Aalto University
School of Science

CS-E4610
Modern Database Systems
05.01.2018-05.04.2018

Tutorial 1

SQL and Semi-structured data with PostgreSQL

FREDERICK AYALA-GÓMEZ
PHD STUDENT IN COMPUTER SCIENCE, ELTE UNIVERSITY
VISITING RESEARCHER, AALTO UNIVERSITY

Agenda

Tutorials

Virtual Machine

PostgreSQL

- PostgreSQL At a Glance
- SQL
  - Data Control Language (DCL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
  - Transaction Control Language (TCL)
- Query Optimization
  - EXPLAIN ANALYZE
  - Indexing

Tutorials and Assignments (Programming)

Goal

- Practice and get experience:
  - PostgreSQL
  - ElasticSearch
  - Apache Spark (Batch and Streaming)

Expectations

- Get to known the technologies
- Be able to differentiate the use cases
- Hand-on

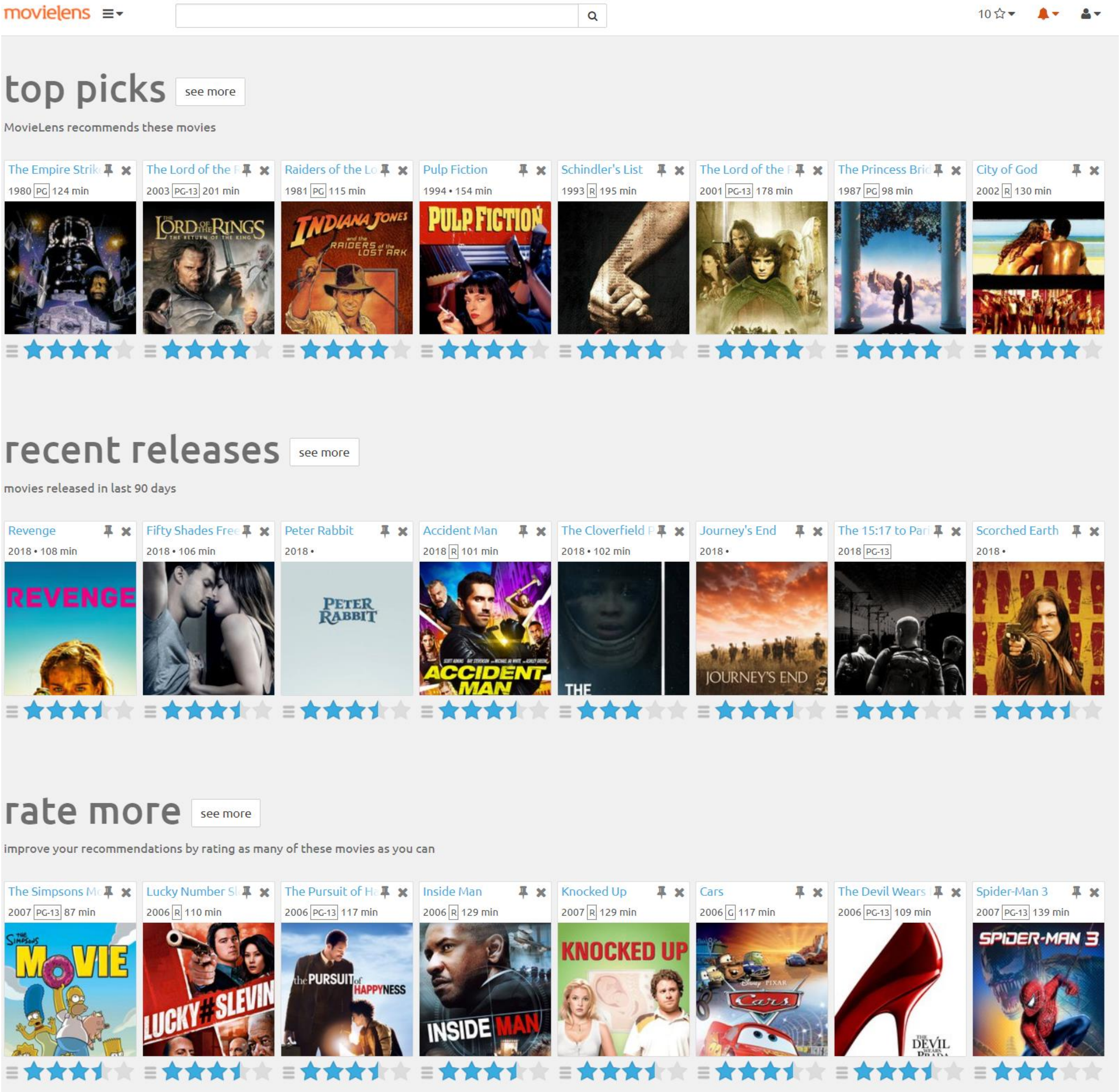
However…

- Mastering each technology is a course on its own
- Other interesting options (e.g., datomic, voltDB)

Dataset: Movielens

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

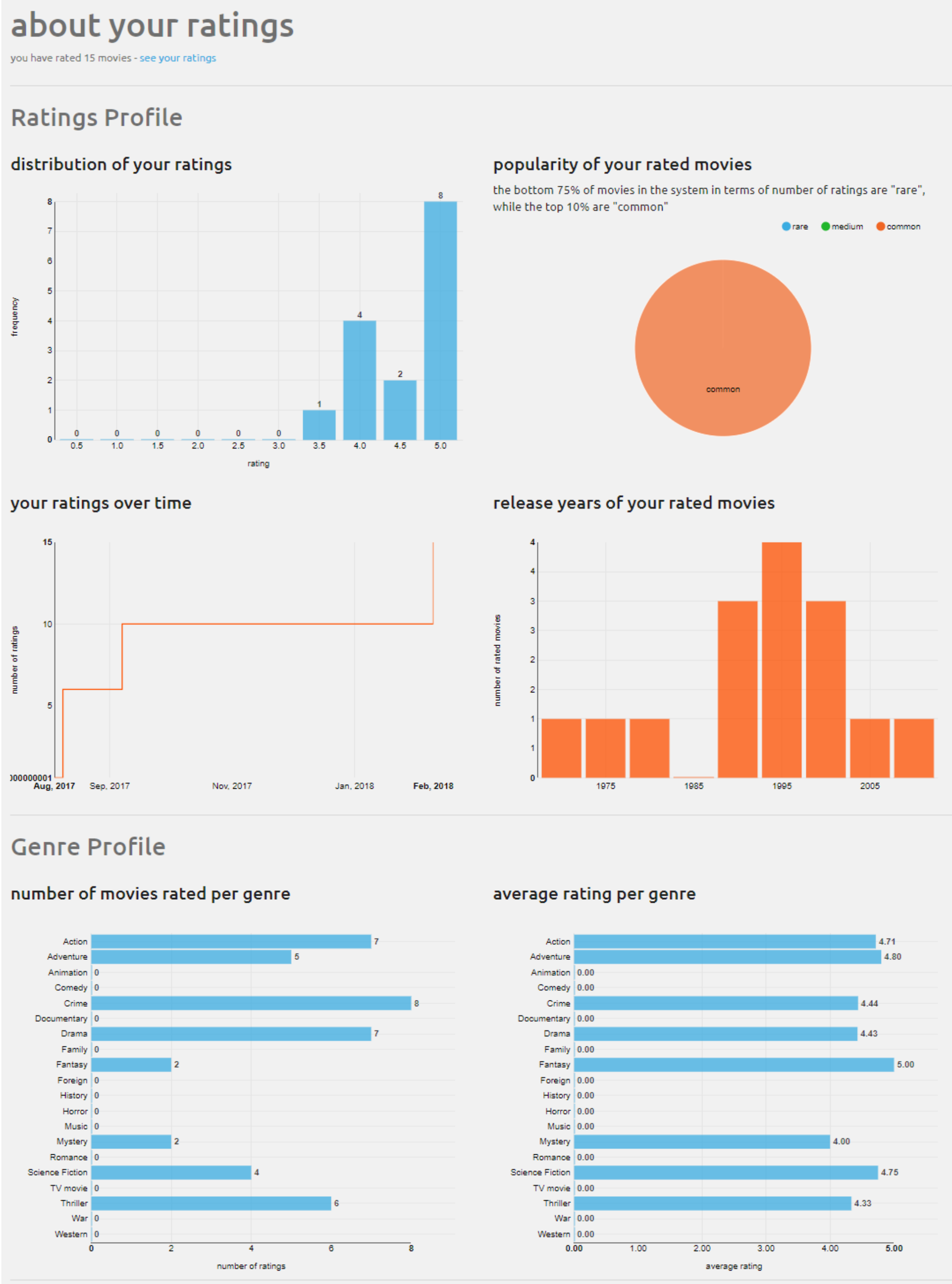
movielens.org



Dataset: Movielens

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

movielens.org



Dataset: Movielens

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

movielens.org

Movies (~3.9K)

