

Databases

04 Query Languages (SQL)

Lucas Iacono

Know-Center & Graz University of Technology, Austria

Computer Science and Biomedical Engineering
Institute of Human Centred Computing

Course calendar

- Part 2: Queries and database APIs
 - **(4) Mon. 17.11.2025: Query languages (SQL) + Exercise 2 (Available Wednesday)**

Exercise 1 submission: Tue. 18.11.2024 23:59

- (5) Mon. 24.11.2025: no lecture (time for project concept)

Course project concept submission: Tue. 25.11.2024 (groups of 5! less not allowed)

- Mon. 01.12.2025: Data handling and manipulation with Python (Pandas)

Last lecture

- **Relational data model**

- **Entities** → **Relationships** → **Simplify** → **Attributes**

- **Database normalization**

- **NF1:** only atomic values
 - e.g. phone numbers
- **NF2:** all non-key attributes functional depended on primary key
 - Orders(ProductID, StoreID, ProductName) →
 - Products(ProductID, ProductName) & Orders(ProductID, StoreID)
- **NF3:** no functional dependencies among non-key attributes
 - Employees(EmpID, EmpName, Department, DeptLocation) →
 - Employees(EmpID, EmpName, Department) & Departments(Department, DeptLocation)

This week: Agenda

- **Structured Query Language (SQL)**
 - Data Definition Language
 - **Data Manipulation Language**
 - Data Control Language (just a little bit)

- **Exercise 2 (online on Wednesday)**

Structured Query Language (SQL)

who has ever written a SQL query/statement?

What is a SQL Query?

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

What is a SQL Query?

```

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



Firstname	Lastname	Affiliation	Location
Volker	Markl	TU Berlin	Berlin, GER
Thomas	Neumann	TU Munich	Munich, GER

What is a 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



Firstname	Lastname	Affiliation	Location
Volker	Markl	TU Berlin	Berlin, GER
Thomas	Neumann	TU Munich	Munich, GER

#2 **Flexibility:**
composability

#3 **Automatic Optimization**

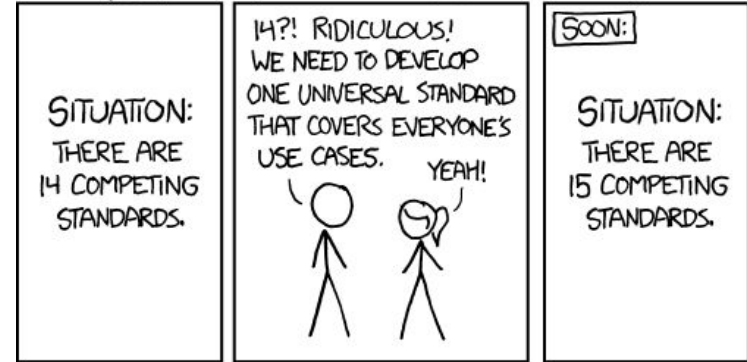
#4 **Physical Data Independence**

Why should I care?

- **SQL as a Standard**

- Standards ensure **interoperability**, and protect **application investments**
- **Mature standard** with heavy industry support for decades
- **Rich ecosystem** (existing apps, services, frameworks, drivers, design tools, systems)

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)



[<https://xkcd.com/927/>]

Why should I care?

- **SQL is here to stay**
 - Foundation of mobile/server **applications**
 - **Highly relevant for data science / data analytics**
 - Complemented by NoSQL abstractions

Why should I care?

- **SQL is here to stay**
 - Foundation of mobile/server **applications**
 - **Highly relevant for data science / data analytics**
 - Complemented by NoSQL abstractions



Overview SQL

- **Structured Query Language (SQL)**
 - **Data Definition Language (DDL)** → Manipulate the database schema
 - E.g., CREATE TABLE statements (**Exercise 1**)
 - **Data Manipulation Language (DML)** → Update and query database
 - Main content of today (**Exercise 2**)
 - **Data Control Language (DCL)** → Modify permissions
 - Very small part of today

Overview SQL

- **Dialects**

- Spectrum of system-specific dialects for **non-core features**
 - Data types
 - Built-in functions, and tools
 - Support for new/optional features

- **Web-Resource**

- <https://www.w3schools.com/sql/default.asp>

Name	Examples
T-SQL	Microsoft
PL/SQL	Oracle
PL/pgSQL	PostgreSQL

Data Definition Language (DDL)

Create Tables

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

- Typed attributes
- Primary and foreign keys
- **NOT NULL** constraint
 - Primary key has **NOT NULL + UNIQUE** constraints
- **DEFAULT** values

Recap: Example UniversityDB

Professors

<u>PID</u>	Title	Firstname	Lastname
1	Univ.-Prof. Dipl.-Inf. Dr.	Stefanie	Lindstaedt
3	Univ.-Prof. Dr.-Ing.	Matthias	Boehm
7	Dipl. Ing.	Adrian	Spataru

