Distributed Data Management
(Handout)

Dr. Eike Schallehn
Organization
Organization of Lecture and Exercises

- Weekly lecture
  - Teacher: Eike Schallehn (eike@iti.cs.uni-magdeburg.de)
- Weekly exercises with two alternative time slots
  - Starting in the third week of the lecture period
  - Teachers: Xiao Chen, Juliana Alves Pereira
- Written exam at the end of the semester (registration using HISQUIS system)

Prerequisites

- Required: knowledge about database basics from database introduction course
  - Basic principles, Relational Model, SQL, database design, ER Model
- Helpful: advanced knowledge about database internals
  - Query processing, storage structures
- Helpful hands-on experience:
  - SQL queries, DDL and DML

Content Overview

1. Foundations
2. Distributed DBMS: architectures, distribution, query processing, transaction management, replication
3. Parallel DBMS: architectures, query processing
4. Federated DBS: architectures, conflicts, integration, query processing
5. Peer-to-peer Data Management

English Literature /1

English Literature /2


German Literature

Part I

Introduction

Overview

Contents

I Introduction 5
1 Motivation 6
2 Classification of Multi-Processor DBMS 12
3 Recapitulation 21
1 Motivation

Centralized Data Management

• New requirements
  – Support for de-centralized organization structures
  – High availability
  – High performance
  – Scalability

Client Server Data Management in a Network
Distributed Data Management

Distributed Data Management: Example

Advantages of Distributed DBMS

- Transparent management of distributed/replicated data
- Availability and fault tolerance
- Performance
- Scalability
Transparent Data Management

- Transparency: "hide" implementation details
- For (distributed) database systems
  - Data independence (physical, logical)
  - Network transparency
    * "hide" existence of the network
    * "hide" physical location of data
  - To applications a distributed DBS looks just like a centralized DBS

Transparent Data Management/2

- continued:
  - Replication transparency
    * Replication: managing copies of remote data (performance, availability, fault-tolerance)
    * Hiding the existence of copies (e.g. during updates)
  - Fragmentation transparency
    * Fragmentation: decomposition of relations and distribution of resulting fragments
    * Hiding decomposition of global relation

Who provides Transparency?

- Application
  - Different parts/modules of distributed application
  - Communication / data exchange using standard protocols (RPC, CORBA, HTTP, SOAP, …)

- DBMS
  - Transparent SQL-access to data on remote DB-instances
  - Requires query decomposition, transaction coordination, replication

- Operating system
  - Operating systems provides network transparency e.g. on file system level (NFS) or through standard protocols (TCP/IP)
Fault-Tolerance

- Failure of one single node can be compensated
- Requires
  - Replicated copies on different nodes
  - Distributed transactions

Performance

- Data can be stored, where they are most likely used → reduction of transfer costs
- Parallel processing in distributed systems
  - Inter-transaction-parallelism: parallel processing of different transactions
  - Inter-query-parallelism: parallel processing of different queries
  - Intra-query-parallelism: parallel of one or several operations within one query

Scalability

- Requirements raised by growing databases or necessary performance improvement
  - Addition of new nodes/processors often cheaper than design of new system or complex tuning measures
Differentiation: Distributed Information System

- Distributed Information System
  - Application components communicate for purpose of data exchange (distribution on application level)

- Distributed DBS
  - Distribution solely realized on the DBS-level

Differentiation: Distributed File System

- Distributed File System provides non-local storage access by means of operating system

- DBMS on distributed file system
  - All data must be read from blocks stored on different disks
  - Processing is performed only within DBMS node (not distributed)
  - Distribution handled by operating system

Special Case: Parallel DBS

- Data management on simultaneous computer (multi processor, special hardware)
- Processing capacities are used for performance improvement
- Example
  - 100 GB relation, sequential read with 10 MB/s $\leadsto$ 17 minutes
  - parallel read on 10 nodes (without considering coordination overhead) $\leadsto$ 1:40 minutes

Special Case: Heterogeneous DBS

- Motivation: integration of previously existing DBS (legacy systems)
  - Integrated access: global queries, relationships between data objects in different databases, global integrity

