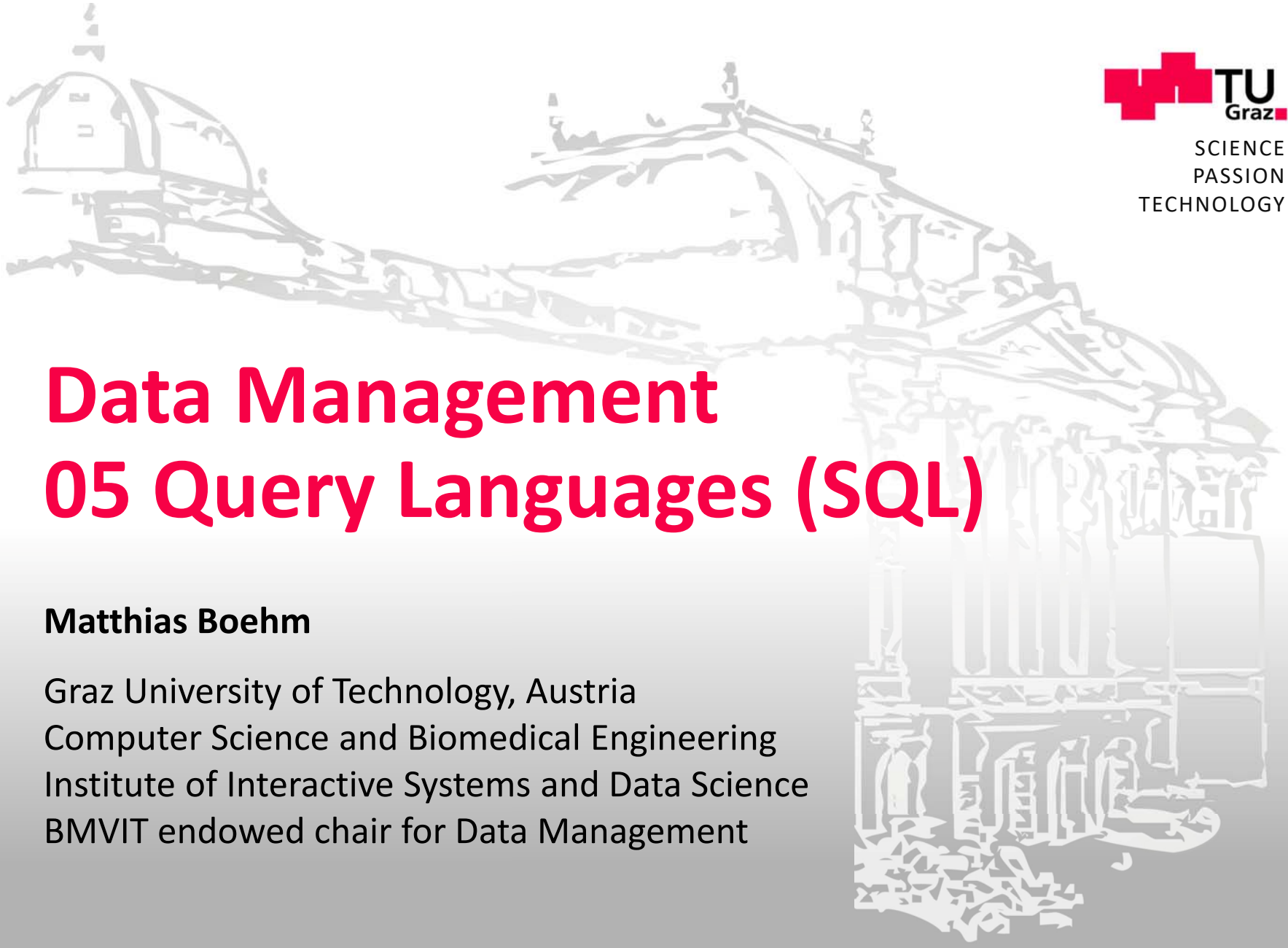


SCIENCE
PASSION
TECHNOLOGY

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
BMVIT endowed chair for Data Management

Last update: Mar 28, 2020

ISDS

Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
  - **Live Streaming** Mo 4.10pm until end of lockdown (end of semester?)

