Databases and Data Warehouses

Lecture BigData Analytics

Julian M. Kunkel

julian.kunkel@googlemail.com

University of Hamburg / German Climate Computing Center (DKRZ)

23-10-2015

Outline

Relational Model [10]

- Database model based on first-order predicate logic
  - Theoretic foundations: relational algebra and calculus
- Data is represented as tuples
- Relation/Table: groups similar tuples
  - Table consists of rows and named columns (attributes)
  - No duplicates of complete rows allowed
- No support for collections in tuples
- Schema: specify structure of tables
  - Datatypes (domain of attributes)
  - Organization and optimizations
  - Consistency via constraints

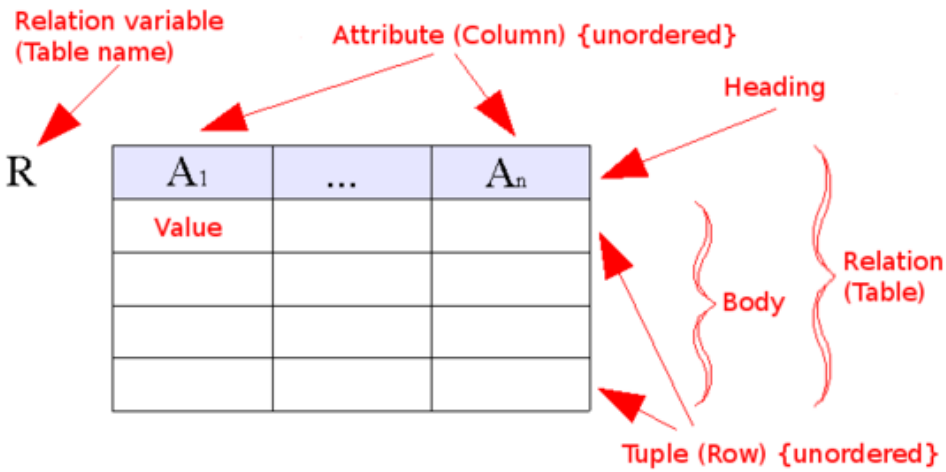

Figure: Source: Relational model concepts [11]

Example Schema for our Students Data

Description

Database for information about students and lectures

Relational model

Matrikel	Name	Birthday
242	Hans	22.04.1955
245	Fritz	24.05.1995

Table: Student table

ID	Name
1	Big Data Analytics
2	Hochleistungsrechnen

Table: Lecture table

Matrikel	LectureID
242	1
242	2
245	2

Table: Attends table representing a relation

Relationships

- Model relationships between data entities
- Cardinality defines how many entities are related
- Relevant relationships are
  - One-to-many: One entity of type A with many entities of type B
  - Many-to-many: One-to-many in both directions
  - One-to-one: One entity of type A with at most one entity of type B
- Relationships can be expressed with additional columns
  - Packing data of entities together in the table
  - Alternatively: provide a “reference” to other tables

Matrikel	Name	Birthday	Lecture ID	Lecture Name
242	Hans	22.04.1955	1	Big Data Analytics
242	Hans	22.04.1955	2	Hochleistungsrechnen
245	Fritz	24.05.1995	2	Hochleistungsrechnen

Table: Student table with attended lecture information embedded

Entity Relationship Diagrams

- Illustrate the relational model and partly the database schema
- Elements: Entity, relations, attributes
  - Additional information about them e.g. cardinality, data types

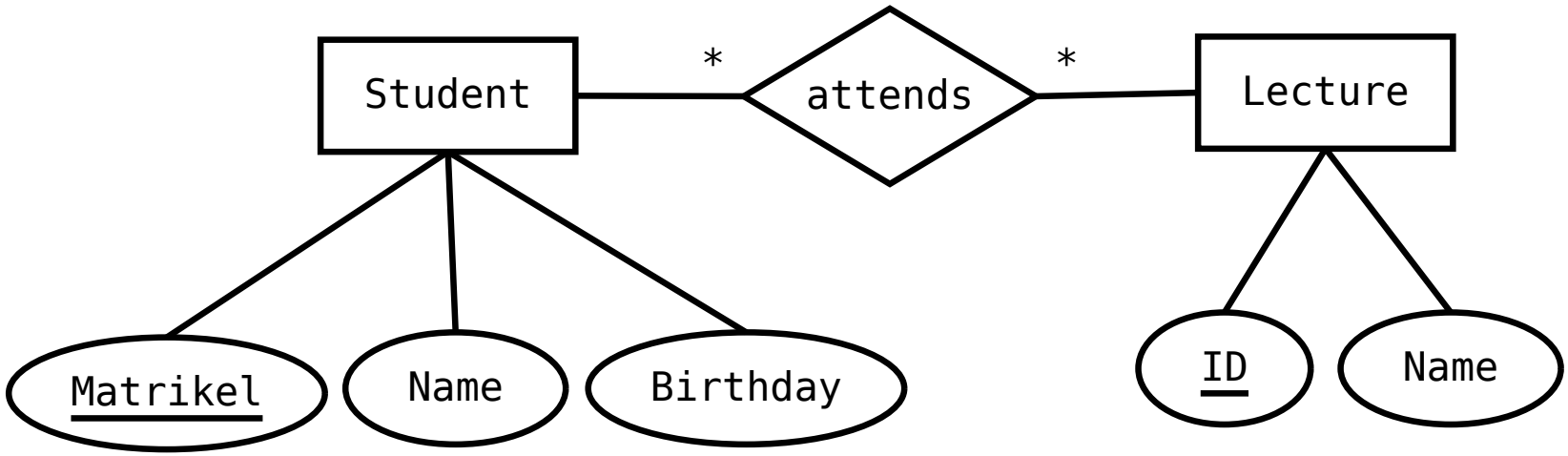

Figure: A student/lecture example in modified Chen notation, * is the cardinality and means any number is fine

Keys [16, 17, 18]