- Problems
  - Heterogeneity on different levels: system, data model, schema, data

- Special importance on the WWW: integration of Web sources $\leadsto$ Mediator concept
Special Case: Peer-to-Peer-Systems

- Peer-to-Peer (P2P): networks without centralized servers
  - All / many nodes (peers) store data
  - Each node knows only some "close" neighbors
    * No global view
    * No centralized coordination

- Examples: Napster, Gnutella, Freenet, BitTorrent, ... 
  - Distributed management of data (e.g. MP3-Files)
  - Lookup using centralized servers (Napster) or distributed (Gnutella)
2 Classification of Multi-Processor DBMS

Multi-Processor DBMS

- In general: DBMS which are able to use multiple processors or DBMS-instances to process database operations [Rahm 94]
- Can be classified according to different criteria
  - Processors with same or different functionality
  - Access to external storage
  - Spatial distribution
  - Processor connection
  - Homogeneous vs. heterogeneous architecture

Classification Overview

- Assumption: each processor provides the same functionality
- Classification [Rahm94]
Criterion: Access to External Storage

- Partitioned access
  - External storage is divided among processors/nodes
    * Each processor has only access to local storage
    * Accessing different partitions requires communication
- Shared access
  - Each processor has access to full database
  - Requires synchronisation

Criterion: Spatial Distribution

- Locally distributed: DB-Cluster
  - Fast inter-processor communication
  - Fault-tolerance
  - Dynamic load balancing possible
  - Little administration efforts
  - Application: parallel DBMS, solutions for high availability
- Remotely distributed: distributed DBS in WAN scenarios
  - Support for distributed organization structures
  - Fault-tolerant (even to major catastrophes)
  - Application: distributed DBS

Criterion: Processor Connection

- Tight connection
  - Processors share main memory
  - Efficient co-operation
  - Load-balancing by means of operating system
  - Problems: Fault-tolerance, cache coherence, limited number of processors \( \leq 20 \)
  - Parallel multi-processor DBMS
Criterion: Processor Connection /2

- Loose connection:
  - Independent nodes with own main memory and DBMS instances
  - Advantages: failure isolation, scalability
  - Problems: expensive network communication, costly DB operations, load balancing

- Close connection:
  - Mix of the above
  - In addition to own main memory there is connection via shared memory
  - Managed by operating system
Class: Shared-Everything

- Simple realization of DBMS
- Distribution transparency provided by operating system
- Expensive synchronization
- Extended implementation of query processing
Class: Shared-Nothing

- Distribution of DB across various nodes
- Distributed/parallel execution plans
- TXN management across participating nodes
- Management of catalog and replicas
**Class: Shared-Disk**

- Avoids physical data distribution
- No distributed TXNs and query processing
- Requires buffer invalidation

---

**Class: Shared-Disk /2**

- Avoids physical data distribution
- No distributed TXNs and query processing
- Requires buffer invalidation
Criterion: Integrated vs. Federated DBS

- Integrated:
  - Shared database for all nodes $\rightarrow$ one conceptual schema
  - High distribution transparency: access to distributed DB via local DBMS
  - Requires co-operation of DBMS nodes $\rightarrow$ restricted autonomy

- Federated:
  - Nodes with own DB and own conceptual schema
  - Requires schema integration $\rightarrow$ global conceptual schema
  - High degree of autonomy of nodes

Criterion: Integrated vs. Federates DBS /2
Criterion: Centralized vs. De-centralized Coordination

- Centralized:
  - Each node has global view on database (directly or via master)
  - Central coordinator: initiator of query/transaction → knows all participating nodes
  - Provides typical DBS properties (ACID, result completeness, etc.)
  - Applications: distributed and parallel DBS
    * Limited availability, fault-tolerance, scalability

