STAATLICH
ANERKANNTE
HOCHSCHULE

BIG DATA & BUSINESS ANALYTICS

DATA ENGINEERING

SRH HOCHSCHULE HEIDELBERG
Intelligence in Learning

DATA ENGINEERING

1. SQL
2. Hardware
3. MapReduce and Hadoop
4. NoSQL and CAP
5. InMemory
6. Streaming

RELATIONAL DATABASES

Invention of **relational database**s at IBM

- Edgar F. Codd:
  "A Relational Model of Data for Large Shared Data Banks"
  Communications of the ACM, 1970.
- Donald Chamberlin und Raymond Boyce:
  SEQUEL (Structured English Query Language)
  later renamed to **SQL**

Edgar F. Codd (1923-2003)

In the beginning, relational databases had to compete against a **hierarchical database** at IBM, the Information Management System (IMS), which was commercially highly successful in the 1960s.

CODD'S TWELVE RULES (1985)

Codd provided a list of 12 rules formalizing his definition of the relational database model.

https://en.wikipedia.org/wiki/Codd%27s_12_rules

Relational database model

Relation	= table
Tuple	= row
Attribute	= columm
Primary key	= one or more attributes that uniquely identify a row
Foreign key	= attribute in a row that is a primary key in another table

DATABASE NORMALIZATION

First normal form (1NF)
Each attribute contains only atomic values (i.e. values that cannot be decomposed by the database into smaller components). A comma-separated list of values or a JSON string in one field violates the 1NF.

Second normal form (2NF)
First normal form + no partial dependencies. Any attribute not in the primary key depends on the complete attributes of any candidate key, not on a subset.

Third normal form (3NF)
Second normal form + no transitive dependencies. All attributes not in the primary key depend only on the primary key.
Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key, so help me Codd.

+ 4NF + 5NF

DATABASE NORMALIZATION

this is not atomic

Given Name	Surname	Company	Phone	Company Country Code	Company Country
John	Doe	IBM	617-123, 617-456	US	United States
Hans	Muster	SAP	06221-3456, 06221-9876	DE	Germany
John	Doe	SAP	06221-1111	DE	Germany

1NF

Given Name	Surname	Company	Phone 1	Phone 2	Company Country Code	Country
John	Doe	IBM	617-123	617-456	US	United States
Hans	Muster	SAP	06221-3456	06221-9876	DE	Germany
John	Doe	SAP	06221-1111	NULL	DE	Germany

2NF

Given Name	Surname	Company	Phone 1	Phone 2
John	Doe	IBM	617-123	617-456
Hans	Muster	SAP	06221-3456	06221-9876
John	Doe	SAP	06221-1111	NULL

Company	Code	Country
IBM	US	United States
SAP	DE	Germany

3NF

Given Name	Surname	Company	Phone 1	Phone 2
John	Doe	IBM	617-123	617-456
Hans	Muster	SAP	06221-3456	06221-9876
John	Doe	SAP	06221-1111	NULL

Company	Code
IBM	US
SAP	DE

Code	Country
US	United States
DE	Germany

PRIMARY KEY

Natural key
A natural primary key is composed of one more fields of the application data.

Surrogate key
A surrogate key is an artificial primary key. The value is generated when inserting the record. For example, a UUID (128 bit universally unique identifier) or an automatically incremented integer.

```
CREATE TABLE participants(
  id           INTEGER PRIMARY KEY AUTOINCREMENT,
  first_name   TEXT
);
```

Advantages of surrogate keys

- Stable reference, applications can be sure that it does not change
- Support of requirement changes. If the natural key would have to be modified (e.g. expanded), the surrogate key remains stable.

Disadvantages

- Foreign keys cannot be interpreted, more JOINs necessary, performance impact

HISTORY OF SQL

Language for managing databases and their content, standardised as ISO/IEC 9075

Version	Notable Changes
SQL-86	First standard pf ANSI (American National Standards Institute)
SQL-92	Introduction of new data types and operators
SQL:1999	Extension for object-relational databases
SQL:2003	Support of XML, additional operators
SQL:2006	Extended functionality for XML
SQL:2008	More operators
SQL:2011	Support of time functions (time intervals, ...)
SQL:2016	Support of JSON (Javascript Object Notation)

COMPONENTS OF SQL

Data Definition Language (DDL)

- CREATE TABLE - Creation of a table
- ALTER TABLE - Changing a table (e.g. add or remove columns)
- DROP TABLE - Deletion of a table

Data Query Language (DQL)

- SELECT - Reading from a table

Data Manipulation Language (DML)

- SELECT ... INTO - Reading from a table and inserting
- INSERT - Inserting into a table
- UPDATE - Changing existing entries of a table
- DELETE - Deletion from a table
- COMMIT / ROLLBACK - Transaction control

Data Control Language (DCL)

- GRANT / REVOKE - Assignment and revocation of permissions to users

... and many more statements in these components.