- A Superkey[1] allows addressing specific tuples in a table
- Superkey: Set of attributes that identify each tuple in a table
  - There is at most one tuple for each possible key value
  - A superkey does not have to be minimal
    - e.g. all columns of a table are a Superkey
    - After removing an attribute it can still be a key
  - Simple key: key is only one attribute
  - Compound key: consists of at least two attributes
- Candidate key: a minimal key i.e. no attribute can be removed
- Primary key: the selected candidate key for a table
- Foreign key: inherited key of another table
- Natural key: key that naturally is unique, e.g. matrikel
- Surrogate key: artificial key, e.g. numeric ID for a row

[1] Often it is just called key

Example Keys

Table: Student table

Matrikel	Name	Birthday	...
242	Hans	22.04.1955	
245	Fritz	24.05.1995	

Table: Lecture table

ID	Name	Semester
1	Big Data Analytics	SS15
2	Hochleistungsrechnen	WS1516

Table: Attends table representing a relation

Matrikel	LectureID
242	1
242	2
245	2

- Student table
  - Candidate keys: Matrikel, (name, birthday, city), social insurance
  - Primary key: Matrikel (also a natural key)
- Lecture table
  - Candidate keys: ID, (Name, Semester)
  - Primary key: ID (also a Surrogate Key)
- Attends table
  - Candidate key: (Matrikel, Lecture ID)
  - Primary key: (Matrikel, Lecture ID)

Normalization: My Simplified Perspective [10]

- Normalization is the process of organizing the columns and tables to minimize redundancy [19]
  - Reduces dependencies
  - Prevents inconsistency across replicated information
  - Normally, reduces required storage space and speeds up updates
- There are different normal forms with increasing requirements
  - 1NF: It follows our notion of a table.
    - No collections in the table. A primary key exists.
  - 2NF: No redundancy of data
    - I.e. entities of many-to-many relations are stored in separate tables
    - Every column must depend on each candidate key and not a subset
  - 3NF: Columns are not functional dependent to sth. else than a candidate key
  - 4NF: Do not store multiple relationships in one table
  - 4NF is a good choice[2]

[2] It has been shown that 4NF can always be achieved for relational data

Example for Unnormalized Data

Matrikel	**Name**	Birthday	**Name**
242	Hans	22.04.1955	**[Big Data Analytics, Hochleistungsrechnen]**
245	Fritz	24.05.1995	Hochleistungsrechnen

Table: Not normalized Student and lecture table/relation, contains identical column names and collections

Matrikel	Name	Birthday	Lecture Name
242	Hans	22.04.1955	Big Data Analytics
242	Hans	22.04.1955	**Hochleistungsrechnen**
245	Fritz	24.05.1995	**Hochleistungsrechnen**

Table: Student and lecture table/relation in 1NF, it contains a many-to-many relation. Changing lecture name requires to touch multiple rows

Example for Unnormalized Data

Matrikel	Name	Birthday	**Age**
242	Hans	22.04.1955	40
245	Fritz	24.05.1995	20

Table: In 2NF but not 3NF: Age is functional depending on birthday

Matrikel	Attended lecture	Attended seminar
242	BDA	SIW
242	HR	SIW
242	BDA	NTH
242	HR	NTH

Table: In 3NF but not 4NF: Candidate key depends on all three columns

Databases [29]

- **Database**: an organized collection of data
  - Includes layout (schemes), queries, views
  - Database models: Relational, graph, document, ...
- **Database management system (DBMS)**: software application that interacts with the user, other applications and the database itself to capture and analyze data [29]
  - Definition, creation, update, querying and administration of databases

DBMS functions for managing databases

- Data definition: Creation, modification of definitions for data organization
- Update: Insertion, modification and deletion of data
- Query/Retrieval: retrieving stored and computing derived data
- Administration: user management, security, monitoring, data integrity, recovery

Structured Query Language (SQL) [20]

- Declarative language: specify what to achieve and not how
- Evolving standard that includes more and more features

Language elements

- Query: retrieve (and compute) data based on criteria
- Statement: instructions to perform, terminate by ;
- Clause: components of statements
- Predicates: conditions limiting the affected rows/columns
- Expressions: produce scalar values or tables
- Operators: compare values, change column names
- Functions: transform/compute values

PostgreSQL [10]

A popular database implementation

- Semantics: ACID support for transactions
  - A transaction is a batch of operations
  - It either fails or succeeds
- Implements majority of SQL:2011 standard
- Note that syntax may differ from the standard
- Typically databases offer extensions to the standard
- Interactive shell via `psql`

Excerpt of features

- Materialized views (create virtual tables from logical tables)
- Fulltext search
- Regular expression
- Statistics and histograms
- User defined objects (functions, operators)
- Triggers: events upon insert or update statements; may invoke functions
- New versions support semi-structed data in arrays, XML, JSON[3]

Schemas (in Postgres)

Creation of our database and table

```
1 CREATE ROLE "bigdata" NOSUPERUSER LOGIN PASSWORD 'mybigdata';
2 CREATE DATABASE bigdata OWNER "bigdata";
```

To connect to the database use `psql -W -U BigData bd`

Create our tables

```
1 CREATE TABLE students (matrikel INT, name VARCHAR, birthday DATE, PRIMARY KEY(matrikel));
2 CREATE TABLE lectures (id SERIAL, name VARCHAR, PRIMARY KEY(id));
3 CREATE TABLE attends  (matrikel INT, lid INT,
4   FOREIGN KEY (matrikel) REFERENCES students(matrikel),
5   FOREIGN KEY (lid) REFERENCES lectures(id));
6 --\d <TABLE>  prints the schema
```

Additional constraints

```
1 -- minimum length of the name shall be 5
2 ALTER TABLE students ADD CONSTRAINT length CHECK (char_length(name) > 3);
3 -- to remove the constraint later: ALTER TABLE students DROP CONSTRAINT length ;
4 -- minimum age of students should be 10 years
5 ALTER TABLE students ADD CONSTRAINT age CHECK (extract('year' from age(birthday)) > 10);
6 -- disallow NULL values in students
7 ALTER TABLE students ALTER COLUMN birthday SET NOT NULL; -- during CREATE with "birthday DATE NOT NULL"
8 ALTER TABLE students ALTER COLUMN name SET NOT NULL;
```