Criterion: Centralized vs. De-centralized Coordination /2

- De-centralized:
  - No global view on schema → peer knows only neighbors
  - Autonomous peers; global behavior depends on local interaction
  - Can not provide typical DBMS properties
  - Application: P2P systems
    * Advantages: availability, fault-tolerance, scalability
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Parallel DBS</th>
<th>Distributed DBS</th>
<th>Federated DBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>High TXN rates</td>
<td>↑</td>
<td>➔</td>
<td>→</td>
</tr>
<tr>
<td>Intra-TXN-Parallelism</td>
<td>↑ ➔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>➔</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>➔</td>
<td>➔</td>
<td>➔</td>
</tr>
<tr>
<td>Geo. Distribution</td>
<td>➔</td>
<td>➔</td>
<td></td>
</tr>
<tr>
<td>Node Autonomy</td>
<td>➔</td>
<td>➔</td>
<td></td>
</tr>
<tr>
<td>DBS-Heterogeneity</td>
<td>➔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Administration</td>
<td>➔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3 Recapitulation

Database Management Systems (DBMS)

- Nowadays commonly used
  - to store huge amounts of data persistently,
  - in collaborative scenarios,
  - to fulfill high performance requirements,
  - to fulfill high consistency requirements,
  - as a basic component of information systems,
  - to serve as a common IT infrastructure for departments of an organization or company.

Database Management Systems

A database management system (DBMS) is a suite of computer programs designed to manage a database and run operations on the data requested by numerous clients.

A database (DB) is an organized collection of data.

A database system (DBS) is the concrete instance of a database managed by a database management system.
Codd’s 9 Rules for DBMS

- Differentiate DBMS from other systems managing data persistently, e.g. file systems

1 **Integration**: homogeneous, non-redundant management of data

2 **Operations**: means for accessing, creating, modifying, and deleting data

3 **Catalog**: the data description must be accessible as part of the database itself

4 **User views**: different users/applications must be able to have a different perception of the data

5 **Integrity control**: the systems must provide means to grant the consistency of data

6 **Data security**: the system must grant only authorized accesses

7 **Transactions**: multiple operations on data can be grouped into a logical unit

8 **Synchronization**: parallel accesses to the database are managed by the system

9 **Data backups**: the system provides functionality to grant long-term accessibility even in case of failures
3 Level Schema Architecture

- Important concept of DBMS
- Provides
  - transparency, i.e. non-visibility, of storage implementation details
  - ease of use
  - decreased application maintenance efforts
  - conceptual foundation for standards
  - portability
- Describes abstraction steps:
  - Logical data independence
  - Physical data independence

Data Independence

Logical data independence: Changes to the logical schema level must not require a change to an application (external schema) based on the structure.

Physical data independence: Changes to the physical schema level (how data is stored) must not require a change to the logical schema.
Architecture of a DBS

Schema architecture roughly conforms to general architecture of a database systems

- Applications access database using specific views (external schema)
- The DBMS provides access for all applications using the logical schema
- The database is stored on secondary storage according to an internal schema

Client Server Architecture

Schema architecture does not directly relate to client server architecture (communication/network architecture)

- Clients may run several applications
- Applications may run on several clients
- DB servers may be distributed
- ...
The Relational Model

- Developed by Edgar F. Codd (1923-2003) in 1970
- Derived from mathematical model of n-ary relations
- Colloquial: data is organized as tables (relations) of records (n-tuples) with columns (attributes)
- Currently most commonly used database model
- Relational Database Management Systems (RDBMS)
- First prototype: IBM System R in 1974
- Implemented as core of all major DBMS since late ’70s: IBM DB2, Oracle, MS SQL Server, Informix, Sybase, MySQL, PostgreSQL, etc.
- Database model of the DBMS language standard SQL

Basic Constructs

A relational database is a database that is structured according to the relational database model. It consists of a set of relations.

\[
R \rightarrow \{R_1, \ldots, R_n\}
\]

Integrity Constraints

- Static integrity constraints describe valid tuples of a relation
  - Primary key constraint
  - Foreign key constraint (referential integrity)
  - Value range constraints
  - ...
- In SQL additionally: uniqueness and not-NULL
- Transitional integrity constraints describe valid changes to a database
The Relational Algebra

A relational algebra is a set of operations that are closed over relations.

