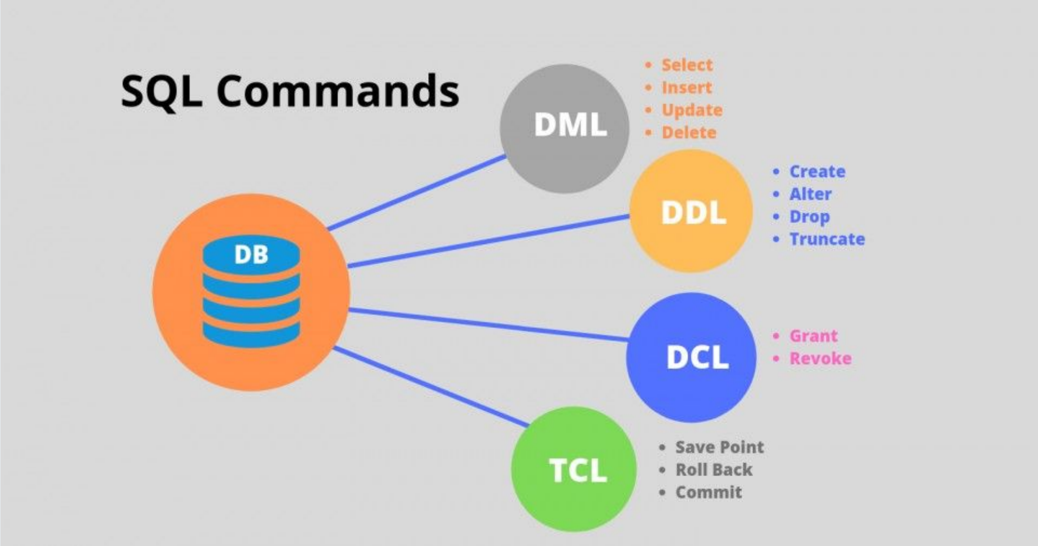
A **declarative programming language** specifics what actions should be performed rather than how to perform those actions.

SQL Consists of 3 sub-languages

1. **DDL** - Data Definition Language - defines the structure of the data
2. **DML** - Data Manipulation Language - query and modify the data
3. **DCL** - Data Control Language - used to administer the database

# DDL, DML ...

- Data definition language (DDL): modify or alter the structure of the database.
- Data manipulation language (DML): perform operations like storing data in database tables, modifying and deleting existing rows, retrieving data, or updating data.
- Data control language (DCL): perform operations like giving or removing database access for a user.
- Transaction Control Language (TCL): control the execution of a transaction.



© 2022 Tech Elevator, LLC Confidential

**SQL** - Structured Query Language :  a language that lets you access and manipulate databases

**ANSI-SQL** - A standard that databases must follow to be considered a SQL Database.

All SQL databases support the ANSI-SQL language, however, most databases extend it with their own proprietary additions.

# Tables, Columns, Rows

An **entity** is a set of data being stored a *table*.

A **Table** defines a set of data elements and the structure to store them. Tables are structured into *columns* and *rows*.

**Columns** - attributes of a table and define the name and data type. A table has a set number of defined columns.

**Rows** - the data being stored. A table has an unlimited number or rows.

A **Cell** is the location where a *column* and *row* intersect, and is used to refer to a specific value or row of data (*entity*).

© 2022 Tech Elevator, LLC Confidential

# Column

| id | name | countrycode | district | population |
|---|---|---|---|---|
| 3793 | New York | USA | New York | 8008278 |
| 3794 | Los Angeles | USA | California | 3694820 |
| 3795 | Chicago | USA | Illinois | 2896016 |
| 3796 | Houston | USA | Texas | 1953631 |
| 3797 | Philadelphia | USA | Pennsylvania | 1517550 |
| 3798 | Phoenix | USA | Arizona | 1321045 |
| 3799 | San Diego | USA | California | 1223400 |
| 3800 | Dallas | USA | Texas | 1188580 |
| 3801 | San Antonio | USA | Texas | 1144646 |
| 3802 | Detroit | USA | Michigan | 951270 |

**Column**

- Tables have a set number.
- Define the data the table will hold
- Provides a label for each part of the data being stored

