Data Management
05 Query Languages (SQL)

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

Last update: Apr 09, 2021
Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - [https://tugraz.webex.com/meet/m.boehm](https://tugraz.webex.com/meet/m.boehm)
  - Corona traffic light **RED** until end of April

- **#2 Reminder Communication**
  - **Newsgroup:** [news://news.tugraz.at/tu-graz.lv.dbase](news://news.tugraz.at/tu-graz.lv.dbase)
  - **Office hours:** Mo 12.30-1.30pm ([https://tugraz.webex.com/meet/m.boehm](https://tugraz.webex.com/meet/m.boehm))

- **#3 Exercise Submissions**
  - **Exercise 1:** Mar 30 + 7 late days, grading in progress
  - **Exercise 2:** Apr 27, published Apr 07
  - [https://mboehm7.github.io/teaching/ss21_dbs/02_ExerciseQueriesAPIs.pdf](https://mboehm7.github.io/teaching/ss21_dbs/02_ExerciseQueriesAPIs.pdf)
Agenda

- Structured Query Language (SQL)
- Other Query Languages (XML, JSON)
- Exercise 2: Query Languages and APIs
Structured Query Language (SQL)
What is a(n) SQL Query?

```
SELECT Firstname, Lastname, Affiliation, Location
FROM Participant AS R, Locale AS S
WHERE R.LID = S.LID
    AND Location LIKE '%, GER'
```

#1 Declarative:
what not how

#2 Flexibility:
closed → composability

#3 Automatic Optimization

#4 Physical Data Independence

<table>
<thead>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Affiliation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volker</td>
<td>Markl</td>
<td>TU Berlin</td>
<td>Berlin, GER</td>
</tr>
<tr>
<td>Thomas</td>
<td>Neumann</td>
<td>TU Munich</td>
<td>Munich, GER</td>
</tr>
</tbody>
</table>
Why should I care?

- **SQL as a Standard**
  - Standards ensure **interoperability**, avoid **vendor lock-in**, and protect **application investments**
  - **Mature standard** with heavy industry support for decades
  - **Rich eco system** (existing apps, BI tools, services, frameworks, drivers, design tools, systems)

- **SQL is here to stay**
  - Foundation of mobile/server **application data management**
  - **Adoption of existing standard** by new systems (e.g., SQL on Hadoop, cloud DBaaS)
  - Complemented by NoSQL abstractions, see lecture **10 NoSQL (key-value, document, graph)**
Overview SQL

- **Structured Query Language (SQL)**
  - **Data Definition Language (DDL)** → Manipulate the database schema
  - **Data Manipulation Language (DML)** → Update and query database
  - **Data Control Language (DCL)** → Modify permissions

- **Dialects**
  - Spectrum of system-specific dialects for **non-core features**
  - Data types and size constraints
  - Catalog, builtin functions, and tools
  - Support for new/optional features
  - Case-sensitive identifiers

<table>
<thead>
<tr>
<th>Name</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-SQL</td>
<td>Microsoft, Sybase</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>Oracle, (IBM)</td>
</tr>
<tr>
<td>PL/pgSQL</td>
<td>PostgreSQL, derived</td>
</tr>
<tr>
<td>Unnamed</td>
<td>Most systems</td>
</tr>
</tbody>
</table>
The History of the SQL Standard

- **SQL:1986**
  - ‘87 international edition

- **SQL:1989** *(120 pages)*
  - *Database Language SQL with Integrity Enhancements*,
    ANSI X3.135-1989, ISO-9075-1989(E)

- **SQL:1992** *(580 pages)*
  - ‘95 SQL/CLI (part 3), ‘96 SQL/PSM (part 4)

- **SQL:1999** *(2000 pages)*
  - Complete reorg, ‘00 OLAP, ’01 SQL/MED, ’01 SQL/OLB, ‘02 SQL/JRT

- **SQL:2003** *(3764 pages)*

[C. J. Date: A Critique of the SQL Database Language. SIGMOD Record 1984]
The History of the SQL Standard, cont.

