GEO 874 I HS25
Universität Zürich

3. Lecture Introduction to Databases

Complex SQL Queries & Database design

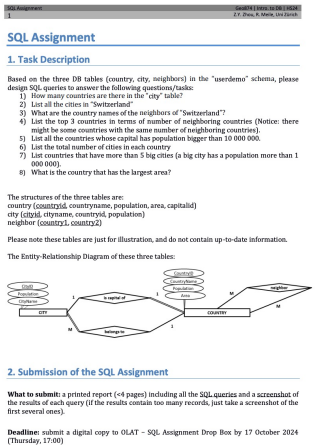
Esra Suel
Dept. of Geography, University of Zürich

Rolf Meile
Eidg. Forschungsanstalt für Wald, Schnee und Landschaft (WSL)

kudos to Dr. Zhiyong Zhou, Dr. Cheng Fu, Dr. Haosheng Huang for providing this document

SQL Assignment

- Based on the three DB tables in the „userdemo“ schema, design SQL queries to answer tasks.
  - More details on: https://www.geo.uzh.ch/microsite/geo874/scripte/geo874_HS25_SQL_Assignment.pdf



SQL Assignment

1. Task Description

Based on the three DB tables (country, city, neighbors) in the "userdemo" schema, please design SQL queries to answer the following questions/tasks:

1) How many countries are there in the "city" table?
2) List all the cities in "Switzerland"
3) What are the country names of the neighbors of "Switzerland"?
4) List the top 3 countries in terms of number of neighboring countries (Notice: there might be some countries with the same number of neighboring countries).
5) List all the countries whose capital has population bigger than 10 000 000.
6) List the total number of cities in each country
7) List countries that have more than 5 big cities (a big city has a population more than 1 000 000).
8) What is the country that has the largest area?

The structures of the three tables are:
country (countryid, countryname, population, area, capitalid)
city (cityid, cityname, countryid, population)
neighbor (country1, country2)

Please note these tables are just for illustration, and do not contain up-to-date information.

The Entity-Relationship Diagram of these three tables:

2. Submission of the SQL Assignment

What to submit: a printed report (<4 pages) including all the SQL queries and a screenshot of the results of each query (if the results contain too many records, just take a screenshot of the first several ones).

Deadline: submit a digital copy to OLAT – SQL Assignment Drop Box by 17 October 2024 (Thursday, 17:00)

- 30% of the final grade
- Individual work

- **What to submit:** a report (<4 pages) including all the SQL queries and a screenshot of each query.
- **How to submit:** submit to OLAT
- **Deadline:** by **16 October 2025** (Thursday, 18:00)

Recap

- SQL (Structured Query Language) – language for relational databases
- Queries over tables
  SELECT → Projection (columns), Selection (rows)
- Joins – combination of tables

```
SELECT name, age
FROM students;
```

Projection (columns)

```
SELECT name, age
FROM students
WHERE age > 20;
```

Selection (rows)

Recap

- SQL (Structured Query Language) – language for relational databases
- Queries over tables
  SELECT → Projection (columns), Selection (rows)
- Joins – combination of tables

- Modification of tables
  (CREATE, ALTER, DROP)
- Modification of data in tables
  (INSERT, UPDATE, DELETE)

Today's Practical 

Querying data: SELECT - statement

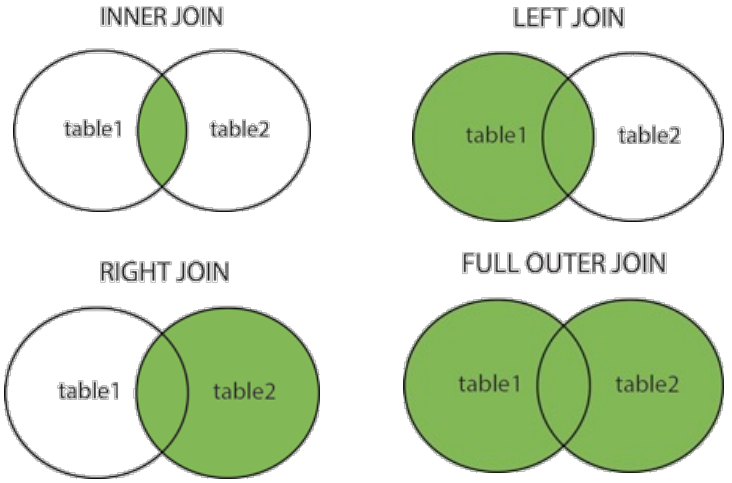
Basic SQL statement for data querying: SELECT

```
SELECT <Column1>, <Column2>, …
FROM   <TableA>
WHERE  <Column1> = …;
```

SFW-Block

Different types of JOIN

- Join type
  - INNER JOIN (a.k.a. JOIN)
  - LEFT OUTER JOIN (a.k.a. LEFT JOIN)
  - RIGHT OUTER JOIN (a.k.a. RIGHT JOIN)
  - FULL OUTER JOIN (a.k.a. FULL JOIN)



- T1 join_type T2 ON [join_condition]
  - Join condition normally compares two columns, which must be of the same data type. (Number = Number, String = String)
  - Most often „Foreign Key“ = „Primary Key“

https://www.postgresql.org/docs/13/queries-table-expressions.html

Why SELECT Comes First (but Runs Last)

Example query

```
SELECT T1.name, T2.course
FROM Students T1
INNER JOIN Enrollments T2
    ON T1.s_id = T2.s_id
WHERE T1.age > 20;
```

How it's written

1. SELECT …
2. FROM … INNER JOIN … ON …
3. WHERE

How its executed (logical order)

1. FROM + JOIN
   - Combine tables T1 and T2 using the join condition.
   - Result: all columns from both tables, but only rows where `T1.s_id = T2.s_id`.
2. WHERE
   - Filter rows: keep only those where T1.age > 20.
3. SELECT
   - Project the requested columns (name, course) from the intermediate result.

Basic steps of solving query tasks

- 1. Which **table(s)** are needed?
- 2. If two or more tables are involved, which **type of JOIN** to be used, and the **join condition**?
- 3. **Other filter conditions** (to be used in WHERE part)?
- 4. **GROUP BY** and aggregation (e.g., Avg, Max, Min) needed?
- 5. Which columns/attributes/fields to be shown in the results (to be used in the SELECT part)?

Basic steps of solving query tasks – Example (1)

- **In which town(s) we can find restaurant Castello?**

Zip: zipcode, zipname

Foodcountrycat (food country categories): fcccode, fccdescription

Person: persid, persname, persfirstname

Bestrest (list of best restaurants): bestrestid, bestrestname, streetname, streetno, zipcode, fcccode

Report (report and opinion for a best restaurant): bestrestid, persid, opiniontext, opiniondate

- 1. Which **table(s)** are needed? **Bestrest, Zip**
- 2. If two or more tables are involved, Which **type of JOIN** to be used, and the **join condition**? **Inner join, bestrest.zipcode = zip.zipcode**
- 3. **Other filter conditions** (to be used in WHERE part)? **bestrestname='Castello'**
- 4. **GROUP BY** and aggregation (e.g., Avg, Max, Min) needed? **No**
- 5. Which columns/attributes/fields to be shown in the results (to be used in the SELECT part)? **bestrestname, zipname**

```
SELECT r.bestrestname ,z.zipname FROM userdemo.bestrest r
INNER JOIN userdemo.zip z ON r.zipcode=z.zipcode
WHERE r.bestrestname = 'Castello';
```

Basic steps of solving query tasks – Example (2)

- **Which restaurants are listed although there is no report about it?**

Zip: zipcode, zipname

Foodcountrycat (food country categories): fcccode, fccdescription

Person: persid, persname, persfirstname

Bestrest (list of best restaurants): bestrestid, bestrestname, streetname, streetno, zipcode, fcccode

Report (report and opinion for a best restaurant): bestrestid, persid, opiniontext, opiniondate