- MovieID::Title::Genres

Users (~6K)

- UserID::Gender::Age::Occupation::Zip-code

Ratings (~1M)

- UserID::MovieID::Rating::Timestamp

DBpedia

Auer, Sören, et al. "Dbpedia: A nucleus for a web of open data." The semantic web. Springer, Berlin, Heidelberg, 2007. 722-735.

dbpedia.org

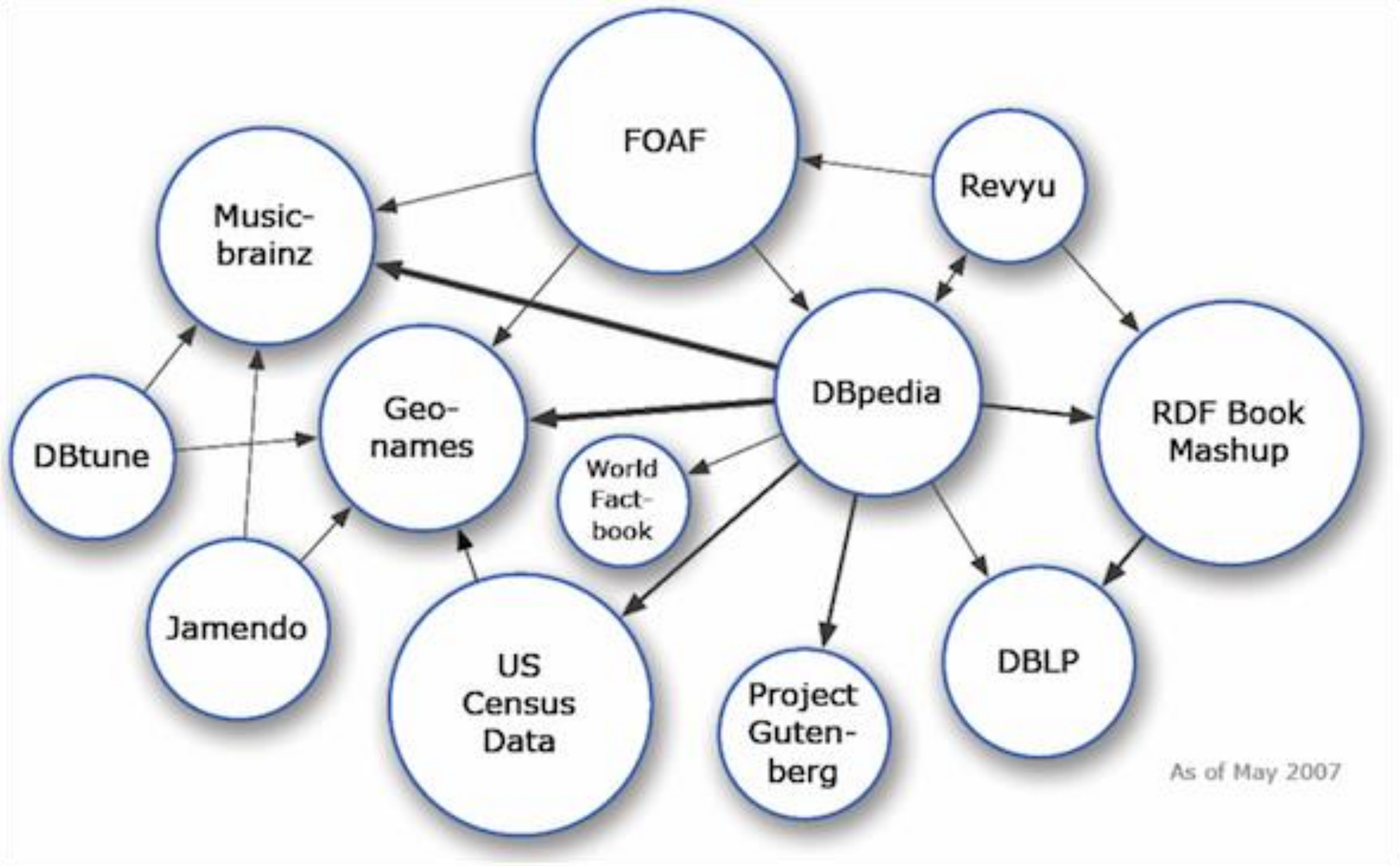
Mapping of Movielens to DBpedia taken from:

