University of Zurich^UZH

# Language Technology and Web Applications

## Databases

Johannes Graën

Wednesday 11th October, 2023

Department of Computational Linguistics & Linguistic Research Infrastructure, University of Zurich

# Overview

Data Structure & Diagrams (UML & ER)

Defining the Database Schema (DDL)

Manipulating a Database (DML)

Querying a Database (DQL)

# Learning Goals for this Week

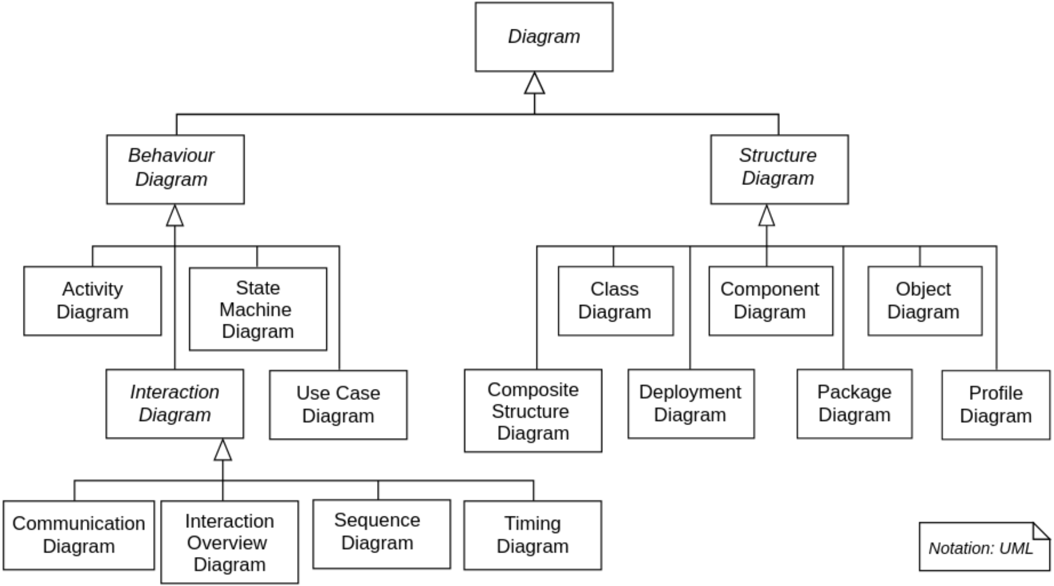
- You know how to represent relational structures in ER and UML diagrams
- You can translate such diagrams to a database schema and vice versa
- You know how to query and manipulate data from a database comprehending its schema

# Data Structure & Diagrams (UML & ER)

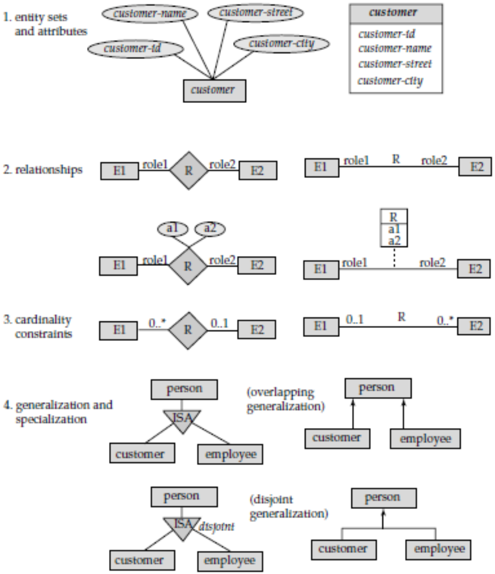
# UML

- UML = Unified Modeling Language
- a large number of diagram types, class diagrams for modeling data structures
- class diagrams are similar to ER (Entity Relationship) diagrams

# UML Diagram Hierarchy



# Comparion ER ↔ UML



# Entities & Relations vs. Tables

- tables are used to represent entity types
- each row represents a concrete entity
- attributes are realized as columns
- each column has a name and a data type
- columns can be *nullable* (cardinalities!)
- relations are translated to tables if both sides have cardinalities > 1

# Defining the Database Schema (DDL)

# SQL

SQL consist of different sublanguages, namely

- DDL: Data Definition Language
- DML: Data Manipulation Language
- DQL: Data Query Language
- DCL: Data Control Language