Populating the Tables

```
1  -- Explicit specification of columns, not defined values are NULL
2  INSERT INTO students (matrikel, name, birthday)
3    VALUES (242, 'Hans', '22.04.1955');
4  -- This should be prevented using a constraint
5  INSERT INTO students (matrikel, name) VALUES (246, 'Hans');
6  -- Order is expected to match the columns in the table
7  INSERT INTO students VALUES (245, 'Fritz', '24.05.1995');
8  INSERT INTO lectures VALUES (1, 'Big Data Analytics');
9  INSERT INTO lectures VALUES (2, 'Hochleistungsrechnen');
10
11 -- Populate relation
12 INSERT into attends VALUES(242, 1);
13 INSERT into attends VALUES(242, 2);
14 INSERT into attends VALUES(245, 2);
15
16 -- Insertations that will fail due to table constraints:
17 INSERT INTO students (matrikel, name) VALUES (250, 'Hans');
18 -- ERROR:  null value in column "birthday" violates not-null constraint
19 INSERT INTO students VALUES (250, 'Hans', '22.04.2009');
20 -- ERROR:  new row for relation "students" violates check constraint "age"
21 INSERT INTO students VALUES (245, 'Fritz', '24.05.1995');
22 -- ERROR:  duplicate key value violates unique constraint "students_pkey"
23 -- DETAIL:  Key (matrikel)=(245) already exists.
```

Queries [20]

- A query retrieves/computes a (sub)table from tables
- It does not change/mutate any content of existing tables
- Statement: SELECT $< column1 >, < column2 >, ...$
- Subqueries: nesting of queries is possible

Supported clauses

- FROM: specify the table(s) to retrieve data
- WHERE: filter rows returned
- GROUP BY: group rows together that match conditions
- HAVING: filters grouped rows
- ORDER BY: sort the columns

```
1 SELECT Matrikel, Name FROM students
2 WHERE Birthday='22.04.1955';
3 -- Returns a table with one row:
4 --  matrikel | name
5 -- ----------+------
6 --       242 | Hans
```

More Queries

Ordering of tables

```
1 -- Example comment, alternatively /* */
2 select * from students
3   where (name != 'fritz' and name != 'nena') -- two constraints
4   order by name desc; -- descending sorting order
```

Aggregation functions

```
1 -- There are several aggregate functions such as max, min, sum, avg
2 select max(birthday) from students;
3 -- 1995-05-24
4
5 -- It is not valid to combine reductions with non-reduced columns e.g.
6 select matrikel, max(birthday) from students; -- ERROR!
```

Counting the number of students

```
1 -- Number of students in the table and rename the column to number
2 SELECT count(*) AS number FROM students;
3 -- number
4 --      2
```

Subqueries

A subquery creates a new (virtual) named table to be accessed

Identify the average age

```
1 -- Identify the min, max and average age, therefore, we create a new table and convert
      ↪ the date
2 select min(age), avg(age), max(age) from
3     -- Here we create the virtual table with the name ageTbl
4     (SELECT age(birthday) as age from students) as ageTbl;
5 --            min            |                avg                 |          max
6 -- 20 years 3 mons 30 days | 40 years 4 mons 15 days 12:00:00 | 60 years ...
```

Identify students which are not attending any course

```
1 -- We use a subquery and comparison with the set
2 select matrikel from students
3 where matrikel not in  -- compare a value with entries in a column
4   (select matrikel from attends);
```

Subquery expressions: exists, in, some, all, (operators, e.g. $<$) [4]

[4] See http://www.postgresql.org/docs/9.4/static/functions-subquery.html

Grouping of Data

Data can be grouped by one or multiple (virtual) columns. It is not valid to include non-grouped or non-reduced values

Identify students with the same name and birthday, count them

```
1 select name, birthday, count(*) from students group by name, birthday;
2 -- name  |    max     | count
3 ---------+------------+-------
4 -- Fritz | 1995-05-24 |     1
5 -- Hans  | 1955-04-22 |     1
```

Figure out the number of people starting with the same letter

```
1 select upper(substr(name,1,1)) as firstletter, count(*) from students
2    group by firstletter;
3 -- firstletter | count
4 ---------------+------------
5 -- F           | 1
6 -- H           | 1
```

Filtering Groups of Data

- With the HAVING clause, groups can be filtered
- ORDER BY is the last clause and can be applied to aggregates

Identify students with the same name and birthday, and return the total number of non-“duplicates”

```
1  select sum(mcount) from
2    (select count(*) as mcount from students
3     group by name, birthday having count(*) = 1 order by count(*)) as groupCount;
4  -- sum
5  -- 2
6
7  -- Alternatively in a subquery you can use:
8  select sum(count) from
9    (select count(*) as count from students
10    group by name, birthday) as groupCount
11    where count = 1;
```

Joins [10]

A join combines records from multiple tables

- Usually filtering tuples according to a condition
- Used to resolve relations of entities in normalized schemes

Types of joins

- CROSS JOIN: Cartesian product of two tables
- NATURAL JOIN: All combinations that are equal on their common attributes
- INNER JOIN: Return all rows that have matching records based on a condition
- OUTER JOIN: Return all rows of both tables even if they are not matching the condition
  - LEFT OUTER JOIN: Return all combinations and all tuples from the left table
  - RIGHT OUTER JOIN: ... from the right table
  - FULL OUTER JOIN: Return all combinations

Example Joins

```
1  select * from students as s1 CROSS JOIN students as s2;
2  -- matrikel | name  |  birthday  | matrikel | name  |  birthday
3  ------------+-------+------------+----------+-------+------------
4  --       242 | Hans  | 1955-04-22 |      242 | Hans  | 1955-04-22
5  --       242 | Hans  | 1955-04-22 |      245 | Fritz | 1995-05-24
6  --       245 | Fritz | 1995-05-24 |      242 | Hans  | 1955-04-22
7  --       245 | Fritz | 1995-05-24 |      245 | Fritz | 1995-05-24
8
9  select * from students  NATURAL JOIN attends;
10 -- matrikel | name  |  birthday  | lid
11 -- ----------+-------+------------+-----
12 --       242 | Hans  | 1955-04-22 |   1
13 --       242 | Hans  | 1955-04-22 |   2
14 --       245 | Fritz | 1995-05-24 |   2
15
16 select * from students INNER JOIN attends ON students.matrikel = attends.matrikel;
17 -- matrikel | name  |  birthday  | matrikel | lid
18 ------------+-------+------------+----------+-----
19 --       242 | Hans  | 1955-04-22 |      242 |   1
20 --       242 | Hans  | 1955-04-22 |      242 |   2
21 --       245 | Fritz | 1995-05-24 |      245 |   2
```