Fernández-Tobías, Ignacio, et al. "Accuracy and diversity in cross-domain recommendations for cold-start users with positive-only feedback." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016

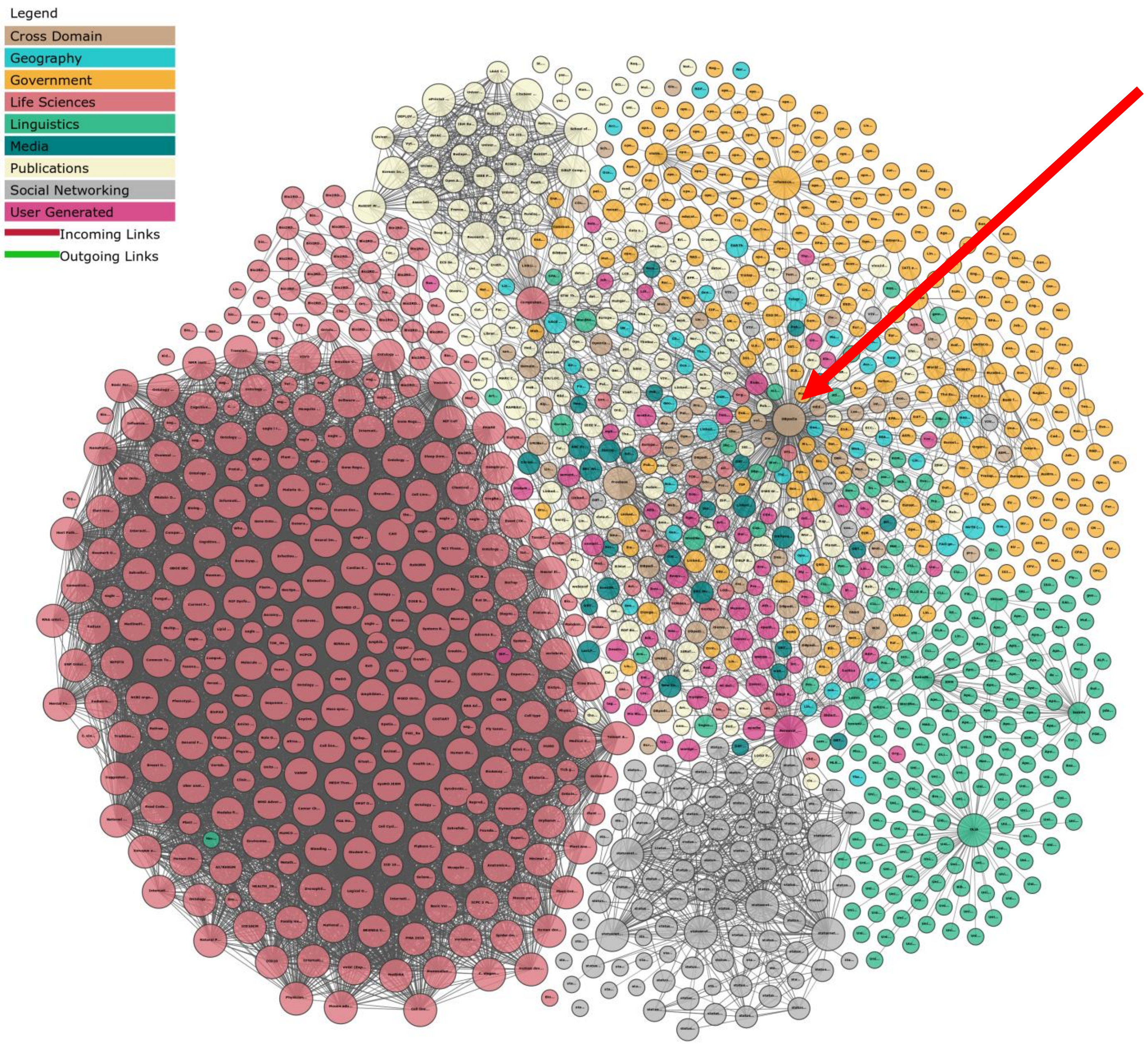
"DBpedia is a community effort to **extract structured information from Wikipedia** and to make this information available on the Web. DBpedia allows you to **ask sophisticated queries against datasets derived from Wikipedia** and to **link other datasets** on the Web to Wikipedia data."