# DDL: Data Definition Language

DDL Commands:

- *CREATE*: create something (eg. a table)
- *DROP*: remove something (eg. a table)
- *ALTER*: modify something (eg. a table)
- *TRUNCATE*: remove all data from a table

*https://www.postgresql.org/docs/current/ddl.html*

# CREATE TABLE statement

Creates an empty table with the columns defined.

```
CREATE TABLE person (
    name    varchar   NOT NULL,
    age     int
);
```

- tables can be TEMPORARY
- *NOT NULL* needs to be specified if a column is not nullable

*https://www.postgresql.org/docs/current/sql-createtable.html*

# ALTER TABLE statement

```
ALTER TABLE person ADD COLUMN country varchar (2);
ALTER TABLE person ALTER COLUMN age SET NOT NULL;
ALTER TABLE person ALTER COLUMN name DROP NOT NULL;
ALTER TABLE person ADD CONSTRAINT adults_only CHECK (age >= 18);
```

- the command will fail if conditions are not met (eg. ***NOT NULL*** for a new column)

*https://www.postgresql.org/docs/current/sql-altertable.html*

# DROP TABLE statement

Removes the table specified.

```
DROP TABLE person;
DROP TABLE person CASCADE;
```

- the statement will fail if anything depends on the table (referential integrity!)
- *CASCADE* will remove everything that directly or indirectly depends on the table

*https://www.postgresql.org/docs/current/sql-droptable.html*

# Constraints

Constraints restrict the permitted values for columns.

- a **check constraint** defines a condition on an individual row that needs to be met (`age >= 18`)
- the **not null** constraint disallows the `NULL` value (ie. undefined or unknown)
- **primary keys** identify a record
- **foreign keys** reference a record
- the **unique** constraint requires each value of a column to be unique in the table

`https://www.postgresql.org/docs/16/ddl-constraints.html`

# Primary Keys

- used to identify records (rows)
- need to be unique
- can be generated with the help of sequences
- UUIDs can be used instead of integers (security, multi-master setups)

```
CREATE TABLE person (
    person_id      int       PRIMARY KEY,
    name           varchar   NOT NULL,
    age            int
);
```

There are two options regarding the generated values for the primary key column:

- *GENERATED BY DEFAULT AS IDENTITY* (queries can define the value)
- *GENERATED ALWAYS AS IDENTITY* (queries can't define the value)

# Foreign Keys

```
CREATE TABLE driver (
    person_id      int        REFERENCES person (person_id),
    numberplate    varchar    NOT NULL
);
```

- every value in the *person_id* column in the *driver* table must be present in the *person_id* column of table *person* (referential integrity!)
- columns referenced must be unique (not necessarily primary keys)
- primary and foreign keys can consist of multiple columns

# Manipulating a Database (DML)

# DML: Data Manipulation Language

DML Commands:

- *`INSERT INTO`*: insert data into a table
- *`COPY`*: bulk insert data
- *`DELETE FROM`*: remove rows from a table
- *`UPDATE`*: update data in a table

*`https://www.postgresql.org/docs/current/dml.html`*

# INSERT INTO statement

```
INSERT INTO person VALUES ('John Doe', 42);
INSERT INTO person (id, name, age) VALUES (1, 'Jane Doe', 39);
INSERT INTO person
      VALUES ('Tick', 10), ('Trick', 10), ('Track', 11);
```

- rows can be inserted by specifying all required columns in the given order
- … or by explicitly listing the columns
- instead of the individual rows (VALUES), a query can be provided

*https://www.postgresql.org/docs/current/dml-insert.html*

# COPY statement

```
COPY person (name, age) FROM STDIN WITH DELIMITER E'\t'
John Doe    42
Jane Doe    39
Tick        10
Trick       10
Track       11
\.
```

- *STDIN* and *STDIN* can be used to pipe data in and out when accessing the database via network connection (as opposed to locally)

*https://www.postgresql.org/docs/current/sql-copy.html*

# DELETE FROM statement

```
DELETE FROM person;
DELETE FROM person WHERE age < 18;
```

- *`WHERE`* condition defines which criteria need to be met for a row be deleted
- *`TRUNCATE`* is more efficient than *`DELETE FROM`* without any constraint