EXAMPLES

```
-- Creation of a table
CREATE TABLE participants(
  id            INT           NOT NULL,
  lastname      VARCHAR(30)   NOT NULL,
  firstname     VARCHAR(30),
  age           INT,
  PRIMARY       KEY(id)
);


-- Inserting into a table
INSERT INTO participants VALUES (1, "Codd", "Edgar", 79);


-- Reading all entries from a table
-- Sorted by name ascending
SELECT * FROM participants  ORDER BY surname ASC
```

SELECT QUERIES

```
-- Reading of surname and firstname of all people with age > 30
-- where the first name starts with letter E (% is wildcard symbol)
SELECT lastname, firstname
  FROM participants
  WHERE age > 30  AND firstname LIKE 'E%'


-- Counting of all entries
SELECT COUNT(*) FROM participants


-- Return all different first names (without duplicates)
SELECT DISTINCT firstname FROM participants
```

ACID

Desirable properties / guarantees of a database system

Atomicity

- Transactional behaviour: "all or nothing"

Consistency

- After each transaction, the database is in a consistent state
  (i.e. valid and without internal contradictions)

Isolation

- Each transaction is executed as if it were the only action on the system.

Durability

- Persistency of data: After successful transaction commit, the data persists
  even after database crash, power outage, etc… (physical integrity).

ATOMICITY

A **transaction** is the logical unit of work in a database.

Transactions are composed of one or more statements which are treated as unit.
A transaction is either successfully executed completely or not at all.

Example: Bank transfer of 100 € from Alice to Bob.

```
BEGIN;
UPDATE accounts SET balance = balance – 100  WHERE name = 'Alice';
UPDATE accounts SET balance = balance + 100  WHERE name = 'Bob';
COMMIT;
```

A database can be configured for automatic commit after each statement (`AUTOCOMMIT ON`).

In case of `AUTOCOMMIT OFF`, the statements `COMMIT` or `ROLLBACK` have to be executed explicitly.

See: https://www.postgresql.org/docs/current/tutorial-transactions.html

CONSISTENCY

Consistency means that after each transaction, the database is in a valid state.

In a valid state, the database invariants are maintained:

- Data integrity
  - Entity integrity: each record has a primary key which is unique and not null.
  - Referential integrity: a foreign key value refers to a primary key value in another table
  - Domain integrity: each attribute value matches the column type specified in the schema
- Cascading rollbacks are executed correctly:
  if a transaction depends on another transaction, and the second transaction is rolled back, the first transactions must be rolled back as well
- Triggers are executed correctly:
  A trigger is a procedure / script that executes SQL statements on some event. For example when a new record is created (e.g. a new book), a trigger could automatically create another record in another table (e.g. add author if not present yet). Used to maintain application level integrity.

ISOLATION LEVELS

Isolation means that each transaction is executed as if it were the only action on the system.

This can always be enforced by the application using **locking** and thus forcing a strictly sequential ordering of database transactions. Usually this leads to performance degradation, because the application has to wait until the lock is released before accessing the database.

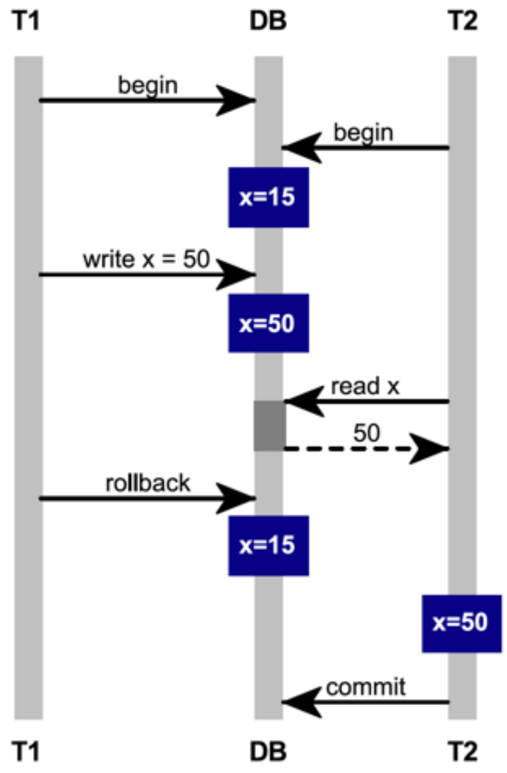
Alternatively, the database system can be configured with **isolation level "serializable"**, which is the highest isolation level and has the same effect.