- **#2 Reminder Communication**
  - **Newsgroup:** news://news.tugraz.at/tu-graz.lv.dbase; **no TeachCenter forum!** (https://news.tugraz.at/cgi-bin/usenet/nntp.csh?tu-graz.lv.dbase)
  - **Office hours:** Mo 1pm-2pm (https://tugraz.webex.com/meet/m.boehm)

- **#3 Reminder Exercise 1**
  - Submission through **TeachCenter** (max 5MB, **draft possible**)
  - **Submission open** (deadline **Mar 31, 11.59pm**) + (7+3) late days

75/601

- **#4 Preview Exercise 2**
  - Will be published **Apr 7**, Py/Java examples (schema **Apr 10**, deadline **Apr 28**)
  - **Preview today**, fully introduced **next lecture**

Agenda

- **Structured Query Language (SQL)**
- **Other Query Languages (XML, JSON)**
- **Preview Exercise 2**

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

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

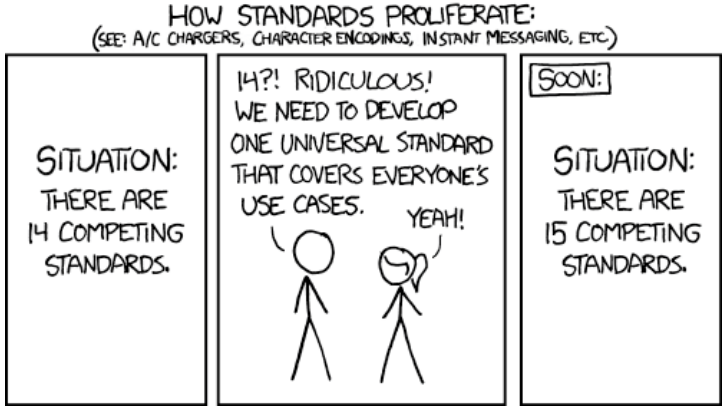
#2 Flexibility: closed → composability

#3 Automatic Optimization

#4 Physical Data Independence

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)



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

- **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)**
  - Current Standard: ISO/IEC 9075:2016 (SQL:2016)
  - **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

Name	Examples
T-SQL	Microsoft, Sybase
PL/SQL	Oracle, (IBM)
PL/pgSQL	PostgreSQL, derived
Unnamed	**Most systems**