**Columns on this Table**

id, name, countrycode, district, population

# Row

| id | name | countrycode | district | population |
|---|---|---|---|---|
| 3793 | New York | USA | New York | 8008278 |
| 3794 | Los Angeles | USA | California | 3694820 |
| 3795 | Chicago | USA | Illinois | 2896016 |
| 3796 | Houston | USA | Texas | 1953631 |
| 3797 | Philadelphia | USA | Pennsylvania | 1517550 |
| 3798 | Phoenix | USA | Arizona | 1321045 |
| 3799 | San Diego | USA | California | 1223400 |
| 3800 | Dallas | USA | Texas | 1188580 |
| 3801 | San Antonio | USA | Texas | 1144646 |
| 3802 | Detroit | USA | Michigan | 951270 |

**Row**

- Tables have a unlimited number (0...n)
- Contain the data
- Has a value for each column

# Cell

| id | name | countrycode | district | population |
|---|---|---|---|---|
| 3793 | New York | USA | New York | 8008278 |
| 3794 | Los Angeles | USA | California | 3694820 |
| 3795 | Chicago | USA | Illinois | 2896016 |
| 3796 | Houston | USA | Texas | 1953631 |
| 3797 | Philadelphia | USA | Pennsylvania | 1517550 |
| 3798 | Phoenix | USA | Arizona | 1321045 |
| 3799 | San Diego | USA | California | 1223400 |
| 3800 | Dallas | USA | Texas | 1188580 |
| 3801 | San Antonio | USA | Texas | 1144646 |
| 3802 | Detroit | USA | Michigan | 951270 |

**Cell**

- The intersection of a column and row
- Used to identify a specific row of data

In this example we would identify the row we want to access by saying the ROW where the CELL in the COLUMN labelled ‘name’ has the value ‘Chicago’

# Datatypes

- Character Types
  - `char(#)` - character.  # defined the length of the data.
  - `varchar(#)` - varying character. # defined the length of the data.
  - `text` - text based data that is not limited by a predefined size
- Numeric Types
  - `int` - similar to Java’s int
  - `serial` - works similar to the integers except these are automatically generated in the columns by PostgreSQL
  - `numeric(d,p)` - floating point numbers with d number of digits and p number of decimal places
- Other types
  - `boolean` - true/false
  - `date` - yyyy-mm-dd
  - `time` - hh:mm:ss
  - `timestamp` - yyyy-mm-dd hh:mm:ss

© 2022 Tech Elevator, LLC Confidential

# SELECT

The most basic SQL statement is a `SELECT` query, and it follows the following format:

```
SELECT [column], [column-n] FROM [table];
```

- `SELECT [column],[column-n]` indicates which columns that you want returned from your query.
- `FROM [table]` indicat ing

```
SELECT city_name, population FROM city;
```

```
SELECT * FROM city;
```

- `SELECT *` indicates ALL columns returned from your query

city table

| city_id | city_name | state_abbreviation | population |
|---|---|---|---|
| 3793 | New York | NY | 8008278 |
| 3794 | Los Angeles | CA | 3694820 |
| 3795 | Chicago | IL | 2896016 |
| 3796 | Houston | TX | 1953631 |
| 3797 | Philadelphia | PA | 1517550 |
| 3798 | Phoenix | AZ | 1321045 |
| 3799 | San Diego | CA | 1223400 |

# SELECT WHERE

The `WHERE` clause is used to filter the result set based on criteria rules.  Can include:

- **=, <>, !=, >, >=, <, <=**
- **IN(values), NOT IN(values)**
- **BETWEEN** value **AND** value
- **IS NULL, IS NOT NULL**
- **LIKE** (with wildcard character)

```
SELECT city_name, population FROM city WHERE state_abbreviation = ‘PA’;
```

```
SELECT city_name, population FROM city WHERE population > 3000000
```

city table

| city_id | city_name | state_abbreviation | population |
|---|---|---|---|
| 3793 | New York | NY | 8008278 |
| 3794 | Los Angeles | CA | 3694820 |
| 3795 | Chicago | IL | 2896016 |
| 3796 | Houston | TX | 1953631 |
| 3797 | Philadelphia | PA | 1517550 |
| 3798 | Phoenix | AZ | 1321045 |
| 3899 | Pittsburgh | PA | 301286 |

