Database Systems

Structured Query Language

Prof. Dr. Agnès Voisard Muhammed-Ugur Karagülle

Institute of Computer Science, Databases and Information Systems Group

Fraunhofer FOKUS

2025

Outline

Notes

Notes

Introduction

- SQL: “Structured Query Language”
- Originally, SEQUEL (Structured English Query Language), SYSTEM R, IBM Research.
- ANSI (American National Standard Institute)
  + ISO (International Standard Organization)
  => SQL1, standard version ANSI 1986.
- 1992: SQL2, 1994: SQL3, SQL/MM: SQL Multimedia and Application Packages
- Now query language of most DBMS.

Statements for data definition, query and update:
=> DDL & DML.

Table ≡ relation, row ≡ tuple, column ≡ attribute.

Schema:

- Group together tables that belong to the same DB application.
- SQL schema: schema name
  + user (or account) who owns it (authorization identifier)
  + descriptors for tables, views, etc.
- **CREATE SCHEMA** statement:

Example:

CREATE SCHEMA Company **AUTHORIZATION** Jackson

Catalog:

- Named collection of schemas in an SQL environment.

CREATE TABLE Company.EMPLOYEE

Data types for attributes:
numeric, character-string, bit-string + date and time (SQL2).

Example:

Date: YYYY-MM-DD.

Table creation (cont'd)


```
CREATE TABLE EMPLOYEE(
        FName           VARCHAR(15) NOT NULL,
        LName           VARCHAR(15) NOT NULL,
        SSN             CHAR(10) NOT NULL,
        BDate           DATE,
        Address         VARCHAR(30),
        Salary          DECIMAL,
        BossSSN         CHAR(10),
        NumDept         INT NOT NULL,
PRIMARY KEY (SSN),
FOREIGN KEY (NumDept) REFERENCES DEPARTMENT(DNumber),
FOREIGN KEY (BossSSN) REFERENCES EMPLOYEE(SSN),
);
```

Table creation (cont'd)


```
CREATE TABLE DEPARTMENT(
        DName           VARCHAR(15) NOT NULL,
        DNumber         INT NOT NULL,
        MGRSSN          CHAR(10) NOT NULL,
        MGRStartDate    DATE,
PRIMARY KEY (DNumber),
UNIQUE (DName),
FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN),
...
);
```

UNIQUE: alternate key.

Domain creation

Easier to change a data type in a domain (change only at one place).

Improves the readability:

CREATE DOMAIN SSN-TYPE **AS** CHAR(10);

One statement:

SELECT < attribute list >
FROM < table list >
WHERE < condition >

where:

- < attribute list >: list of attribute names whose values are to be retrieved
- < table list >: list of relation names necessary to process the query
- < condition >: Boolean expression.

Example queries - schema used

Example:

EMPLOYEE(FName, LName, SSN, BDate, Address, Salary, BossSSN, NumDept)
DEPENDENT(Name, Gender, BDate, Relation, ESSN)
DEPARTMENT(DName, DNumber, MGRSSN, MGRStartDate)
DEPTLOCATION(DNumber, Location)
PROJECT(PName, PNumber, Location, DNumber)
WORKSON(ESSN, PNO, Hours)

Example queries (cont'd)

Example:

Query 0: *Birthdate and address of employee John Doe.*

```
SELECT BDate, Address
  FROM EMPLOYEE
  WHERE FName = "John"
  AND LName = "Doe"
```

$$\pi_{BDate,Address}(\sigma_{FName=John \wedge LName=Doe}(EMPLOYEE))$$

Example queries (cont'd)

Example:

Query 1: *Name and address of all employees in the Research department*.

```
SELECT LName, Address
  FROM EMPLOYEE, DEPARTMENT
  WHERE NumDept = DNumber
  AND DName = "Research"
```

$$R = EMPLOYEE \underset{NumDept=DNumber}{\bowtie} DEPARTMENT$$

$$\pi_{LName,Address}(\sigma_{DName=Research}(R))$$

Attributes ambiguities and aliases

Problem of same attribute names allowed in different relations.
Identification: prefix the relation name to the attribute name.