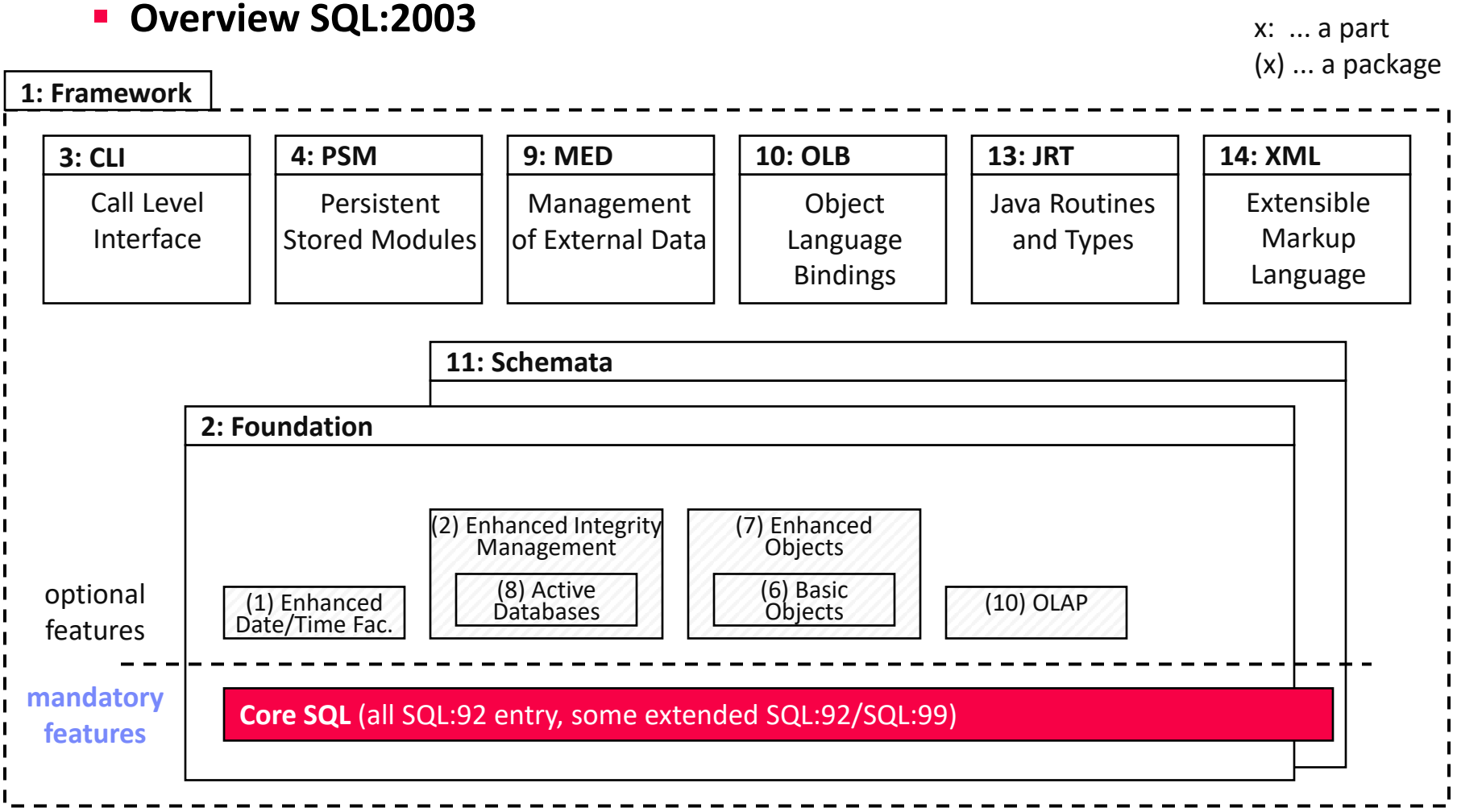
The History of the SQL Standard

[C. J. Date: A Critique of the SQL Database Language. **SIGMOD Record 1984**]

- **SQL:1986**
  - Database Language SQL, ANSI X3.135-1986, ISO-9075-1987(E)
  - '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)
  - Database Language SQL, ANSI X3-1992, ISO/IEC-9075 1992, DIN 66315
  - '95 SQL/CLI (part 3), '96 SQL/PSM (part 4)
- **SQL:1999** (2000 pages)
  - Information Technology – Database Language – SQL, ANSI/ISO/IEC-9075 1999
  - Complete reorg, '00 OLAP, '01 SQL/MED, '01 SQL/OLB, '02 SQL/JRT
- **SQL:2003** (3764 pages)
  - Information Technology – Database Language – SQL, ANSI/ISO/IEC-9075 2003

The History of the SQL Standard, cont.

- **Overview SQL:2003**



The History of the SQL Standard, cont.

Since SQL:2003 overall structure remained unchanged ...

