ifis

Institut für Informationssysteme
Technische Universität Braunschweig

Relational Database Systems I

Wolf-Tilo Balke
Simon Barthel

Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de

14 Active Databases

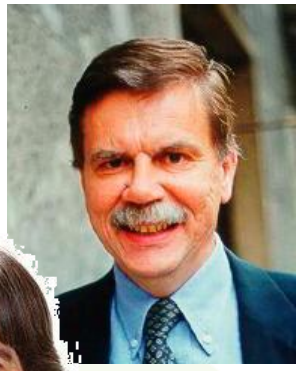
- Active databases
  - Integrity constraints
  - Triggers
  - UDFs and stored procedures
- Basic security
  - Access control
  - SQL injection

14.0 Active Databases

- There is a growing need for databases that can **actively manipulate their data**
  - In particular, the first wave of object databases encouraged many RDBMS vendors to provide active functionalities within their systems

14.0 Active Databases

- **Active databases** are RDBMS that can
  - **recognize predefined situations** and
  - **respond** to those situations with individual **predefined actions.**
- Initially proposed by S. Ceri and J. Widom in 1990
  - "Deriving Production Rules for Constraint Maintenance" In 16th International Conference on Very Large Data Bases, Brisbane, Australia,1990.



14.0 Active Databases

- Active databases allow programmers and admins to **enhance the functionality of the DBMS** by defining:
  - **Constraints**
  - **Triggers**
  - **User-defined data types (UDTs)**
  - **User-defined functions (UDFs)**
  - **Stored procedures**

14.0 Active Databases

- Most of these active extensions started as **proprietary technologies**
  - The exact syntax strongly differs among database vendors
- Some of them have been **standardized** in SQL:
  - Constraints and assertions
  - Triggers
  - Procedural statements

14.1 Integrity Constraints

- The original aim of active components in database systems was to respond to attempted **violations of integrity constraints**
  - Integrity constraints describe:
    - What is a **valid database state**
    - How to make **valid transitions** between database states
  - **Examples:**
    - Primary/foreign key constraints
    - Data types and domains
    - "CHECK conditions" in SQL

14.1 Integrity Constraints

- **Types of constraints** include
  - **Static** integrity constraints
    - Bound to a single DB state (Examples: data types, key constraints)
  - **Dynamic** integrity constraints
    - **Transitional integrity constraints** are bound to a change of the DB state (Examples: update, insert, delete)
    - **Temporal integrity constraints** are bound to a sequence of DB states (Examples: transactions, periodical checks)
- Some constraints may be difficult to evaluate and require **predicate logic** for specification
  - Master course: **Knowledge-Based Systems and Deductive Databases**

14.1 Integrity Constraints

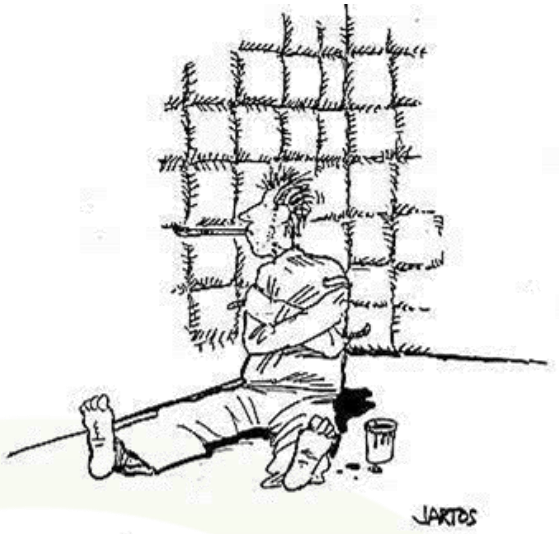
- An integrity constraint is called
  - **Local,** if it only concerns a single relation
    - Examples: value domains, data types of attributes
  - **Global,** if more than one relation is concerned
    - Example: foreign keys
  - **Implicit,** if it is a consequence of the data model
    - Example: data types of attributes
  - **Explicit,** if it is not implicit, but can be expressed in DDL
    - Example: primary key
  - **External,** if it is neither implicit, nor explicit
    - Example: semantic check clauses

14.1 Defining Constraints

- **Constraints** (or **assertions**) are conditions which have to be true for all data in the database instance
  - We already introduced constraints briefly (SQL)
- Constraints may be defined
  - **Explicitly** by the `CREATE CONSTRAINT` statement
  - **Implicitly** within the DDL table/column definition (`CREATE TABLE ... CHECK ...`)
- An SQL statement is executed only if it does not result in a constraint violation
  - Usually critical: insert, delete, and update operations

14.1 Defining Constraints

- Summary of constraint types:
  - **Data type constraint, NOT NULL constraint, UNIQUE constraint**
    - Usually within column definition
  - **Primary key constraint (key integrity)**
    - Usually within table or column definition
  - **Foreign key constraints (referential integrity)**
    - Usually within table or column definition
  - **Check constraints**
    - **Support any arbitrary complex condition expressible in SQL**
    - Usually defined explicitly or within a table definition
  - **Informational constraints**
    - This type of constraint is **not enforced**
    - Used by the query optimizer to better understand the data



14.1 Defining Constraints

- **Example:**
  Aliases of superheroes
  - Data types, primary key, foreign key, check clause


```
CREATE  TABLE hasAlias (
  hero_id INTEGER REFERENCES hero
                  ON DELETE CASCADE ON UPDATE CASCADE,
  alias VARCHAR(100) NOT NULL,
  PRIMARY KEY (hero_id, aliasName),
  CONSTRAINT noSillyAliases CHECK (alias <> 'Stupid Man')
)
```

14.1 Constraint Definitions

- Constraints are used to enforce valid DB states by **rejecting** all operations resulting in invalid DB states
  - **Simple and robust** tool for enforcing some basic (static) constraints
- But invalid DB operations cannot be "**repaired**" depending on the type of constraint violation
  - Example: If a tuple in some insert statement refers to a non-existing foreign key, why not simply add the respective foreign key before the insert is committed?

14.2 Server-Side Code

- We will cover three main technologies for executing code on server side
  - **Triggers**
    - A trigger is automatically executed by the DBMS when a predefined event occurs
  - **UDFs (user-defined functions)**
    - A UDF can used in any SQL statement as a function (similar to MIN, MAX, and COUNT)
  - **Stored procedures**
    - A stored procedure can be executed using SQL's CALL statement (also, parameters may be specified)

14.2 Triggers

- **Triggers** link user-defined actions to standard database operations
  - Whenever a certain DB operation is performed, the trigger "**fires**"
  - Very helpful to implement **dynamic** integrity constraints
  - Each operation can have assigned **several triggers**
    - Sequence of execution is usually non-deterministic
  - Several triggers can fire within a transaction
  - Again, different vendors use **different syntax**…

14.2 Triggers

- Standardized in **SQL:1999**
- Some DBMS offer **native extensions** to SQL for specifying the triggers
  - Examples: PL/SQL (Oracle), Transact SQL (MS SQL Server)
- Some DBMS allow the use of **general purpose programming languages**
  - Examples: Java (Oracle, DB2), C#/VB (MS SQL Server)
- Some DBMS use an **extended trigger concept**
  - Example: Triggers on views (Oracle)

14.2 Triggers

- Triggers implement the **event-condition-action** model
  - Triggers are **active rules**
    - Typical syntax: ON **<event>** IF **<condition>** DO **<action>**
  - **Events** activate a rule
    - Usually, triggers are fired upon data modifications
    - In general, it may be any external event
  - The **condition** determines whether the action is executed
    - Optional; contains a Boolean expression
  - The **action** is executed for every event satisfying the condition
    - Usually, this is done as a series of SQL (update) statements within the same transaction as the triggering event
    - But an action may also be the call of an external program

14.2 Triggers

- **Types of events** include
  - Timed events
    - Absolute, relative, or periodic
  - Database events
    - Begin/end of some insert, delete, or update statement
  - DBMS events
    - DDL commands
    - Transaction events: begin, commit, or abort
    - Changes in user accounts, or access control

- Today's commercial databases typically support triggers **only for database events**

14.2 Triggers

- **What to use triggers for?**
  - **Auditing table operations**
    - Write a protocol of each data access
  - **Tracking record value changes**
    - Write a modification log and archive all previous data
  - **Preserving a database's referential integrity**
    - Retaining referential integrity by actively changing all affected records

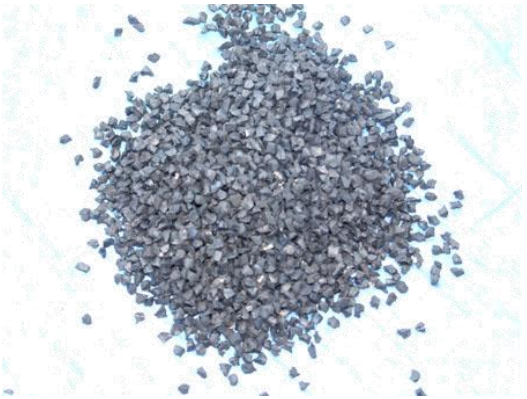
14.2 Triggers

- **Maintenance of semantic integrity**
  - Example: When a super villain is caught, all henchmen should become unemployed
- **Storing derived data**
  - Customized update of materialized views
  - Computing complex aggregations that cannot be expressed easily using pure SQL
- **Access control**
  - Checking user privileges when accessing sensitive information

14.2 Triggers

- When **creating a trigger,** the following information needs to be specified
  - **Trigger name**
    - Triggers use qualified names within a given schema
  - **Trigger event**
    - Trigger events may either monitor row updates (`ON INSERT`/ `ON DELETE`) or column updates (`ON UPDATE`)
    - A trigger gets **attached** to the table mentioned in the event
  - **Activation time**
    - The trigger can be activated either **before** or **after** the event occurred
    - `BEFORE` or `AFTER` keywords