Example Joins

```
1  -- This join returns NULL values for Fritz as he has not the selected matrikel
2  select * from students LEFT OUTER JOIN attends ON students.matrikel = 242;
3  -- matrikel | name  |  birthday  | matrikel | lid
4  ----------+-------+------------+----------+-----
5  --       242 | Hans  | 1955-04-22 |      242 |   1
6  --       242 | Hans  | 1955-04-22 |      242 |   2
7  --       242 | Hans  | 1955-04-22 |      245 |   2
8  --       245 | Fritz | 1995-05-24 |          |
9
10 select * from students as s FULL OUTER JOIN attends as a ON s.matrikel = a.lid;
11 -- matrikel | name  |  birthday  | matrikel | lid
12 ------------+-------+------------+----------+-----
13 --          |       |            |      242 |   1
14 --          |       |            |      242 |   2
15 --          |       |            |      245 |   2
16 --       242 | Hans  | 1955-04-22 |          |
17 --       245 | Fritz | 1995-05-24 |          |
18
19 -- Now identify all lectures attended by Hans
20 select s.name, l.name from students as s INNER JOIN attends as a ON s.matrikel
       ↪ = a.matrikel INNER JOIN lectures as l ON a.lid=l.id;
21 -- name  |         name
22 ---------+----------------------
23 -- Hans  | Big Data Analytics
24 -- Hans  | Hochleistungsrechnen
25 -- Fritz | Hochleistungsrechnen
```

Views

- View: virtual table based on a query
  - Can be used to re-compute complex dependencies/apply joins
  - The query is evaluated at runtime, which may be costly
- Materialized view: copies data when it is created/updated[5]
  - Better performance for complex queries
  - Suitable for data analytics of data analysts
  - Export views with permissions and reduce knowledge of schema

```
1  CREATE  VIEW studentsView AS
2    SELECT s.matrikel, s.name as studentName, l.name as lectureName, age(birthday) as age
         ↪ from students as s INNER JOIN attends as a ON s.matrikel = a.matrikel INNER
         ↪ JOIN lectures as l ON a.lid=l.id;
3
4  select * from studentsView;
5  -- matrikel | studentname |     lecturename      |           age
6  ------------+-------------+----------------------+------------------------
7  --       242 | Hans        | Big Data Analytics   | 60 years 5 mons 1 day
8  --       242 | Hans        | Hochleistungsrechnen | 60 years 5 mons 1 day
9  --       245 | Fritz       | Hochleistungsrechnen | 20 years 3 mons 30 days
10 -- To replace the data with new data
11 REFRESH MATERIALIZED VIEW studentsView;
```

[5] www.postgresql.org/docs/9.4/static/sql-creatematerializedview.html

Regular Expressions

- PostgreSQL supports several styles of regular expressions[6]
- We will look at POSIX regular expressions (regex)
- Operator: ~for matching and ~* for not matching
- `regexp_matches(string, pattern)` returns text array with all matches

Examples

```
1  -- Any lecture which name contains Data
2  select name from lectures where name~*'data';
3  -- Big Data Analytics
4
5  -- Lectures starting with Big
6  select name from lectures where name~'^Big.*$';
7  -- Big Data Analytics
8
9  -- Students whose name contain at least two vocals
10 select name from students where name~'(i|a|o|u).*(a|i|o|u)';
11
12 -- Students whose name contain at least one vacal and at most three
13 select name from students where name~'^([^auiu]*(i|a|o|u)[^aiou]*){1,3}$';
14
15 -- Retrieve all lower case letters in the names
16 select regexp_matches(name, '[a-z]', 'g') as letter from students;
17 -- {a}, {n} ...
```

Array Operations

- Operations allow manipulation of multidimensional arrays[7]
- Useful operators: `unnest`, `array_agg`, `array_length`
- JSON support in new postgres version (not discussed here)

```
1  -- Alternative schema for our student/lecture example using an array for the attends relationship
2  CREATE TABLE studentsA (matrikel INT, name VARCHAR, birthday DATE, attends INT[], PRIMARY KEY(matrikel));
3  CREATE TABLE lectures (id SERIAL, name VARCHAR, PRIMARY KEY(id));
4
5  INSERT INTO studentsA VALUES (242, 'Hans', '22.04.1955', '{1,2}' );
6  INSERT INTO studentsA VALUES (245, 'Fritz', '24.05.1995', '{2}');
7
8  -- Addressing array elements: first lecture attended by each student
9  SELECT attends[1] from studentsA;
10 -- Slicing is supported: First three lectures
11 SELECT attends[1:3] from studentsA;
12
13 -- Retrieve the lecture name attended for each student
14 SELECT s.name, l.name from studentsA AS s INNER JOIN lectures AS l ON l.id = ANY(s.attends);
15 -- Hans  | Big Data Analytics
16 -- Hans  | Hochleistungsrechnen
17 -- Fritz | Hochleistungsrechnen
18
19 -- Now retrieve the lectures in an array per person
20 SELECT s.name, array_agg(l.name) from studentsA AS s INNER JOIN lectures AS l ON l.id = ANY(s.attends)  GROUP by s.matrikel;
21 -- Hans  | {"Big Data Analytics",Hochleistungsrechnen}
22 -- Fritz | {Hochleistungsrechnen}
```