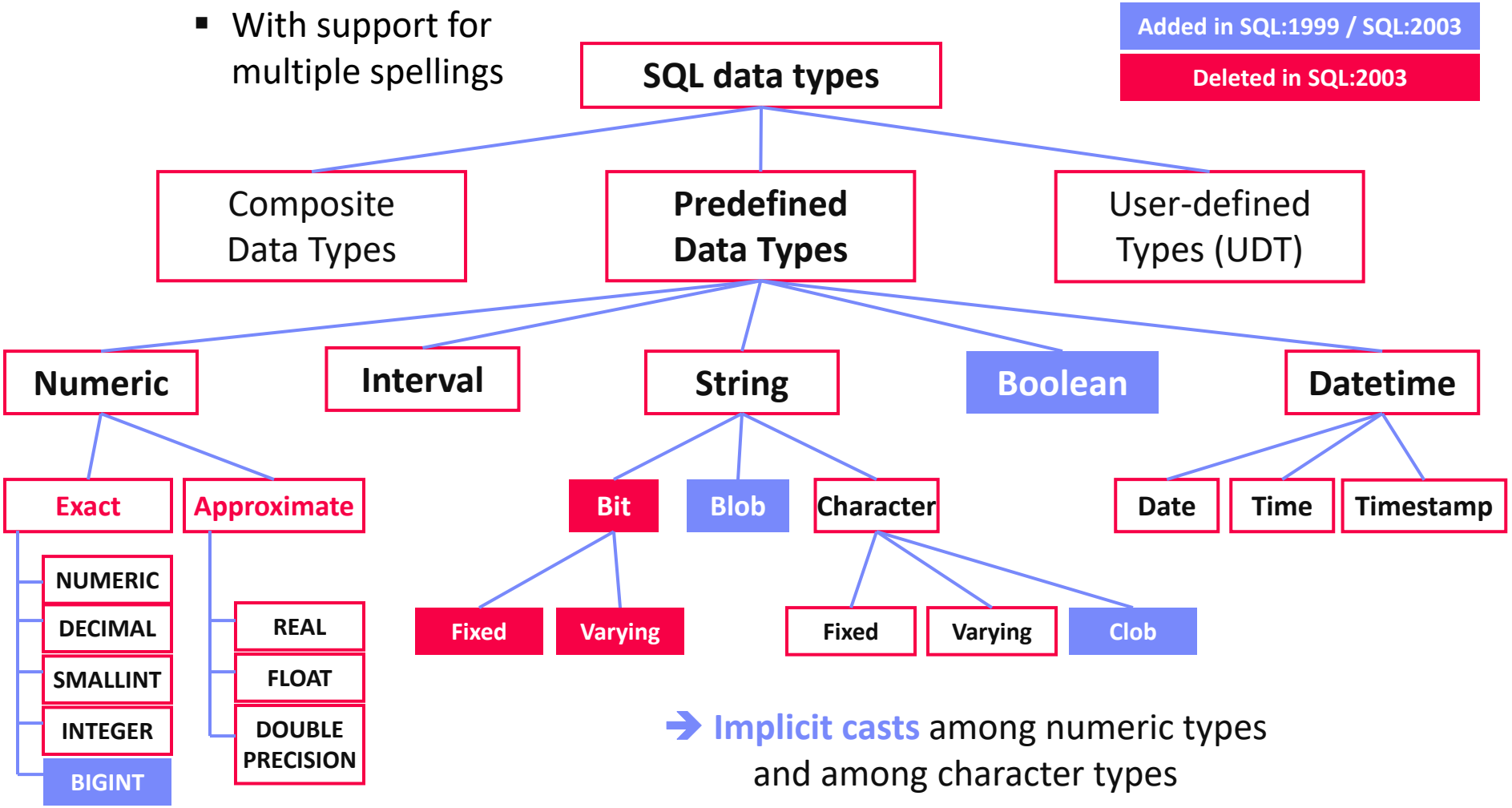
- **SQL:2008** (???? pages)
  - **Information Technology – Database Language – SQL**, ANSI/ISO/IEC-9075 2003
  - E.g., **XML** XQuery extensions, case/trigger extension
- **SQL:2011** (4079 pages)
  - **Information Technology – Database Language – SQL**, ANSI/ISO/IEC-9075 2011
  - E.g., time periods, temporal constraints, time travel queries
- **SQL:2016** (???? pages)
  - **Information Technology – Database Language – SQL**, ANSI/ISO/IEC-9075 2016
  - E.g., **JSON** documents and functions (optional)

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

[Working Draft SQL:2011: https://www.wiscorp.com/SQLStandards.html]

Data Types in SQL:2003

- **Large Variety of Types**
  - With support for multiple spellings



➔ **Implicit casts** among numeric types and among character types

Data Types in PostgreSQL

Appropriate, Brief, Complete

- **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

Templates in SQL
Examples in PostgreSQL

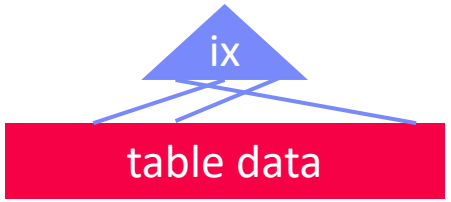
- **Create Table**
  - Typed attributes
  - Primary and foreign keys
  - NOT NULL, UNIQUE constraints
  - DEFAULT values
  - CHECK constraints
- **Alter Table**
  - ADD/DROP columns
  - ALTER data type, defaults, constraints, etc
- **Delete Table**
  - Delete table
  - **Note:** order of tables matters due to referential integrity