14.2 Triggers



- **Granularity:**
  - A trigger's **actions** may be executed **per statement** (statement trigger) or **per row** (row trigger)
  - Per statement:
    - Default
    - The whole body is executed once **per event**
    - `FOR EACH STATEMENT` keyword
  - Per row:
    - The body is executed once **per affected row**
    - `FOR EACH ROW` keyword

14.2 Triggers

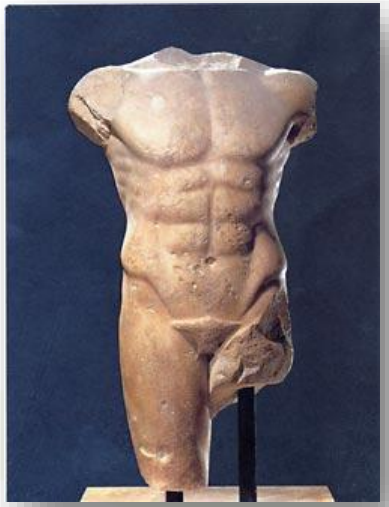
- **Transition variables**
  - Optional
  - Often triggers need **access** to the updated (new and old values), deleted, or added data
  - `REFERENCING` clause
  - There are four types of transition variables:
    - Old row (`OLD`):
      References the modified row before the triggering event
    - New row (`NEW`):
      References the modified row after the triggering event
    - Old table (`OLD_TABLE`):
      References the table as it was before the triggering event (read-only)
    - New table (`NEW_TABLE`):
      References the table as it is after the triggering event

14.2 Triggers

- Not all **combinations** of trigger **events,** activation **times, granularities,** and **transition variables** are possible

Event and time	ROW TRIGGER	STATEMENT TRIGGER
BEFORE INSERT	NEW	
BEFORE UPDATE	OLD, NEW	
BEFORE DELETE	OLD	
AFTER INSERT	NEW, NEW_TABLE	NEW_TABLE
AFTER UPDATE	OLD, NEW, OLD_TABLE, NEW_TABLE	OLD_TABLE , NEW_TABLE
AFTER DELETE	OLD, OLD_TABLE	OLD_TABLE

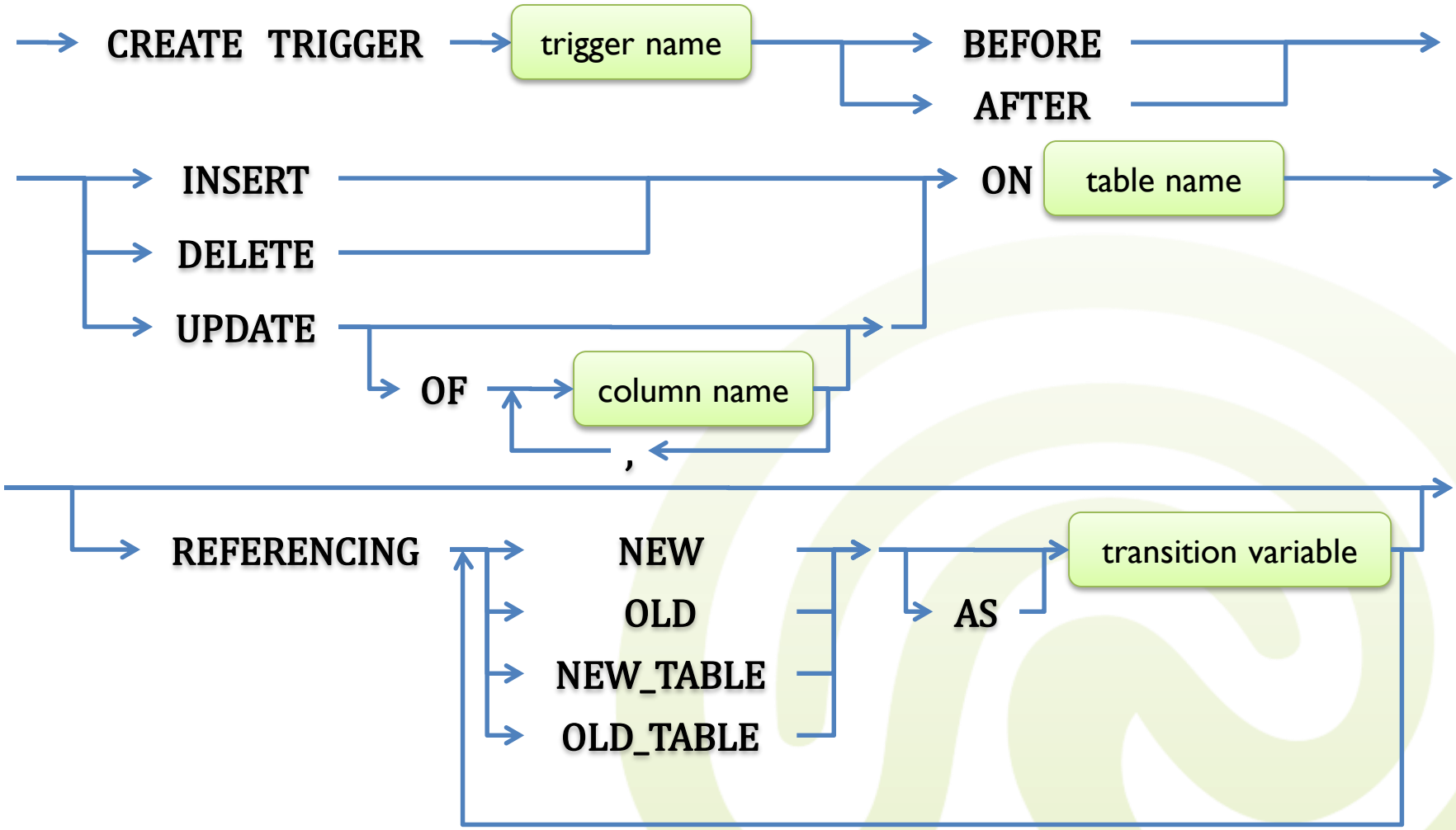
14.2 Triggers



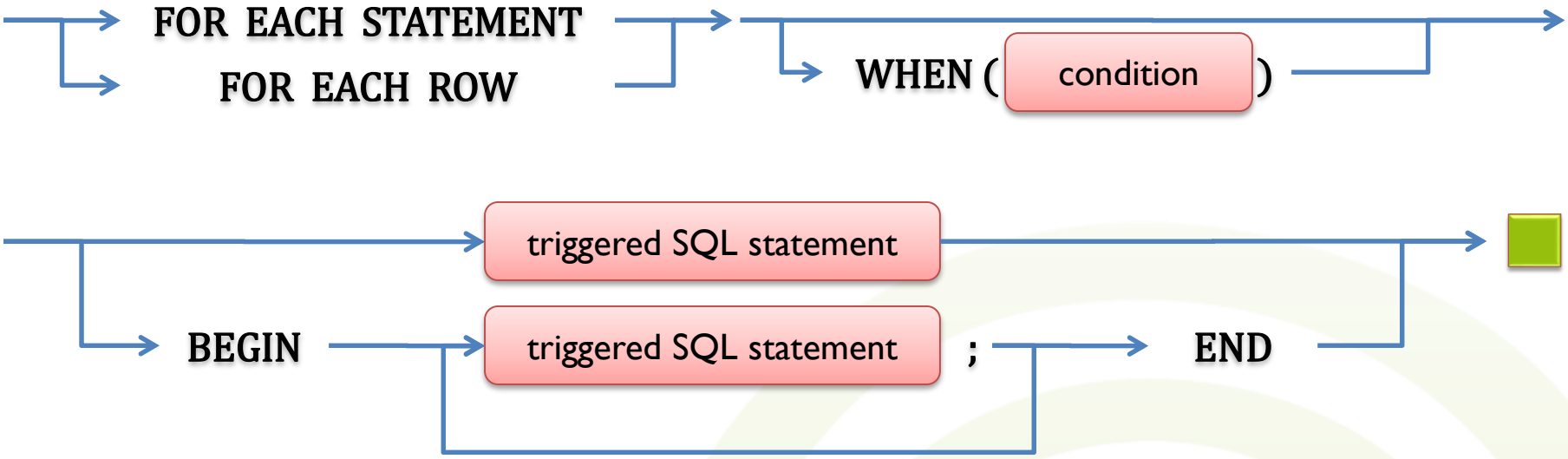
- **Trigger condition**
  - Optional
  - WHEN clause
  - Use any Boolean expression (as in SQL's WHERE clause)
- **Trigger body**
  - Can be any number of SQL statements, separated by semicolon
  - Embedded into a BEGIN-END block
  - Some DBMS also allow calling code written in other languages or even binary programs

14.2 Triggers

CREATE TRIGGER STATEMENT



14.2 Triggers



14.2 Triggers

- **Example:**
  - A DB storing the current location of things and heroes
  - Trigger: As soon as Superman comes near kryptonite, **delete him!**

```
CREATE TRIGGER kill_superman
    AFTER UPDATE OF location ON heroes
    REFERENCING NEW AS hn
    FOR EACH ROW
    WHEN hn.name = 'Superman'
            AND EXISTS(SELECT * FROM stuff s
                                WHERE s.name = 'Kryptonite'
                                        AND s.location = hn.location)
    BEGIN
        DELETE FROM heroes h WHERE h.id = hn.id;
    END
```