- Overview SQL:2003

1: Framework

3: CLI
- Call Level Interface

4: PSM
- Persistent Stored Modules

9: MED
- Management of External Data

10: OLB
- Object Language Bindings

13: JRT
- Java Routines and Types

14: XML
- Extensible Markup Language

2: Foundation

(1) Enhanced Date/Time Fac.

(2) Enhanced Integrity Management

(7) Enhanced Objects

(8) Active Databases

(6) Basic Objects

11: Schemata

Core SQL (all SQL:92 entry, some extended SQL:92/SQL:99)

x: ... a part
(x) ... a package

10: OLAP

optional features

mandatory features
The History of the SQL Standard, cont.

Since SQL:2003 overall structure remained unchanged ...

- **SQL:2008** (???? pages)
  - E.g., XML XQuery extensions, case/trigger extension

- **SQL:2011** (4079 pages)
  - E.g., time periods, temporal constraints, time travel queries

- **SQL:2016** (???? pages)
  - E.g., JSON documents and functions (optional)

➤ **Note:** We can only discuss common primitives

Data Types in SQL:2003

- Large Variety of Types
  - With support for multiple spellings

SQL data types

- Composite Data Types
- Predefined Data Types
- User-defined Types (UDT)

Numeric
  - Exact
    - NUMERIC
    - DECIMAL
    - SMALLINT
    - INTEGER
    - BIGINT
  - Approximate
    - REAL
    - FLOAT
    - DOUBLE PRECISION

Interval

String
  - Bit
  - Blob
  - Character
    - Fixed
    - Varying

Boolean

Datetime
  - Date
  - Time
  - Timestamp

Implicit casts among numeric types and among character types
Data Types in PostgreSQL

- **Strings**
  - CHAR(n) → fixed-length character sequence (padded to n)
  - VARCHAR(n) → variable-length character sequence (n max)
  - TEXT → variable-length character sequence

- **Numeric**
  - SMALLINT → 2 byte integer (signed short)
  - INT/INTEGER → 4 byte integer (signed int)
  - SERIAL → INTEGER w/ auto increment
  - NUMERIC(p, s) → exact real with p digits and s after decimal point

- **Time**
  - DATE → date
  - TIMESTAMP/TIMESTAMPTZ → date and time, timezone-aware if needed

- **JSON**
  - JSON → text JSON representation (requires reparsing)
  - JSONB → binary JSON representation
Create, Alter, and Delete Tables

- **Create Table**
  - Typed attributes
  - Primary and foreign keys
  - **NOT NULL, UNIQUE** constraints
  - **DEFAULT** values
  - **CHECK** constraints

- **Alter Table**
  - **ADD/DROP** columns
  - **ALTER** data type, defaults, constraints, etc

- **Delete Table**
  - Delete table
  - **Note:** order of tables matters due to referential integrity

Template in SQL

```sql
CREATE TABLE Students ( 
  SID INTEGER PRIMARY KEY, 
  Fname VARCHAR(128) NOT NULL, 
  Lname VARCHAR(128) NOT NULL, 
  Mtime DATE DEFAULT CURRENT_DATE 
);
```

```sql
ALTER TABLE Students
ADD DoB DATE;
```

```sql
ALTER TABLE Students
ADD CONSTRAINT PKStudent PRIMARY KEY(SID);
```

```sql
DROP TABLE Students; -- sorry
DROP TABLE Students CASCADE;
DROP TABLE IF EXISTS Countries, Cities, Airports, Airlines, Routes, Planes, Routes_Planes;
```
Create and Delete Indexes

- **Create Index**
  - Create a secondary (nonclustered) index on a set of attributes
  - **Clustered**: tuples sorted by index
  - **Non-clustered**: sorted attribute with tuple references
  - Can specify uniqueness, order, and indexing method
  - **PostgreSQL methods**: btree, hash, gist, and gin
    → see lecture 07 Physical Design and Tuning

