Database Systems
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: Apr 01, 2019
Announcements/Org

- **#1 Video Recording**
  - Since lecture 03, video/audio recording (setting)
  - Link in TeachCenter & TUbe (but not public yet)

- **#2 Exercise 1**
  - Submission through TeachCenter (max 5MB, draft possible)
  - Submission open (deadline Apr 02, 12.59pm)

- **#3 Exercise 2**
  - Modified published date Apr 3 → Apr 08
  - Introduced during next lecture

- **#4 Participation**
  - Awesome to see active involvement via newsgroup, PRs, etc
  - Reminder: office hours Monday 1pm-2pm
Agenda

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

[https://xkcd.com/927/]
Structured Query Language (SQL)
Overview SQL

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

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

- **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**
  - 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)
  - ‘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]
### Overview SQL:2003

<table>
<thead>
<tr>
<th>1: Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3: CLI</strong> Call Level Interface</td>
</tr>
<tr>
<td><strong>4: PSM</strong> Persistent Stored Modules</td>
</tr>
<tr>
<td><strong>9: MED</strong> Management of External Data</td>
</tr>
<tr>
<td><strong>10: OLB</strong> Object Language Bindings</td>
</tr>
<tr>
<td><strong>13: JRT</strong> Java Routines and Types</td>
</tr>
<tr>
<td><strong>14: XML</strong> Extensible Markup Language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2: Foundation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Enhanced Date/Time Fac.</td>
</tr>
<tr>
<td>(2) Enhanced Integrity Management</td>
</tr>
<tr>
<td>(8) Active Databases</td>
</tr>
<tr>
<td>(3) Enhanced Objects</td>
</tr>
<tr>
<td>(6) Basic Objects</td>
</tr>
<tr>
<td>(10) OLAP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11: Schemata</th>
</tr>
</thead>
</table>

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

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

- Numeric
  - Exact
    - NUMERIC
    - DECIMAL
    - SMALLINT
    - INTEGER
  - Approximate
    - FLOAT
    - REAL
    - 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/TIMESTAMPZ → 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

---

**Templates in SQL**

**Examples in PostgreSQL**

```sql
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 ...;
```

```sql
ALTER TABLE Students ADD DoB DATE;

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

```sql
DROP TABLE Students;  -- sorry
DROP TABLE Students CASCADE;
```
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
    
    \[
    \text{INSERT INTO Students (SID, Lname, Fname, MTime, DoB)} \\
    \text{VALUES (7,'Boehm','Matthias','2002-10-01','1982-06-25')};
    \]
  - Insert attribute key-value pairs to use auto increment, defaults, NULLs, etc
    
    \[
    \text{INSERT INTO Students (Lname, Fname, DoB)} \quad \text{SERIAL SID,} \\
    \text{VALUES ('Boehm','Matthias','1982-06-25'), \text{DEFAULT MTime} (\ldots), (\ldots));
    \]

- Insert Table
  - *Redirect query result into* INSERT (append semantics)
    
    \[
    \text{INSERT INTO Students} \\
    \text{SELECT * FROM NewStudents;}
    \]
  - *Analogy Linux redirect (append):*
    
    \[
    \text{cat NewStudents.txt >> Students.txt}
    \]
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

```
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 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

- **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 lecture **04 Relational Algebra and Tuple Calculus**
  - Distinct (set) and all (bag) alternatives
  - Set operations w/ **set semantics** by default

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

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

SELECT Firstname, Lastname
FROM Participant2018
UNION
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

```sql
SELECT Product, sum(Quantity), sum(Quantity*Price)
FROM Sales
GROUP BY Product
```
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 $\exists x$)

```sql
SELECT S.Fname, S.Lname, C.Name
FROM Students AS S,
     (SELECT CID, Name FROM County
      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

```
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

- 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

  ![XKCD comic](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

- **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 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**

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

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

```sql
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

```json
{
  "students": [
    {
      "id": 1,
      "courses": [
        {
          "id": "INF.01014UF",
          "name": "Databases"
        },
        {
          "id": "706.550",
          "name": "AMLS"
        }
      ]
    },
    {
      "id": 5,
      "courses": [
        {
          "id": "706.004",
          "name": "Databases 1"
        }
      ]
    }
  ]
}
```

**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}
```

Other Query Languages (XML, JSON)

JSON in PostgreSQL, cont.

- **Overview JSON in PostgreSQL**
  - Alternative data types: **JSON** (text), **JSONB** (binary, with restrictions)
  - Implements RFC 7159, builtins 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
```
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**
  - All background to solve tasks 1.1-1.3 since last lecture
  - Submission: since Mar 25, Deadline: **Apr 02 11.59pm**

- **Next Lectures**
  - Apr 08: **06 APIs (ODBC, JDBC, OR frameworks)**, incl. **Exercise 2**