[7] See http://www.postgresql.org/docs/9.4/static/arrays.html

Updating Rows

- UPDATE statement allows changes values
- DELETE statement removes rows
- Each individual operation follows the ACID semantics[8]
- Transactions allow to batch operations together

```
1 -- Change the name of Fritz
2 UPDATE students SET name='Fritzchen' WHERE matrikel=245;
3
4 -- Remove Fritzchens attendance in Hochleistungsrechnen
5 DELETE FROM attends WHERE matrikel=242 and lid=2;
6
7 -- Subqueries can be used to select rows that are updated/deleted
8 -- Remove Fritzchen attendence with the name
9 DELETE from attends WHERE matrikel=242 and lid = (SELECT id from lectures where name =
      ↪ 'Hochleistungsrechnen');
```

[8] In fact, when AUTOCOMMIT is enabled, every statement is wrapped in a transaction. To change this behavior on the shell, invoke: SET AUTOCOMMIT [OFF|ON]

Transactions

- Transaction: A sequence of operations executed with ACID semantics
  - It either succeeds and becomes visible and durable; or it fails
  - Note: Complex data dependencies of concurrent operations may create a unresolvable state that require restart
- All queries access data in the version when it started
  - The isolation level can be relaxed, e.g. see committed or uncommited changes
- Internally, complex locking schemes ensure conflict detection

Example: Atomic money transfer between bank accounts

```
1 START TRANSACTION;
2 UPDATE account SET balance=balance-1000.40 WHERE account=4711;
3 UPDATE account SET balance=balance+1000.40 WHERE account=5522;
4
5 -- if anything failed, revert to the original state
6 IF ERRORS=0 COMMIT; -- make the changes durable
7 IF ERRORS!=0 ROLLBACK; -- revert
```

Performance Aspects

Indexes

- Full scan: when searching for a variable with a condition e.g. x=y, the table data needs to be read completely
- Index allows lookup of rows for which a condition (likely) holds
- Postgres supports B-tree, hash, GiST, SP-GiST and GIN indexes[9]

```
1  CREATE INDEX ON students (name);
```

Optimizing the execution of operations (query plan)

- Postgres uses several methods to optimize the query plan
- The query planer utilizes statistics about access costs
  - Knowing how values are distributed helps optimizing access
- ANALYZE statement triggers collection of statistics
- Alternatively: automatically collect statistics
- EXPLAIN statement: describes the query plan

[9] See http://www.postgresql.org/docs/9.4/static/sql-createindex.html

Performance Aspects (2) [22]

Bulk Loads/Restores

- Combine several INSERTS into one transaction
- Perform periodic commits
- Create indexes/foreign key/constraints after data was inserted

Vacuuming: Cleaning empty space

- When changing or inserting rows additional space is needed
- It is expensive to identify empty rows and compact them
  - $\Rightarrow$ Just append new data
    - Mark data e.g. in a bitmap as outdated
- Periodically space is reclaimed and data structures are cleaned
- VACCUUM statement also triggers cleanup
- ANALYZE also estimates the amount of garbage to optimize queries

Data Warehouse

"A data warehouse (DW or DWH), also known as an enterprise data warehouse (EDW), is a system used for reporting and data analysis." [27]

- Central repository
- Integrates data from multiple inhomogeneous sources
- Provides tools for the business analyst in descriptive analysis
- Many queries are executed periodically and used in reports
- May provide some tools for predictive analysis
- Data analysts use a simplified data model: a multidimensional data cube

Databases vs. Big Data

Database management systems (DBMS)

- Standardized systems and methods to process structured data
- Use the relational model for data representation
- Use SQL for processing

Online Transaction Processing (OLTP)

- Real-time processing
- Offer ACID qualities
- Relies on normalized schemes (avoid redundant information)

Online Analytical Processing (OLAP)

- Systems and methods to analyze large quantities of data
- Utilizes data warehouses with non-normalized schemes
- Extract, Transform and Load (ETL): import data from OLTP

OLAP

- Online analytical process with large quantities of business data
- Utilizes denormalized dimensional model to avoid costly joins
- Technology alternatives:
  - MOLAP (Multidimensional OLAP): problem-specific solution
  - With relational databases (ROLAP)
    - Star schema
    - Snowflake schema
- Dimensional modeling: design techniques and concepts [26]
  1. Choose the business process e.g. sales situation
  2. Declare the grain: what does the model focus on e.g. item purchased
  3. Identify the dimensions
  4. Identify the facts

The OLAP Cube: Typical Operations [27]

- Slice: Fix one value to reduce the dimension by one
- Dice: Pick specific values of multiple operations
- Roll-up: Summarize data along a dimension
  - Formulas can be applied, e.g. profit = income - expense
- Pivot: Rotate the cube to see the faces

The OLAP Cube: Slice [27]