- **Delete Index**
  - Drop indexes by name

- **Tradeoffs**
  - Indexes often automatically created for **primary keys / unique** attributes
  - **Lookup/scan performance vs insert performance**
Database Catalog

- **Catalog Overview**
  - **Meta data** of all database objects (tables, constraints, indexes) → mostly read-only
  - Accessible through SQL
  - Organized by schemas (`CREATE SCHEMA tpch;`)

- **SQL Information_Schema**
  - Schema with tables for all tables, views, constraints, etc
  - **Example:** check for existence of accessible table

  ```sql
  SELECT 1 FROM information_schema.tables
  WHERE table_schema = 'tpch'
  AND table_name = 'customer'
  ```

  (defined as views over PostgreSQL catalog tables)
Insert

- **Insert Tuple**
  - **Insert a single tuple** with implicit or explicit attribute assignment
    
    ```sql
    INSERT INTO Students (SID, Lname, Fname, MTime, DoB) VALUES (7, 'Boehm', 'Matthias', '2002-10-01', '1982-06-25');
    ```
  - Insert attribute key-value pairs to use auto increment, defaults, NULLs, etc
    
    ```sql
    INSERT INTO Students (Lname, Fname, DoB) VALUES ('Boehm', 'Matthias', '1982-06-25'), (...), (...);
    ```

- **Insert Table**
  - **Redirect query result into**
    
    ```sql
    INSERT INTO Students
    SELECT * FROM NewStudents;
    ```
  - **Analogy Linux redirect (append):**
    
    ```bash
    cat NewStudents.txt >> Students.txt
    ```
Update and Delete

- **Update Tuple/Table**
  - Set-oriented update of attributes
  - Update single tuple via predicate on **primary key**

- **Delete Tuple/Table**
  - Set-oriented delete of tuples
  - Delete single tuple via predicate on **primary key**

- **Note:** Time travel and multi-version concurrency control
  - Deleted tuples might be just **marked as inactive**
  - See lecture 09 Transaction Processing and Concurrency

```sql
UPDATE Students
SET MTime = '2002-10-02'
WHERE LName = 'Boehm';

DELETE FROM Students
WHERE extract(year FROM mtime) < 2010;
```
Basic Queries

- Basic Query Template
  - Select-From-Where
  - Grouping and Aggregation
  - Having and ordering
  - Duplicate elimination

Example

```
SELECT [DISTINCT] Fname, Affil, Location
FROM Participant AS P,
     Locale AS L
WHERE P.LID = L.LID;
```

Structured Query Language (SQL)

```
SELECT [DISTINCT] <column_list>
FROM [<table_list>] |
     <table1> [RIGHT | LEFT | FULL] JOIN
     <table2> ON <condition>]
WHERE <predicate>]
GROUP BY <column_list>]
HAVING <grouping predicate>]
ORDER BY <column_list> [ASC | DESC]]
```
Basic Queries, cont.

- **Distinct and All**
  - Distinct and all alternatives
  - Projection w/ **bag semantics** by default

- **Sorting**
  - Convert a **bag** into a **sorted list** of tuples; order lost if used in other ops
  - Single order: \((Lname, Fname)\) **DESC**
  - Evaluated last in a query tree

- **Set Operations**
  - See **04 Relational Algebra and Calculus**
    - **UNION**, **INTERSECT**, **EXCEPT**
  - Set operations **set semantics** by default
    - **DISTINCT** (set) vs **ALL** (bag)

Structured Query Language (SQL)

```sql
SELECT DISTINCT Lname, Fname
FROM Students;
```

```sql
SELECT * FROM Students
ORDER BY Lname DESC, Fname DESC;
```

```sql
(SELECT Firstname, Lastname
FROM Participant2018)
UNION DISTINCT
(SELECT Firstname, Lastname
FROM Participant2013)
```
Grouping and Aggregation

- **Grouping and Aggregation**
  - **Grouping**: determines the distinct groups
  - **Aggregation**: compute aggregate f(B) per group
  - Column list can only contain grouping columns, aggregates, or literals
  - **Having**: selection predicate on groups and aggregates

