1. Motivation

![Diagram](Energy Informatics, WS 2015/16)
Energy Management

- Synchronizing supply and demand
- Prognosis of supply and demand
- Load shifting
- Controlling power generating systems, managing storage devices
- Ancillary services for distributed resources: balancing power, reactive power,...
- Energy efficiency, user behavior

All these tasks are based on data.

Need to

- Model data,
- query and update data,
- react on data changes and model what has to happen,
- construct messages to communicate.
2. Data Modelling

Example

To be able to forecast energy consumption we maintain a database to record the consumption of cities households.

To this end we need data about cities, their buildings, the corresponding households and their energy consumption values taken as readings with respect to periods. To make the data and the relationships between data explicit we develop a model.

The model should show what we need to know about

- cities: Name, Population, Area, Elevation, Latitude, Longitude,
- buildings: Address, Housekeeper
- households: Area
- readings: Date, Value, Fuel.


**Entity-Relationship Diagram (ER-Diagram, ERD)**

- **Entity-Sets** are represented by rectangles, their *properties (attributes)* by ovals, **Relationship-Sets** by diamonds, which are further described by *cardinalities* attached to the connecting lines.

---

**City**
- CityID
- Name
- Population
- Latitude
- Longitude
- Area

**Building**
- BuildingID
- Caretaker
- Address

**Household**
- HouseholdID
- Space

**CityBuilding**
- (1,*)

**Period**
- Year
- Month

**PeriodBuilding**
- (0,*)

**Fuel**
- FuelID
- Type

**FuelReading**
- (1,1)
- Description

**Reading**
- ReadingID
- ReadingValue

**HouseReading**
- (1,1)

**ReadingDate**
- ReadingDate

---

Energy Informatics, WS 2015/16
Entities and Relationships

- **Entities** must be uniquely identifiable by a *key*, i.e. a selected number of attributes graphically indicates by underlining. Typically the key is a certain artificial attribute, e.g. cityID, buildingID, etc. In general it may be defined by more than one attribute, e.g. to identify a building the name of the city and the address would suffice, as well.

- **Relationships** must be uniquely identifiable by the keys of the involved entities. Relationships may be defined over more than two entities, in general. Relationships over relationships are not allowed.
Example

Let \( E \rightarrow R \) be an edge connecting, entity-set \( E \) and relationship-set \( R \) which is labelled by \((\text{min}, \text{max})\), \( \text{min} \leq \text{max} \).

\((\text{min}, \text{max})\) is called cardinality of \( E \) with respect to \( R \).

A cardinality \((\text{min}, \text{max})\) of \( E \) with respect to \( R \) states that each entity \( e \in E \) is involved in at least \( \text{min} \) and at most \( \text{max} \) relationships \( r \in R \).

* used for \( \text{max} \) means arbitrarily many.
Examples.

**recursive relationships**

- An relationship-set is called *recursive*, whenever it is connected to the same relationship-set several times.

For recursive relationship-sets we have to introduce *roles*.

- A relationship-set may be defined over more than two entity-sets.
Decomposition of relationship-sets.

(a) and (b) describe the same world;

(c) and (d) do not.
Example: why decomposition is not (always) allowed!

<table>
<thead>
<tr>
<th>SHT</th>
<th>Supplier</th>
<th>Household</th>
<th>Tariff</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>HouseID</td>
<td>TaID</td>
<td></td>
</tr>
<tr>
<td>Energiedienst</td>
<td>1020</td>
<td>Eco</td>
<td></td>
</tr>
<tr>
<td>Badenova</td>
<td>1020</td>
<td>Maxi</td>
<td></td>
</tr>
<tr>
<td>Badenova</td>
<td>1030</td>
<td>Eco</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S0</th>
<th>Supplier</th>
<th>Tariff</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>TaID</td>
<td></td>
</tr>
<tr>
<td>Energiedienst</td>
<td>Eco</td>
<td></td>
</tr>
<tr>
<td>Badenova</td>
<td>Maxi</td>
<td></td>
</tr>
<tr>
<td>Badenova</td>
<td>Eco</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TH</th>
<th>Tariff</th>
<th>Household</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaID</td>
<td>HouseID</td>
<td></td>
</tr>
<tr>
<td>Eco</td>
<td>1020</td>
<td></td>
</tr>
<tr>
<td>Maxi</td>
<td>1020</td>
<td></td>
</tr>
<tr>
<td>Eco</td>
<td>1030</td>
<td></td>
</tr>
</tbody>
</table>
The same entity-sets may be involved in more than one relationship-set.

(a) and (b) describe different worlds.
3. Data Representation

Example: Mapping ER-Diagrams to Tables (Relations)

Tables for:

- City, with columns for CityID, Name, Population, ...,
- Building, with columns for BuildingID, CityID, Caretaker, Address,
- etc.