- Auer, Sören, et al., 2007

Linked Open Data as of 2007



"Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/"



Linked Open Data as of 2017

"Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/"

Roadmap

Tutorial 1:

- Exploring Movielens with PostgreSQL
- Tools: pgsql, pgAdmin4
- Languages: SQL, JSON

Tutorial 2:

- Information retrieval and similarity search on movies
- Tools: Elasticsearch, wget, Jupyter Notebook
- Languages: Python, JSON

Tutorial 3:

- Batch: Building recommender systems
- Streaming: TBD… Twitter(?), BTC transactions (?)… ideas (?)
- Tools: Apache Spark. Python

Accessing the VM

Instructions on: https://github.com/frederickayala/mds2018

Two steps to access the VM:

- Configure key in https://version.aalto.fi
- Configure key, X11Forwarding, and compression for ssh

Feel free to work in your computer:

- Install PostgreSQL, ElasticSearch, Apache Spark. Python, Jupyter Notebook, and required python modules

PostgreSQL

For fundamental contributions to the concepts and practices underlying modern database systems.

PostgreSQL at a Glance

SQL

Indexes (**B-tree**, Hash, GiST, SP-GiST, **GIN** and BRIN)

Transactions and Locking

Log Files and System Statistics

Optimizing Queries (e.g., **ANALYZE EXPLAIN**)

Stored Procedures

Security (e.g., users, roles, permissions. Row-Level Security (RLS), encryption)

Backup and Recovery

Replication

Extensions: PostGIS

Migraton (e.g., **loading files from CSV**)

SQL – Data Definition Language (DDL)

Aalto University
School of Science

- CREATE
- DROP
- ALTER
- RENAME
- TRUNCATE

SQL – Data Definition Language (DDL)

Aalto University
School of Science

```
DROP TABLE IF EXISTS movie CASCADE;
CREATE TABLE movie
(movie_id int not null unique, title varchar, genres varchar, PRIMARY KEY(movie_id));
-- PostgreSQL creates an index on PRIMARY KEYS automatically, so there is no need to run:
-- CREATE INDEX movie_id ON movie USING btree (movie_id);

DROP TABLE IF EXISTS user_profile CASCADE;
CREATE TABLE user_profile
(user_profile_id int not null unique, gender varchar, age int, occupation int, zip_code varchar, PRIMARY KEY(user_profile_id));
-- PostgreSQL creates an index on PRIMARY KEYS automatically, so there is no need to run:
-- CREATE INDEX user_profile_user_profile_id ON user_profile USING btree (user_profile_id);

DROP TABLE IF EXISTS rating;
CREATE TABLE rating
(user_profile_id  int not null, movie_id int not null, rating int, rating_timestamp timestamp,
 FOREIGN KEY (user_profile_id) REFERENCES user_profile(user_profile_id),
 FOREIGN KEY (movie_id) REFERENCES movie(movie_id));

CREATE INDEX rating_user_profile_id ON rating USING btree (user_profile_id);
CREATE INDEX rating_movie_id ON rating USING btree (movie_id);

DROP TABLE IF EXISTS dbpedia;
CREATE TABLE dbpedia
(movie_id int not null unique, title varchar, dbpedia_url varchar, json_url varchar, dbpedia_content json,
 FOREIGN KEY (movie_id) REFERENCES movie(movie_id));

CREATE INDEX dbpedia_movie_id ON dbpedia USING btree (movie_id);
```

JSON vs JSONB

Differences	Major practical difference is efficiency.
JSON	Stores an exact copy of the input text (must reparse on each execution)
JSONB	Stores a decomposed binary format
	Slightly slower to input
	Significantly faster to process
	Also supports indexing