- Slice: Fix one value to reduce the dimension by one

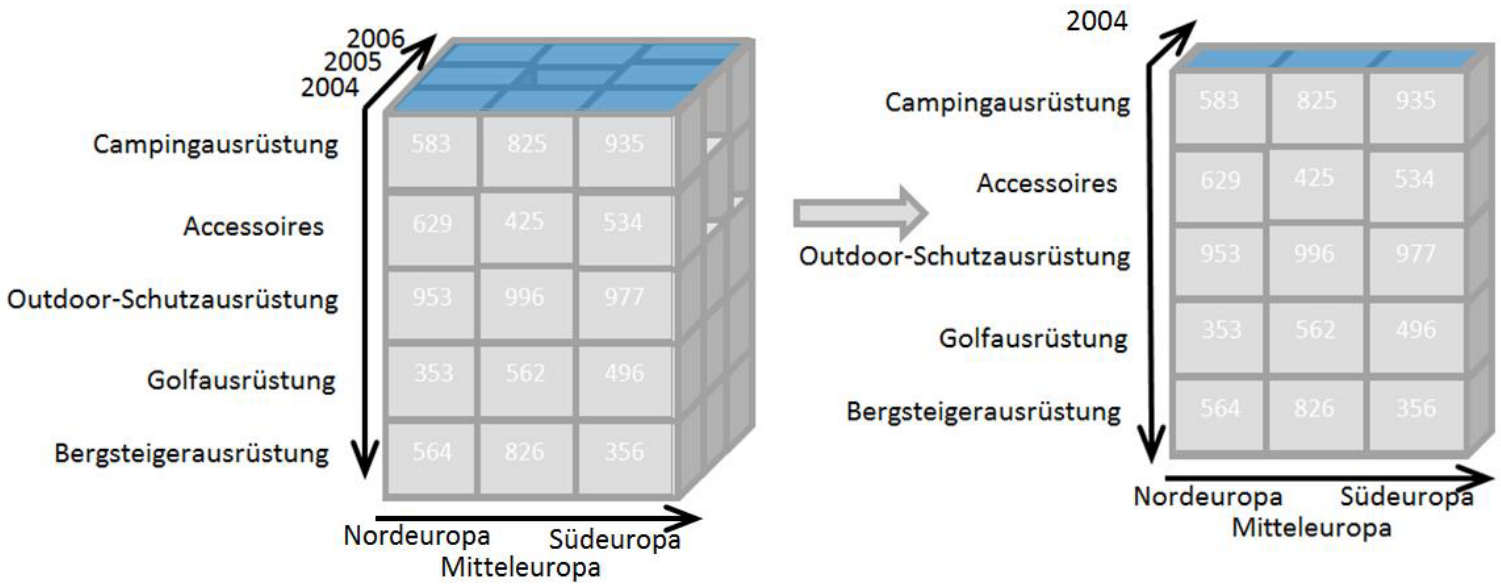

Figure: Source: Infopedian, OLAP Slicing [27]

The OLAP Cube: Dice [27]

- Dice: Pick specific values of multiple operations

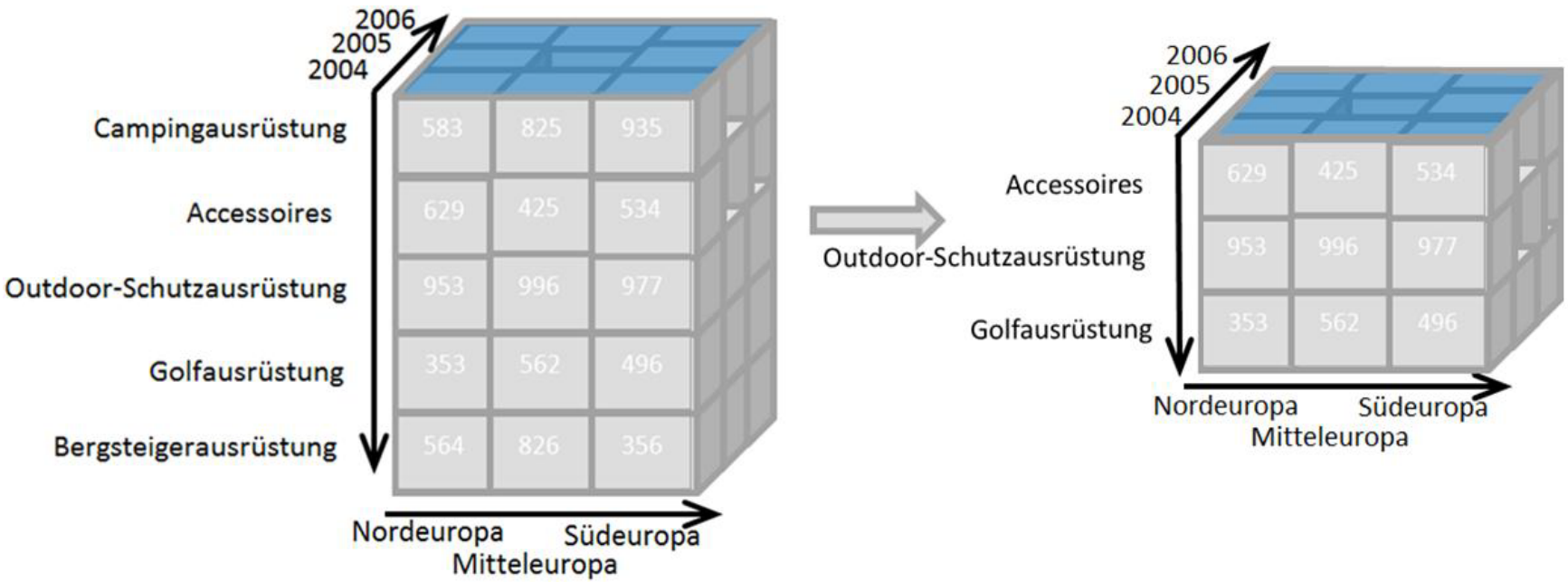

Figure: Source: Infopedian, OLAP dicing [27]

The OLAP Cube: Drill Down/Up [27]

- Drill Down/Up: Navigate the aggregation level
  - Drill down increases the detail level