```
CREATE TABLE Professors (
  PID INTEGER SERIAL PRIMARY KEY,
  Title VARCHAR(128),
  Firstname VARCHAR(128),
  Lastname VARCHAR(128)
);
```

→ alternative for composite key:

```
...
PRIMARY KEY(A1, A2)
...
```

→ **SERIAL** keyword for auto-increment

Recap: Example UniversityDB

Courses

<u>CID</u>	Title	ECTS	PID
INF.01017UF	Data Management (VO)	3	7
INF.02018UF	Data Management (KU)	1	7
706.010	Databases	3	7

```
CREATE TABLE Courses (
  CID INTEGER PRIMARY KEY,
  Title VARCHAR(256),
  ECTS INTEGER NOT NULL,
  PID INTEGER
  REFERENCES Professors
```

```
);
```

→ alternative for composite key:

```
...
FOREIGN KEY(A1, A2)
  REFERENCES R(A1, A2)
```

...

→ R could be **Professors**

Recap: Referential Integrity Constraints

- **Foreign Keys:**

- Reference of a primary key in another relation
- **Referential integrity:** FK need to reference existing tuples or NULL

- **Enforcing Referential Integrity**

- **#1 Error** (default)
- **#2 Propagation** on request
 - E.g., for existential dependence
- **#2 Set NULL** on request
 - E.g., for independent entities

DELETE FROM Professors WHERE PID=18



```
CREATE TABLE Courses (...
  PID INTEGER REFERENCES Professors
  ON DELETE CASCADE);
```

```
CREATE TABLE Courses (...
  PID INTEGER REFERENCES Professors
  ON DELETE SET NULL);
```

Recap: Domain and Semantic Constraints

- **Domain/Semantic Constraints**

- Value constraints of individual attributes
- **CHECK:** Value ranges or enumerated valid values

```
CREATE TABLE Courses (  
    CID INTEGER PRIMARY KEY,  
    ECTS INTEGER  
    CHECK (ECTS BETWEEN 1 AND 10)  
);
```

- + NOT_NULL, UNIQUE, etc.

Alter Tables

- **ADD/DROP** columns

```
ALTER TABLE Students ADD DoB DATE;  
ALTER TABLE Students DROP DateOFBirth;
```

- **ALTER** data type, defaults, constraints, etc

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

Delete Tables

- **CASCADE** for ensuring referential integrity

DROP TABLE Students;

DROP TABLE Students **CASCADE**;



Delete Tables

```
DROP TABLE IF EXISTS Countries,  
Cities, Airports, Airlines,  
Routes, Planes, Routes_Planes;
```

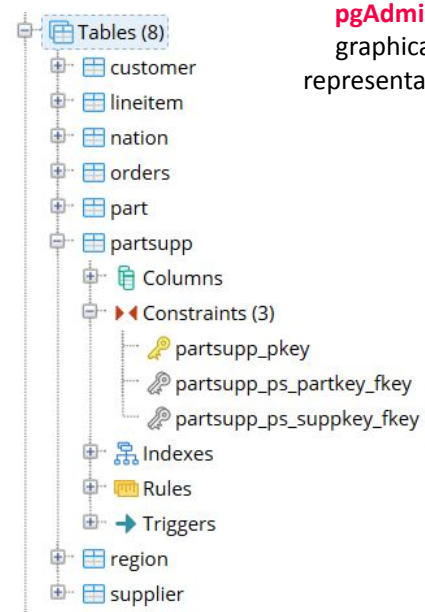
- **Note:** often used at the **beginning of CREATE TABLE scripts**

Database Catalog

- **Catalog Overview**

- **Metadata** of all database objects (e.g., tables, constraints) → **read-only**
- **Accessible through SQL**
- Organized by schemas

```
SELECT * FROM information_schema.tables;
```



pgAdmin
graphical
representation

Data Manipulation Language (DML)

Insert

- **Insert Tuple**

- **Insert a single tuple** with implicit or **explicit** attribute assignment
 - **No need to specify** the attributes in case you **add values for all attributes**

```
INSERT INTO Students (SID, Lname, Fname, MTime, DoB)  
  VALUES (7, 'Smith', 'John', '2022-10-01', '1969-04-20');
```

Insert

- **Insert Tuple**

- Example to use **auto increment attributes and defaults**
- With **commas** we can insert multiple tuples at once