Note how we treated relationship-set CityBuilding - it became part of table Building. This does not work for PeriodBuilding! However, works for HouseholdBuilding, HouseReading, FuelReading analogously.
Definition of tables representing the information content modelled by the ER-Diagram

City(CityID, Name, Population Elevation, Lat, Long, Area)
Building(BuildingID, CityID, Caretaker, Address)
Household(HouseID, BuildingID, Area)
Reading(ReadingID, HouseID, Date, Fuel, ReadingValue)
Fuel(FuelID, Type, Description)

Period(PeriodID, Year, Month)
PeriodBuilding(PeriodID, BuildingID, FuelID, Consumption)

The rows of the tables contain the data - each column for each row contains one value.

A system, which is able to process a set of tables, each containing a (very) large number of rows, is called Relational Database System (DBS).

- To store, access and process the data managed by a DBS we need a language: SQL
- To make this working for a large number of users, a DBS provides a layered architecture: the user communicates with a Data Base Management Software (DBMS), which organizes the access to the data efficiently.
- The data typically is stored on disks.

We concentrate on SQL.
Basic Architecture of a DBMS

- **transaction programs and queries**
  - SQL
  - T1
  - T2
  - ... Tn

- **DBMS**
  - transaction management
  - access path operators
  - logical page references
  - DB buffer management
  - DB buffer
  - physical page references
  - storage management
  - disk accesses
  - disk

Energy Informatics, WS 2015/16
http://dbissql.informatik.uni-freiburg.de/dbis/energy/sql.php

There you can work with a database of size:

<table>
<thead>
<tr>
<th>Name of City</th>
<th>Buildings</th>
<th>Households</th>
<th>Readings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freiburg im Breisgau</td>
<td>10877</td>
<td>97061</td>
<td>2329464</td>
</tr>
<tr>
<td>Karlsruhe</td>
<td>14446</td>
<td>128028</td>
<td>3072672</td>
</tr>
<tr>
<td>Kehl</td>
<td>1715</td>
<td>14827</td>
<td>355848</td>
</tr>
<tr>
<td>Stuttgart</td>
<td>29383</td>
<td>262710</td>
<td>6305040</td>
</tr>
<tr>
<td><strong>Σ</strong></td>
<td><strong>56421</strong></td>
<td><strong>502626</strong></td>
<td><strong>12063024</strong></td>
</tr>
</tbody>
</table>

Note: artificial numbers!
### How to create a table:

```
CREATE TABLE City {
    CityID       NUMBER,
    Name         VARCHAR(80),
    Population   NUMBER,
    Area         NUMBER,
    Elevation    NUMBER,
    Latitude     NUMBER,
    Longitude    NUMBER,
    PRIMARY KEY (CityID) ;
}
```

The primary key guarantees unique references to the rows of the table. Alternatively: PRIMARY KEY (Latitude, Longitude).
How to avoid dangling references between tables:

CREATE TABLE Building {
    BuildingID    NUMBER,
    CityID        NUMBER,
    Address       VARCHAR(40),
    Caretaker     VARCHAR(40),
    PRIMARY KEY (BuildingID),
    FOREIGN KEY (CityID) REFERENCES City (CityID) ;
}

The references clause guarantees that there will be no tuples in relation Building, for which the referenced city does not exist in table City.
The referential integrity is guaranteed. Later discussed: referential actions.
### How to pose simple queries to a table:

- **Give me all rows of a table.**
  
  ```sql
  SELECT * FROM City;
  ```

- **Give me for all rows only the values of certain columns of a table.**
  
  ```sql
  SELECT CityID, Name, Area FROM City;
  ```

- **Give me all (column values of) rows of a table which fulfill certain conditions.**
  
  ```sql
  SELECT CityID, Name, Area FROM City WHERE Area > 500;
  ```

- **Give me all cities which are 'near' to Freiburg.**
  
  ```sql
  SELECT CityID, Name FROM City WHERE ????????????;
  ```
How to combine (join) tables:

- Give me a listing of building addresses with name of the city.
  
  ```sql
  SELECT City.Name, Building.Address FROM City, Building
  WHERE City.CityID = Building.CityID;
  ```

- Compute all pairs of cities.
  
  ```sql
  SELECT A.Name, B.Name FROM City A, City B
  WHERE A.CityID <> B.CityID;
  ```

- Give me all cities which are 'near' to Kehl.
  
  ```sql
  SELECT B.Name FROM City A, City B
  WHERE A.Name = 'Kehl' AND ABS(A.Latitude - B.Latitude) < 0.5
  AND B.Name <> 'Kehl';
  ```

- Give me all the readings of Kehl nicely sorted.
  
  ```sql
  SELECT D.HouseholdID, D.Fuel, D.ReadingValue
  FROM City A, Building B, Household C, Reading D
  WHERE A.Name = 'Kehl' AND A.CityID = B.CityID
  AND B.BuildingID = C.BuildingID AND C.HouseholdID = D.HouseholdID
  ORDER BY D.HouseholdID ASC, D.Fuel ASC, D.ReadingValue DESC;
  ```