JSON vs JSONB

Querying a JSON field

movielens on movielens_user@movielens

```
-- Comparing JSON vs JSONB
ALTER TABLE dbpedia
  ALTER COLUMN dbpedia_content
  SET DATA TYPE json
  USING dbpedia_content::json;

EXPLAIN ANALYZE
SELECT * FROM dbpedia
WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Seq Scan on dbpedia (cost=0.00..633.99 rows=16 width=156) (actual time=0.103..162.309 rows=12 loops=1)
2	Filter: ((dbpedia_content ->> 'cinematography'::text) = 'Tak_Fujimoto'::text)
3	Rows Removed by Filter: 3254
4	Planning time: 0.378 ms
5	Execution time: 162.348 ms

JSON vs JSONB

Querying a JSONB field

movielens on movielens_user@movielens

```sql
-- Comparing JSON vs JSONB
ALTER TABLE dbpedia
  ALTER COLUMN dbpedia_content
  SET DATA TYPE jsonb
  USING dbpedia_content::jsonb;

EXPLAIN ANALYZE
SELECT * FROM dbpedia
WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Seq Scan on dbpedia (cost=0.00..581.99 rows=16 width=156) (actual time=0.069..44.109 rows=12 loops=1)
2	Filter: ((dbpedia_content ->> 'cinematography'::text) = 'Tak_Fujimoto'::text)
3	Rows Removed by Filter: 3254
4	Planning time: 0.479 ms
5	Execution time: 44.152 ms

SQL – Data Definition Language (DDL)

```
DROP TABLE IF EXISTS movie CASCADE;
CREATE TABLE movie
(movie_id int not null unique, title varchar, genres varchar, PRIMARY KEY(movie_id));
-- PostgreSQL creates an index on PRIMARY KEYS automatically, so there is no need to run:
-- CREATE INDEX movie_id ON movie USING btree (movie_id);

DROP TABLE IF EXISTS user_profile CASCADE;
CREATE TABLE user_profile
(user_profile_id int not null unique, gender varchar, age int, occupation int, zip_code varchar, PRIMARY KEY(user_profile_id));
-- PostgreSQL creates an index on PRIMARY KEYS automatically, so there is no need to run:
-- CREATE INDEX user_profile_user_profile_id ON user_profile USING btree (user_profile_id);

DROP TABLE IF EXISTS rating;
CREATE TABLE rating
(user_profile_id  int not null, movie_id int not null, rating int, rating_timestamp timestamp,
 FOREIGN KEY (user_profile_id) REFERENCES user_profile(user_profile_id),
 FOREIGN KEY (movie_id) REFERENCES movie(movie_id));

CREATE INDEX rating_user_profile_id ON rating USING btree (user_profile_id);
CREATE INDEX rating_movie_id ON rating USING btree (movie_id);

DROP TABLE IF EXISTS dbpedia;
CREATE TABLE dbpedia
(movie_id int not null unique, title varchar, dbpedia_url varchar, json_url varchar,  dbpedia_content json,
 FOREIGN KEY (movie_id) REFERENCES movie(movie_id));

CREATE INDEX dbpedia_movie_id ON dbpedia USING btree (movie_id);
```

JSONB Index

Creating a JSONB field index

ONLY WORKS ON JSONB fields

GIN stands for Generalized Inverted Index

```
DROP INDEX IF EXISTS dbpedia_content_cinematography;
CREATE INDEX dbpedia_content_cinematography ON dbpedia USING gin ((dbpedia_content -> 'cinematography'));
```

Querying on JSONB index

Using the equals operator "="

movielens on movielens_user@movielens

```
EXPLAIN ANALYZE
SELECT * FROM dbpedia
WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
--WHERE dbpedia_content -> 'cinematography' ? 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Seq Scan on dbpedia (cost=0.00..581.99 rows=16 width=156) (actual time=0.061..48.483 rows=12 loops=1)
2	Filter: ((dbpedia_content ->> 'cinematography'::text) = 'Tak_Fujimoto'::text)
3	Rows Removed by Filter: 3254
4	Planning time: 0.082 ms
5	Execution time: 48.512 ms

It does not use the index!

Querying on JSONB index

Using the operator "?“ which is indexable

movielens on movielens_user@movielens

```
EXPLAIN ANALYZE
SELECT * FROM dbpedia
--WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
WHERE dbpedia_content -> 'cinematography' ? 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