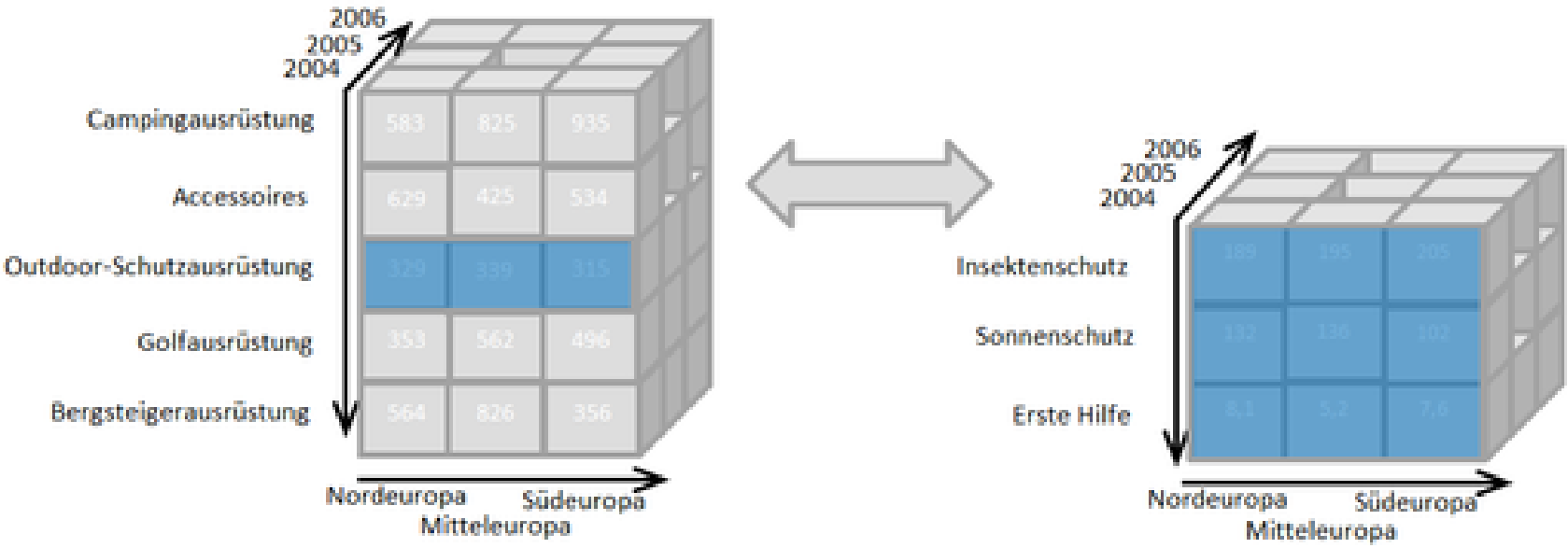

Figure: Source: Infopedian, OLAP Drill-up and drill-down [27]

Star (and Snowflake) Schemas [23]

Implement the OLAP cube in relational databases

Data model

- Fact table: records measurements/metrics for a specific event
  - Center of the star
  - Transaction table: records a specific event e.g. sale
  - Snapshot table: record facts at a given point in time e.g. account balance at the end of the month
  - Accumulating table: aggregate facts for a timespan e.g. month-to-date sales for a product

  $\Rightarrow$ A fact table retains information at a low granularity and can be huge
- Dimension tables: describe the facts in one dimension
  - Contains e.g. time, geography, product (hierarchy), employee, range
  - The fact table contains a FOREIGN KEY to all dimension tables

  $\Rightarrow$ Comparably small table

Snowflake schema normalizes dimensions to reduce storage costs

Star Schema Example Model

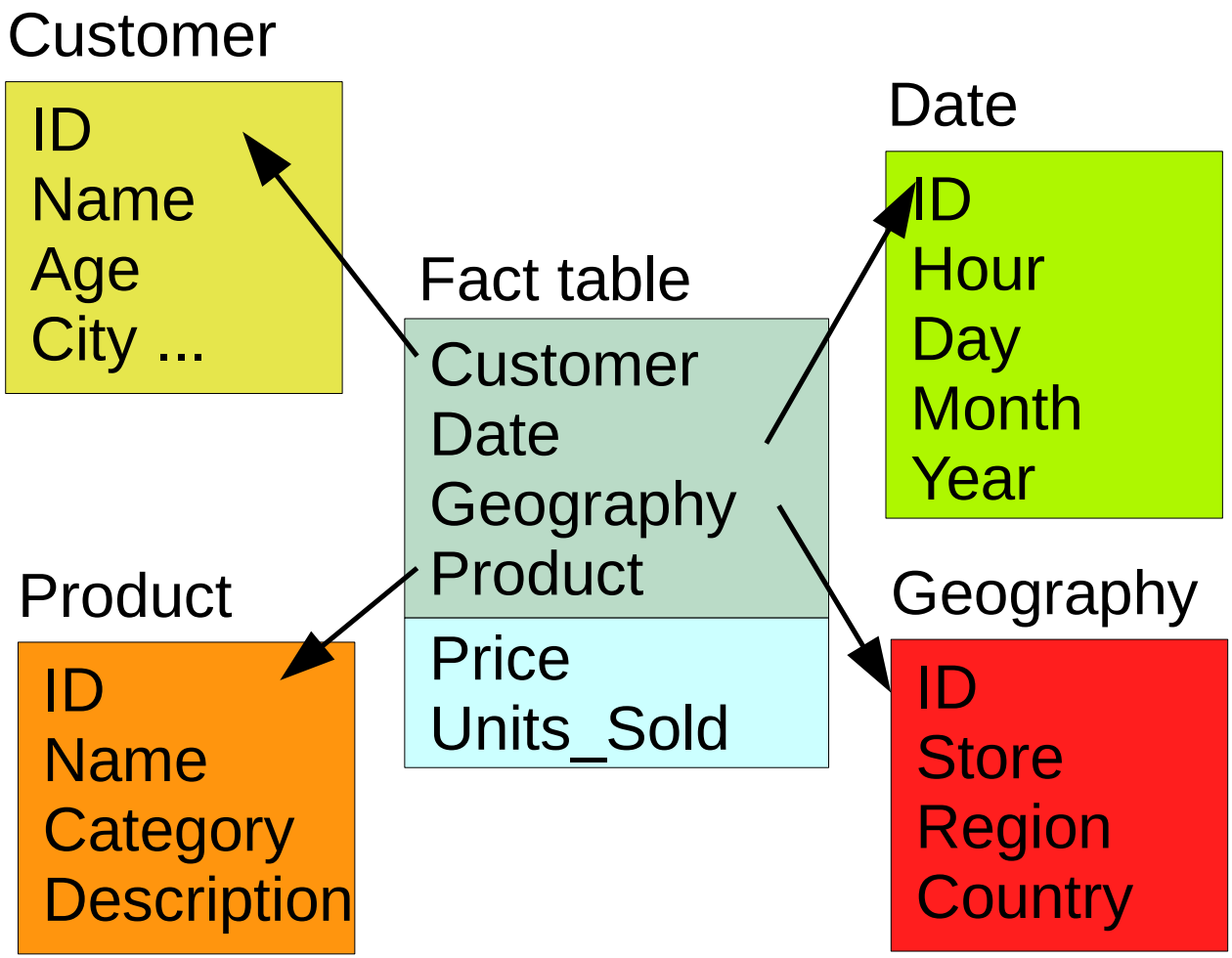

Figure: Star schema

Star Schema: Example Query

Analyze the sales of TVs per country and brand [23]

```
1  SELECT  P.Brand, S.Country AS Countries,
2    SUM(F.Units_Sold)
3  FROM Fact_Sales F
4  INNER JOIN Date    D ON (F.Date_Id = D.Id)
5  INNER JOIN Store   S ON (F.Store_Id = S.Id)
6  INNER JOIN Product P ON (F.Product_Id = P.Id)
7
8  WHERE D.Year = 1997 AND  P.Product_Category = 'tv'
9
10 GROUP BY
11   P.Brand,
12   S.Country
```