```
INSERT INTO Students (Lname, Fname, DoB)  
VALUES ('Smith', 'John', '1969-04-20'),  
        (...), (...);
```

```
SERIAL SID,  
DEFAULT MTime
```

Insert

- **Insert from query**
 - Redirect query result into **INSERT**

```
INSERT INTO Students  
  SELECT * FROM OldStudents;
```

Insert

- **Insert from CSV file**

- Read and insert from a file-name using **COPY**
 - The attributes in the csv file need to match the ones in the DB table
 - file-name is the path, e.g., 'C:\Users\UserName\FolderName\file.csv'
 - Arguments can be added, e.g., **DELIMITER**
 - **CSV** argument if you have a .csv file
 - **HEADER** argument if you have a .csv file with a header

```
COPY table-name FROM file-name CSV HEADER DELIMITER ',';
```

```
/* this is a comment to document what is happening across  
multiple lines - make sure that user “everyone” from Windows has  
read access to the files */
```

Update and Delete

- **Update Tuple/Table**

- **Update** of attributes
- Without **Where** all tuples are updated

- **Delete Tuple/Table**

- **Delete** of tuples
- Without **Where** all tuples are deleted

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

```
DELETE FROM Students  
  WHERE LName = 'Smith';
```

Basic Queries

```
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

```
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 Query Template

- **Select-From-Where** (with joins or without)
- Grouping and Aggregation (e.g., sum / avg) + Having
- Ordering
- Also possible at the end: **LIMIT** *n* (only show first *n* results)

Basic Queries: SELECT

- **Distinct and All**
 - **Distinct** to get a set
 - Without **Distinct** you get a bag

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

Basic Queries: SELECT

- **Distinct and All**

- **Distinct** to get a set
 - Without **Distinct** you get a bag

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

- **Aliases**

- Give **table** or **column** another name
- Can then be used in the query
 - E.g., **S.Last**

```
SELECT DISTINCT Lname AS Last,  
Fname AS First  
FROM Students AS S;
```

- **Select *** → all columns

Basic Queries: FROM

- **Specifies the tables**

- Possible to combine rows from tables with related columns using **JOINS**
 - **Used for foreign key relationships**

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

Basic Queries: FROM

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

- **Specifies the tables**

- Possible to combine rows from tables with related columns using **JOINS**
 - **Used for foreign key relationships**

- **(Inner) Join**

- Get Customer and Product-Name of all orders

```
SELECT O.Customer, P.Name
FROM (Orders AS O
INNER JOIN Products AS P
ON O.PID = P.PID);
```

Orders

OID	Customer	Date	Quantity	PID
1	A	'2019-06-22'	3	2
2	B	'2019-06-22'	1	3
3	A	'2019-06-22'	1	4
4	C	'2019-06-23'	2	2
5	D	'2019-06-23'	1	4
6	C	'2019-06-23'	1	1

Products

PID	Name	Price
1	X	100
2	Y	15
4	Z	75
3	W	120

Basic Queries: FROM – INNER JOIN

- **Another example**

- Join conference participants with their location (LID, Location)

→ **NULL values are omitted with INNER JOIN**

```
SELECT *
      FROM (Participants AS P
INNER JOIN Location AS L
      ON P.LID = L.LID);
```

<u>PID</u>	Firstname	Lastname	Affiliation	LID
102	Anastasia	Ailamaki	EPFL	1
104	Peter	Bailis	Stanford	NULL
105	Magdalena	Balazinska	U Washington	3

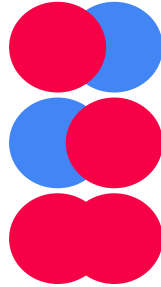
<u>LID</u>	City
1	Lausanne
3	Seattle

<u>PID</u>	Firstname	Lastname	Affiliation	LID	LID	City
102	Anastasia	Ailamaki	EPFL	1	1	Lausanne
105	Magdalena	Balazinska	UW	3	3	Seattle

Basic Queries: FROM – Outer JOINS

- **Outer Joins**

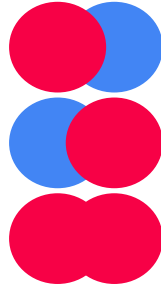
- LEFT
- RIGHT
- FULL OUTER Join



Basic Queries: FROM – Outer JOINS

- Outer Joins

- LEFT
- RIGHT
- FULL OUTER Join