- **Example**
  - Sales (Customer, Location, Product, Quantity, Price)
  - **Q**: Compute number of sales $\text{sumQ}$ and revenue per product $\text{sumQP}$

  ```sql
  SELECT Product, 
  sum(Quantity) AS SumQ, 
  sum(Quantity*Price) AS SumQP 
  FROM Sales 
  GROUP BY Product
  ```

<table>
<thead>
<tr>
<th>Product</th>
<th>SumQ</th>
<th>SumQP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>Quantity</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>A</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>
BREAK (and Test Yourself)

- Task: SQL queries for the following query trees.

\[ \text{Orders} \]

\[ \text{Products} \]

\[ \text{SELECT DISTINCT Customer, Date} \]
FROM Orders O, Products P
WHERE O.PID = P.PID
AND Name IN('Y','Z')
Subqueries

Subqueries in Table List
- Use a subquery result like a base table
- Modularization with `WITH C AS (SELECT ...)`

Subqueries w/ IN
- Check containment of values in result set of sub query

Other subqueries
- `EXISTS`: existential quantifier \( \exists x \) for correlated subqueries
- `ALL`: comparison (w/ universal quantifier \( \forall x \))
- `SOME/ANY`: comparison (w/ existential quantifier \( \exists x \))

```
SELECT S.Fname, S.Lname, C.Name
FROM Students AS S,
     (SELECT CID, Name FROM Country
      WHERE ...) AS C
WHERE S.CID=C.CID;
```

```
SELECT Product, Quantity, Price
FROM Sales
WHERE Product NOT IN(
    SELECT Product FROM Sales
    GROUP BY Product
    HAVING sum(Quantity*Price)>1e6)
```
Correlated and Uncorrelated Subqueries

- **Correlated Subquery**
  - Evaluated subquery for every tuple of outer query
  - Use of attribute from table bound in outer query inside subquery

- **Uncorrelated Subquery**
  - Evaluate subquery just once
  - No attribute correlations between subquery and outer query

- **Query Unnesting (de-correlation)**
  - Rewrite during query compilation
  - See lecture 08 Query Processing

Structured Query Language (SQL)

```
SELECT P.Fname, P.Lname
FROM Professors P,
WHERE NOT EXISTS(
    SELECT * FROM Courses C
    WHERE C.PID=P.PID);
```

```
SELECT P.Fname, P.Lname
FROM Professors P,
WHERE P.PID NOT IN(
    SELECT PID FROM Courses);
```

[Thomas Neumann, Alfons Kemper: Unnesting Arbitrary Queries. BTW 2015]
Recursive Queries

**Approach**
- WITH RECURSIVE `<name> (<arguments>)`
- Compose recursive table from non-recursive term, union all/distinct, and recursive term
- Terminates when recursive term yields empty result

**Example**
- Courses(CID, Name), Precond(pre REF CID, suc REF CID)
- Dependency graph (pre → suc)

WITH RECURSIVE rPrereq(p,s) AS(

(SELECT pre, suc
 FROM Precond WHERE suc=5)

UNION DISTINCT
(SELECT B.pre, B.suc
 FROM Precond B, rPrereq R
 WHERE B.suc = R.p)
)

SELECT DISTINCT p FROM rPrereq
Procedures and Functions

- **Overview Procedures and Functions**
  - Stored programs, written in PL/pgSQL or other languages
  - Control flow (loops, branches) and SQL queries

- **(Stored) Procedures**
  - Can be called standalone via `CALL <proc_name>(<args>);`
  - Procedures return no outputs

- **Functions**
  - Can be called standalone or inside queries
  - Functions are value mappings
  - Table functions can return sets of records with multiple attributes

```sql
CREATE FUNCTION sampleProp(FLOAT)
RETURNS FLOAT
AS 'SELECT $1 * (1 - $1);'
LANGUAGE SQL;
```

```sql
CREATE PROCEDURE prepStud(a INT)
LANGUAGE PLPGSQL AS $$
BEGIN
  DELETE FROM Students;
  INSERT INTO Students
  SELECT * FROM NewStudents;
END; $$;
```
Triggers