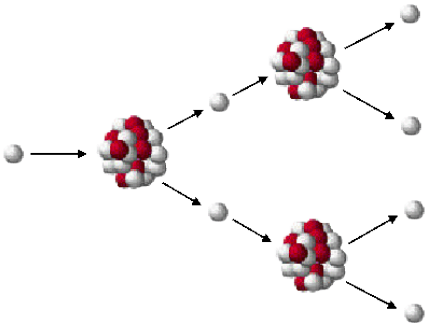
14.2 Triggers

- The previous example is **standard SQL:1999**
  - It won't necessarily work on all DBMS
  - Example **DB2:**
    - Replace `BEGIN` by `BEGIN ATOMIC`
      - Or just don't use `BEGIN-END` at all
    - Add `MODE DB2SQL` before `WHEN`
  - Read the technical documentation of your DBMS!
- There are some **prototype implementations** for **active databases** based on ECA rules, thus also supporting a larger group of events

14.2 Triggers

- **Trigger execution order**
  1. Execute all BEFORE STATEMENT triggers
  2. Temporarily disable all integrity constraints defined on each affected table
  3. Loop for each row in the table
     1. Execute all BEFORE ROW triggers
     2. Execute the SQL statement against the row and perform integrity constraint checks of the data
     3. Execute all AFTER ROW triggers
  4. Complete deferred integrity constraint checks on the table
  5. Execute all AFTER STATEMENT triggers

14.2 Triggers



- **Trigger chaining**
  - When using triggers, the actions of one trigger might **activate another trigger**
    - That trigger may actually activate even more triggers
  - It is very easy to lose track of what happens…
  - You need to be **very careful here!**
- **Recursive triggers**
  - Special case of chaining: A trigger **activates itself** again
  - It is easy to create **infinite loops**
  - Even if you do not create infinite loops, most **DBMS don't handle this too well**
    - Example: DB2 aborts with a `TOO COMPLEX` error, if a trigger activates itself more than once

14.3 Complex Database Programs

- Next, we will introduce two flavors of complex **database programs**
  - **Stored procedures**
  - **User-defined functions (UDFs)**
- Both reside **within the DBMS** and may be called explicitly
  - Exchange of information possible via input and/or output **parameters** and **result sets**
  - Application programmers and users may define and create those database programs

14.3 Complex Database Programs

- **Stored procedures** are **called directly** by the application or by other procedures
  - `CALL removeInactiveHeroes(00200000)`
- **UDFs** can be used within any SQL statement as a **functional expression**

```
SELECT *
  FROM villains v
WHERE notoriety(v.id) > 100
```

14.3 Complex Database Programs

- What are possible **advantages?**
  - **Move** parts of program **logic** (code!) to the server
  - **Improve** application **performance** by reducing client/server communication
    - Database program is executed in the DBMS
  - **Control access** to database objects
    - Database programs can be used instead of queries, thus enabling fine-grained access control
  - Integrate some **"non-database functionality"** into the DBMS
  - **Readability** and **reliability** of common, complex queries can be increased by encapsulation of some functionality

14.3 Complex Database Programs

- What **problems** can you encounter?
  - Database server may end up being a **performance bottleneck**
  - Writing database programs **"disturbs"** your usual application development and **deployment process**
    - They are usually written in a different language
    - They have to be installed and registered with the DBMS
  - Database programs can be **tricky to debug**
    - It can be cumbersome to get debug information from DBMS
    - Your normal debugging environment may not work
    - There may be complex dependencies among DB programs
  - You can easily **lose track** of your database programs and versions
    - They reside outside your normal source control programs

14.3 Stored Procedures in DB2

Detour

- DB2 offers three kinds of stored procedures:
  - **SQL stored procedures**
    - Directly written in procedural SQL as defined by SQL:1999
  - **External stored procedures**
    - Written in one of the many higher programming languages supported by DB2
      - Examples: C, CL, RPG, Cobol, ...
  - **Java stored procedures**
    - Actually, they are also external stored procedures
    - Due to the different implementation and deployment mechanics, they are treated as an extra case

14.3 Stored Procedures in DB2

Detour

- Stored procedures
  - Defined by the **CREATE PROCEDURE** statement
  - General structure is

    CREATE PROCEDURE name
    (list of input and output variables)
    Procedure properties
    Generation options
    Procedure body

14.3 Stored Procedures in DB2

Detour

- As procedures are stored within a specific schema, you may use a fully **qualified name**
- The **input/output list** contains all interface variables and their respective types
- In **properties,** additional meta-data is provided
  - **Programming language,** number of result sets, read/write mode (read only, data update, full access), ...
- **Generation** options provide optional advanced features for storing the procedure
  - Debug modes, storage locations, ..
- The **procedure body** provides the actual code
  - If you choose SQL as language, the body contains a compound SQL statement
  - In the other cases, a respective linker information is provided

14.3 Stored Procedures in DB2

Detour

- Example:
  - There is a table storing the names of heroes and their last heroic deed
  - Create a stored procedure that disables all heroes who did not do anything within the last *x* days

```
CREATE PROCEDURE setInactive
    (IN x DECIMAL(8,0))
    LANGUAGE SQL
    UPDATE heroes h SET h.status = 'inactive' WHERE
        CURRENT DATE - x > h.deedDate
```

Durations are given in DECIMAL(8,0) format: YYYYMMDD

Reserved keyword for returning the current date

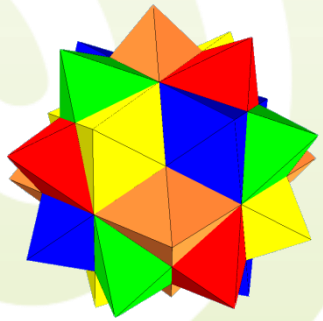
```
CALL setInactive(00050600)
```

Date duration for 5 years, 6 month, 0 days

14.3 Stored Procedures in DB2

Detour

- You may use SQL **compound statements** as procedure body
  - Compound statements defined in SQL:1999 standard
  - Contain definition of **cursors**, **variables**, **control-flow** statements, ...
  - Compound statements are embedded in **BEGIN...END**
  - Also, **dynamic statements** can be used within compound SQL



14.3 Stored Procedures in DB2

- Example: Using compound SQL
  - Create a new table 'numbers' and fill it with rows containing all numbers between 0 and *x*

```
CREATE PROCEDURE createNumbers
      (IN  x  INTEGER)
      LANGUAGE  SQL
      MODIFIES  SQL  DATA
BEGIN
      DECLARE  v_counter INTEGER DEFAULT 0;
      CREATE  TABLE  numbers (num INTEGER);
      WHILE  v_counter < x DO
            INSERT INTO number VALUES (v_counter);
            SET v_counter = v_counter + 1;
      END WHILE;
END
```

14.3 Stored Procedures in DB2

Detour

- Example: Return a result set
  - A return cursor needs to be declared

```
CREATE PROCEDURE getInactiveHeroes
    RESULT SETS 1
    LANGUAGE SQL
BEGIN
    DECLARE c1 CURSOR WITH RETURN FOR
        SELECT * FROM heroes WHERE status = 'inactive';
    OPEN c1;
END
```

  - Stored procedure result sets are not very useful in a **CALL** statement
    - But may be used by other stored procedures or host languages
    - Caller is responsible for closing the result set

14.3 Stored Procedures in DB2

Detour

- You may also use dynamic SQL:

```
CREATE PROCEDURE createSomeTable
    (IN name VARCHAR(20))
    LANGUAGE SQL
    MODIFIES SQL DATA
BEGIN
    SET stmt1 = 'CREATE TABLE ' || name || ' (id INTEGER NOT NULL)';
    PREPARE s1 FROM stmt1;
    EXECUTE s1;
END
```

- To drop a procedure, use the DROP statement

```
DROP PROCEDURE createNumbers;
```

14.3 Stored Procedures in DB2

Detour

- DB2 also allows to create **stored procedures** written in **Java**
  - DB2 comes with its own Java virtual machine that will run the statements
  - Class files containing the procedure are uploaded and bound to the DBMS
  - A single Java class can define multiple stored procedures
  - Classes have to inherit from StoredProc
    - Provided by DB2's JDK

IBM DB2

14.3 Stored Procedures in DB2

Detour

- The signature of a Java stored procedure is public static void
- **Input parameters** are defined within the method signature
- **Output** and **input/output** variables are defined in the signature as arrays of length 1

```
package ifis;
public class SomeJavaStoredProcedures extends StoredProc {
        public static void procedure1(String inValue) {
                // do something
        }
        public static void getRandomNumber(double[] number) {
                number = new double[] { Math.random() };
}
```

14.3 Stored Procedures in DB2

- Within a stored Java procedure, data may be freely manipulated using either **JDBC** or **SQLJ**
  - JDBC connections may be obtained by calling DriverManager.getConnection("**jdbc:default:connection**");
- To enable the use of the stored procedure, you must **upload** and **bind it**
  - (Manually) upload class file into the function directory
  - Bind the procedure

```
CREATE PROCEDURE getRandomNumber
    (OUT number double)
    LANGUAGE JAVA
    PARAMETER STYLE JAVA
    EXTERNAL NAME 'ifis.SomeJavaStoredProcedures!getRandomNumber';
```

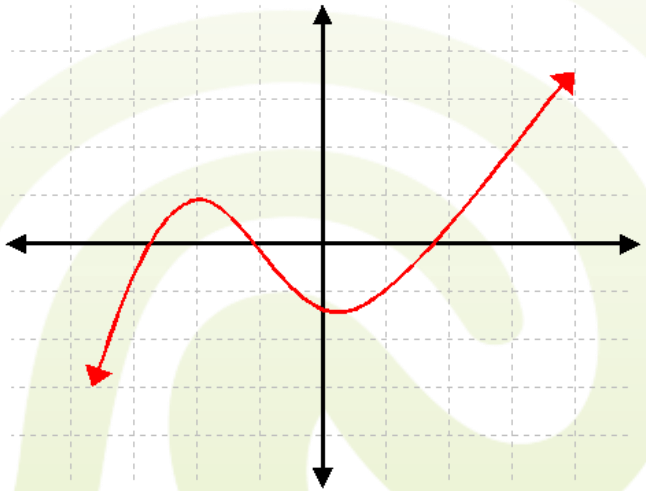
14.3 User-Defined Functions in DB2

- Similar to stored procedures, UDFs may also either implemented in **SQL, external languages, or Java**
  - Also, there are **sourced** UDF which just link to an existing function
- There are two types of UDFs
  - **Scalar** functions returning just a **single value**
    - Good in computing values or statistics, formatting strings, etc.
  - **Table** functions returning a whole **table**
    - Can be used instead of a query
    - Also, they can import non-relational data into SQL
      - Reading text files or any other input stream, access web services, etc.