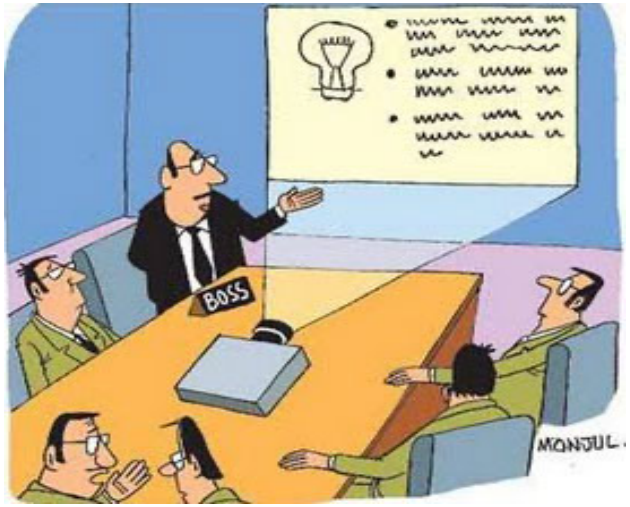
- 1. Which **table(s)** are needed? **bestrest, report**
- 2. If two or more tables are involved, Which **type of JOIN** to be used, and the **join condition**? **Left outer join on bestrestid (because we want to see a full list of restaurants; restaurants without reports will have a NULL persid)**
- 3. **Other filter conditions** (to be used in WHERE part)? **Persid IS NULL**
- 4. **GROUP BY** and aggregation (e.g., Avg, Max, Min) needed? **No**
- 5. Which columns/attributes/fields to be shown in the results (to be used in the SELECT part)? **bestrestname, …**

```
SELECT r.bestrestname, r.streetname, r.streetno,
       rp.persid, rp.opiniondate, rp.opiniontext
FROM userdemo.bestrest r
LEFT OUTER JOIN userdemo.report rp ON r.bestrestid=rp.bestrestid
WHERE rp.persid IS NULL;
```

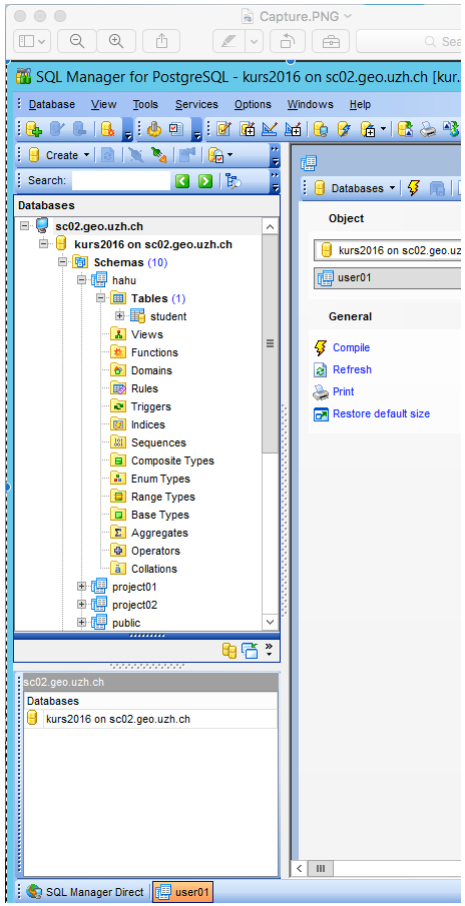
SQL - Today

- View
- more complex SQL queries (e.g., GROUP BY, HAVING, nested queries)

Database Design



Real World



Database

Learning Objectives 3

- ✓ You will learn about **views** and will be able to create them.
- ✓ You will be able to write more complex SQL queries.
- ✓ You will be able to explain and illustrate the phases of DB design
- ✓ You will know what it means to perform a **requirements analysis**
- ✓ You will understand the difference between Entity, Entity type, and Entity collection
- ✓ You will be able to enumerate diverse attribute types

Contents

Views (virtual tables) in SQL

- A view is a virtual table based on the result-set of an SQL statement (e.g., a “SELECT statement”).

- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- You can use a view like a real table in SQL queries.

View

- Create/delete a view

```
CREATE VIEW user01.test1 AS
    SELECT *
    FROM        user01.student
    WHERE       Course = 'Math'
```

```
DROP VIEW user01.test1
```

Table: student

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

- Use a view

```
SELECT * FROM user01.test1
```

The view is not physically materialized/stored. Instead, the query is run every time the view is referenced in a query.

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Why views?

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual tables stored in the database).

- Consider a user who needs to know an instructor's name and department, but not the salary. This user should see a table described, in SQL, by

```
CREATE VIEW user01.test1 AS
SELECT ID, name, dept_name
FROM user01.instructor
```

- A view provides a mechanism to hide certain data from the view of certain users.

Contents

Data Queries: Aggregate Functions

- Aggregate functions: Avg(), Max(), Min(), Sum(), Count(), …

- **Student A's average grade**

```
SELECT Avg (Grade)
FROM user01.student_grades
WHERE Name = 'A'
```

- **Lowest grade of Course "Math"**

```
SELECT Min (Grade)
FROM user01.student_grades
WHERE Course = 'Math'
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Data Queries: Group By

- **GROUP BY:** is used in conjunction with the aggregate functions to group the result-set by one or more columns.
  - E.g., to get the average/highest/lowest grade of each course
  - Aggregate functions: Avg(), Max(), Min(), Sum(), Count(), …

```
SELECT Course, Avg (Grade)
FROM user01.student_grades
GROUP BY Course
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Name	Course	Grade
A	Math	80
B	Math	75
C	Math	60
A	English	90
B	English	75
C	English	80
A	Physics	70
B	Physics	75
C	Physics	50

Results:

Course	Avg
Math	72
English	82
Physics	63

Exercise: Group By

- **Write an SQL Query for "What is the average grade of each student?"**

```
SELECT Name, Avg (Grade)
FROM user01.student_grades
GROUP BY Name
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Table: student

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Results:

Name	Avg
A	80
B	75
C	63.33

Very time consuming! It will take a long time on large tables.

Data Queries: Having (Group By)

- **HAVING:** similar to WHERE, often used with aggregate functions and GROUP BY.
  - It was added to SQL because WHERE could **not** be used with aggregate functions
  - E.g., to get the students whose average grade is bigger than 75

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

```
SELECT Name, Avg(Grade)
FROM user01.student_grades
GROUP BY Name
HAVING Avg(Grade)>75
```

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Name	Avg
A	80
B	75
C	63

Results:

Name	Avg
A	80

Data Queries: Ranking functions

- Ranking functions: RANK(), Dense_RANK(), ROW_Number()…

```
RANK() OVER (
  [PARTITION BY expression, ]
  ORDER BY expression (ASC | DESC) );
```

RANK	It assigns the rank number to each row in a partition. It **skips** the number for similar values.
Dense_RANK	It assigns the rank number to each row in a partition. It **does not skip** the number for similar values.
ROW_Number	It assigns the **sequential rank number** to each unique record.

Data Queries: Ranking functions

- Ranking functions: RANK(), Dense_RANK(), ROW_Number()…

```
RANK() OVER (
  [PARTITION BY expression, ]
  ORDER BY expression (ASC | DESC) );
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

```
SELECT Name, RANK() OVER
(ORDER BY Grade DESC) as StudentRank
FROM student_grades
```

Name	Course	studentrank
A	English	1
A	Math	2
C	English	2
B	English	4
B	Math	4
B	Physics	4
A	Physics	7
C	Math	8
C	Physics	9

Data Queries: Ranking functions

- Ranking functions: RANK(), Dense_RANK(), ROW_Number()…

```
RANK() OVER (
  [PARTITION BY expression, ]
  ORDER BY expression (ASC | DESC) );
```

SELECT Name, **DENSE_RANK**() OVER (ORDER BY Grade DESC) as StudentRank FROM student_grades

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

ABC Name	ABC Course	123 studentrank
A	English	1
A	Math	2
C	English	2
B	English	3
B	Math	3
B	Physics	3
A	Physics	4
C	Math	5
C	Physics	6

Data Queries: Ranking functions

- Ranking functions: RANK(), Dense_RANK(), ROW_Number()…

```
RANK() OVER (
  [PARTITION BY expression, ]
  ORDER BY expression (ASC | DESC) );
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

```
SELECT Name, RANK() OVER
(PARTITION BY Course
ORDER BY Grade DESC) as StudentRank
FROM student_grades
```

Name	Course	studentrank
A	English	1
C	English	2
B	English	3
A	Math	1
B	Math	2
C	Math	3
B	Physics	1
A	Physics	2
C	Physics	3

Nested subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a select-from-where expression that is nested within another query.
- For more details, please refer to: https://www.postgresql.org/docs/current/functions-subquery.html

- **Who has the best grade in English?**