- **Overview Trigger**
  - Similar to stored procedure but register ON INSERT, DELETE, or UPDATE
  - Allows complex check constraints and active behavior such as replication, auditing, etc (good and bad)

- **Trigger Template**
  ```sql
  CREATE TRIGGER <triggername>
  [BEFORE | AFTER | INSTEAD OF]
  [INSERT | DELETE | (UPDATE OF <column_list>)]
  ON <tablename>
  [REFERENCING <old_new_alias_list>]
  [FOR EACH {ROW | STATEMENT}]
  [WHEN (<search condition>)]
  <SQL procedure statement> |
  BEGIN ATOMIC
  {<SQL Procedure statement>;}...
  END
  ```

Not supported in PostgreSQL (need single UDF)
Views and Authorization

- **Creating Views**
  - Create a logical table from a query
  - Inserts can be propagated back to base relations only in special cases
  - Allows authorization for subset of tuples

- **Access Permissions Tables/Views**
  - **Grant** query/modification rights on database objects for specific users, roles
  - **Revoke** access rights from users, roles (recursively revoke permissions of dependent views via **CASCADE**)

```sql
CREATE VIEW TeamDM AS
SELECT * FROM Employee E, Employee M
WHERE E.MgrID = M.EID
AND M.login = 'mboehm';

GRANT SELECT
ON TABLE TeamDM
TO mboehm;

REVOKE SELECT
ON TABLE TeamDM
FROM mboehm;
```
Beware of SQL Injection

- Problematic SQL String Concatenation
  
  ```
  INSERT INTO Students (Lname, Fname) VALUES ('"+ @lname +"','"+ @fname +"');
  ```

- Possible SQL-Injection Attack

  ```
  HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME
  COMPUER TROUBLE.

  OH, DEAR - DID HE
  BREAK SOMETHING?
  IN A WAY-

  DID YOU REALLY
  NAME YOUR SON
  Robert'); DROP
  TABLE Students; -- ?

  OH, YES. LITTLE
  BOBBY TABLES,
  WE CALL HIM.

  WELL, WE'VE LOST THIS
  YEAR'S STUDENT RECORDS.
  I HOPE YOU'RE HAPPY.

  AND I HOPE
  YOU'VE LEARNED
  TO SANITIZE YOUR
  DATABASE INPUTS.
  ```

  ```
  INSERT INTO Students (Lname, Fname) VALUES (‘Smith‘,’Robert’);
  DROP TABLE Students; --’);
  ```
Other Query Languages
(XML, JSON)
No really, why should I care?

- **Semi-structured XML and JSON**
  - *Self-contained documents* for representing nested data
  - *Common data exchange formats* without redundancy of flat files
  - Human-readable formats → often used for SW configuration

- **Goals**
  - *Awareness of XML and JSON* as data models
  - Query languages and embedded querying in SQL
XML (Extensible Markup Language)

- **XML Data Model**
  - Meta language to define specific *exchange formats*
  - Document format for *semi-structured data*
  - Well formedness
  - XML schema / DTD

- **XPath (XML Path Language)**
  - Query language for *accessing collections of nodes* of an XML document
  - Axis specifies for ancestors, descendants, siblings, etc

- **XSLT (XML Stylesheet Language Transformations)**
  - Schema mapping (transformation) language for XML documents

- **XQuery**
  - Query language to extract, transform, and analyze XML documents

---

```xml
<?xml version="1.0" encoding="UTF-8"?>
<data>
  <student id="1">
    <course id="INF.01017UF" name="DM"/>
    <course id="706.550" name="AMLS"/>
  </student>
  <student id="5">
    <course id="706.520" name="DIA"/>
  </student>
</data>
```

```
/data/student[@id='1']/course/@name
```

```
"DM"
"AMLS"
```
XML in PostgreSQL, cont.