# UPDATE statement

```
ALTER TABLE person ADD COLUMN is_adult boolean;
UPDATE person SET is_adult = FALSE WHERE age < 18;
UPDATE person SET is_adult = TRUE WHERE age >= 18;
ALTER TABLE person ALTER COLUMN is_adult SET NOT NULL;
```

Combine the two statements:

```
ALTER TABLE person ADD COLUMN is_adult boolean;
UPDATE person SET is_adult = age >= 18;
ALTER TABLE person ALTER COLUMN is_adult SET NOT NULL;
```

Even better:

```
ALTER TABLE person ADD COLUMN is_adult boolean NOT NULL GENERATED
    ALWAYS AS (age >= 18) STORED;
```

# Querying a Database (DQL)

# DQL: Data Querying Language

DQL Commands:

- *TABLE*: lists the contents of a table
- *SELECT*: runs a complex query on the database

*https://www.postgresql.org/docs/current/queries-overview.html*

# TABLE statement

```
TABLE person;
```

- equivalent to *`SELECT * FROM person;`*

# SELECT statement

```sql
SELECT 1;
SELECT 1 + 1;
SELECT name, age FROM person;
SELECT * FROM person;
SELECT DISTINCT age FROM person;
```

- the by far most complex statement
- the asterisk (*) stands for all attribute

# SELECT statement: WHERE clause

```
SELECT * FROM person WHERE age >= 18;
```

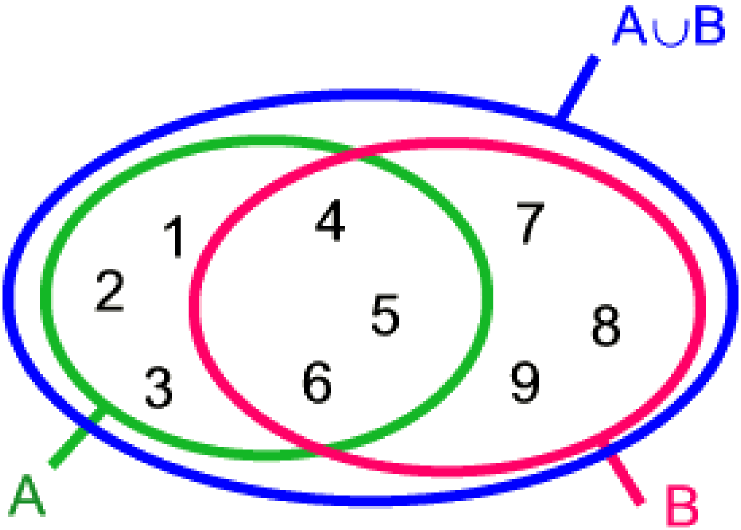
- the condition of the **WHERE** clause can use any attribute, call funtions, use subqueries etc.

# SELECT statement: JOINs

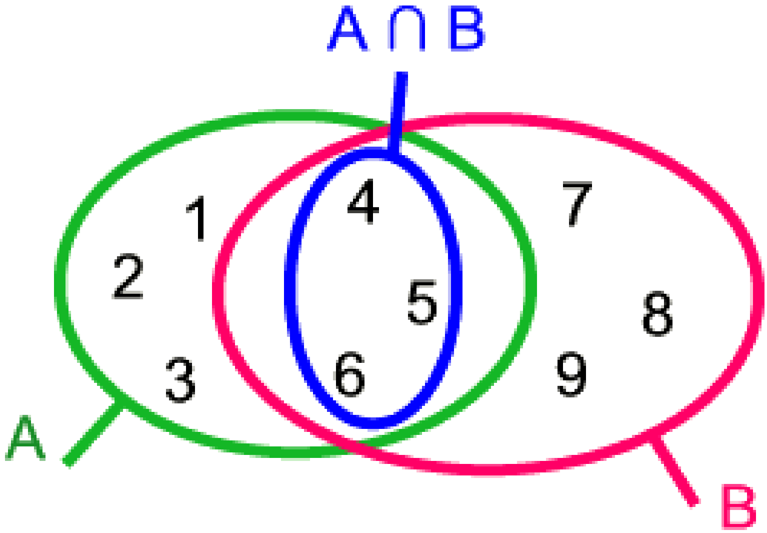
```
SELECT * FROM person JOIN driver
        ON person.person_id = driver.person_id;
SELECT * FROM person JOIN driver USING (person_id);
SELECT * FROM person NATURAL JOIN driver;
```

- the ***NATURAL JOIN*** will be performed on all columns with the same name!