	QUERY PLAN text
1	Bitmap Heap Scan on dbpedia (cost=12.03..23.40 rows=3 width=156) (actual time=0.043..0.276 rows=14 loops=1)
2	Recheck Cond: ((dbpedia_content -> 'cinematography'::text) ? 'Tak_Fujimoto'::text)
3	Heap Blocks: exact=14
4	-> Bitmap Index Scan on dbpedia_content_cinematography (cost=0.00..12.02 rows=3 width=0) (actual time=0.024..0.024 rows=14 loops=1)
5	Index Cond: ((dbpedia_content -> 'cinematography'::text) ? 'Tak_Fujimoto'::text)
6	Planning time: 0.129 ms
7	Execution time: 0.321 ms

Much faster!

Query optimization in PostgreSQL

The task is to make a query perform better over some metric. Usually the *execution time.*

PostgreSQL provides a tool that helps us understand how the queries are executed

This tool is called **analyze**

Analyze

Taken from: Josh Berkus, EXPLAIN Explained, SCALE 14x, Pasadena, CA. USA

Aalto University
School of Science

WHAT EXPLAIN DOES

- Tell us **what** the planner will do
- Stats on **how** the query was executed
- Implies reasons **why** a query was slow
- Tell us **which** step in the query took the longest

WHAT EXPLAIN DOES NOT DO

- Why a particular index is not used
- How to rewrite queries
- What other factors slow the DB
- How much time the request took outside the DB

The Query Planner

Breaks the query down into atomic nodes

↓

Figures out ways to execute the nodes and estimates a cost

↓

Chain the combinations together into "plans"