- **Overview XML in PostgreSQL**
  - Data types **TEXT** or **XML** (well-formed, type-safe operations)
  - ISO/IEC 9075-14 XML-related specifications (SQL/XML)

- **Creating XML**
  - Various **built-in functions** to parse documents, and create elements/attributes
  - `XMLPARSE(<xml_document>)` → **XML type**
  - `XMLELEMENT / XMLATTRIBUTES`

- **Processing XML**
  - Execute **XPath** expressions on XML types
  - `XMLEXIST` with **XPath instead of XQuery**
  - `XPATH` with optional namespace handling

**INSERT INTO** Students
(Fname, Lname, Doc)
VALUES('John', 'Smith',
xmlparse(<source_doc>));

**SELECT** Fname, Lname,
xpath('/student/@id', Doc)
FROM Students
JSON (JavaScript Object Notation)

- **JSON Data Model**
  - Data exchange format for **semi-structured data**
  - Not as verbose as XML (especially for arrays)
  - Popular format (e.g., Twitter)

- **Query Languages**
  - **Most common:** libraries for tree traversal and data extraction
  - **JSONiq:** XQuery-like query language
  - **JSONPath:** XPath-like query language

---

**JSONiq Example:**

```json
declare option jsoniq-version "…";
for $x in collection("students")
  where $x.id lt 10
  let $c := count($x.courses)
return {“sid”:$x.id, “count”:$c}
```

JSON in PostgreSQL, cont.

- **Overview JSON in PostgreSQL**
  - Alternative data types: **JSON** (text), **JSONB** (binary, with restrictions)
  - Implements RFC 7159, built-ins for conversion and access

- **Creating JSON**
  - Built-in functions for creating JSON from tables and tables from JSON input

- **Processing JSON**
  - Specialized operators for tree traversal and data extraction
    - **-> operator**: get JSON array element/object
    - **->>> operator**: get JSON array element/object as text
    - Built-in functions for extracting json (e.g., `json_each`)

---

**SELECT** `row_to_json(t)` FROM (SELECT `Fname`, `Lname` FROM `Students`) t

**SELECT** `Fname`, `Lname`, `Doc->students->>id` FROM `Students`
Exercise 2: Query Languages and APIs

Published: Apr 07, 2021
(updated: data Apr 07, task description Apr 08)
Deadline: Apr 27, 2021
Exercises: Summer Olympics

- **Dataset**
  - Past Summer Olympics (games, teams, athletes, events, medals)
  - Clone or download your copy from [https://github.com/tugraz-isds/datasets.git](https://github.com/tugraz-isds/datasets.git) (summer_olympics)
  - **Finished data cleaning by Apr 07**

- **Exercises**
  - **01** Data modeling (relational schema)
  - **02** Data ingestion and SQL query processing
  - **03** Physical design tuning, query processing, and transaction processing
  - **04** Large-scale data analysis (distributed query processing and ML model training)
Task 2.1: Schema Creation via SQL (3/25 points)

- **Schema creation via SQL**
  - Relies on lectures [04 Relational Algebra](#) and [05 Query Languages (SQL)](#)
  - Setup DBMS PostgreSQL, and start pgAdmin (UI), or psql (terminal)
  - Create database `db<studentID>` and setup relational schema, including primary keys, foreign keys, NOT NULL, UNIQUE

- **Recommended Schema**
  - Feel free to use and submit the provided schema

- **Partial Results**
  - `CreateSchema.sql`
Task 2.2 Data Ingestion via CLI (10/25 points)

- **Data Ingestion Program via ODBC/JDBC**
  - Relies on lectures 05 Query Languages (SQL) and 06 APIs (ODBC, JDBC)
  - Write a program that performs deduplication and data ingestion
  - Programming language of your choosing (Python, Java, C#, C++ recommended)