- Each operation has one or more relations as input
- The output of each operation is a relation

Relational Operations

Primitive operations:

- Selection $\sigma$
- Projection $\pi$
- Cartesian product (cross product) $\times$
- Set union $\cup$
- Set difference $-$
- Rename $\beta$

Non-primitive operations

- Natural Join $\bowtie$
- $\theta$-Join and Equi-Join $\bowtie_{=}$
- Semi-Join $\ltimes$
- Outer-Joins $= \times$
- Set intersection $\cap$
- ...

Notation for Relations and Tuples

- If $R$ denotes a relation schema (set of attributes), than the function $r(R)$ denotes a relation conforming to that schema (set of tuples)
- $R$ and $r(R)$ are often erroneously used synonymously to denote a relation, assuming that for a given relation schema only one relation exists
- $t(R)$ denotes a tuple conforming to a relation schema
- $t(R.a)$ denotes an attribute value of a tuple for an attribute $a \in R$

The Selection Operation $\sigma$

Select tuples based on predicate or complex condition

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>5</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>Houston</td>
<td>1</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
</tbody>
</table>

$\sigma_{PLOCATION='Stafford'}(r(\text{PROJECT}))$

<table>
<thead>
<tr>
<th>PNAME</th>
<th>PNUMBER</th>
<th>PLOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computerization</td>
<td>10</td>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>4</td>
</tr>
</tbody>
</table>
The Projection Operation \( \pi \)

Project to set of attributes - remove duplicates if necessary

<table>
<thead>
<tr>
<th>PROJECT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PNAME</td>
<td>PNUMBER</td>
</tr>
<tr>
<td>ProductX</td>
<td>1</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
</tr>
</tbody>
</table>

\[ \pi_{PLOCATION,DNUM}(r(PROJECT)) \]

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>DNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellaire</td>
<td>5</td>
</tr>
<tr>
<td>Sugarland</td>
<td>5</td>
</tr>
<tr>
<td>Houston</td>
<td>5</td>
</tr>
<tr>
<td>Stafford</td>
<td>4</td>
</tr>
<tr>
<td>Houston</td>
<td>1</td>
</tr>
</tbody>
</table>

Cartesian or cross product \( \times \)

Create all possible combinations of tuples from the two input relations

<table>
<thead>
<tr>
<th>R</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ r(R) \times r(S) \]

<table>
<thead>
<tr>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Set: Union, Intersection, Difference

- All require compatible schemas: attribute names and domains
- Union: duplicate entries are removed
- Intersection and Difference: \( \emptyset \) as possible result

The Natural Join Operation \( \bowtie \)

- Combine tuples from two relations \( r(R) \) and \( r(S) \) where for
  - all attributes \( a = R \cap S \) (defined in both relations)
- is \( t(R.a) = t(S.a) \).

- Basic operation for following key relationships

- If there are no common attributes result is Cartesian product \( R \cap S = \emptyset \implies r(R) \bowtie r(S) = r(R) \times r(S) \)

- Can be expressed as combination of \( \pi, \sigma \) and \( \times \):

\[
r(R) \bowtie r(S) = \pi_{R \cup S}(\sigma_{a \in R \cap S} t(R.a) = t(S.a)(r(R) \times r(S)))
\]

### The Natural Join Operation \( \bowtie \)

#### Table 1

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

### The Semi-Join Operation \( \bowtie \)

- Results all tuples from one relation having a (natural) join partner in the other relation \( r(R) \bowtie r(S) = \pi_{R}(r(R) \bowtie r(S)) \)

#### Table 2

<table>
<thead>
<tr>
<th>PERSON</th>
<th>PID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1273</td>
<td>Dylan</td>
<td></td>
</tr>
<tr>
<td>2244</td>
<td>Cohen</td>
<td></td>
</tr>
<tr>
<td>3456</td>
<td>Reed</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 3

<table>
<thead>
<tr>
<th>CAR</th>
<th>PID</th>
<th>BRAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1273</td>
<td>Cadillac</td>
<td></td>
</tr>
<tr>
<td>1273</td>
<td>VW Beetle</td>
<td></td>
</tr>
<tr>
<td>3456</td>
<td>Stutz Bearcat</td>
<td></td>
</tr>
</tbody>
</table>