```
SELECT Name, Grade
FROM user01.student_grades
WHERE Course='English' AND
        Grade = (
                SELECT MAX (Grade)
                FROM user01.student_grades
                WHERE Course = 'English'
        )
```

Table: **student_grades**

Name	Course	Grade
A	Math	80
A	English	90
A	Physics	70
B	Math	75
B	English	75
B	Physics	75
C	Math	60
C	English	80
C	Physics	50

Results:

Name	Grade
A	90

Nested subqueries: EXISTS

- The EXISTS operator is used to test for the existence of any record in a subquery.
- The EXISTS operator returns true if the subquery returns one or more records.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

➢ **list the suppliers with a product price less than 20:**

Products

ProductID	ProductName	SupplierID	Price
1	Chais	1	18
2	Chang	1	19
3	Aniseed Syrup	1	10
4	Cajun Seasonin	2	22
5	Gumbo Mix	2	21.35

Suppliers

SupplierID	SupplierName	City	Country
1	Exotic Liquid	London	UK
2	New Orleans Cajun Delights	New Orlea	USA
3	Grandma Kelly's Homestead	Ann Arbor	USA
4	Tokyo Traders	Tokyo	Japan

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS
      (SELECT ProductName
       FROM Products
       WHERE Products.SupplierID =
            Suppliers.SupplierID AND Price < 20
       );
```

Results:

SupplierName
Exotic Liquid

This task can be also solved using JOIN, or nested queries with IN

Contents

Permissions / Security

Access rights can be granted for access to diverse database objects

- Tables
- Views
- Stored procedures / Functions

These permissions are granted to a **Role**.

- A role is an entity that can own database objects and has database privileges;
- In PostgreSQL, a role can be considered a "user“ or a "group"

Roles

- `CREATE ROLE rolename;` /* Creating role */
- `DROP ROLE rolename;` /* Deleting roles */

```
Grant SELECT ON TABLE user01.Person TO user02
```

- `GRANT SELECT, DELETE, UPDATE ON TABLE user01.Person TO rolename`
  /* defining access privileges (giving rights) */
- `REVOKE DELETE, UPDATE ON TABLE user01.Person FROM rolename`
  /* removing access privileges (taking rights away) */

- **Role: https://www.postgresql.org/docs/current/user-manag.html**
- **Grant: https://www.postgresql.org/docs/current/sql-grant.html**
- **Revoke: https://www.postgresql.org/docs/current/sql-revoke.html**

Contents

Transaction

- fundamental concept of all database systems
- The essential point of a transaction is that it combines multiple steps (e.g., multiple SQL queries) into a single, all-or-nothing operation.

- The intermediate states between the steps are not visible to other concurrent transactions, and if some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.

- By default, each SQL statement = one transaction

Transaction Control Language

- A transaction is the execution of a set of SQL statements that access the DB and
  - Starts with a BEGIN operation
  - Followed by a number of SQL statements
  - And ends with a COMMIT or ROLLBACK operation
    - COMMIT: apply the transaction by saving the changes to the database
    - ROLLBACK: "undo" changes of a given (entire) transaction – reverts to the state before all of the statements in this transaction

```
BEGIN;
UPDATE accounts SET balance = balance + 100.00 WHERE acctnum = 12345;
UPDATE accounts SET balance = balance - 100.00 WHERE acctnum = 7534;
COMMIT;
```

Transaction: ACID

- Transaction properties:
  - **A**tomicity
    - The set of operations on the database is either executed in its entirely, or not at all.
    - Entire transaction succeeds or fails (all-or-nothing)
    - Example: transfer of funds from account A to account B: either both, the debit on A and the deposit into B are executed, or none of the two.
  - **C**onsistency
    - After a transaction has been executed, the integrity constraints have to be satisfied.
    - Valid state of DB before AND after transaction
    - Example: During the execution, there may be violations, but if they remain until the end, the transaction has to be undone ("aborted").
  - **I**solation
    - Enables transactions to operate independently of and transparently to each other.
    - Other transactions cannot access data that has been modified during a transaction that has not yet completed.
  - **D**urability (persistence)
    - After the successful completion of a transaction, the DBMS commits to make the outcome of the transaction permanent, even in the presence of concurrency and/or breakdowns

Contents

Life cycle of databases

1. **System definition** Definitions of the coverage of the DB, its users and application
2. **Database design** leads to a complete logical and physical database schema
3. **Database implementation** Creation of an empty database
4. **Data (extraction, transformation, and...) loading** Database is loaded with data directly or by converting existing data
5. **Application conversion** potential software applications developed for earlier systems/DB instances are modified to match the new system
6. **Testing and Validation** The new system is tested (!).
7. **Deployment** the DBs and its applications are deployed and used. Often, old and new systems run in parallel during the transition phase.
8. **Monitoring and maintenance** The system is monitored while deployed – the DB contents as well as the client applications can grow →period. Reorganization

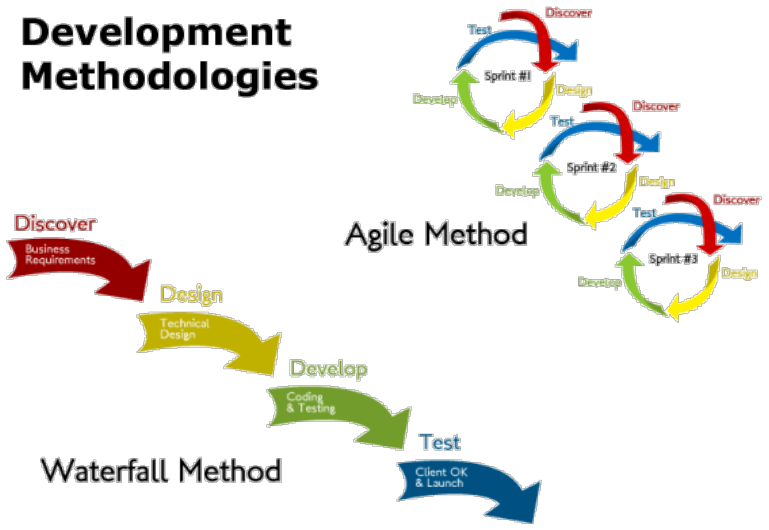
Covered

Note: more iterative lifecycles exist

- This is a very traditional process ("Waterfall"): highly structured, sequential (linear), not responsive to change, slow…

- Agile (or other iterative methods) favours frequent delivery of systems for feedback over static requirements.
  - A pragmatic approach – people change requirements. Very often.
- Will not be covered in this subject, but you will likely encounter it in life – Lots of online materials.

- **Be ready for change**.



DB Design

Def.: The design of the **logical** and **physical structure** of a database (in a given DBMS) so that it contains **all** the **information required** by the **user** and required for the efficient behavior of the whole **information system** for all users.

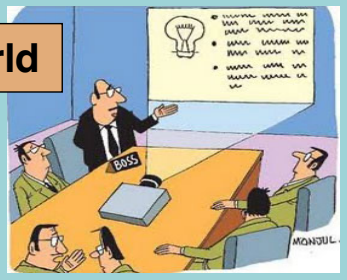
Source: B. Thalheim – Entity-Relationship Modeling. Foundations of Database Technology *(shortened)*

DB Design aims:

- **Identify what to store:** the information required by all applications and use cases
- **Identify what NOT to store:** only store data related to the information needs of the use cases
- **Store it smartly:** To reduce the redundancy of the stored data (structure, computation)

DB Design

Real World

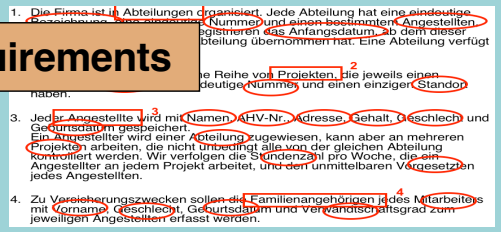


Entities in the real world:
Department head Müller,
Employee Muster,
Project 'Budget 2014'

Lecture 3 Requirements Analysis

Interview of users, study of documentation

Database Requirements



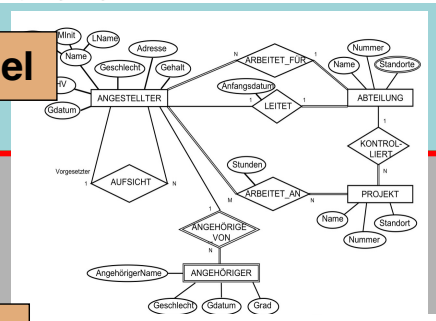
1. Die Firma ist in Abteilungen organisiert. Jede Abteilung hat eine eindeutige Nummer und einen bestimmten Angestellten
2. Eine Reihe von Projekten, die jeweils einen eindeutige Nummer und einen einziger Standor
3. Jeder Angestellte wird mit Namen, AHV-Nr., Adresse, Gehalt, Geschlech und Geburtsdatum gespeichert.
   Ein Angestellter wird einer Abteilung zugewiesen, kann aber an mehreren Projekten arbeiten, die nicht unbedingt alle von der gleichen Abteilung kontrolliert werden. Wir verfolgen die Stundenzahl pro Woche, die ein Angestellter an jedem Projekt arbeitet, und den unmittelbaren Vorgesetzten jedes Angestellten.
4. Zu Versicherungszwecken sollen die Familienangehörigen jedes Mitarbeiters mit Vorname, Geschlecht, Geburtsdatum und Verwandtschaftsgrad zum jeweiligen Angestellten erfasst werden.