Example:

Query 2: *Where is the "Research" department located?*

```
SELECT Location
  FROM DEPARTMENT, DEPTLOCATION
  WHERE DEPARTMENT.DNumber = DEPTLOCATION.DNumber
  AND DName = "Research"
```

Aliases

Example:

Query 3: *Give the last name of each employee together with the one of his/her boss*.

```
SELECT A.LName, B.LName
  FROM EMPLOYEE A, EMPLOYEE B
  WHERE A.BossSSN = B.SSN
```

Also in any query:

Example:

Query 1: *Name and address of all employees in the Research department*.

```
SELECT E.LName, E.Address
  FROM EMPLOYEE E, DEPARTMENT D
  WHERE E.NumDept = D.DNumber
  AND D.DName = "Research"
```

Example query, aliases

Example:

Query 4: *For every project located in Berkeley, give its number, its department and the department manager's last name and address.*

```
SELECT PNumber, DNumber, LName, Address
  FROM PROJECT, EMPLOYEE, DEPARTMENT
  WHERE PROJECT.DNumber = DEPARTMENT.DNumber
  AND DEPARTMENT.MGRSSN = EMPLOYEE.SSN
  AND PROJECT.Location = "Berkeley"
```

Missing WHERE clause: no condition. All tuples qualify.

Example:

Query 5: *Select all combinations of SSN and department in the database*.

```
SELECT SSN, DName
  FROM EMPLOYEE, DEPARTMENT
```

To retrieve all attribute values of selected tuples: **SELECT ***

Example:

Query 6: *Retrieve all attributes values of* Employee *tuples who work in department number 5*.

```
SELECT *
  FROM EMPLOYEE
  WHERE NumDept = 5
```

Handling duplicates

Careful: tables can have identical members: multisets (bags) of tuples.
SQL does not eliminate duplicates in query results:

- Duplicate elimination is expensive.
  (Example: sort + eliminate).
- Sometimes the user wants to see duplicates

To ask for elimination: SELECT **DISTINCT**

Example:

Query 7: *Select all possible salaries in the company*

```
SELECT DISTINCT Salary
  FROM EMPLOYEE
```

Set operations

Union, difference intersection.

Duplicate tuples are eliminated from the result unless the operation is followed by “ALL”.

Careful: relations need to be compatible. I.e., for the attributes: same order and same domain.

Example:

Query 8: *Project numbers for which a “Müller” is working, either as a regular employee or as a manager of the department that controls the project*.

Example:

```
SELECT PNO
  FROM EMPLOYEE E, WORKSON W
  WHERE W.ESSN = E.SSN
  AND E.LName = "Müller"
UNION
SELECT PNumber
  FROM PROJECT, EMPLOYEE, DEPARTMENT
  WHERE PROJECT.DNumber = DEPARTMENT.DNumber
  AND MGRSSN = SSN
  AND LName = "Müller"
```

Complete queries within the WHERE clause of another query called the **outer query**.

IN compares a value with a (multi)set of values.

SELECT …FROM …WHERE …
IN
SELECT …FROM …WHERE …

Example:

Query 8: *Project numbers for which a "Müller" is working, either as a regular employee or as a manager of the department that controls the project*.

Nested queries - Example 1 (2 nested queries)

Example:

```
SELECT DISTINCT PNumber
FROM PROJECT WHERE
  PNumber IN (SELECT W.PNO
              FROM EMPLOYEE E, WORKSON W
              WHERE W.ESSN = E.SSN
              AND E.LName = "Müller") OR
  PNumber IN (SELECT P.PNumber
              FROM PROJECT P, EMPLOYEE E,
                   DEPARTMENT D
              WHERE P.DNumber = D.DNumber
              AND D.MGRSSN = E.SSN
              AND E.LName = "Müller")
```

Example:

Query 9: *Select the SSN of people who work the same amont of time as John Doe does on a (any) project*.

```
SELECT DISTINCT ESSN
    FROM WORKSON
    WHERE (PNO, Hours) IN (SELECT PNO, Hours
                           FROM EMPLOYEE,
                                WORKSON
                           WHERE SSN = ESSN
                           AND FName = “John”
                           AND LName = “Doe”)
```

Nested queries (cont'd)

Attributes with the same name: always take the innermost nested query.

Queries **correlated**:
When a condition in the WHERE of a nested query references attributes declared in the outer query.
Evaluated once for each tuple in the outer query.

Example:

Query 10: *Name of employees who have a dependent with the same last name*.

```
SELECT E.FName, E.LName
    FROM EMPLOYEE E
    WHERE E.SSN IN (SELECT ESSN
                    FROM DEPENDENT D
                    WHERE D.ESSN = E.SSN
                    AND E.LName = D.Name)
```

Nested queries (cont'd)

For each Employee tuple t the nested query is evaluated:

Retrieve the ESSN values for all the dependent tuples which have the same SSN and name as t.

If the SSN value is in the result then keep it.

Query written with nested SELECT...FROM...WHERE and “=”or IN can always be expressed in a single block query.