### Other Join Operations

- **Conditional join**: join condition \( \varphi \) is explicitly specified \( r(R) \bowtie_{\varphi} r(S) = \sigma_{\varphi}(r(R) \times r(S)) \)

- **\( \theta \)-Join**: special conditional join, where \( \varphi \) is a single predicate of the form \( a \theta b \) with \( a \in R, b \in S \), and \( \theta \in \{=, \neq, >, <, \leq, \geq, \ldots \} \)
• **Equi-Join**: special $\theta$-Join where $\theta$ is $=.$

• **(Left or Right) Outer Join**: union of natural join result and tuples from the left or right input relation which could not be joined (requires NULL-values to grant compatible schemas).

### Relational Database Management Systems

A **Relational Database Management System (RDBMS)** is a database management system implementing the relational database model.

- Today, most relational DBMS implement the SQL database model
- There are some significant differences between the relational model and SQL (duplicate rows, tuple order significant, anonymous column names, etc.)
- Most distributed and parallel DBMS have a relational (SQL) data model

### SQL Data Model

- Said to implement relational database model
- Defines own terms

<table>
<thead>
<tr>
<th>Table name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>$A_1 \ldots A_n$</td>
</tr>
</tbody>
</table>

- Some significant differences exist

### Structured Query Language

- **Structured Query Language (SQL)**: declarative language to describe requested query results
- Realizes relational operations (with the mentioned discrepancies)
- Basic form: `SELECT-FROM-WHERE-block (SFW)`

```sql
SELECT FNAME, LNAME, MGRSTARTDATE
FROM EMPLOYEE, DEPARTMENT
WHERE SSN=MGRSSN
```
SQL: Selection σ
\( \sigma_{\text{DNO}=5 \land \text{SALARY}>30000}(r(\text{EMPLOYEE})) \)

```
SELECT *
FROM EMPLOYEE
WHERE DNO=5 AND SALARY>30000
```

SQL: Projection π
\( \pi_{\text{LNAME}, \text{FNAME}}(r(\text{EMPLOYEE})) \)

```
SELECT \text{LNAME}, \text{FNAME}
FROM EMPLOYEE
```

- Difference to RM: does not remove duplicates
- Requires additional DISTINCT

```
SELECT DISTINCT \text{LNAME}, \text{FNAME}
FROM EMPLOYEE
```

SQL: Cartesian Product ×
\( r(\text{EMPLOYEE}) \times r(\text{PROJECT}) \)

```
SELECT *
FROM EMPLOYEE, PROJECT
```

SQL: Natural Join ⊙∽
\( r(\text{DEPARTMENT}) \bowtie r(\text{DEPARTMENT LOCATIONS}) \)

```
SELECT *
FROM DEPARTMENT
NATURAL JOIN DEPARTMENT_LOCATIONS
```

SQL: Equi-Join
\( r(\text{EMPLOYEE}) \bowtie_{\text{SSN}=\text{MGRSSN}} r(\text{DEPARTMENT}) \)

```
SELECT *
FROM EMPLOYEE, DEPARTMENT
WHERE \text{SSN}=\text{MGRSSN}
```

SQL: Union
\( r(R) \cup r(S) \)

```
SELECT * FROM R
UNION
SELECT * FROM S
```
Other set operations: INTERSECT, MINUS

Does remove duplicates (in compliance with RM)

If duplicates required:

```
SELECT * FROM R
UNION ALL
SELECT * FROM S
```

**SQL: Other Features**

- SQL provides several features not in the relational algebra
  
  - Grouping And Aggregation Functions, e.g. SUM, AVG, COUNT, ...
  