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

ALTER TABLE Students ADD DoB DATE;

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

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**

```
CREATE INDEX ixStudLname
   ON Students USING btree
   (Lname ASC NULLS FIRST);
```



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

```
DROP INDEX ixStudLname;
```

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

Database Catalog

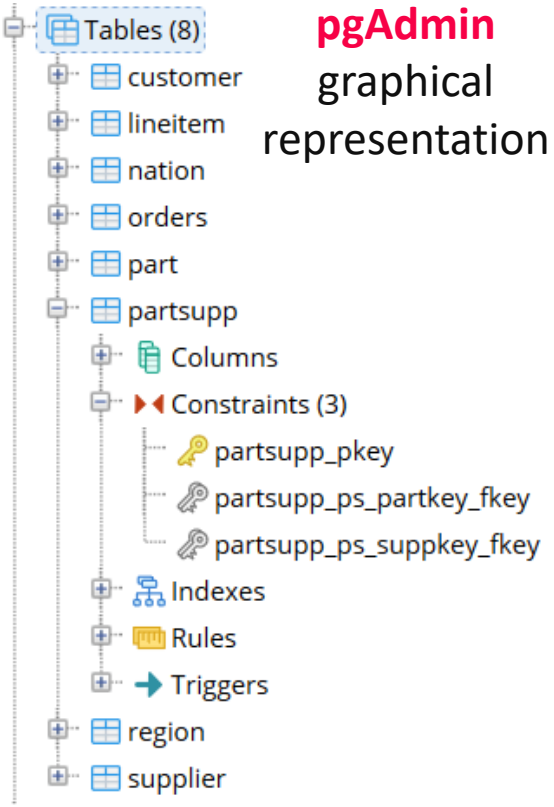
[Meikel Poess: **TPC-H**. **Encyclopedia of Big Data Technologies 2019**]

- **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

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

(defined as views over PostgreSQL catalog tables)



pgAdmin graphical representation

Insert

- **Insert Tuple**
  - Insert a single tuple with implicit or explicit attribute assignment

```
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

```
INSERT INTO Students (Lname, Fname, DoB)
   VALUES ('Boehm','Matthias','1982-06-25'),
          (...), (...);
```

SERIAL SID,
DEFAULT MTime

- **Insert Table**
  - Redirect query result into INSERT (append semantics)

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

Analogy Linux redirect (append):

```
cat NewStudents.txt >> Students.txt
```

Update and Delete

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

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

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

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

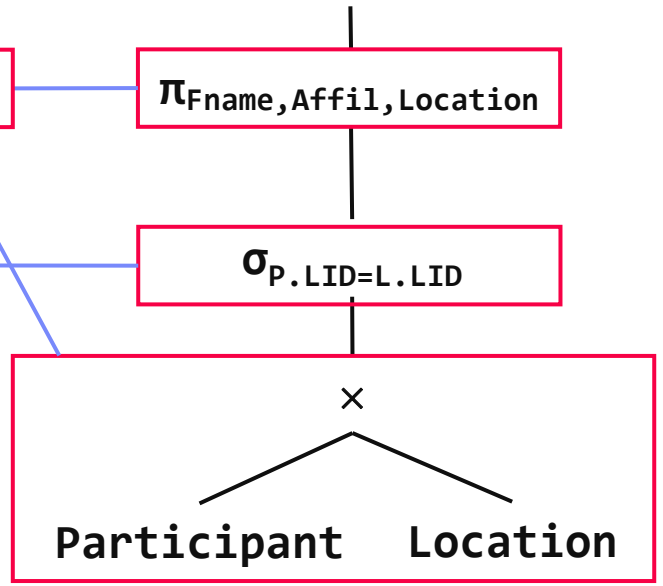
- **Note: Time travel and multi-version concurrency control**
  - Deleted tuples might be just marked as inactive
  - See lecture 09 Transaction Processing and Concurrency

Basic Queries

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