Star Schema [23]

Advantages

- Simplification of queries and performance gains
- Emulates OLAP cubes

Disadvantages

- Data integrity is not guaranteed
- No natural support for many-to-many relations

Snowflake Schema Example Model

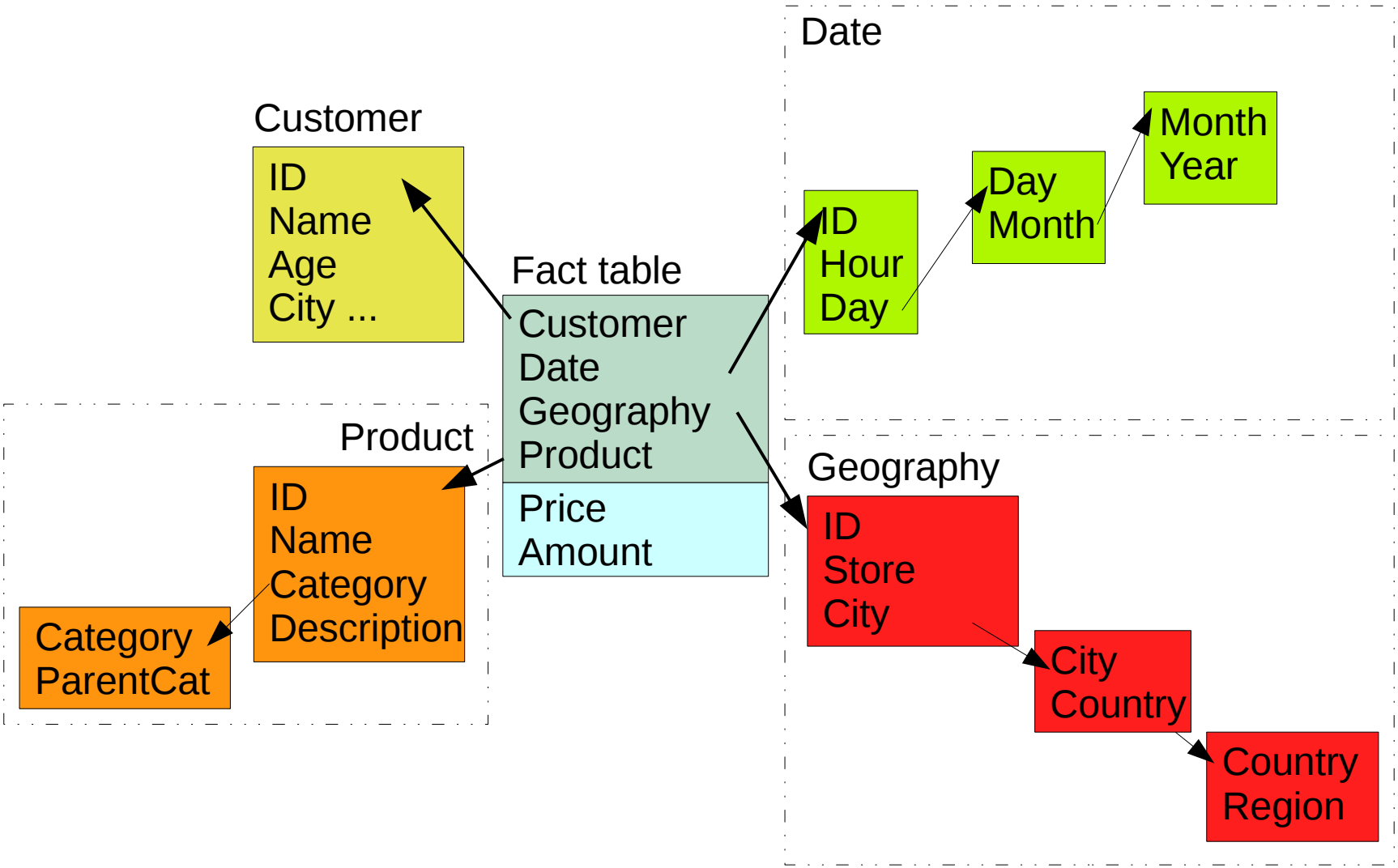

Figure: Snowflake schema

Summary

- ER-diagrams visualize the relational data model
- Keys allow addressing of tuples (rows)
- Normalization reduces dependencies
  - Avoids redundancy, prevents inconsistency
- SQL combines data retrieval/modification and computation
  - Insert, Select, Update, Delete
  - Joins combine records
- Transactions executes a sequence of operations with ACID semantics
- A database optimizes the execution of the queries (query planer)
- Semi-structured data analysis is possible within JSON and XML
- OLAP (Cube) deals with multidimensional business data
- Data warehouses store facts along their dimensions
- Star-schema implements OLAP in a relational schema

Bibliography

10 Wikipedia

11 https://en.wikipedia.org/wiki/Relational_model

16 https://en.wikipedia.org/wiki/Superkey

17 https://en.wikipedia.org/wiki/Candidate_key

18 https://en.wikipedia.org/wiki/Unique_key

19 https://en.wikipedia.org/wiki/Database_normalization

20 https://en.wikipedia.org/wiki/SQL

21 PostgreSQL Documentation http://www.postgresql.org/docs/9.4/static/

22 https://wiki.postgresql.org/wiki/Performance_Optimization

23 https://en.wikipedia.org/wiki/Star_schema

24 https://en.wikipedia.org/wiki/Data_mart

25 https://en.wikipedia.org/wiki/Snowflake_schema

26 https://en.wikipedia.org/wiki/Dimensional_modeling

27 https://en.wikipedia.org/wiki/OLAP_cube

28 https://en.wikipedia.org/wiki/Data_warehouse

29 https://en.wikipedia.org/wiki/Database