Detour

- UDFs
  - Are defined by the **CREATE FUNCTION** statement
  - General structure is similar to stored procedures

    CREATE FUNCTION name
    (list of input parameters)
    Returns
    Function properties
    Generation options
    Routine body



14.3 User-Defined Functions in DB2

- As functions are stored within a specific schema, you may use a fully **qualified name**
- The **input** lists all input variables and their types
  - No output variables here
- The **returns** section provide the name and data type of the return value
- In **properties,** additional meta-data can be specified
  - **Language**, parallelism properties, level of data access, etc.
- **Generation** options provide optional advanced features for storing the function
- The **function body** provides the actual code
  - If you choose SQL as language, the body contains a compound SQL statement
  - Otherwise, linker information is necessary

- Example: Simple function with scalar return value

```
CREATE FUNCTION displayName
      (firstName VARCHAR(50),  lastname VARCHAR(50))
      RETURNS VARCHAR(100)
      LANGUAGE  SQL
      SPECIFIC displayName01
      DETERMINISTIC    CONTAINS SQL
RETURN  firstName || '  ' || lastName;
```

Same input leads to same output

No data read or manipulated

- Example: Simple function with tabular return value

```
CREATE FUNCTION aliasOf(heroname VARCHAR(50))
      RETURNS  TABLE(alias VARCHAR(50))
      LANGUAGE  SQL
      SPECIFIC aliasOf01
      READS  SQL  DATA
RETURN
      SELECT  alias  FROM  aliases a, heroes h
            WHERE  a.heroId = h.id   AND  h.name = heroname
```

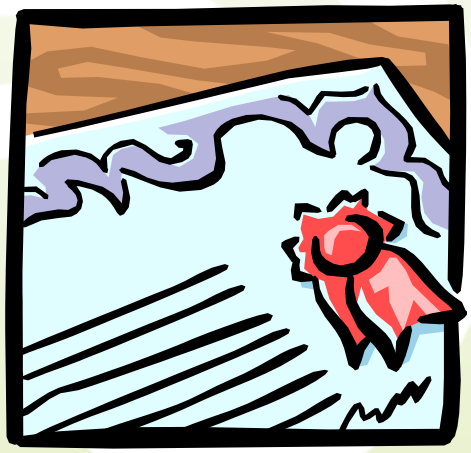
Unique system-wide name

14.4 Basic Access Control

- A major concern in databases is **data security**
  - Remember: **Views** can be used for restricting the data access of some application
    - Example: Salaries of employees are not shown in staff listing
    - Of course, this works only if the **original table** cannot be accessed by the application
  - A basic mechanism to enforce **access rights** to data is so-called **discretionary access control**
    - Grants privileges to users, including the capability to access specific data files, records, or fields in a specific mode (r/w)

14.4 Discretionary Access Control

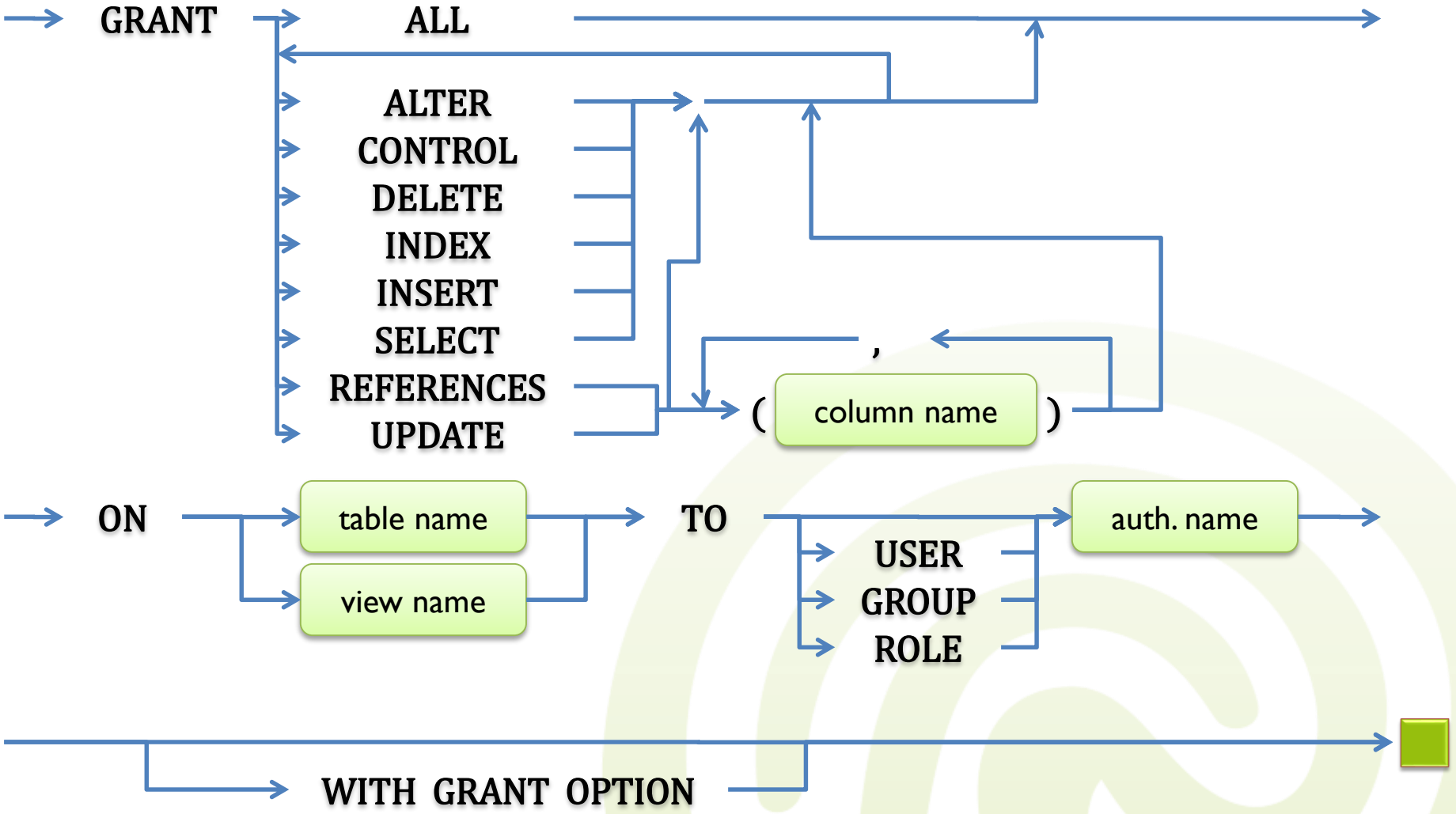
- Discretionary policies require that, **for each user,** authorization rules specify the **privileges granted** on the database objects
  - Access requests are checked against the **granted privileges**
  - Discretionary means that users may **grant/revoke** permissions (usually based on ownership)
  - By grants, access privileges can be **propagated** through the system



14.4 Discretionary Access Control

- The **SQL GRANT/REVOKE** statement can be used to grant privileges to users
  - GRANT privileges ON table(s)/column(s) TO grantees [WITH GRANT OPTION]
  - REVOKE privileges ON table(s)/column(s) FROM grantees
- **Possible privileges** are:
  - SELECT: user can retrieve data
  - UPDATE: user can modify existing data
  - DELETE: user can remove data
  - INSERT: user can insert new data
  - REFERENCES: user can define foreign keys to the table

14.4 Discretionary Access Control



14.4 Discretionary Access Control

- The **WITH GRANT OPTION** option permits the propagation of grant permissions to other users
  - Allows other users to define permissions for certain tables
- The **list of grantees** does not need not be (a set of) usernames
  - It is permitted to specify **PUBLIC**, which means that the privileges are granted to **everyone**
    - **Be very careful with that!**

14.4 Discretionary Access Control

- Checking discretionary access control is often implemented by an **authorization matrix**
  - The **rows** represent users
  - The **columns** represent the database objects
  - The **fields** contain the respective privileges
- Similar concept in Windows file security

14.4 Discretionary Access Control

- Granting or revoking permissions of users manually **for every possible access** is a very time-consuming task
  - More refined concepts of database security exist, for example role-based access control
- But data **security needs** more than simple access control
  - **Authentication:**
    Is the user really who he/she claims to be?
  - Concepts are discussed in detail in master course **Relational Databases 2**

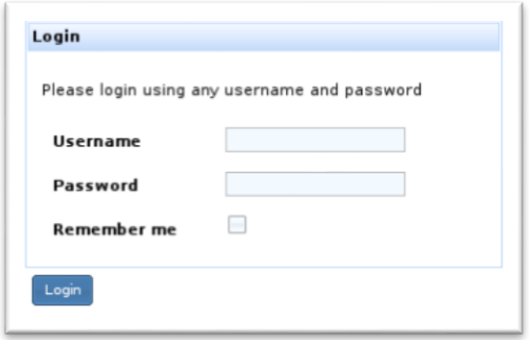
14.5 SQL Injection

Detour

- **SQL injection** is a **security vulnerability** of an application using an SQL database
- Characteristic:
  - **User input** is directly **embedded** into an SQL statement without further checking
  - User is able to **extend** the SQL statement or even **inject** completely new ones
  - Thus, data may be **corrupted, deleted,** or **stolen**