```
SELECT *
  FROM (Participants AS P
LEFT OUTER JOIN Location AS L
ON P.LID = L.LID);
```

- Example: LEFT OUTER Join

- All non-matching fields (in LEFT) are filled with NULL (from RIGHT)

PID	Firstname	Lastname	Affiliation	LID	LID	City
102	Anastasia	Ailamaki	EPFL	1	1	Lausanne
104	Peter	Bailis	Stanford	NULL	NULL	NULL
105	Magdalena	Balazinska	UW	3	3	Seattle

Basic Queries: WHERE

- **To add conditions to the query**

- Combine conditions with **AND / OR**
- **WHERE NOT** to negate conditions
- Single quotes for text fields
- Wild cards for **LIKE** (% placeholder for an char)
- Full list of operators: https://www.w3schools.com/sql/sql_where.asp

```
SELECT DISTINCT Lname, Fname  
FROM Students  
WHERE SID IN (1,2,3) OR Lname LIKE 'S%';
```

Basic Queries: WHERE

- **To add conditions to the query**

- Combine conditions with **AND / OR**
- **WHERE NOT** to negate conditions
- Single quotes for text fields
- Wild cards for **LIKE** (% placeholder for an char)
- Full list of operators: https://www.w3schools.com/sql/sql_where.asp

```
SELECT DISTINCT Lname, Fname
FROM Students
WHERE SID IN (1,2,3) OR Lname LIKE 'S%';
```

- **(Inner) Join**

- Can also be done using WHERE
- **INNER JOIN** syntax is safe option

```
SELECT O.Customer, P.Name
FROM Orders AS O, Products AS P
WHERE O.PID = P.PID;
```

■ if you forget it → all combinations → (maybe) many (!) results

```
SELECT Customer, Name
FROM (Orders AS O
INNER JOIN Product AS P
ON O.PID = P.PID);
```

Grouping and Aggregation

- **Group By**
 - **Grouping:** determines the distinct groups (group rows with same values)
 - **Aggregation:** compute aggregate per group (COUNT, MAX, MIN, SUM, AVG)

Grouping and Aggregation

- **Group By**

- **Grouping:** determines the distinct groups (group rows with same values)
- **Aggregation:** compute aggregate per group (COUNT, MAX, MIN, SUM, AVG)

- **Example**

- **Compute number of sales** SumQ

and revenue per product SumQP

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

Product	SumQ	SumQP
A	3	30
B	4	80



Product	Quantity	Price
A	1	10
B	3	20
A	2	10
B	1	20

Grouping and Aggregation – Conditions with HAVING

- **Having**


- Like **WHERE** but used for **Group By** statements with aggregates
 - **WHERE** does not support aggregates!!!!

- **Example**


- **Compute number of sales SumQ and revenue per product SumQP for products with SumQ > 3**

```
SELECT Product,
       sum(Quantity) AS SumQ,
       sum(Quantity * Price) AS SumQP
FROM Sales
GROUP BY Product
HAVING SumQ > 3;
```

Product	SumQ	SumQP
B	4	80



Product	Quantity	Price
A	1	10
B	3	20
A	2	10
B	1	20



Sorting

- Order By
 - Convert output into a **sorted list** of tuples
 - **DESC** → from highest value to lowest
 - **ASC** → from lowest value to highest
 - Multiple fields possible
 - Order by first and in case of ties by the second, etc.
 - Evaluated last in a query

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

More Examples

- **Task: SQL queries for the following tasks:**

Orders

<u>OID</u>	Customer	Date	Quantity	PID
1	A	'2019-06-22'	3	2
2	B	'2019-06-22'	1	3
3	A	'2019-06-22'	1	4
4	C	'2019-06-23'	2	2
5	D	'2019-06-23'	1	4
6	C	'2019-06-23'	1	1

Products

<u>PID</u>	Name	Price
1	X	100
2	Y	15
4	Z	75
3	W	120

Unique Customer and Date for all orders of products Y or Z

Customer	Date
A	'2019-06-22'
C	'2019-06-23'
D	'2019-06-23'

```
SELECT DISTINCT Customer, Date
FROM (Orders AS O
INNER JOIN Products AS P
ON O.PID = P.PID)
WHERE Name IN('Y', 'Z');
```

More Examples

- Task: SQL queries for the following tasks:

Orders

<u>OID</u>	Customer	Date	Quantity	PID
1	A	'2019-06-22'	3	2
2	B	'2019-06-22'	1	3
3	A	'2019-06-22'	1	4
4	C	'2019-06-23'	2	2
5	D	'2019-06-23'	1	4
6	C	'2019-06-23'	1	1

Products

<u>PID</u>	Name	Price
1	X	100
2	Y	15
4	Z	75
3	W	120

Spent money (Sum) of all customers individually

Customer	Sum
A	120
B	120
C	130
D	75