- **Data Ingestion Process**
  - Data: [https://github.com/tugraz-isds/datasets/tree/master/movies](https://github.com/tugraz-isds/datasets/tree/master/movies)
  - Invoke your ingestion program as follows → script to compile and run

    ```bash
    ./runIngestData.sh ./AthleteEvents.csv ./HostCities.csv ./NOCRegions.csv <host> <port> <database> <user> <password>
    (e.g., localhost 5432 db1234567 postgres postgres)
    ```

- **Partial Results**
  - Source code IngestData.*, and
  - Script runIngestData.sh
Task 2.3: SQL Query Processing (10/25 points)

- **SQL Query Processing**
  - Relies on lecture 05 Query Languages (SQL)
  - Expected results: [https://mboehm7.github.io/teaching/ss21_dbs/Results.zip](https://mboehm7.github.io/teaching/ss21_dbs/Results.zip)

- **List of Queries**
  - **Q01**: Obtain the detailed information of the athlete Usain St. Leo Bolt. (return name, gender, day of birth, height)
  - **Q02**: Compute the distinct cities that hosted the Olympic games. (return distinct {city name, country name} pairs, sorted ascending by country name)
  - **Q03**: List the athletes of team Austria 2012. (return name, gender, day of birth, sorted ascending by gender, day of birth)
  - **Q04**: Compute for each sport type, the number of distinct event types as well as its last (i.e., maximum) year of occurrence. (return sport name, count, last occurrence, sorted descending by count)
  - **Q05**: Determine the years in which female athletes whose names contain Ledecky won (in total) greater or equal than five gold or silver medals. (return year)
Task 2.3: SQL Query Processing (10/25 points)

- **List of Queries, cont.**
  - **Q06:** How many medals of each type did Michael Fred Phelps II win? (return medal, count, sorted descending by count)
  - **Q07:** Determine the top-10 athletes that participated between 1948 and 2016 in the most event occurrences but never won a single medal. (return name, day of birth, participation count, sorted descending by count, name).
  - **Q08:** Compute the athlete-centric Olympic medal table of 2016, which counts every medal awarded (i.e., multiple medals in team events). (return country name, NOC, #gold, #silver, #bronze, #total, sorted descending by {#gold, #silver, #bronze, #total}) // be careful about years

- **Partial Results**
  - SQL Script for each query: Q01.sql, Q02.sql, ..., Q08.sql
Task 2.4: Query Plans (2/25 points)

- Explain Query Plans
  - Relies on lecture 04 Relational Algebra and 05 Query Languages (SQL)
  - Obtain and analyze execution plans of Q05

- Example

  EXPLAIN VERBOSE
  SELECT L.location, count(*)
  FROM Participant P,
      Locale L
  WHERE P.lid = L.lid
  GROUP BY L.location
  HAVING count(*)>1

  "HashAggregate (...)" // grouping
  " Output: l.location, count(*)"
  " Group Key: l.location"
  " Filter: (count(*) > 1)" // selection
  " -> Hash Join (...)" // join
  "     Output: l.location" // projection
  "     Hash Cond: (l.lid = p.lid)"
  "     -> Seq Scan on Locale l (...)"
  "     Output: 1.lid, 1.location"
  "     -> Hash (...)
  "     Output: p.lid" // projection
  "     -> Seq Scan on Participant p (...)"
  "     Output: p.lid"

- Partial Results
  - ExplainQ05.sql
Conclusions and Q&A

- **Summary**
  - History and fundamentals of the **Structured Query Language (SQL)**
  - Awareness of **XML and JSON** (data model and querying)

- **Exercise Submissions**
  - **Exercise 1**: Mar 30 + 7 late days, grading in progress
  - **Exercise 2**: Apr 27, published Apr 07

- **Next Lectures**
  - **06 APIs (ODBC, JDBC, OR frameworks)** [Apr 19] → videos available
  - **07 Physical Design and Tuning** [Apr 26]
  - **08 Query Processing** [May 03]
  - **09 Transaction Processing and Concurrency** [May 10]