14.5 SQL Injection

Detour



- Example scenario:
  - A web interface asking for a **username** and a **password**
  - Following **statement** is used to authenticate the user:

```
String s = "SELECT * FROM users WHERE username = '"
         + user
         + "' AND password = '"
         + passwd
         + "';"
```

  - The application **simply inserts the user input** into the SQL string (using string concatenation)
  - If there is the given username/password combination, the application proceeds to the protected member area

14.5 SQL Injection

Detour

- Possible attacks
  - Authenticate without password
    - User enters a known username, e.g., admin
    - User enters as password ' OR 'a'='a
    - This results to
      SELECT * **FROM** users
      WHERE username='admin' **AND** password=' ' OR 'a'='a';
  - Delete a table
    - password = a'; DROP TABLE users; --
    - **SELECT** * **FROM** users
      WHERE username='admin'
      AND password='a'; DROP TABLE users; --';

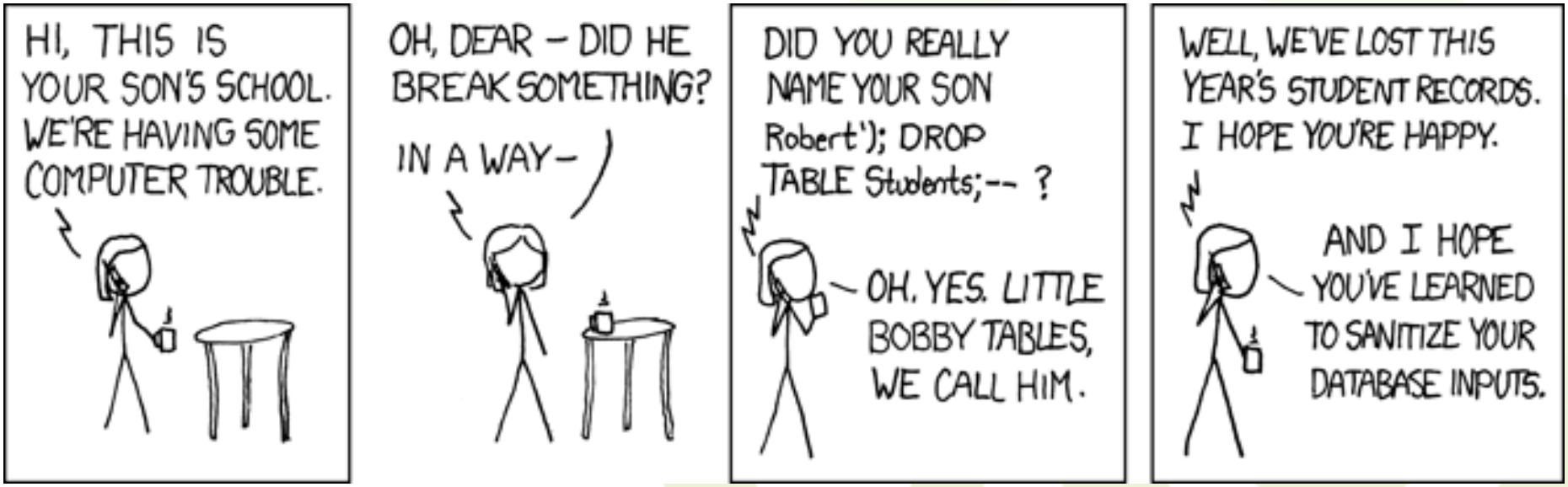
SQL comment

14.5 SQL Injection

Detour

– Even worse – capture the whole system!

- Some DBMS systems provide stored procedures to access the underlying operating system itself (e.g. MS SQL)
- Input: '; **EXEC** xp_cmdshell '**format c: /s**';



14.5 SQL Injection

Detour

- What hackers usually do
  - Hackers usually don't know the queries, tables, and inner workings of applications
    - Vulnerabilities need to be **"discovered"**
  - Start with entering information containing any SQL **control characters** (e.g. ')
    - If this results into an error, the application is potentially prone to injection attacks
  - Inject SQL code in order to guess the **structure** of the tables and columns, and also the **security boundaries** of the system
    - Observe the **error codes** to validate your guesses
  - As soon as the extend of the vulnerability data schema is known, data can be freely manipulated or stolen

14.5 SQL Injection

Detour

- How to **prevent injection attacks?**
- **Sanitize the input!**
  - Restrict all user input to only **safe characters** (i.e. remove control characters)
  - Will also delete characters which might be needed in the input (e.g. ')
  - Won't protect you in case of integer values
    - ... `WHERE id = 17 OR 1 = 1`

14.5 SQL Injection

Detour

- **Quote and escape the input**
  - Escape all control characters
  - Quote the input in a proper way
    - This might be quite tricky and often depends on the DBMS
    - Most database APIs provide special functions for quoting and escaping, use them!
      - e.g. mysql_real_escape_string() in PHP
  - Example:
    Default escape character ' , DBMS specific escape \
    - Input: \'; DROP TABLE users; --
      Default escape and quote: '\'; DROP TABLE users; --'
      Result: ... WHERE email= '\'; DROP TABLE users; --'

14.5 SQL Injection

Detour

- **Use strongly typed parameters**
  - Cast each user input to its intended data type
    - Prevents e.g. integer input with injected code
    - Together with sanitized input or escaping and quoting, typing provides a acceptable minimum amount of protection
- **Use prepared statements**
  - When using prepared statements, the DBMS **automatically** escapes and safely casts the input
    - User input is **"just data"** and won't be interpreted as statements
  - Besides that, prepared statements may **increase your query performance**

14.5 SQL Injection

Detour

- **Isolate your Web/DB server**
  - Put your servers in a secure DMZ (**De**-**M**ilitarized **Z**one) with very limited network capabilities
    - Even if the attacker is able to completely capture the machine, he/she won't be able to do much harm
- **Restrict your error reporting**
  - Many programming frameworks are by default configured into developer mode
  - On failure, they report in detail what went wrong
    - E.g., display the faulty query and excerpts from the call hierarchy or the DB schema
    - This information is very helpful in finding security vulnerabilities, so don't give it to your foes!

That‘s all folks…

Relational Databases 2

Coming up next

- In this lecture you learned
  how to use relational databases
  - Data modeling
  - Querying
  - Relational theory
  - Using DBs in applications
- What we did not tell you:
  How do relational databases **really** work?

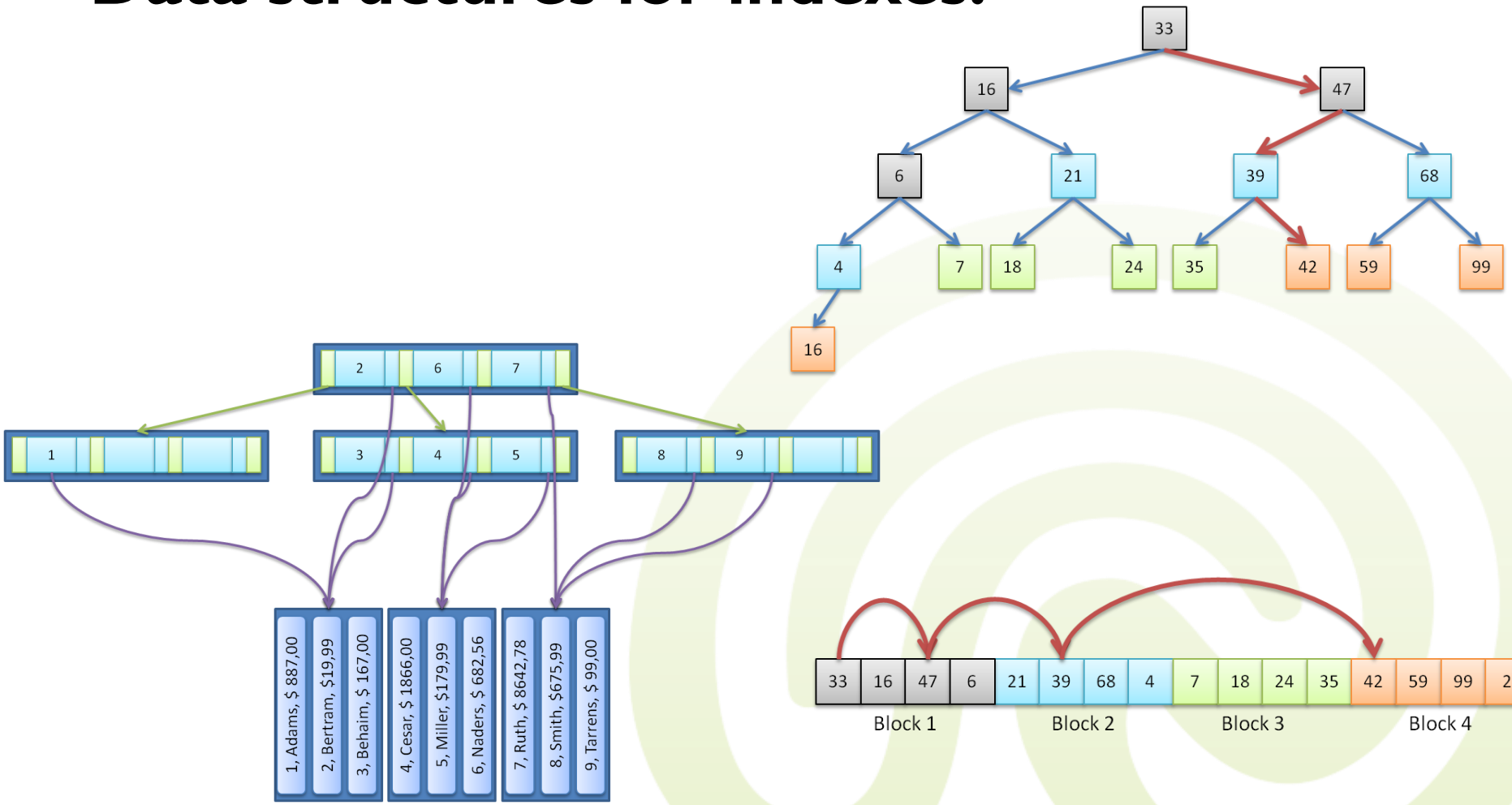
Relational Databases 2

- What we will do in Relational Databases 2:
  - The architecture of a DBMS
  - Storing data on hard disks
  - Indexing
  - Query evaluation and optimization
  - Transactions and ACID
  - Recovery