Written concise description of requirements (**data** reqs and **functional** [operations] reqs)

Lecture 3/4 Conceptual DB design

Text analysis
Entities
Relationships

ER-Model



Entity types with attributes and relationships

DBMS-independent

DBMS-specific

Lecture 5 Logical DB design (data model mapping)

- translation of model to implementation
- Validation through normalization (optional)

DB-Schema (e.g. relational data model)

NGESTELLTER

VNAME	INITAL	NNAME	AHV	GDATUM
DRESSE	GESCHLECHT	GEHALT	SUPERAHV	ANR

BTEILUNG

ANAME	ABTNUMMMER	MGRAHV	MGR_ANFANGSDATUM

BT_STANDORTE

ABTNUMMMER	ASTANDORT

ARBEITET_AN

EAHV	PNR	STUNDEN

PROJEKT

PNAME	PNUMMER	PSTANDORT	ABTNR

ANGEHÖRIGER

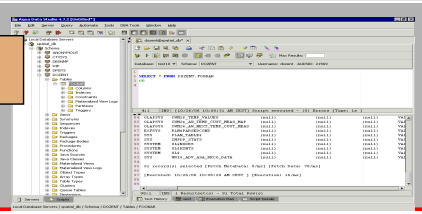
EAHV	ANGEHÖRIGER_NAME	GESCHLECHT	GDATUM	GRAD

Set of relations with attributes (incl. data types + value domains)

Lecture 5 Physical DB design

Description of database implementation (HW, indices, …)

Database

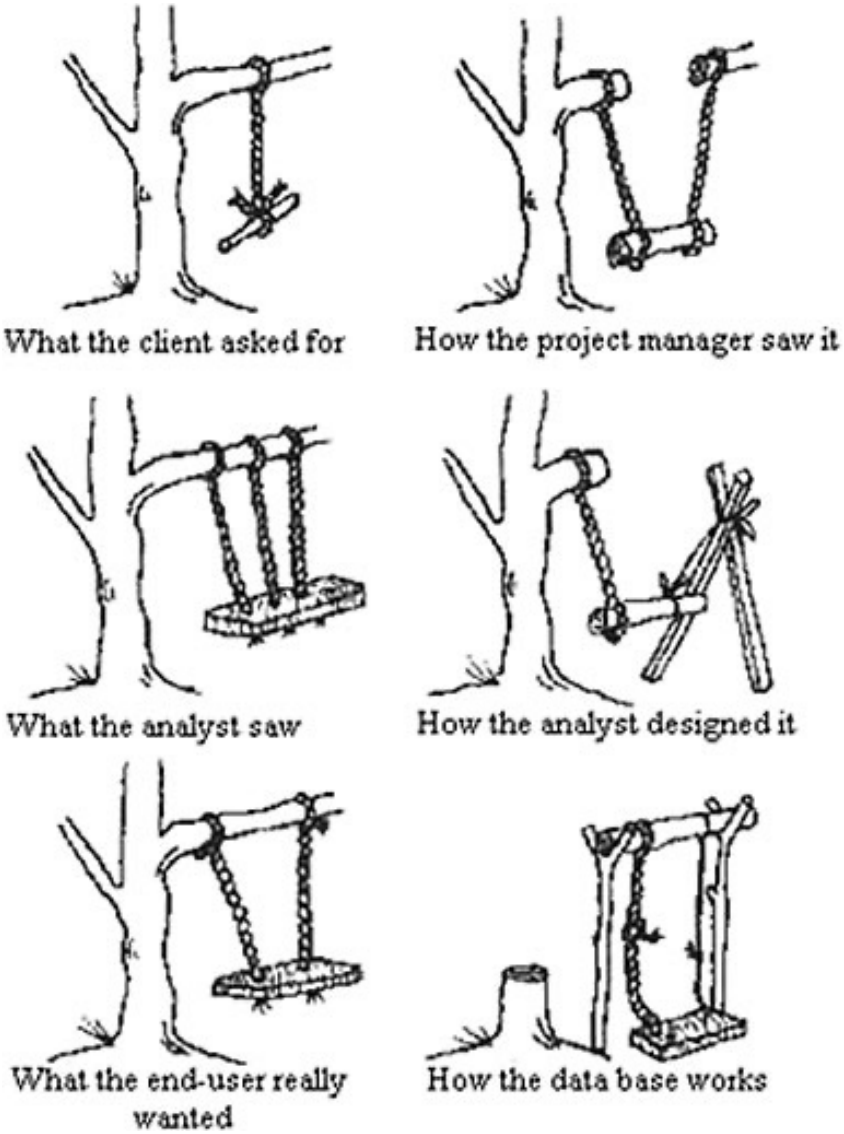


Implemented DB design, Tables (ordered) with rows and columns

➲ Elmasri & Navathe (2014): Ch 7-9

Contents

Requirements Analysis



Requirements Analysis

A recipe:

1. Identification of **organizational entities**
2. Identification of the **business outcomes** they support
3. Requirements gathering plan – identification of **personnel** to be interviewed

Identification of user groups and application areas

4. Requirements collection
5. Filtering: collected information must be checked for ambiguity and clarity, and priority
6. Systematization: identification of objects, relationships between objects and operations and processes
7. Formalization/Structuring

Structuring of the information about application areas

AIM: to provide inputs for the conceptual design – ER diagram

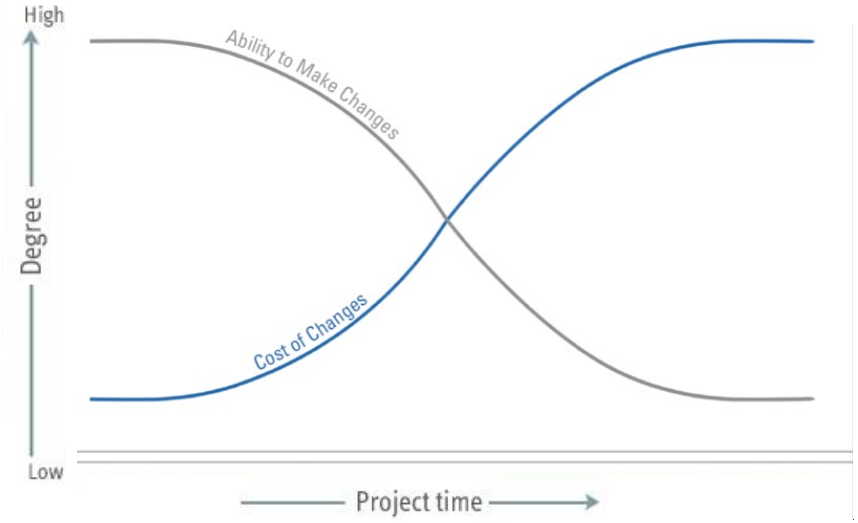
Requirements Analysis

- Initial insights are incomplete, vague, misleading...
- We need to compile the first catalog of requirements in a specification that will be reviewed by users and clients...
- ...the requirements will change!!
  - Meetings and workshops with stakeholders
  - E.g., Joint Application Design (JAD): designers and users work together
  - Prototyping / mocking

- **BUT**: the process is very important!