Example:

Query 10: *Name of employees who has a dependent with the same last name*, Version 2:

```
SELECT E.FName, E.LName
    FROM EMPLOYEE E, DEPENDENT D
    WHERE D.ESSN = E.SSN
    and E.LName = D.Name
```

Other operators

- IN, $=, <, \leq, >, \geq, <>$.
- ANY (or SOME): $\exists$.
  True if the value to be compared is true for some value.
  A θ ANY S $\equiv$ ($\exists$ X) (X is in S $\wedge$ A θ X)
- ALL: $\forall$
  A θ ALL S $\equiv$ ($\forall$ X)(if X is in S then A θ X)

ALL operator example

Example:

Query 11: *Names of employees whose salary is greater than the salary of all employees in Department 5*.

```
SELECT LName
    FROM EMPLOYEE
    WHERE Salary > ALL (SELECT Salary
                        FROM EMPLOYEE
                        WHERE NumDept = 5)
```

Explicit set enclosed in parenthesis

Example:

Query 12: *ESSN of employees who work on project 1, 2 or 3*

```
SELECT DISTINCT ESSN
    FROM WORKSON
    WHERE PNO IN (1,2,3)
```

NULL values: missing, undefined, or not applicable.

Use of **IS NULL** or **IS NOT NULL**.

Example:

Query 13: *Name of employees who do not have a boss*.

```
SELECT LName
    FROM EMPLOYEE
    WHERE BossSSN IS NULL
```

Mathematical **aggregate functions** on collections of values cannot be expressed in the relational algebra.

Built-in SQL functions.

- SUM (**SUM**)
- AVERAGE (**AVG**),
- MAXIMUM (**MAX**),
- MINIMUM (**MIN**)

applied to (multi) sets of numeric values.

Aggregation - Example

Example:

SELECT **SUM**(Salary), **MAX**(Salary), **MIN**(Salary), **AVG**(Salary)
FROM *EMPLOYEE*, *DEPARTMENT*
WHERE NumDept=DNumber
AND DName="Research"

- **COUNT** returns the number of tuples specified in a query.

Example:

Query 14: *Number of employees in the Research department*.

```
SELECT COUNT (*)
    FROM EMPLOYEE, DEPARTMENT
    WHERE NumDept=DNumber
    AND DName="Research"
```

Query 15: *Number of distinct salary values in the database*.

```
SELECT COUNT (DISTINCT Salary)
    FROM EMPLOYEE
```

Grouping attributes

When applying aggregate functions to subgroups of tuples in a relation.
Example: for each project, find the number of employees.

Tuples that have the same value for **grouping attributes** are grouped.

GROUP BY specifies the grouping attributes.

Grouping attributes should appear in the SELECT clause.

Example:

Query 16: *For each department, retrieve its number, its number of employees and their average salary*.

SELECT NumDept, **COUNT**(*), **AVG**(Salary)
FROM *EMPLOYEE*
GROUP BY NumDept

Tuples divided into groups in which COUNT and AVG are computed.

Restricting groups: **HAVING** condition

Example:

Query 17: *For each project on which more than 2 employees work, give the project number and the number of employees*.

```
SELECT PNO, COUNT (*)
    FROM WORKSON
    GROUP BY PNO
    HAVING COUNT (*) > 2
```

Groups of projects with less than 2 people will not appear.

Ordering list of tuples

Ordering tuples of the result of a query by the value of 1 or more attributes. **ORDER BY** clause.

Example:

Query 18: *List of employees with their projects ordered by department, last name, first name*.

```
SELECT DName, LName, FName, PName
    FROM PROJECT, WORKSON, EMPLOYEE, DEPARTMENT
    WHERE DNumber = NumDept
    AND SSN = ESSN
    AND PROJECT.PNumber = WORKSON.PNO
    ORDER BY DName, LName, FName
```

Also **ORDER BY** X **DESC**, Y **ASC**.

Definition Join: Operation of combining data from multiple tables in a database based on a related column, allowing for efficient data retrieval and analysis.

Types of Join: Inner Join, Left (Outer) Join, Right (Outer) Join, Full (Outer) Join, Cross (Cartesian) Join, Self Join

Type - Inner Join:

- Returns matching rows between two or more tables based on the specified join condition.
- Retrieve records that have matching values in the join columns of both tables.

Example:

Query 19: *Retrieve the first name of employees and the department they belong to (DName)*.

```
SELECT EMPLOYEE.FName, DEPARTMENT.DName
FROM EMPLOYEE
JOIN DEPARTMENT ON EMPLOYEE.NumDept =
DEPARTMENT.DNumber
```

Type - Left Join (or Left Outer Join):

- Returns all rows from the left table and matching rows from the right table based on the join condition.