Relational Databases 2

Coming up next

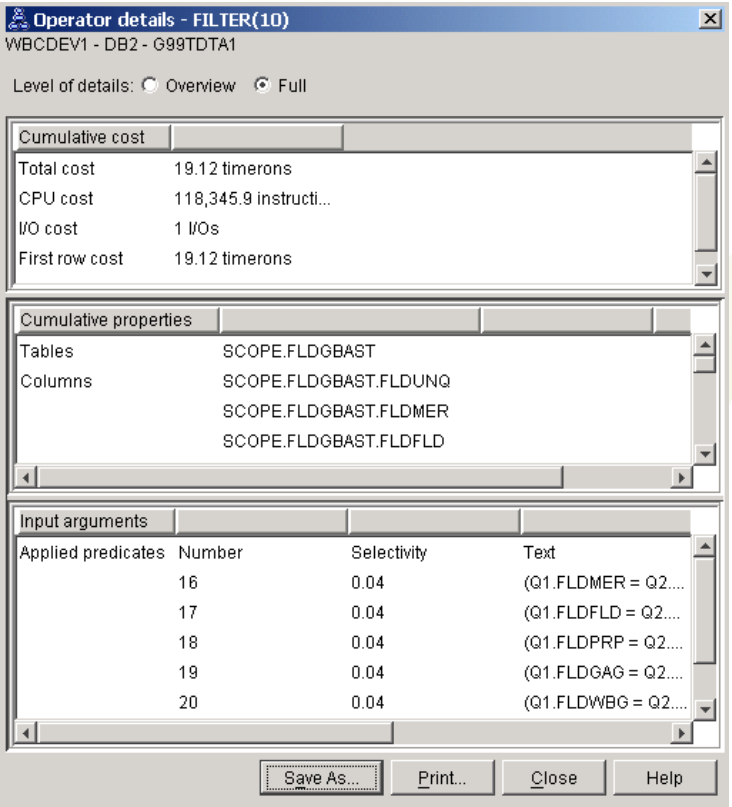
- **Data structures for indexes!**

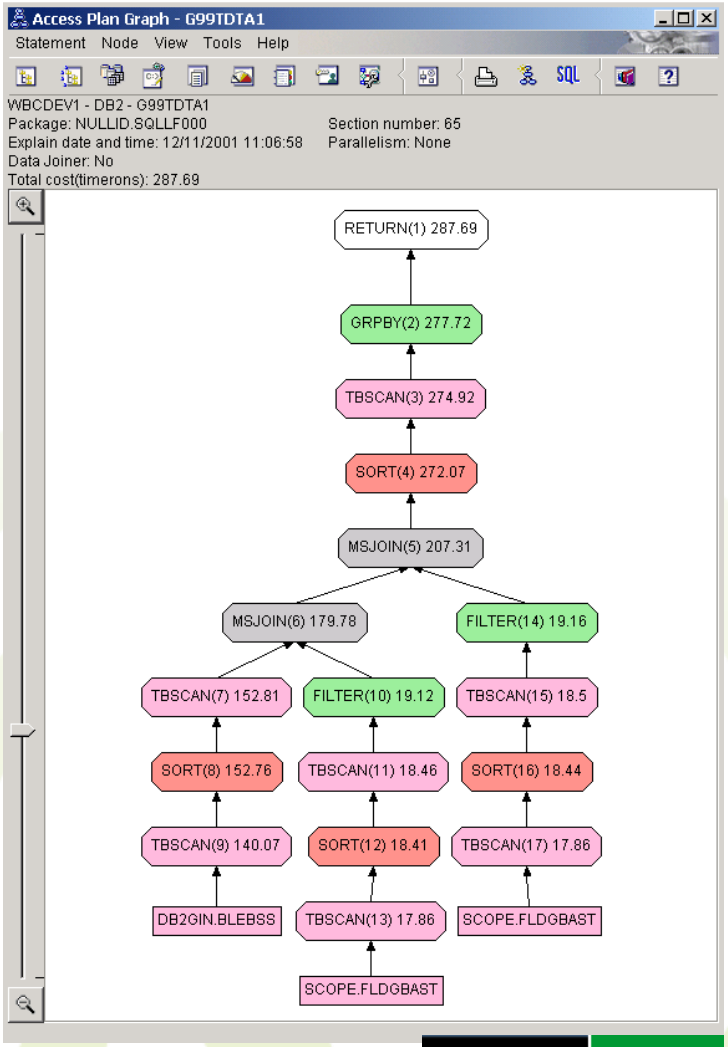


Relational Databases 2

Coming up next

- **Query optimization!**

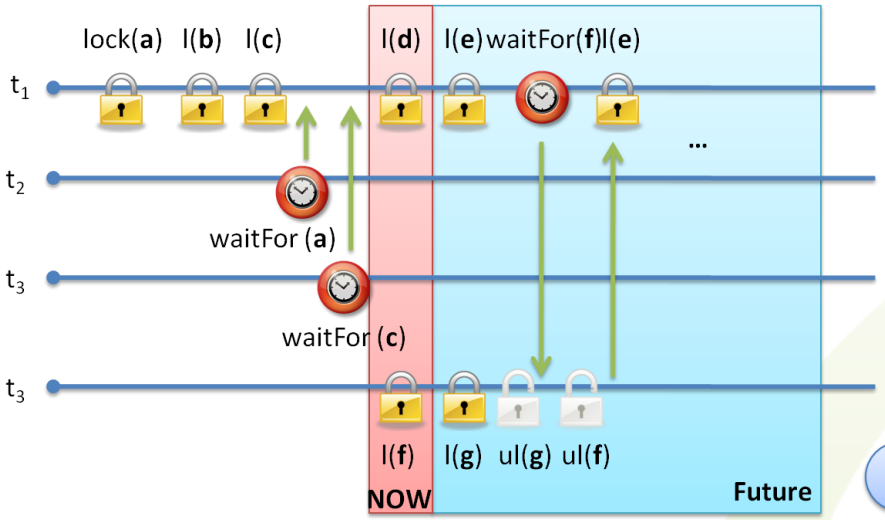


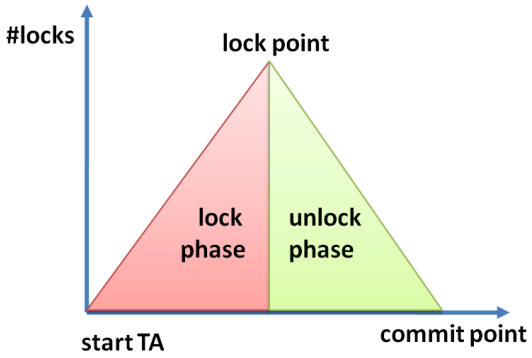


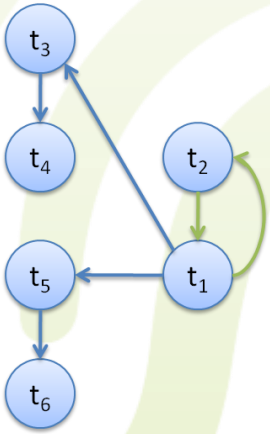
Relational Databases 2

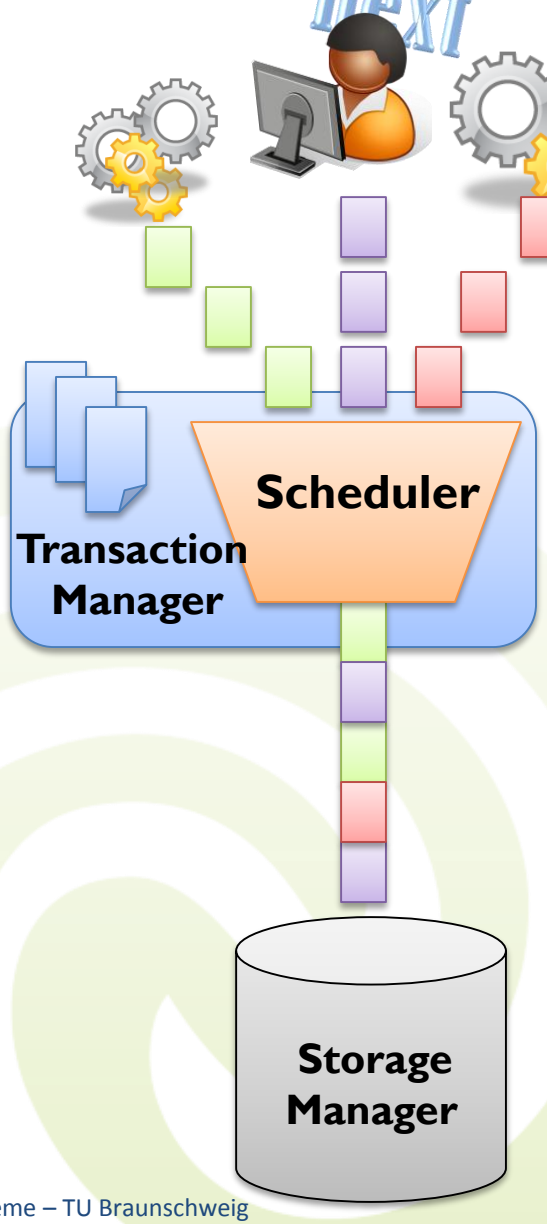
Coming up next

- **Implementing transactions!**









IR & Web Search

Coming up next

- Extremely relevant for practical applications is the retrieval of **textual documents**
  - The importance of **information retrieval** (IR) was already recognized in the 1940s
  - In contrast to relational data, texts are **unstructured**
  - The goal is to find documents from a large collection that are **relevant** with respect to the user's **information need**