- Costs of changes:

  later >> @beginning

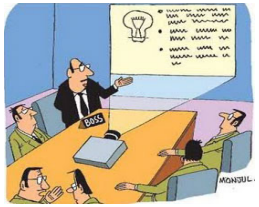


Outcomes of requirements analysis

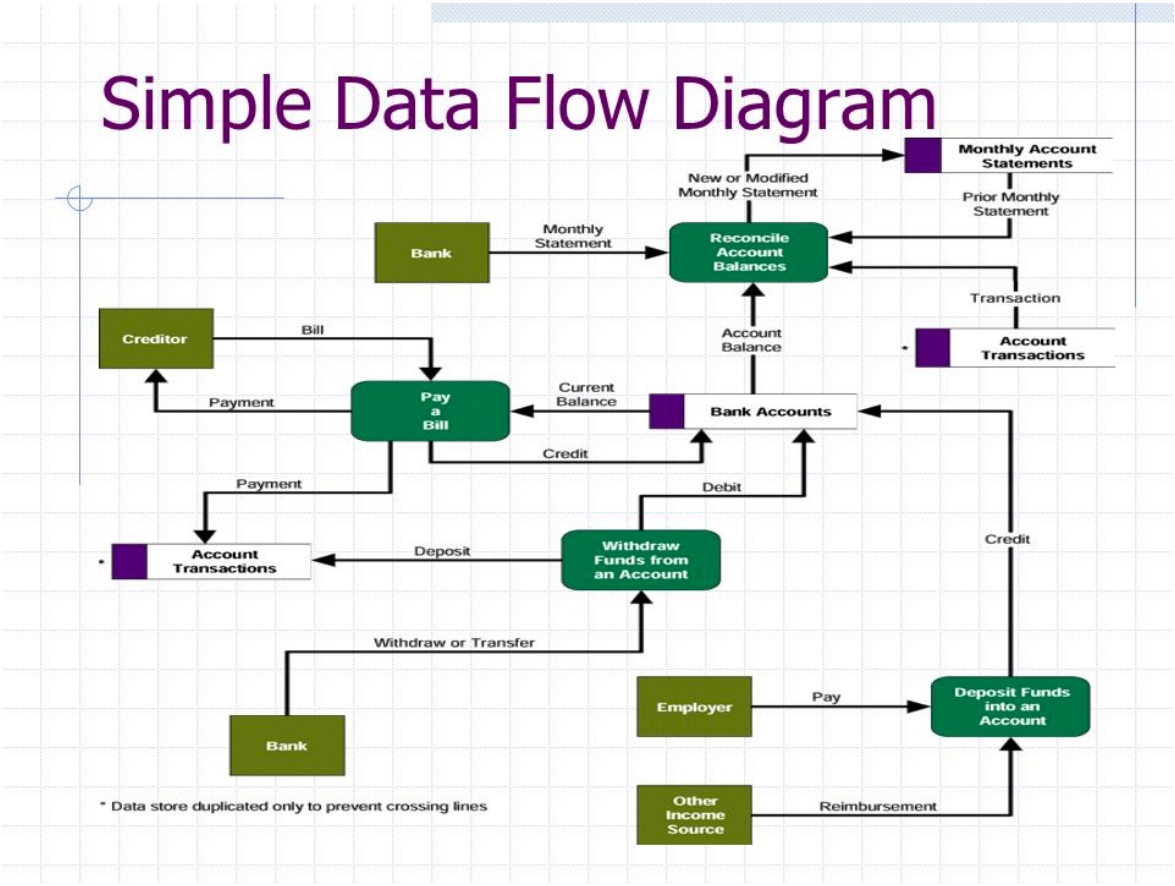
1. Data-Flow models
2. User-based definition of application interfaces
3. Structured text descriptions
4. Other formalized outputs (incl. formally verifiable systems, fast prototypes, etc.)

Outcomes of requirements analysis

1. **Data-Flow models**
2. User-based definition of application interfaces
3. Structured text descriptions



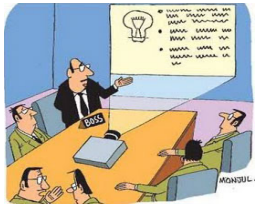
- Used for modeling of processes and software interactions
- For DB design, we focus on the data passed/stored
- Notation understandable to users/clients
- **Advantage**: expressivity
- **Disadvantage**: lack of details



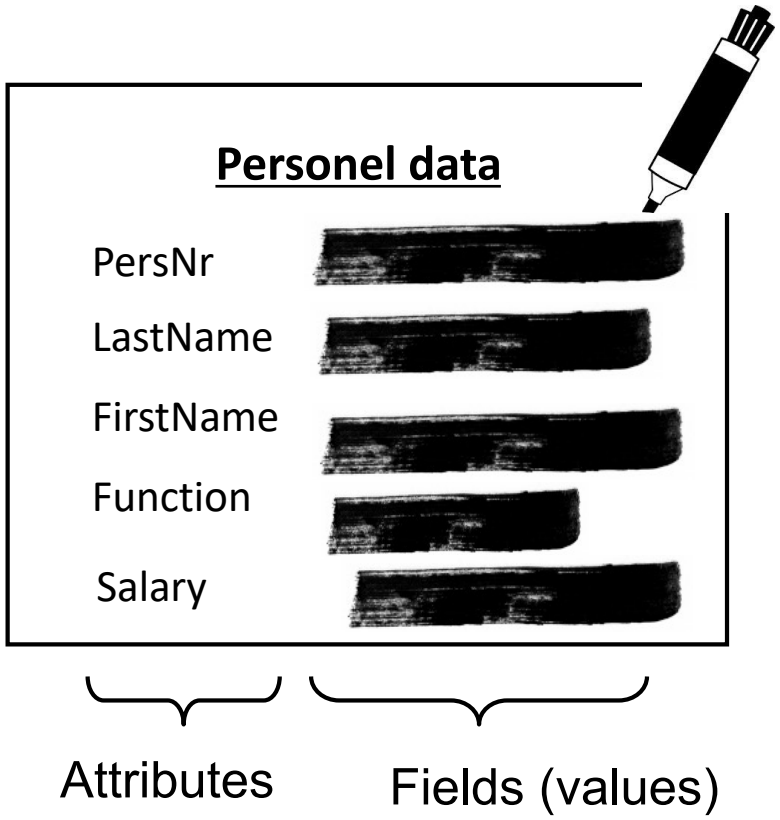
http://images.slideplayer.com/22/6351057/slides/slide_7.jpg

Outcomes of requirements analysis

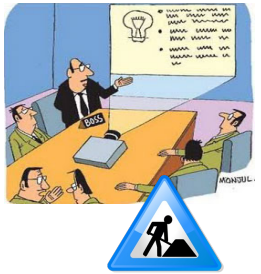
1. Data-Flow models
2. **User-based definition of application interfaces**
3. Structured text descriptions



- End-users design expected user interfaces (mocking)
- Interface between DB and user
- Method leads directly to the first design of DB objects (entity types) and their attributes



Outcomes of requirements analysis



1. Data-Flow models
2. User-based definition of application interfaces
3. **Structured text descriptions**

Example –database
„Company“ see handouts

Applicability: very well defined projects with clearly specified and defined domain (legally defined rules etc. e.g., road vehicle registration database, road rules, etc...)

Text description of the “Company”

1. The company is organized into departments. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department can be spread over multiple locations.

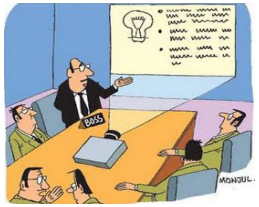
2. Each department controls a number of projects. Each project has a name, number and is located at a single location.

3. We store each employees social security number (AHV-Nr), name, address, salary, sex, and birth- date. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.

4. Each employee may have a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee - for insurance purposes.

Example requirements analysis result

Example of the DB „Company“ managing employees, departments and projects



First result of a requirements analysis

Text description of the “Company”

1. The company is organized into departments. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department can be spread over multiple locations.

2. Each department controls a number of projects. Each project has a name, number and is located at a single location.

3. We store each employees social security number (AHV-Nr), name, address, salary, sex, and birth- date. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.