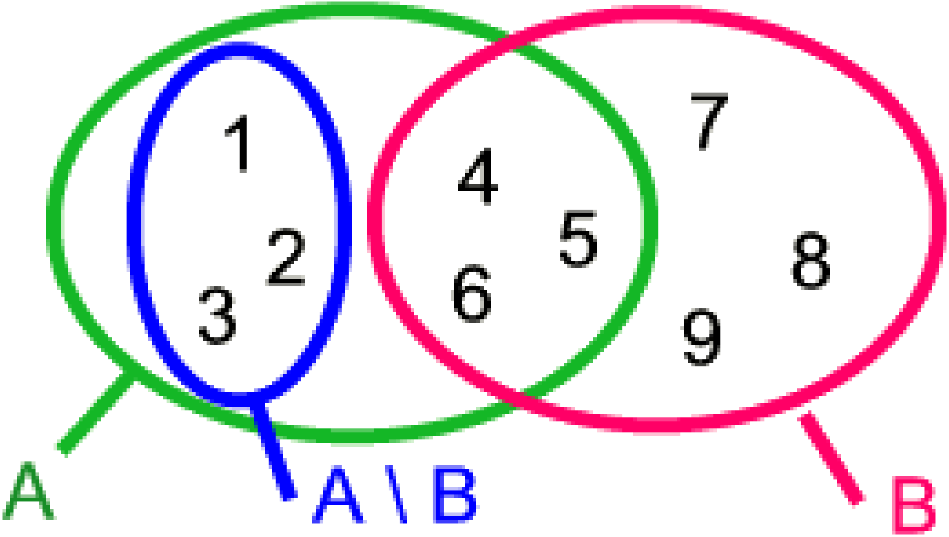
# Set Theory: Union



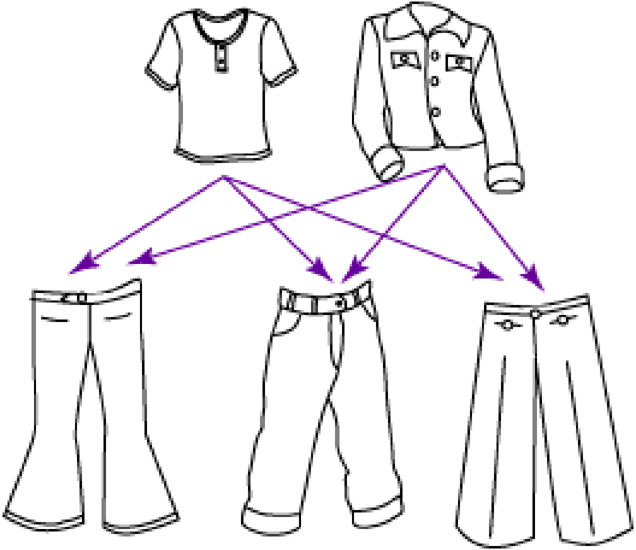
# Set Theory: Intersection



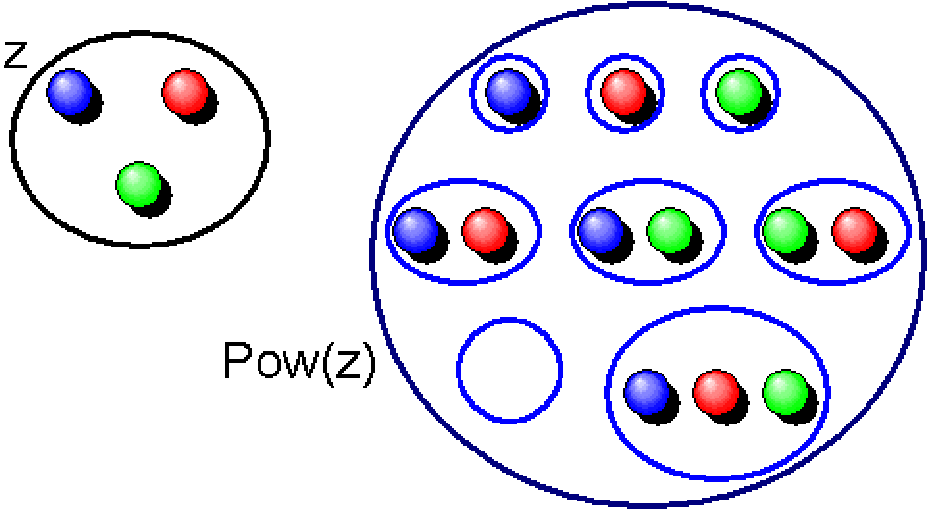
# Set Theory: Difference



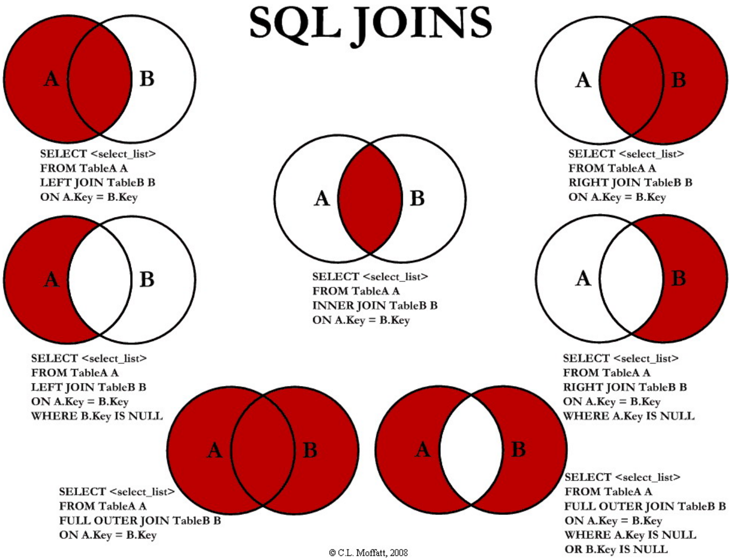
# Set Theory: Cartesion Product



# Set Theory: Power Set



# Joins and Set Theory



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

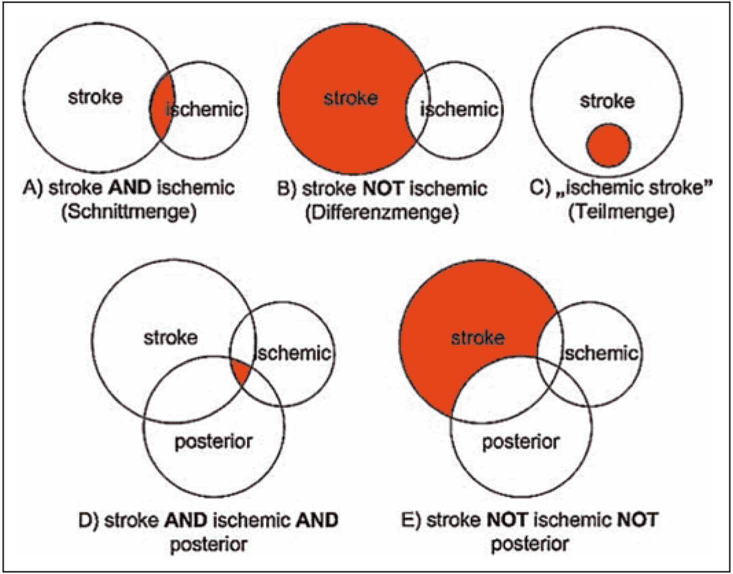
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

# Joins and Set Theory



# SELECT statement: different JOINs

```
SELECT * FROM person INNER JOIN driver USING (person_id);
SELECT * FROM person LEFT JOIN driver USING (person_id);
SELECT * FROM person RIGHT JOIN driver USING (person_id);
SELECT * FROM person FULL OUTER JOIN driver USING (person_id);
```

- a *LEFT JOIN* where the left table references the right one is equivalent to an *INNER JOIN* and vice versa

# SELECT statement: Semi Joins and Anti Joins

```
SELECT * FROM person WHERE EXISTS (
      SELECT 1 FROM driver
      WHERE driver.person_id = person.person_id
);
```

- keywords are *IN*, *EXISTS*, *ANY* / *SOME*, *ALL*
- anti joins are negated semi joins
- for some cases, set operations *UNION*, *INTERSECT* and *EXCEPT* are more efficient than joins

# What is still missing?

- $\Theta$ joins
- data types and domains
- ternary logic
- aggregates and window functions
- indices