```
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]]
```

- **Example**
  - **SELECT** Fname, Affil, Location
    FROM Participant AS R,
    Locale AS S
    WHERE R.LID=S.LID;



Basic Queries, cont.

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

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

- **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

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

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

```
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 and revenue per product**

```
SELECT Product, sum(Quantity), sum(Quantity*Price)
   FROM Sales
   GROUP BY Product
```

BREAK (and Test Yourself)

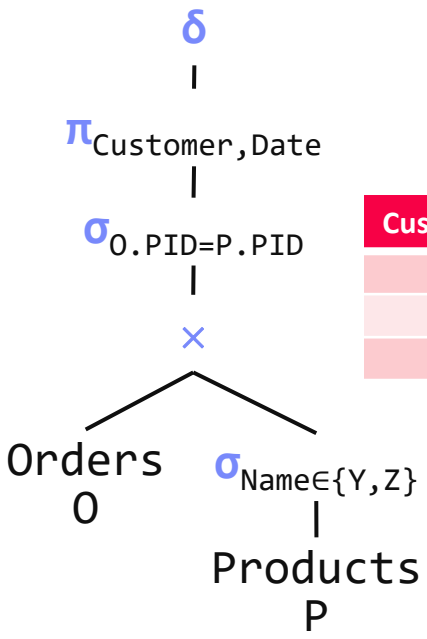
- **Task: SQL queries for the following query trees.**

Orders

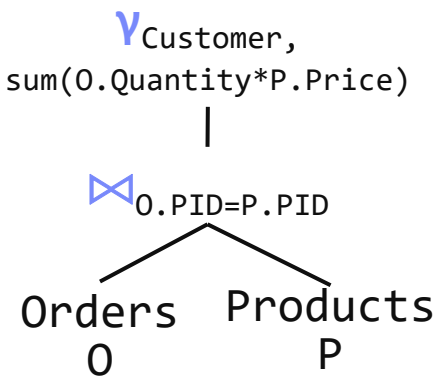
OID	Customer	Date	Quantity	PID
1	A	'2019-06-22'	3	2
2	B	'2019-06-22'	1	3
3	A	'2019-06-22'	1	4
4	C	'2019-06-23'	2	2
5	D	'2019-06-23'	1	4
6	C	'2019-06-23'	1	1

Products

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



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



Customer	Sum
A	120
B	120
C	130
D	75

```
SELECT DISTINCT Customer, Date
  FROM Orders O, Products P
  WHERE O.PID = P.PID
    AND Name IN('Y','Z')
```

```
SELECT Customer,
    sum(O.Quantity * P.Price)
  FROM Orders O, Products P
  WHERE O.PID = P.PID
  GROUP BY Customer
```

Subqueries

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

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

- **Subqueries w/ 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)>1e6)
```

- **Other subqueries**
  - **EXISTS:** existential quantifier **∃x** for correlated subqueries
  - **ALL:** comparison (w/ universal quantifier **∀x**)
  - **SOME/ANY:** comparison (w/ existential quantifier **∃x**)

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

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

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

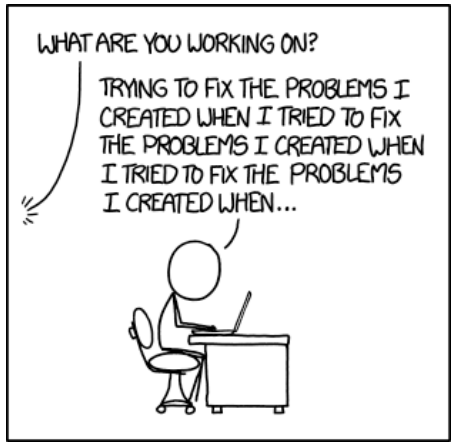
```
SELECT P.Fname, P.Lname
FROM Professors P,
WHERE P.PID NOT IN(
   SELECT PID FROM Courses);
```

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

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

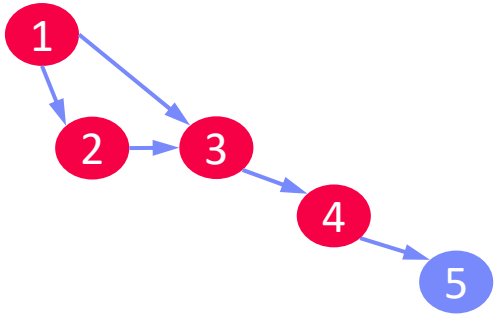
Recursive Queries

- **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



[https://xkcd.com/1739/]

- **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
```