4. Each employee may have a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee - for insurance purposes.

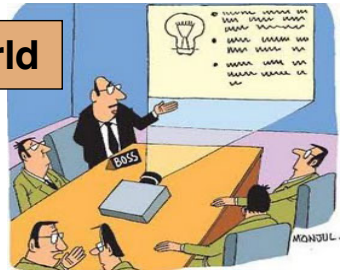
Contents

DB Design

Lecture 3 Requirements Analysis

Interview of users, study of documentation

Real World



Entities in the real world: Department head Müller, Employee Muster, Project 'Budget 2014'

Database Requirements

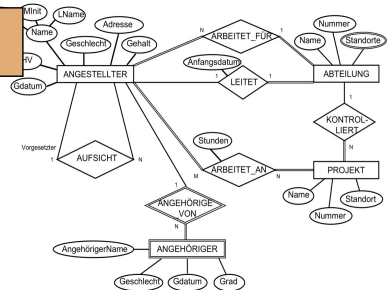
1. Die Firma ist in Abteilungen organisiert. Jede Abteilung hat eine eindeutige Nummer und einen bestimmten Angestellten, [illegible] das Anfangsdatum, ab dem dieser Abteilung übernommen hat. Eine Abteilung verfügt [illegible] Reihe von Projekten, die jeweils einen eindeutigen Namen und einen einzigen Standort haben.
3. Jeder Angestellte wird mit Namen, AHV-Nr., Adresse, Gehalt, Geschlecht und Geburtsdatum gespeichert. Ein Angestellter wird einer Abteilung zugewiesen, kann aber an mehreren Projekten arbeiten, die nicht unbedingt alle von der gleichen Abteilung kontrolliert werden. Wir verfolgen die Stundenzahl pro Woche, die ein Angestellter an jedem Projekt arbeitet, und den unmittelbaren Vorgesetzten jedes Angestellten.
4. Zu Versicherungszwecken sollen die Familienangehörigen jedes Mitarbeiters mit Vorname, Geschlecht, Geburtsdatum und Verwandtschaftsgrad zum jeweiligen Angestellten erfasst werden.

Written concise description of requirements (data reqs and functional [operations] reqs)

Lecture 3/4 Conceptual DB design

Text analysis
Entities
Relationships

ER-Model



Entity types with attributes and relationships

DBMS-independent
DBMS-specific

Lecture 5 Logical DB design (data model mapping)

- translation of model to implementation
- Validation through normalisation

DB-Schema (e.g. relational data model)

ANGESTELLTER

VNAME	INITAL	NNAME	AHV	GDATUM
ADRESSE	GESCHLECHT	GEHALT	SUPERAHV	ANR

ABTEILUNG

ANAME	ABTNUMMMER	MGRAHV	MGR_ANFANGSDATUM

ABT_STANDORTE

ABTNUMMMER	ASTANDORT

ARBEITET_AN

EAHV	PNR	STUNDEN

PROJEKT

PNAME	PNUMMER	PSTANDORT	ABTNR

ANGEHÖRIGER

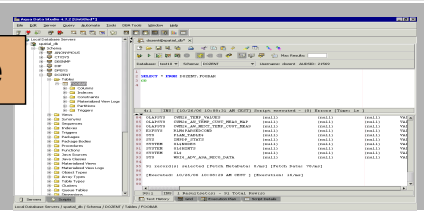
EAHV	ANGEHÖRIGER_NAME	GESCHLECHT	GDATUM	GRAD

Set of relations with attributes (incl. data types + value domains)

Lecture 5 Physical DB design

Description of database implementation (HW, indices, …)

Database



Implemented DB design, Tables (ordered) with rows and columns

➲ Elmasri & Navathe (2014): Ch 7-9

Conceptual design

Identification of the *things* (objects, entity types), their relationships and attributes based on requirements analysis

- Identification and definition of entities
  - *Things* (or objects) abstracted into entity types directly
  - *Attributes* that describe and/or identify these objects
- Description of relationships
  - Relationships between *things* (can also be abstracted into types of relationships)
  - Alternatively, relationships can be initially defined as attributes

- Example.:

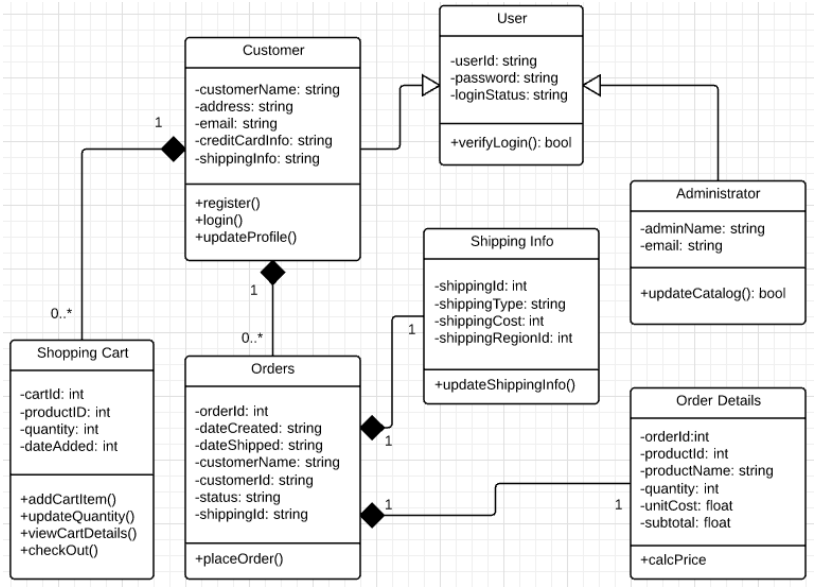
Department
Name, Number, {Locations}, Manager, Manager start date

Formalizations of conceptual models

- Possible formalizations of conceptual models
  - **Entity-Relationship Diagrams (ER-diagrams)**
  - UML (Unified Modeling Language)
  - ...

- ER diagrams - relational DB design
- UML models - ideal for OO (object-oriented) DB design, sometimes OR (object-relational) models

UML class diagram



https://www.lucidchart.com/pages/class-diagram-for-login-page-UML

➔ We will only cover relational DB design – hence ER models.

ER-Diagram: Entity and its attributes

Definition of ENTITY:

A *thing* in the real world with an independent physical or conceptual existence that can be characterized through a set of attributes

- Basic element of ER models;
- Representation of an object that has an independent existence;
- Representation of an object that can exist physically (e.g., cat, Rolf, river);
- ... or conceptually (e.g., Google [a company], GEO 874, love).

Entity and its attributes

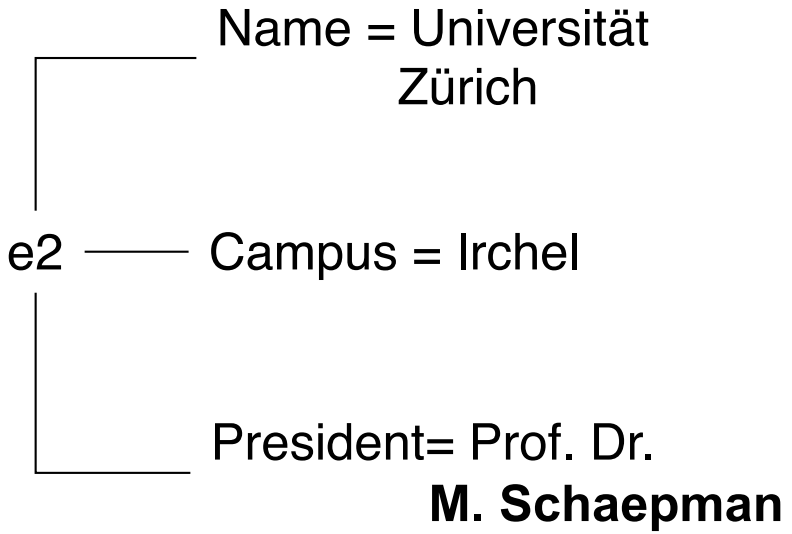
Definition of Attribute:

A particular property that describes an Entity. Attribute consists of a name and data type.

- Each entity is described through its properties, e.g. Employee is described by its Name, Age, Address, Salary and Position.
- Each attribute of an entity has a **value**.

Entity and its attributes

Entity of an employee



Entity representing a University

Contents

Entity types, entity sets

Entity types and entity sets:

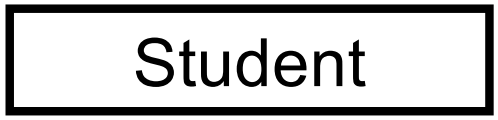
- Databases may contain not only entities, but also groups of similar entities
- An **Entity type** *defines* a set of entities that have the same attributes (as a template), but different attribute values (e.g.: STUDENT)
- An **Entity set** is *the collection* of entities of the same entity type.