```
SELECT Customer,
       sum(O.Quantity * P.Price) AS Sum
FROM (Orders AS O
      INNER JOIN Products AS P
      ON O.PID = P.PID)
GROUP BY O.Customer;
```

Subqueries

- **Subqueries in Table List**
 - Use a subquery result like a table

```
SELECT S.Fname, S.Lname, C.Name
FROM (Symposiums AS S
INNER JOIN
    (SELECT CID, Name FROM Country
    WHERE Name LIKE 'A%') AS C
ON S.CID=C.CID);
```

Subqueries

- **Subqueries in Table List**

- Use a subquery result like a table

```
SELECT S.Fname, S.Lname, C.Name
FROM (Students AS S
INNER JOIN
  (SELECT CID, Name FROM Country
  WHERE Name LIKE 'A%') AS C
ON S.CID=C.CID);
```

- **Subqueries with IN**

- Check containment of values in result set of sub query

```
SELECT Product, Quantity, Price
FROM Sales
WHERE Product NOT IN (
  SELECT Product FROM Sales
  GROUP BY Product
  HAVING sum(Quantity*Price) > 1000000);
```

Subqueries – WITH Statements

- **Give subquery a name using WITH**

- Enhances readability
- Enhances reusability

```
SELECT Product, Quantity, Price
FROM Sales
WHERE Product NOT IN (
    SELECT Product FROM Sales
    GROUP BY Product
    HAVING sum(Quantity*Price) > 1000000);
```



```
WITH ProductsRevenueMore1M AS (
    SELECT Product FROM Sales
    GROUP BY Product
    HAVING sum(Quantity*Price) > 1000000
)
SELECT Product, Quantity, Price
FROM Sales
WHERE Product NOT IN ProductsRevenueMore1M;
```

- **Can be used in WHERE or SELECT**

Stored Procedures

- **Overview Procedures**

- Stored programs, written in **PL/pgSQL** or other languages
- **Control flow (loops, ifs) and parameters are possible**

- **Stored Procedures**

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

```
CREATE PROCEDURE prepStud
LANGUAGE PLPGSQL AS $$
BEGIN
    DELETE FROM Students;
    INSERT INTO Students
        SELECT * FROM NewStudents;
END; $$;
```

Data Control Language

- **Set access permissions on tables**
 - **Grant** query/modification rights on database objects for specific users, roles
 - **Revoke** access rights from users, roles
 - Same syntax for **UPDATE**, **INSERT**, etc

```
GRANT SELECT  
ON TABLE TeamDM  
TO liacono;
```

```
REVOKE SELECT  
ON TABLE TeamDM  
FROM liacono;
```

Exercise 2

Exercise 2 – Part 1

- **Database Creation and Data Ingestion (9 / 25 points)**

- Install PostgreSQL
- Create new database
- Download our SQL schema
- Set up your database using this schema
- Download our processed dataset files
- Write SQL COPY statements
- Ingest the data using COPY statements (e.g. `COPY Teams FROM 'C:\Users\user\Folder\teams.csv'`
`DELIMITER ';' CSV HEADER;`)
-

- **Deliverable** → IngestData.sql file

Exercise 2 – Part 2

- **SQL Query Processing (16 / 25 points)**
 - 6 SQL SELECT queries to write
 - Queries 1 – 3 2 points each
 - Queries 4 + 5 → 3 points each
 - Query 6 → 4 points
 - You can check the output of your queries using our results files
 - Check assignment sheet for all the details!
- **Deliverables** → Q01.sql ... Q06.sql files
- **Please document** your scripts using `/* ... */` in case things need extra documentation

Exercise 2 – Part 2

- Q01
 - Show all male users who are younger than 30. (Return: userID)
- Q02
 - Find the top 10 lowest-rated gift cards (by average rating attribute). (Return: title, average rating)
- Q03
 - Calculate how many users have never written a review in their time on the platform (Return: number of such users)

→ 2 points each

Exercise 2 – Part 2

- Q04
 - Find users who wrote more than 10 reviews and only rated gift cards in them with 5 stars. (Return: userID)
 - Q05
 - List all users who only reviewed gift cards from one single category. (Return: userID)
- 3 points each

Exercise 2 – Part 2

- Q06
 - Get gift cards more expensive than the average price of their category. (Return: title, price, category name)
- 4 points

Conclusions

- **Summary**
 - Query languages
 - SQL (DML: Insert, Update, Delete, Select)
 - Exercise 2

Conclusions

- **Summary**

- Query languages
- SQL (DML: Insert, Update, Delete, Select)
- Exercise 2

- **Next week**

- No lecture!
- **Submission group project concept: Tue. 25.11.2025 (groups of 5!)**