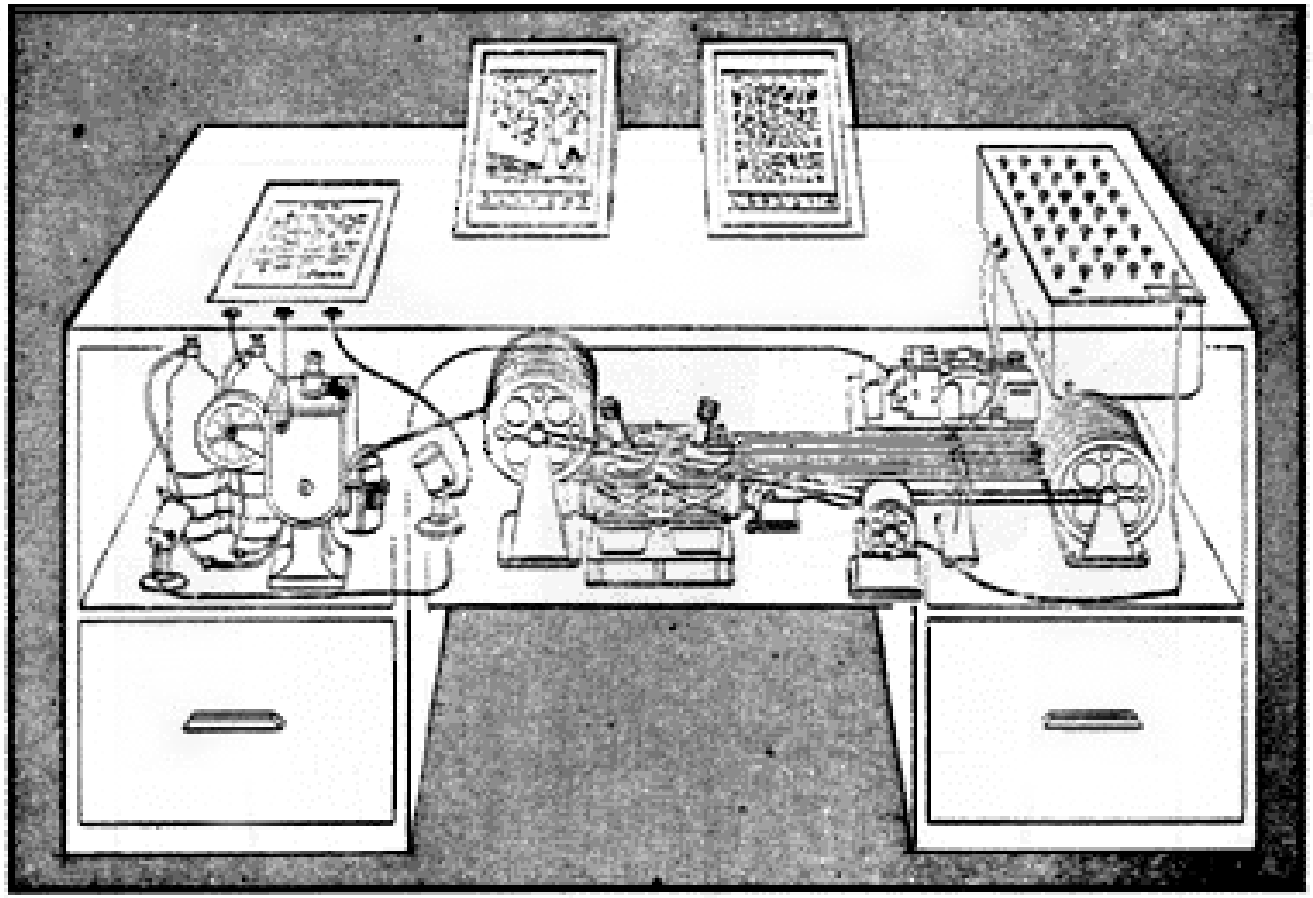
IR & Web Search

Coming up next

- Origins in period immediately after World War II
  - Tremendous scientific progress during the war
  - Rapid growth in amount of scientific publications available
- The **"Memex Machine"**
  - Conceived by Vannevar Bush, one of President Roosevelt's science advisors
  - Outlined in 1945 Atlantic Monthly article titled "As We May Think"
  - Foreshadows the development of hypertext (the Web) and information retrieval systems

IR & Web Search

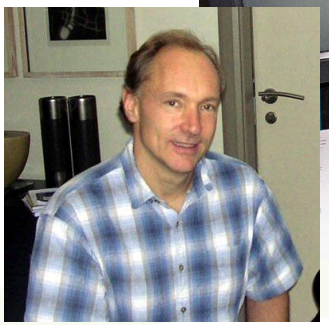
Coming up next

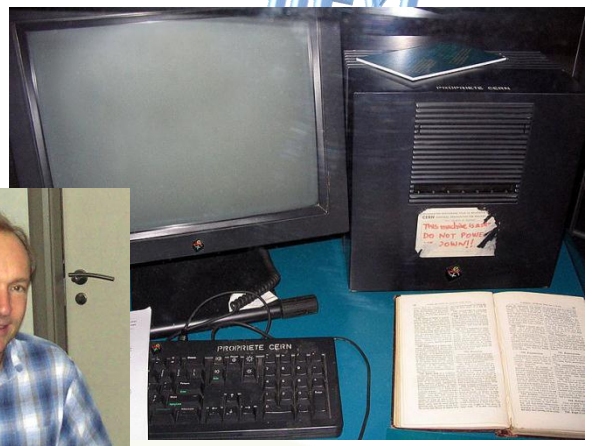


Memex in the form of a desk would instantly bring files and material on any subject to the operator's fingertips. Slanting translucent viewing screens magnify supermicrofilm filed by code numbers. At left is a mechanism which automatically photographs longhand notes, pictures and letters, then files them in the desk for future reference (*LIFE* 19(11), p. 123).

IR & Web Search

Coming up next





- The **WWW** started with a **telephone book** maintained by **Tim Berners-Lee** at CERN and quickly developed into a vast variety of interconnected servers

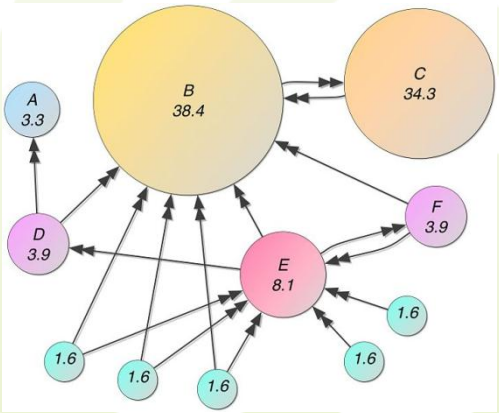
- The first Web search engines (1993-1994) all relied **only on then-classical IR techniques**
  - Infoseek, Lycos, AltaVista, Inktomi, HotBot, etc.

IR & Web Search

Coming up next

Google™

- In 1998, **Google** was founded
  - They also took the **structure of the Web** into account
  - Link analysis favored pages that are pointed to **from many others** and **from more important** pages
- The innovation was the **PageRank algorithm** invented by Google's founders Larry Page and Sergey Brin
  - The **relative importance** of a site is derived from the hyperlinks pointing to it
  - Links propagate scores through the system
  - Getting many links from important sites improves the "belief" that a site is relevant regarding a topic