  - Sorting

```
SELECT PLOCATION, AVG(HOURS)
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER
GROUP BY PLOCATION
HAVING COUNT(*) > 1
ORDER BY PLOCATION
```

**SQL DDL**

- **Data Definition Language** to create, modify, and delete schema objects

```
CREATE DROP ALTER TABLE mytable ( id INT, ...)
DROP TABLE ...
ALTER TABLE ...
CREATE VIEW myview AS SELECT ...
DROP VIEW ...
CREATE INDEX ...
DROP INDEX ...
...
```

**Simple Integrity Constraints**

```
CREATE TABLE employee(
   ssn INTEGER,
   lname VARCHAR2(20) NOT NULL,
   dno INTEGER,
   ...
   FOREIGN KEY (dno)
      REFERENCES department(dnumber),
   PRIMARY KEY (ssn)
)
```

- Additionally: triggers, explicit value domains, ...
SQL DML

- **Data Manipulation Language** to create, modify, and delete tuples
  
  \[
  \text{INSERT INTO (\langle\text{COLUMN}\rangle) mytable VALUES (...)}
  \]
  
  \[
  \text{INSERT INTO (\langle\text{COLUMN}\rangle) mytable SELECT ...}
  \]
  
  \[
  \text{UPDATE mytable}
  \]
  \[
  \text{SET ...}
  \]
  \[
  \text{WHERE ...}
  \]
  
  \[
  \text{DELETE FROM mytable}
  \]
  \[
  \text{WHERE ...}
  \]

Other Parts of SQL

- Data Control Language (DCL): **GRANT, REVOKE**
- Transaction management: **START TRANSACTION, COMMIT, ROLLBACK**
- Stored procedures and imperative programming concepts
- Cursor definition and management

Transactions

- Sequence of database operations
  - Read and write operations
  - In SQL sequence of INSERT, UPDATE, DELETE, SELECT statements
- Build a semantic unit, e.g. transfer of an amount from one bank account to another
- Has to conform to **ACID** properties

Transactions: **ACID Properties**

- Atomicity means that a transaction can not be interrupted or performed only partially
  - TXN is performed in its entirety or not at all
- Consistency to preserve data integrity
  - A TXN starts from a consistent database state and ends with a consistent database state
• **Isolation**
  – Result of a TXN must be independent of other possibly running parallel TXNs

• **Durability or persistence**
  – After a TXN finished successfully (from the user’s view) its results must be in the database and the effect can not be reversed

**Functional Dependencies**

• A functional dependency (FD) \( X \rightarrow Y \) within a relation between sets \( r(R) \) of attributes \( X \subseteq R \) and \( Y \subseteq R \) exists, if for each tuple the values of \( X \) determine the values for \( Y \)

• i.e.
  \[ \forall t_1, t_2 \in r(R) : t_1(X) = t_2(X) \Rightarrow t_1(Y) = t_2(Y) \]

**Derivation Rules for FDs**

- \( R_1 \) Reflexivity if \( X \supseteq Y \) \( \implies X \rightarrow Y \)
- \( R_2 \) Accumulation \( \{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ \)
- \( R_3 \) Transitivity \( \{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z \)
- \( R_4 \) Decomposition \( \{X \rightarrow YZ\} \Rightarrow X \rightarrow Y \)
- \( R_5 \) Unification \( \{X \rightarrow Y, X \rightarrow Z\} \Rightarrow X \rightarrow YZ \)
- \( R_6 \) Pseudotransitivity \( \{X \rightarrow Y, WY \rightarrow Z\} \Rightarrow WX \rightarrow Z \)

\( R_1-R_3 \) known as *Armstrong-Axioms* (sound, complete)

**Normal Forms**

• Formal criteria to force schemas to be free of redundancy

• First Normal Form (1NF) allows only *atomic attribute values*

  – i.e. all attribute values \( a \) of basic data types like *integer* or *string* but not further structured like e.g. an *array* or a *set* of values

• Second Normal Form (2NF) avoids *partial dependencies*

  – A partial dependency exist, if a non-key attribute is functionally dependent on a real subset of the primary key of the relation
Normal Forms /2

- Third Normal Form (3NF) avoids \textit{transitive dependencies}
  - Disallows functional dependencies between non-key attributes

- Boyce-Codd-Normal Form (BCNF) disallows \textit{transitive dependencies} also for primary key attributes