© 2022 Tech Elevator, LLC Confidential

# Objectives

- Write and execute simple select statements using SELECT, FROM, and WHERE
- **Use aggregate information using GROUP BY to group rows together**
- Use the ORDER BY clause to order results from the database
- Write select statements using subqueries

© 2023 Tech Elevator, LLC Confidential

# Execution Order

SQL is a declarative language and does not run from top to bottom and left to right. The order it runs is:

1. `FROM` clause - The database needs to know which table(s) you're selecting from first of all
2. `WHERE` clause - The database then needs to know which rows you'll work with
3. `GROUP BY` clause - The database then groups those rows according to your `GROUP BY` clause
4. `SELECT` clause - The database then collapses those rows down and selects the columns that you want data from
5. `ORDER BY` clause - The database orders the rows in the order that you ask for
6. `LIMIT` clause - The database only returns the number of resulting rows that you want

# Obnoxious tip to memorize the order

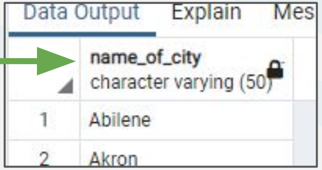
| | |
|---|---|
| **S**elect | **S**ome |
| **F**rom | **F**rench |
| **W**here | **W**aiters |
| **G**roup by | **G**row |
| **H**aving | **H**ealthy |
| **O**rder by | **O**ranges & |
| **L**imit | **L**emons |

© 2022 Tech Elevator, LLC Confidential

# Modifiers

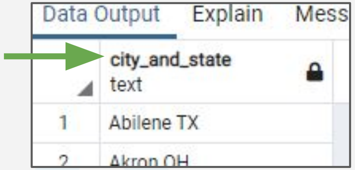
- AS can be used with a column name to give it an alias (new name)

```
SELECT city_name AS name_of_city FROM city;
```



| | name_of_city<br>character varying (50) |
|---|---|
| 1 | Abilene |
| 2 | Akron |

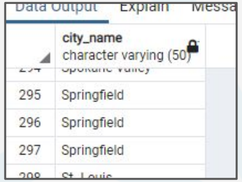
```
SELECT city_name || ' ' || state_abbreviationAS city_and_state FROM city;
```



| | city_and_state<br>text |
|---|---|
| 1 | Abilene TX |
| 2 | Akron OH |

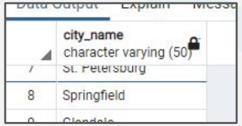
- DISTINCT can be used with a column name to return only unique values from that column.

```
SELECT city_name from city;
```



| | city_name<br>character varying (50) |
|---|---|
| 294 | Spokane Valley |
| 295 | Springfield |
| 296 | Springfield |
| 297 | Springfield |
| 298 | St. Louis |

```
SELECT DISTINCT city_name from city;
```



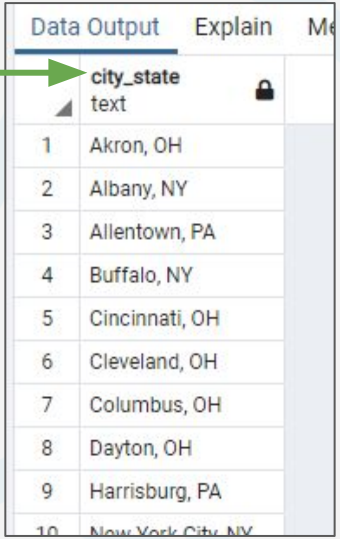
| | city_name<br>character varying (50) |
|---|---|
| 7 | St. Petersburg |
| 8 | Springfield |
| 9 | Glendale |

© 2022 Tech Elevator, LLC Confidential

# Concatenate

- In Postgres, we can concatenate the values across multiple columns into a single field using the `||` operator to concatenate strings.
- We can use the `AS` keyword to give the concatenated column a name.

```
SELECT city_name || ', ' || state_abbreviation
AS city_state FROM city
WHERE state_abbreviation IN ('PA', 'OH', 'NY');
```



| | city_state<br>text |
|---|---|
| 1 | Akron, OH |
| 2 | Albany, NY |
| 3 | Allentown, PA |
| 4 | Buffalo, NY |
| 5 | Cincinnati, OH |
| 6 | Cleveland, OH |
| 7 | Columbus, OH |
| 8 | Dayton, OH |
| 9 | Harrisburg, PA |
| 10 | New York City, NY |

# Aggregate Functions

Aggregate function performs a calculation on a set of values and returns a single value.