↓

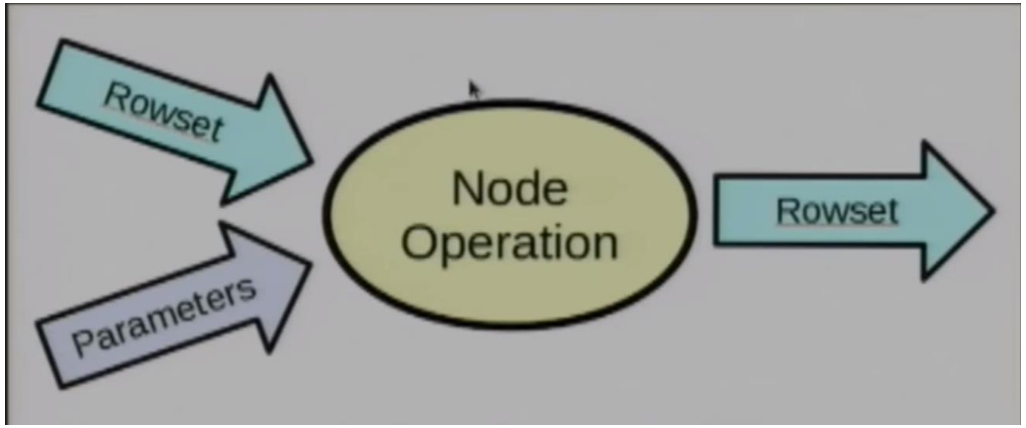
Calculate the total "cost" of each plan

↓

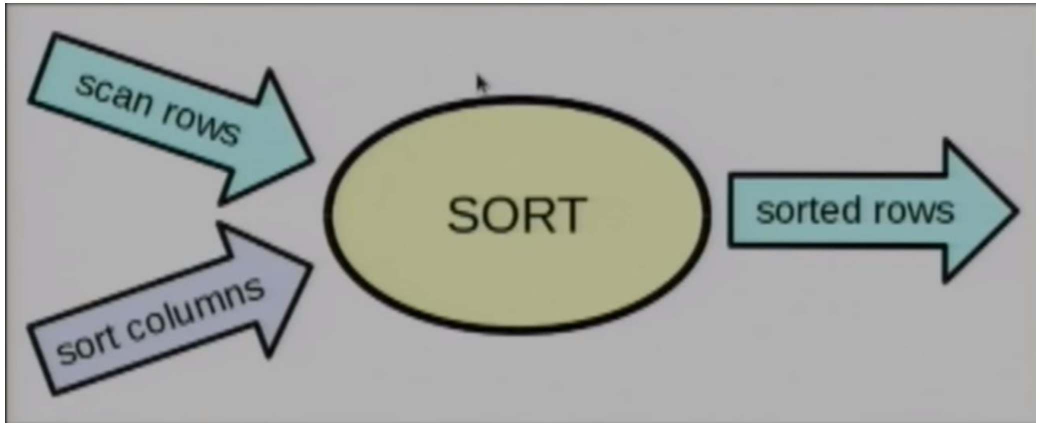
Picks the plan with the lowest cost

Query planner

Nodes in the query planner



For example, the sort node



Cost

- Cost is a unit that is meaningful to the query planner.
- The cost is relevant for a specific query.
- It's not comparable between different queries.
- It does not represent time.

EXPLAIN and EXPLAIN ANALYZE

EXPLAIN

- Shows what the planner decided to do

EXPLAIN ANALYZE

- Shows what the planner decided to do
- Shows that actually happened
- OK with safe operations
- Be careful with **delete, update**

EXPLAIN

movielens on movielens_user@movielens

```sql
EXPLAIN
SELECT * FROM dbpedia
WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

Operation | table | Espected cost, rows, and width (bytes per row)

	QUERY PLAN
1	Seq Scan on dbpedia (cost=0.00..581.99 rows=16 width=156)
2	Filter: ((dbpedia_content ->> 'cinematography'::text) = 'Tak_Fujimoto'::text)

EXPLAIN and EXPLAIN ANALYZE

movielens on movielens_user@movielens

```sql
EXPLAIN ANALYZE
SELECT * FROM dbpedia
WHERE dbpedia_content->>'cinematography' = 'Tak_Fujimoto';
```

Data Output | Explain | Messages | Query History

Operation | table | Espected cost, rows, and width (bytes per row)

	QUERY PLAN
1	Seq Scan on dbpedia (cost=0.00..581.99 rows=16 width=156) (actual time=0.098..78.392 rows=12 loops=1)
2	Filter: ((dbpedia_content ->> 'cinematography'::text) = 'Tak_Fujimoto'::text)
3	Rows Removed by Filter: 3254
4	Planning time: 0.346 ms
5	Execution time: 78.431 ms

Adds execution time

EXPLAIN ANALYZE BUFFERS

movielens on movielens_user@movielens

```
EXPLAIN (analyze on, buffers on)
SELECT * FROM rating
WHERE rating > 3;
```

Data Output | Explain | Messages | Query History

Operation | table | Espected cost, rows, and width (bytes per row)

1	Seq Scan on rating (cost=0.00..18873.61 rows=573153 width=20) (actual time=0.014..189.819 rows=575281 loops=1)
2	Filter: (rating > 3)
3	Rows Removed by Filter: 424928
4	Buffers: shared hit=6371
5	Planning time: 0.223 ms
6	Execution time: 228.384 ms

Adds readings from cache (shared) or filesystem (reads)

Execution time

PostgreSQL functions on queries

movielens on movielens_user@movielens

```sql
SELECT unnest(string_to_array(genres, '|')) as genre, title from movie
```

Data Output | Explain | Messages | Query History

	genre text	title character varying
1	Animation	Toy Story (1995)
2	Children's	Toy Story (1995)
3	Comedy	Toy Story (1995)
4	Adventure	Jumanji (1995)
5	Children's	Jumanji (1995)
6	Fantasy	Jumanji (1995)
7	Comedy	Grumpier Old Men (...
8	Romance	Grumpier Old Men (...
9	Comedy	Waiting to Exhale (19...
10	Drama	Waiting to Exhale (19...

Transactions

ACID

- **A**tomicity:
  - An transaction is either completed or not initialized at all (All or nothing).
- **C**onsistency:
  - At the end of the transaction the system is in a valid state (e.g., constraints, cascades, triggers).
- **I**solation:
  - Each transaction has exclusive rights on the resources
- **D**urability:
  - A transaction once completed, all of the changes are permanent.

That's all for now!

Aalto University
School of Science

Thanks!

Questions?

Frederick Ayala-Gómez
frederick.ayala@aalto.fi

Credits

The analyze section was based on Josh Berkus talk "EXPLAIN Explained" at SCALE 14x, Pasadena, CA. USA. Watch it here: https://www.youtube.com/watch?v=mCwwFAl1pBU