If the performance penalty is not desired, lower isolation levels allow concurrent transactions at the risk of accessing invalid data. The issues are:

- reading uncommited data (**"dirty read"**)
- reading data that changes during the transaction (**"non-repeatable read"**)
- accessing a wrong set of records because the data of the WHERE-condition changes during the transaction (**"phantom read"**)

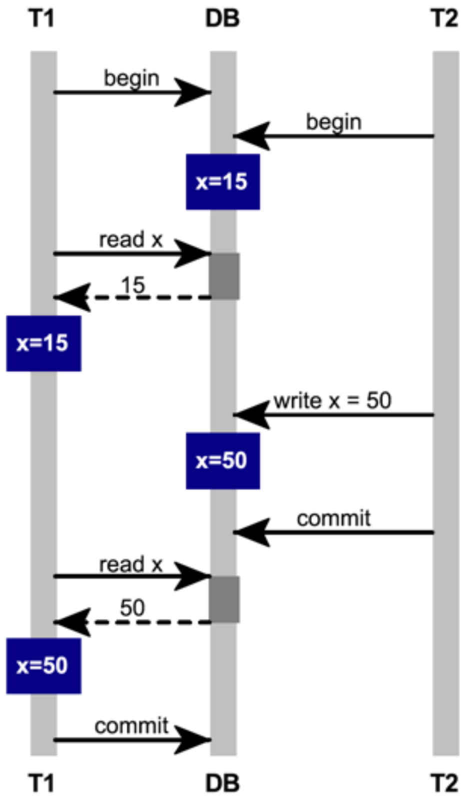
See: https://www.postgresql.org/docs/current/transaction-iso.html

ISOLATION LEVELS



Dirty Read

Transaction T1 writes the value 50 to x and later rolls back the transaction. Therefore transaction T2 should not read the value 50.



Non-repeatable read

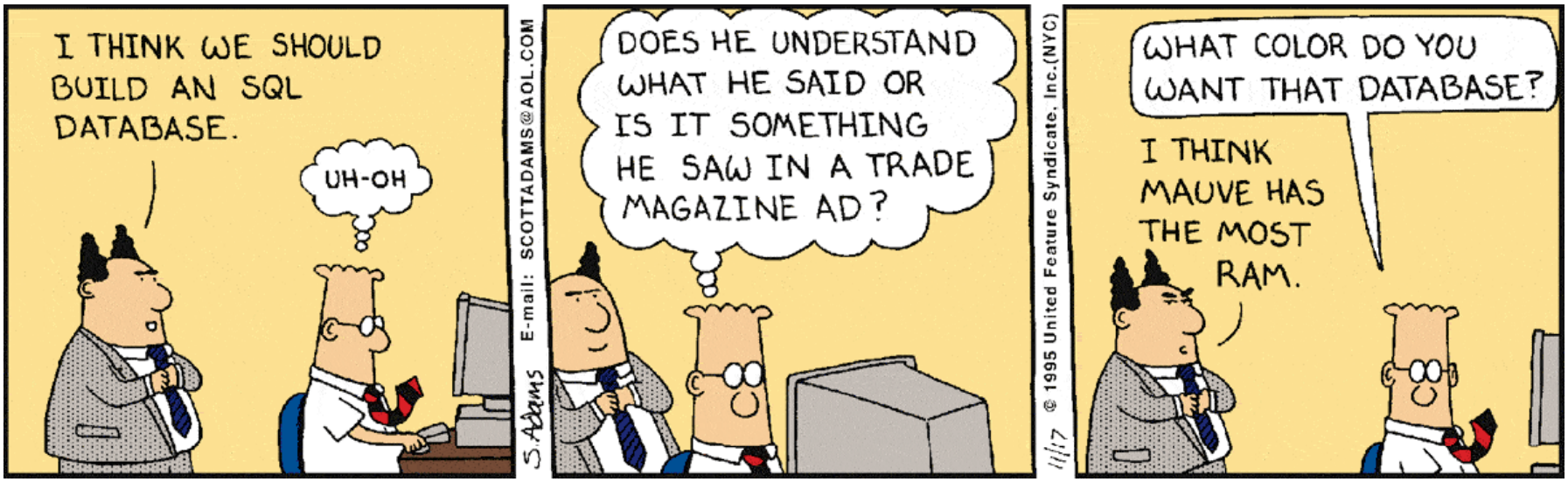
Transaction T1 reads the value 15. Transaction T2 writes the value 50 to x and performs a commit. Transaction T1 rereads the value of x and it is different from the first read action. Therefore transaction T2 should not write the value 50 to x.

See: Stefan Lütolf, Seminar Thesis, ANSI SQL Isolation Levels, 2014.

ISOLATION LEVELS

Based on these anomalies, the SQL ANSI standard defines four isolation levels

Isolation Level	Dirty Read	Non-repeatable Read	Phantom Read
Read uncommited	possible	possible	possible
Read commited	not possible	possible	possible
Repeatable read	not possible	not possible	possible
Serializable	not possible	not possible	not possible



http://dilbert.com/strip/1995-11-17