- AVG returns the average value of a numeric column
- SUM returns the total sum of a numeric column
- COUNT returns the number of rows matching criteria
- MIN returns the smallest value of the selected column
- MAX returns the largest value of the selected column

Except for COUNT(*), aggregate functions ignore null values. It's important to be aware of the difference between COUNT(*) and COUNT on a specific field.

If you specify a column name to count, only the rows in the table that have a value for that column will be returned. For instance, in our state data, this query returns 51 rows, while changing it to use COUNT(*) instead returns 56. This is because only 51 rows have a value (that is do not have a NULL value) for state_nickname.

```
SELECT COUNT(state_nickname) FROM state;
```

```
51
```

```
SELECT COUNT(*) FROM state;
```

```
56
```

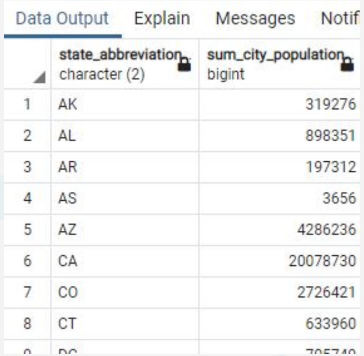
# GROUP BY

An aggregate function performs a calculation on one or more values and returns a single value.

Very often with aggregate functions such as `SUM`, we want the results to be grouped by the value of some attribute.  The `GROUP BY` clause groups the result set into groups of values and the aggregate function returns a single value for each group.

For example if our database contains city information, including state abbreviation  and population, we can report the total population in each state abbreviation  by using `SUM` and `GROUP BY` the state abbreviation .

```
SELECT state_abbreviation, SUM(population) as sum_city_population
FROM city
GROUP BY state_abbreviation;
```



| | state_abbreviation<br>character (2) | sum_city_population<br>bigint |
|---|---|---|
| 1 | AK | 319276 |
| 2 | AL | 898351 |
| 3 | AR | 197312 |
| 4 | AS | 3656 |
| 5 | AZ | 4286236 |
| 6 | CA | 20078730 |
| 7 | CO | 2726421 |
| 8 | CT | 633960 |
| 9 | DC | 705749 |