4
3
1
2

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

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

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

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**

```
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
```

Event

Condition

Action

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

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

- **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**)

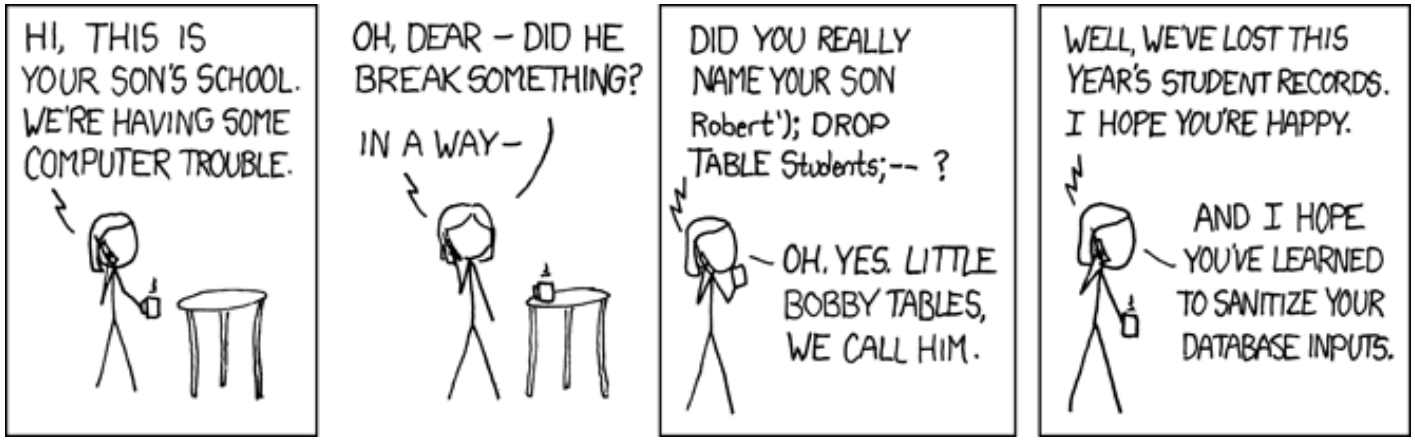
```
GRANT SELECT
    ON TABLE TeamDM
    TO mboehm;

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**



[https://xkcd.com/327/]

```
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

```
<?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>
```

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

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

↓

"DM"
"AMLS"

- **XSLT** (XML Stylesheet Language Transformations)
  - Schema mapping (transformation) language for XML documents
- **XQuery**
  - Query language to extract, transform, and analyze XML documents

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 **builtin functions** to parse documents, and create elements/attributes
  - XMLPARSE(<xml_document>) → **XML type**
  - XMLELEMENT / XMLATTRIBUTES

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

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

```
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)

```
{“students:”[
  {“id”: 1, “courses”:[
    {“id“:“INF.01017UF”, “name“:“DM”},
    {“id“:“706.550”, “name“:“AMLS”}]},
 {“id”: 5, “courses”:[
    {“id“:“706.520”, “name“:“DIA”}]},
]}
```

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

JSONiq Example:

```
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}
```

[http://www.jsoniq.org/docs/JSONiq/html-single/index.html]

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

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

- **Processing JSON**
  - Specialized operators for **tree traversal and data extraction**