>> Entity vs Entity Type

>> Martin vs. Person

>> UZH vs. University

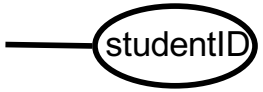
In ER-Diagrams drawn as a rectangle with the name of the Entity type inside

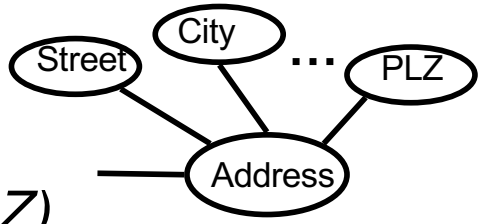


Contents

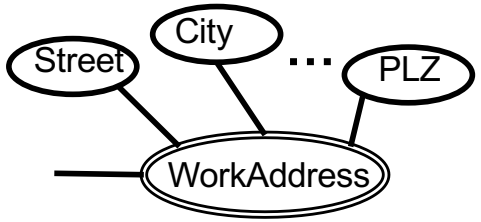
Attribute types

Attribute types:





- Simple vs. composite
  - *E.g.: studentID vs. Address (Street, City, State, PLZ)*
- Single-valued vs. multi-valued
  - *E.g.: studentID vs. Languages spoken (languages)*
- Derived
  - *E.g.: Age computed based on DOB and current date*
- Complex & nested (composite + multi-valued)
  - *E.g.: A Person with multiple Employers has multiple work addresses*



In ER-Diagrams attributes are shown as ellipses, connected to their entity type.

Attributes value domains

- Each simple attribute of an entity type is associated with a **value domain** (also value set)
- Value domains define, which values are valid for a given attribute, e.g.,:
  - Value domain for the attribute **AGE** of the Entity type **Employee**: integers in the range 16-70
  - Value domain for the attribute NAME: alphanumeric characters separated by blank spaces
- Setup of value domain (in SQL): through selection of a data type and definition of constraints
  - E.g., `numeric(10,3)` or `varchar(250)`
  - E.g., `NOT NULL`
- Value domains are not represented in ER Diagrams (conceptual design), but in physical design.

Attributes value domains

Problem: Attributes with values that are

- *Not (yet) known* – e.g., New employee that does not yet have a bank number

or

- Is not *applicable* – e.g. Fax No. for a Person without FAX connection

Practical solution: NULL

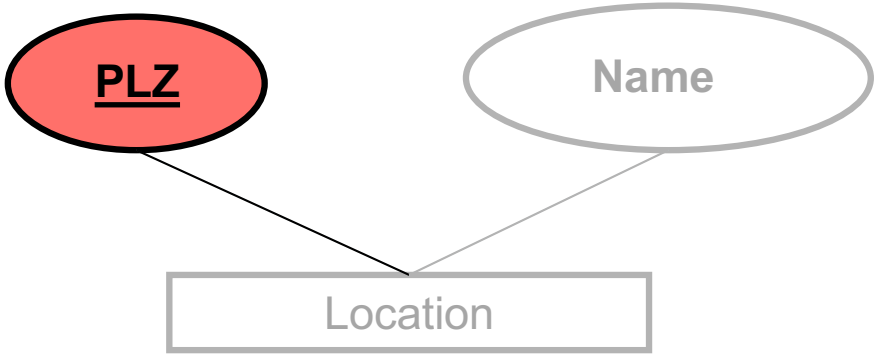
NULL differs from all other values of a domain. It is not comparable (NULL != NULL).

- Not equal to 0 (NULL !=0)
- Not equal to blank spaces (NULL !=“ “)

Key attributes

- **Key**: Each entity type *usually* has an attribute that can be used to identify the entity uniquely (e.g., AHV-Number) – this is a **Key** attribute
- A key can consist of a single attribute or a combination of attributes
- For an entity type, there might exist many possible keys.
- During mapping to the relational model (logical design), one key is picked as a primary key (if a key attribute exists)
- The value of a primary key **can not** be NULL!

- Representation of key in an ER-Diagram: Oval with an **underline** Name



Primary key PLZ

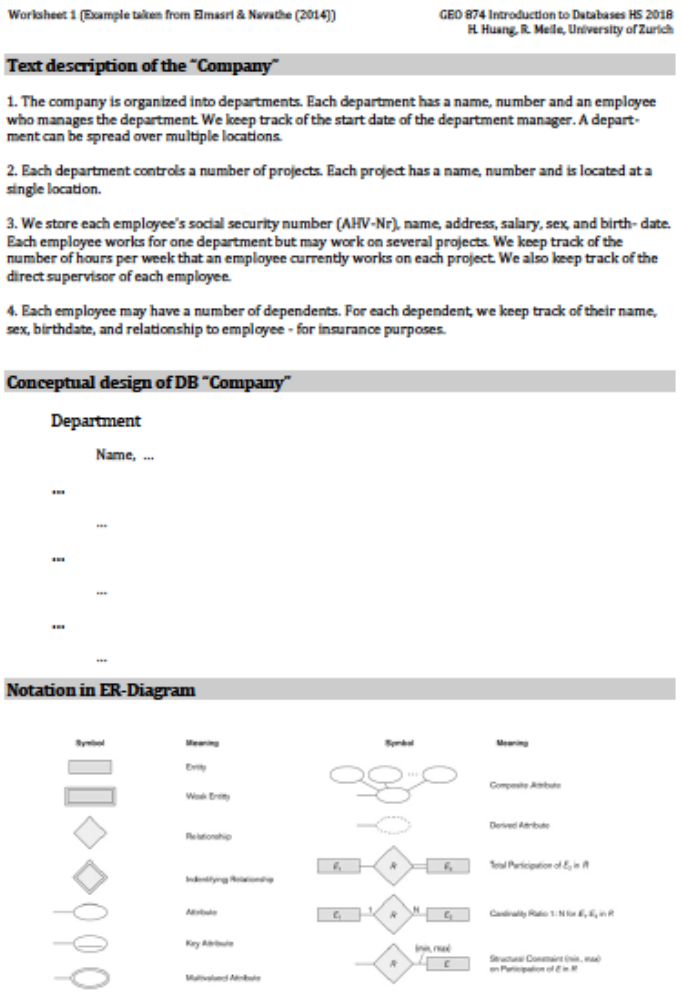
Location

PLZ	Name
6330	Cham
9000	St. Gallen
8002	Zürich
6300	Zug
8400	Winterthur
8003	Zürich
3000	Bern
8001	Zürich
8051	Zürich

Exercise 3.1: Conceptual design of the DB "COMPANY"

Coarse conceptual design based on *Entity types and Attributes*, later refinement through the introduction of relationships (see exercise sheet)

1. Analyze all **NOUNS** in the text (next Slide), they could be Entities or Attributes
2. Analyze all the **VERBS** in the text, they could refer to relationships
3. Analyze all **ADJECTIVES** in the text, they could relate to attributes



Worksheet 1 (Example taken from Elmasri & Navathe (2014)) GEO 874 Introduction to Databases HS 2018
H. Huang, R. Meile, University of Zurich

Text description of the "Company"

1. The company is organized into departments. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department can be spread over multiple locations.

2. Each department controls a number of projects. Each project has a name, number and is located at a single location.

3. We store each employee's social security number (AHV-Nr), name, address, salary, sex, and birth- date. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.

4. Each employee may have a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee - for insurance purposes.

Conceptual design of DB "Company"

Department

Name, ...

...

...

...

...

...

Notation in ER-Diagram

Symbol | Meaning | Symbol | Meaning

Entity

Weak Entity

Relationship

Identifying Relationship

Attribute

Key Attribute

Multivalued Attribute

Composite Attribute

Derived Attribute

Total Participation of E_2 in R

Cardinality Ratio 1: N for E_1, E_2 in R

(min, max)

Structural Constraint (min, max) on Participation of E in R

Exercise 3.1: Conceptual design of the DB "COMPANY"

1. The company is organized into departments. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department can be spread over multiple locations.
2. Each department controls a number of projects. Each project has a name, number and is located at a single location.
3. We store each employee's social security number, name, address, salary, sex, and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.
4. Each employee may have a number of dependents. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.


Entity type


Attribute

Exercise 3.1: Conceptual design of the DB "Company"

Perform a coarse conceptual design based on *Entity types and Attributes (ignoring relationships)*

DEPARTMENT

_____

_____

_____

{} – multivalued attribute

() – composite attribute

Exercise 3.1: Conceptual design of the DB "COMPANY"

1. The company is organized into departments[1]. Each department has a name, number and an employee who manages the department. We keep track of the start date of the department manager. A department can be spread over multiple locations.
2. Each department controls a number of projects[2]. Each project has a name, number and is located at a single location.
3. We store each employee's[3] social security number (AHV-Nr), name address, salary, sex, and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.
4. Each employee may have a number of dependents[4]. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

Entity type Attribute

Exercise 3.1: Conceptual design of the DB "Company"

Perform a coarse conceptual design based on *Entity types and Attributes (ignoring relationships)*

Department

Name, Number, {Location}, Manager, ManagerStartDate

Project

Name, Number, Location, ManagingDepartment

Employee

Name (FName, LName), AHV, Sex, Address, BDate, Dept, Supervisor, {Works on(Project, Hours)}

Dependent

Employee, DependentName, Sex, BDate, Relationship

{} – multivalued attribute

() – composite attribute

Summary 3.1

- SQL Queries
  - View
  - Aggregate functions: Avg(), Max(), Min(), Sum(), Count(), …
  - Group By, Having: often used with aggregate functions
  - Nested queries

- Data Control Language (DCL), and Transaction Control Language (TCL).