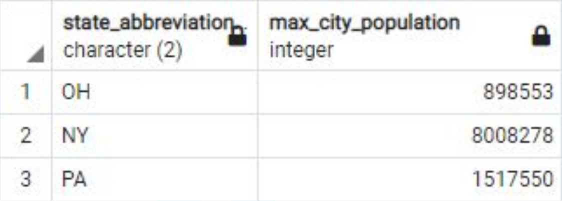
© 2022 Tech Elevator, LLC Confidential

# GROUP BY

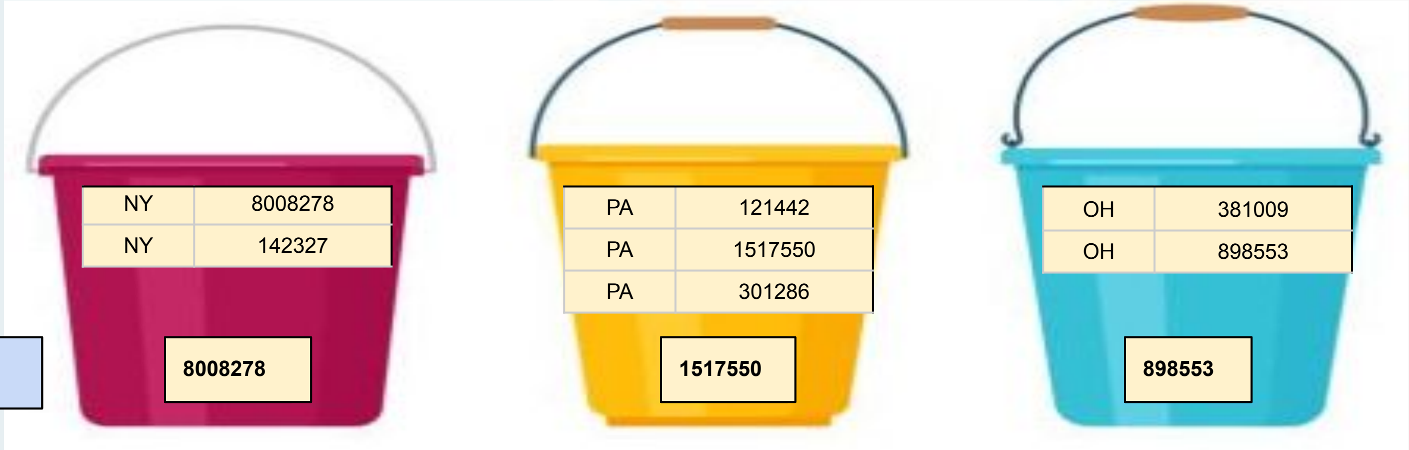
city table

| city_name | state_abbreviation | population |
|---|---|---|
| New York | NY | 8008278 |
| Allentown | PA | 121442 |
| Syracuse | NY | 142327 |
| Cleveland | OH | 381009 |
| Philadelphia | PA | 1517550 |
| Columbus | OH | 898553 |
| Pittsburgh | PA | 301286 |

```
SELECT
state_abbreviation,
MAX(population) as max_city_population
FROM city
GROUP BY state_abbreviation;
```



| | state_abbreviation<br>character (2) | max_city_population<br>integer |
|---|---|---|
| 1 | OH | 898553 |
| 2 | NY | 8008278 |
| 3 | PA | 1517550 |



| | |
|---|---|
| NY | 8008278 |
| NY | 142327 |

8008278

| | |
|---|---|
| PA | 121442 |
| PA | 1517550 |
| PA | 301286 |

1517550

| | |
|---|---|
| OH | 381009 |
| OH | 898553 |

898553

```
MAX(population)
```

# Objectives

- Write and execute simple select statements using SELECT, FROM, and WHERE
- Use aggregate information using GROUP BY to group rows together
- **Use the ORDER BY clause to order results from the database**
- Write select statements using subqueries

© 2023 Tech Elevator, LLC Confidential

# ORDER BY

A result set can be sorted using the `ORDER BY` clause:

```
SELECT col1, col2
FROM tablename
WHERE col1 = 'value'
ORDER BY col1 [ASC | DESC], col2 [ASC | DESC]
```

- Sort columns must exist in the table being queried or can be aliased columns
- Multiple column names can be provided which assigns a priority sort.
- Each column in `ORDER BY` clause can be specified as ascending (`ASC`) or descending (`DESC`). If not specified the default is `ASC`.

```
SELECT census_region, state_name, population
FROM state ORDER BY census_region, population DESC;
```

# LIMIT

By using `LIMIT` n at the end of a query, we can limit the size of our result set to n results.

```
SELECT state_name FROM state  LIMIT 10;
```

This tends to work best with `ORDER BY` as it allows you to construct lists like the top 10 states by population.

```
SELECT state_name, population
FROM state ORDER BY population DESC
LIMIT 10;
```

© 2022 Tech Elevator, LLC Confidential

# Objectives

- Write and execute simple select statements using SELECT, FROM, and WHERE
- Use aggregate information using GROUP BY to group rows together
- Use the ORDER BY clause to order results from the database
- **Write select statements using subqueries**

© 2023 Tech Elevator, LLC Confidential

# Subqueries

A subquery is referred to as an inner query and can provide the results of one query as input to another.

- Often used in the WHERE clause

```
SELECT city_name, state_abbreviation FROM city
WHERE state_abbreviation IN (SELECT state_abbreviation FROM state WHERE census_region = 'Northeast');
```

- Can be used in the SELECT clause (more on joins next week)

```
SELECT s.state_abbreviation, (SELECT c.city_name FROM city AS c WHERE c.city_id = s.capital)
FROM state AS s;
```

- Or even in the FROM clause.

```
SELECT s.city_state, s.population
FROM (SELECT city_name || ', ' || state_abbreviation AS city_state, population FROM city) AS s;
```

© 2022 Tech Elevator, LLC Confidential