```
SELECT Fname, Lname,
   Doc->students->>id
   FROM Students
```

  - **-> operator:** get JSON array element/object
  - **->> operator:** get JSON array element/object as text
  - Built-in functions for extracting json (e.g., json_each)

Preview Exercise 2: Query Languages and APIs

Exercises: DBLP Publications

- **Dataset**
  - CC0-licensed, derived (extracted, cleaned) from **DBLP** (https://dblp.org Feb 1, 2020) for publication **year ≥ 2011 + DM venues**
  - Clone or download your copy from https://github.com/tugraz-isds/datasets.git
- **Exercises**
  - **01** Data modeling (relational schema)
  - **02** **Data ingestion and SQL query processing**
    - **Relational schema** + **ingestion**
    - SQL query processing + extra credit
  - **03** Physical design tuning, query processing, and transaction processing
  - **04** Large-scale data analysis (distributed data ingestions and query processing)

persons.csv: The persons file contains author information inclu websites. It's detailed structure and examples look as follows.

```
#PID | name | aliases | affiliation | url
A261789|Matthias Boehm 0001|Matthias Böhm 0001|Graz Unive
A1537639|Stefanie N. Lindstaedt|Stefanie N. Lindstädt||ht
A977823|Denis Helic||Graz University of Technology, Austr
```

theses.csv: The theses file contains the information of public Ph look as follows.

```
#TKey | author | title | year | type | school | pages | i
T25621|A261789|Cost-based optimization of integration flo
T30052|A1399369|An Architecture for Fast and General Data
```

pubs.csv: The pubs file (or better, its individual parts) contains detailed structure and examples look as follows.

```
#PKey | authors| title | year | type | journal | volumne
P519327|A382693:A261789:A261428:A2051042:A69590|MNC: Stru
P1640801|A261789:A2051042:A2047447:A472485:A261428:A38856
P12485|A1399369:A1703306:A1416241:A557115:A650354:A863102
```

confs.csv: The confs file contains the information on conferenc likely be further improved soon.

```
#CKey | title | editors | year | isbn
C8036|Proceedings of the 2019 International Conference on
C76|Proceedings of the 9th USENIX Symposium on Networked
```

Task 2.1: Schema Creation via SQL

- **Schema creation via SQL**
  - Relies on lectures **04 Relational Algebra** and **05 Query Languages (SQL)**
  - Setup DBMS PostgreSQL
  - Create database `db<studentID>` and **setup relational schema**
    - **Ignore** (1) person aliases, and (2) conference editors
    - Primary keys, foreign keys, `NOT NULL`, `UNIQUE`
    - `CreateSchema.sql`

- **Recommended Schema**
  - **TBA (after Apr 10, 11.59pm)**

Task 2.2 Data Ingestion via CLI

- **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/dblp_publications
  - Invoke your ingestion program as follows → script to compile and run

```
IngestData ./confs.csv ./journals.csv \
  ./persons.csv ./pubs.csv ./theses.csv \
  <host> <port> <database> <user> <password>
```

Getting Started w/ Task 2.2

- **Exercise 2: Would you know how to start?**
  If not, please ask.

"I feel a bit lost on how to start this task and also a bit overwhelmed by the amount of lists and the two phases of loading the data." (Apr 26, 2019)

- **Some Pointers**
  - Download and install PostgreSQL, use pgAdmin for schema creation / querying
  - Download and install an IDE (e.g., PyCharm, Eclipse/IntelliJ, VS Code)
  - Run `CreateSchema.sql` through query tool (pgAdmin or psql terminal)
  - #1 Setup the database connection
  - #2 Read the csv files into lists of string arrays (create a function for that)
  - #3 For all target tables (in order of reference, create a function for each)
    - Extract necessary data from respective lists (incl. deduplication)
    - Insert data via CLI into table using PK lookup tables
    - Create tailor-made lookup tables if necessary

Conclusions and Q&A

- **Summary**
  - History and fundamentals of the **Structured Query Language (SQL)**
  - Awareness of **XML and JSON** (data model and querying)
- **Exercise 1 Reminder**
  - Submission deadline: **Mar 31, 11.59pm**
  - **Late day policy:** 7+3 days; **replacements** possible via email
- **Exercise 2**
  - To be published **Apr 07** on website and TeachCenter
  - Submission deadline **Apr 28, 11.59pm**
- **Next Lectures**
  - **06 APIs (ODBC, JDBC, OR frameworks)** [Apr 20], **incl. Exercise 2**