Summary 3.2

- Phases of DB design:
  - **Requirements analysis**
  - **Conceptual DB design (part I)**
  - Logical DB design
  - Physical DB design
- Requirements analysis
  - Identifies user groups and application domain
  - Gathers and structures information about application areas
  - Prepares for the conceptual design by identifying objects and attributes
- Intro to Entity-Relationship modeling
  - Entities, entity types, entity sets
  - Attributes and Attribute types

Summary 3.3

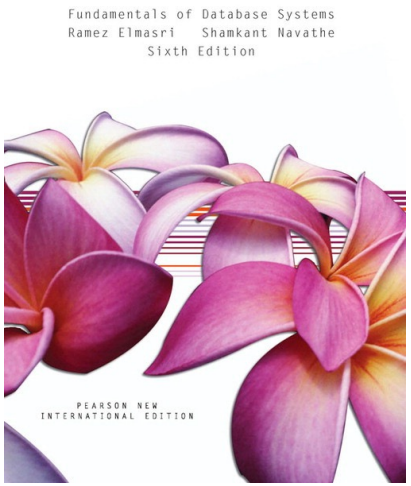
- Differentiable objects in the real world are modeled as **ENTITIES**
- **ENTITY TYPEs** – named templates that are used to summarise the charactersitics of entities. Only entity types are modeled in an ER model.
- Entities can belong to collections of entities with the same characteristics (attributes).
- Different attribute types: simple, composite, multi-valued, Derived, Complex & nest
- The unique identification of an entity amongst other entities of the same type is based on its **key**.

Resources

- Ramez A. Elmasri, Shamkant B. Navathe (2014). Fundamentals of Database Systems, 6th edition, ISBN: 978-1-292-02560-5, Pearson, 1081 pages.

  Ch 7: Data Modelling Using the Entity-Relationship (ER) Model

  Ch 8: The EER Model (optional)



Later, in lab

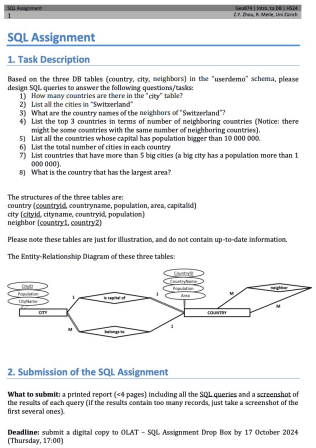
- **10:15-12:00**

- **Y25-J-09, Y25-J-10**

- **SQL demo:** Create, alter, drop, insert, update, delete
- **SQL Exercise**

SQL Assignment

- Based on the three DB tables in the „userdemo“ schema, design SQL queries to answer tasks.
  - More details on: https://www.geo.uzh.ch/microsite/geo874/scripte/geo874_HS25_SQL_Assignment.pdf



SQL Assignment

1. Task Description

Based on the three DB tables (country, city, neighbors) in the "userdemo" schema, please design SQL queries to answer the following questions/tasks:

1) How many countries are there in the "city" table?
2) List all the cities in "Switzerland"
3) What are the country names of the neighbors of "Switzerland"?
4) List the top 3 countries in terms of number of neighboring countries (Notice: there might be some countries with the same number of neighboring countries).
5) List all the countries whose capital has population bigger than 10 000 000.
6) List the total number of cities in each country
7) List countries that have more than 5 big cities (a big city has a population more than 1 000 000).
8) What is the country that has the largest area?

The structures of the three tables are:
country (countryid, countryname, population, area, capitalid)
city (cityid, cityname, countryid, population)
neighbor (country1, country2)

Please note these tables are just for illustration, and do not contain up-to-date information.

The Entity-Relationship Diagram of these three tables:

2. Submission of the SQL Assignment

What to submit: a printed report (<4 pages) including all the SQL queries and a screenshot of the results of each query (if the results contain too many records, just take a screenshot of the first several ones).

Deadline: submit a digital copy to OLAT – SQL Assignment Drop Box by 17 October 2024 (Thursday, 17:00)

- 30% of the final grade
- Individual work

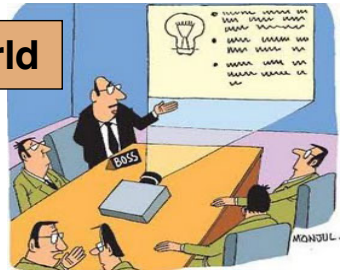
- **What to submit:** a report (<4 pages) including all the SQL queries and a screenshot of each query.
- **How to submit:** submit to OLAT
- **Deadline:** by **16 October 2025** (Thursday, 17:00)

Next week: Conceptual DB design (Cont.)

Lecture 3 Requirements Analysis

Interview of users, study of documentation

Real World



Entities in the real world: Department head Müller, Employee Muster, Project 'Budget 2014'

Lecture 4 Conceptual DB design

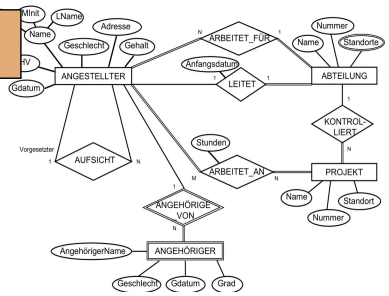
Text analysis
Entities
Relationships

Database Requirements

1. Die Firma ist in Abteilungen organisiert. Jede Abteilung hat eine eindeutige Nummer und einen bestimmten Angestellten, der das Anfangsdatum, ab dem dieser Abteilung übernommen hat. Eine Abteilung verfügt
2. Projekten, die jeweils einen eindeutigen Nummer und einen einzigen Standor haben.
3. Jeder Angestellte wird mit Namen, AHV-Nr., Adresse, Gehalt, Geschlech und Geburtsdatum gespeichert. Ein Angestellter wird einer Abteilung zugewiesen, kann aber an mehreren Projekten arbeiten, die nicht unbedingt alle von der gleichen Abteilung kontrolliert werden. Wir verfolgen die Stundenzahl pro Woche, die ein Angestellter an jedem Projekt arbeitet, und den unmittelbaren Vorgesetzten jedes Angestellten.
4. Zu Versicherungszwecken sollen die Familienangehörigen jedes Mitarbeiters mit Vorname, Geschlecht, Geburtsdatum und Verwandtschaftsgrad zum jeweiligen Angestellten erfasst werden.

Written concise description of requirements (data reqs and functional [operations] reqs)

ER-Model



Entity types with attributes and relationships

DBMS-independent
DBMS-specific

Lecture 5 Logical DB design (data model mapping)

- translation of model to implementation
- Validation through normalisation

DB-Schema (e.g. relational data model)

ANGESTELLTER

VNAME	INITAL	NNAME	AHV	GDATUM
ADRESSE	GESCHLECHT	GEHALT	SUPERAHV	ANR

ABTEILUNG

ANAME	ABTNUMMMER	MGRAHV	MGR_ANFANGSDATUM

ABT_STANDORTE

ABTNUMMMER	ASTANDORT

ARBEITET_AN

EAHV	PNR	STUNDEN

PROJEKT

PNAME	PNUMMER	PSTANDORT	ABTNR

ANGEHÖRIGER

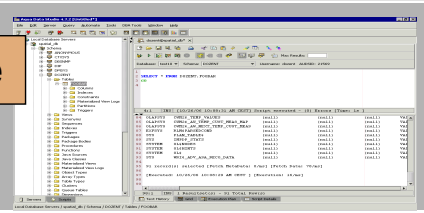
EAHV	ANGEHÖRIGER_NAME	GESCHLECHT	GDATUM	GRAD

Set of relations with attributes (incl. data types + value domains)

Lecture 5 Physical DB design

Description of database implementation (HW, indices, …)

Database



Implemented DB design, Tables (ordered) with rows and columns

➲ Elmasri & Navathe (2014): Ch7-9