Example:

Query 20: *Retrieve the first name of employees and the department they belong to (DName) .*

```
SELECT EMPLOYEE.FName, DEPARTMENT.DName
FROM EMPLOYEE
LEFT JOIN DEPARTMENT ON EMPLOYEE.NumDept =
DEPARTMENT.DNumber
```

Type - Right Join (or Right Outer Join):

- Returns all rows from the right table and matching rows from the left table based on the join condition.

Example:

Query 21: *Retrieve the first name of employees and the department they belong to (DName).*

```
SELECT EMPLOYEE.FName, DEPARTMENT.DName
FROM EMPLOYEE
RIGHT JOIN DEPARTMENT ON EMPLOYEE.NumDept =
DEPARTMENT.DNumber
```

Type - Full Join (or Full Outer Join):

- Returns all rows from both the left and right tables.
- Retrieve all records from both tables, including unmatched rows.

Example:

Query 22: *Retrieve the first name of employees and the department name they belong to, including all employees and departments*.

```
SELECT EMPLOYEE.FName, DEPARTMENT.DName
FROM EMPLOYEE
FULL JOIN DEPARTMENT ON EMPLOYEE.NumDept =
DEPARTMENT.DNumber
```

Type - Cross Join (or Cartesian Join):

- Returns the Cartesian product of two tables, combining each row from the first table with every row from the second table.
- Generate all possible combinations between two tables.

Example:

Query 23: *Retrieve the first name of employees and the department name of all possible combinations*.

```
SELECT EMPLOYEE.FName, DEPARTMENT.DName
FROM EMPLOYEE
CROSS JOIN DEPARTMENT
```

Type - Self Join:

- Joins a table to itself, treating it as two separate instances, often using different aliases.
- Retrieve related data within the same table, such as hierarchical or hierarchical relationships.

Example:

Query 24: *Retrieve the first name of employees and the first name of their respective bosses*.

```
SELECT E1.FName, E2.FName
FROM EMPLOYEE E1
JOIN EMPLOYEE E2 ON E1.BossSSN = E2.SSN
```

SELECT < attribute list >
FROM < table list >
[**JOIN** < condition >]
[**WHERE** < condition >]
[**GROUP BY** < grouping attribute(s) >]
[**HAVING** < group condition >]
[**ORDER BY** < attribute list >]

Only SELECT and FROM are mandatory.

Data manipulation: 3 commands to modify the database

- INSERT: **INSERT INTO** *R* **VALUES** $(v_1, \ldots, v_n)$

Example:

INSERT INTO *EMPLOYEE*(FName, LName, SSN) **VALUES** (”Juliette”, ”Montgomery”, 4055467)

- UPDATE: **UPDATE** *R* **SET** $A_1 = v_1, A_n = v_n$ **WHERE** *cond*

Example:

UPDATE *EMPLOYEE* E **SET** Salary = Salary*1.1 **WHERE** E.FName = ”Juliette” **AND** E.Lastname = ”Mont-gomery”

- DELETE: **DELETE FROM** *R* **WHERE** *cond*

Example:

DELETE FROM *EMPLOYEE* **WHERE** LName = ”Mongomery”

Views in SQL

- View: ”virtual table”, logical relation $\neq$ physical
- Table derived from other tables or views
- Advantage: simplification and security

- Specification of views:
  CREATE VIEW V ($A_1, \ldots, A_n$) **AS** *query that defines the view*

Example:

CREATE VIEW *MY-WORKON* **AS**
SELECT FName, LName, PName, Hours
FROM *EMPLOYEE*, *PROJECT*, *WORKSON*
WHERE SSN = ESSN **AND** PNO = PNumber

Update: problem!

- View defined on a single table without any aggregate function: OK
- Views defined on multiple tables using joins are generally not updatable
- View using grouping or aggregate functions are not updatable
- Views with a single defining table is updatable if the view attributes contain a primary key of the base relation (maps each virtual view tuple to a single base tuple)

Strategies

- **Query modification**: query on the underlying tables instead of the view
- **View materialization**: physical temporary table creation

Questions?

What will come next?

1 Welcome to Database Systems
2 Introduction to Database Systems
3 Entity Relationship Design Diagram (ERM)
4 Relational Model
5 Relational Algebra
6 Structured Query Language (SQL)
7 Relational Database Design - Functional Dependencies
8 Relational Database Design - Normalization
9 Online Analytical Processing + Embedded SQL
10 Data Mining
11 Physical Representation - Storage and File Structure
12 Physical Representation - Indexing and Hashing
13 Transactions
14 Concurrency Control Techniques
15 Recovery Techniques
16 Query Processing and Optimization