Multimedia Databases

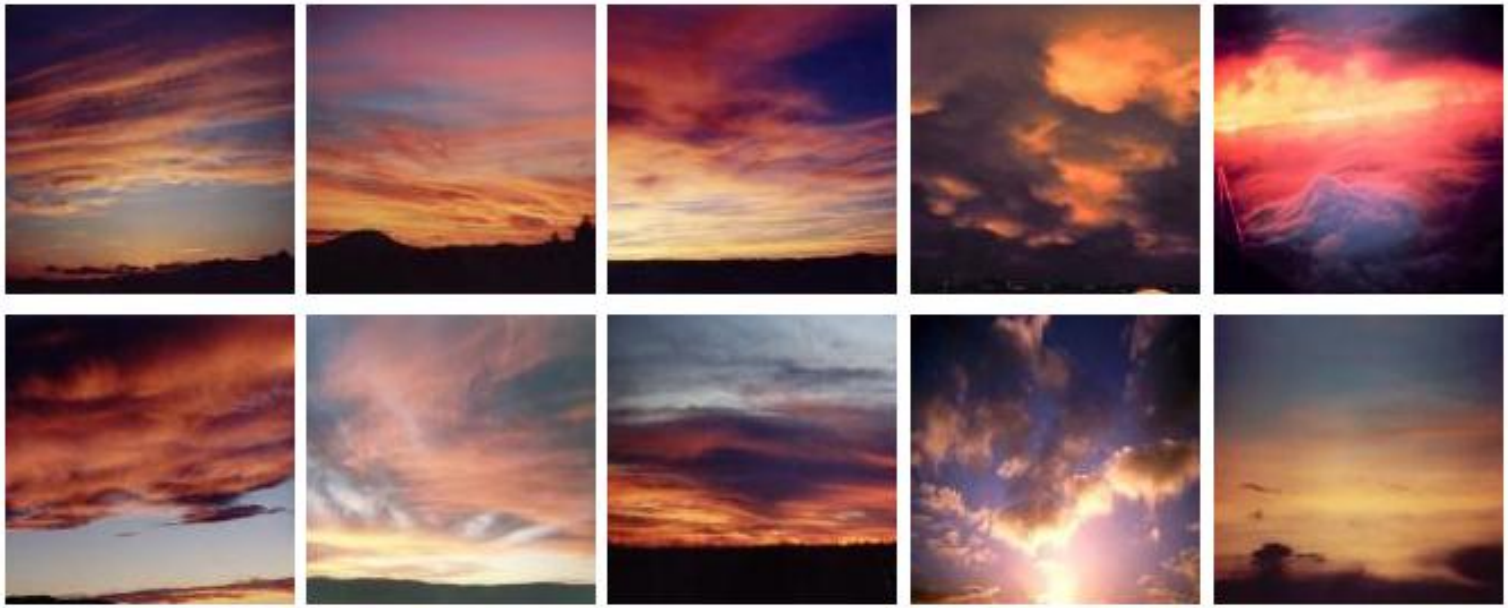
Coming up next

- Relational databases efficiently store and retrieve **structured data**
  - Bank accounts, customer data,…
- How to achieve **persistent storage of media** like
  - Text documents
  - Vectorgraphics, CAD
  - Image, audio, video
- What about **content-based retrieval**?
  - Efficiently searching media content
  - Standardization of meta-data (e.g., MPEG-7, MPEG-21)

Multimedia Databases

Coming up next

- Find all images in the database that show a **sun set**!

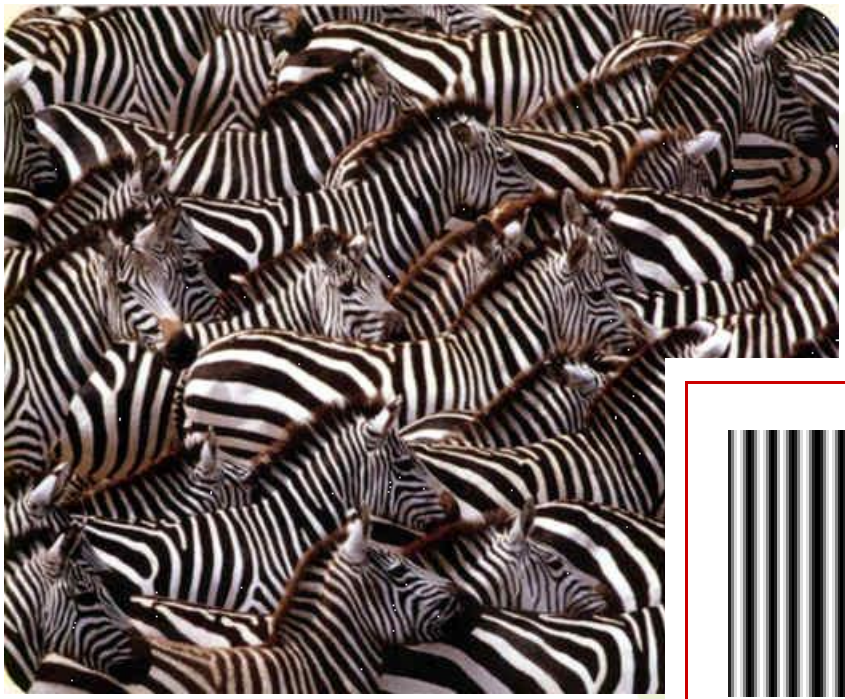


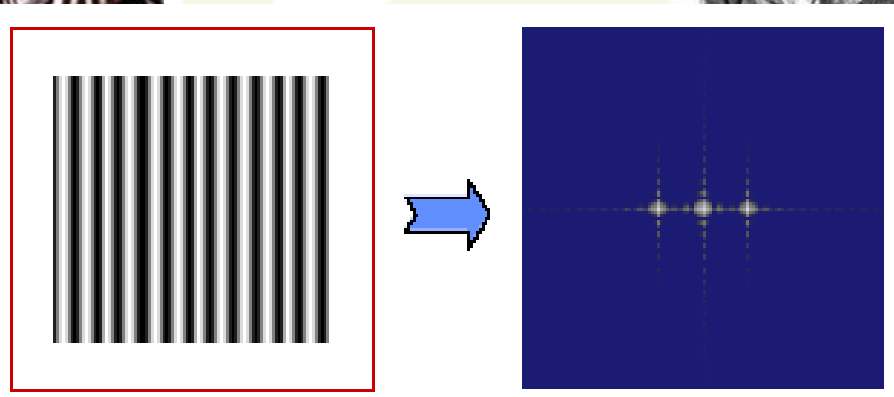
- What are their **common characteristics**? Can we only retrieve them by meta-data annotations?

Multimedia Databases

Coming up next

- **Different types of features** can be combined with other features to aid retrieval
  - For instance Fourier transform for textures

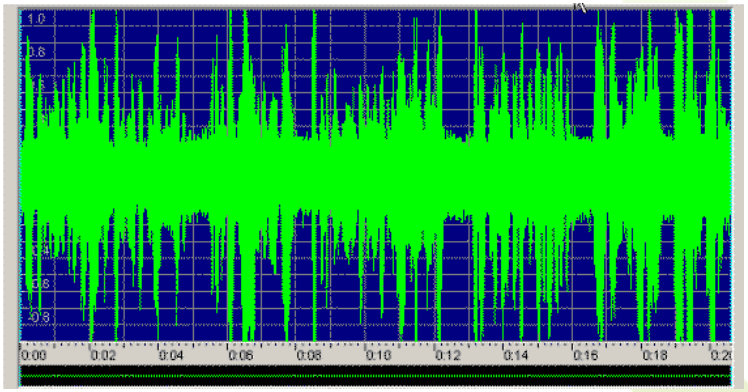


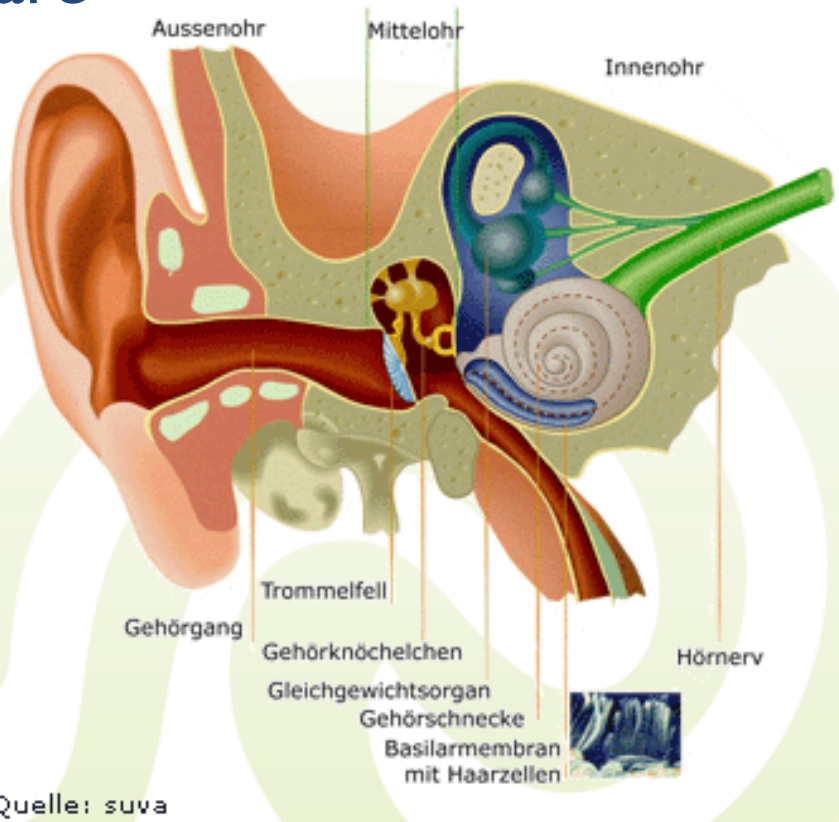


Multimedia Databases

Coming up next

- Also **continuous data** can be described
  - For perception of audio data **psychoacoustic models** are helpful
  - Waveforms can actually be described by some features similar to image features

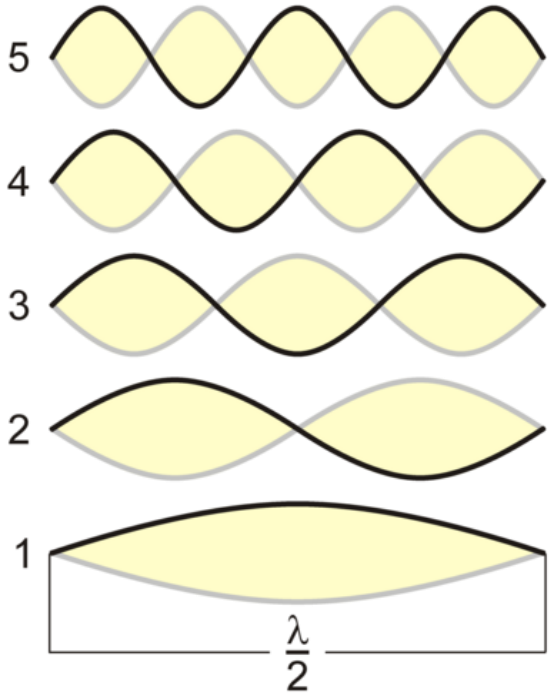


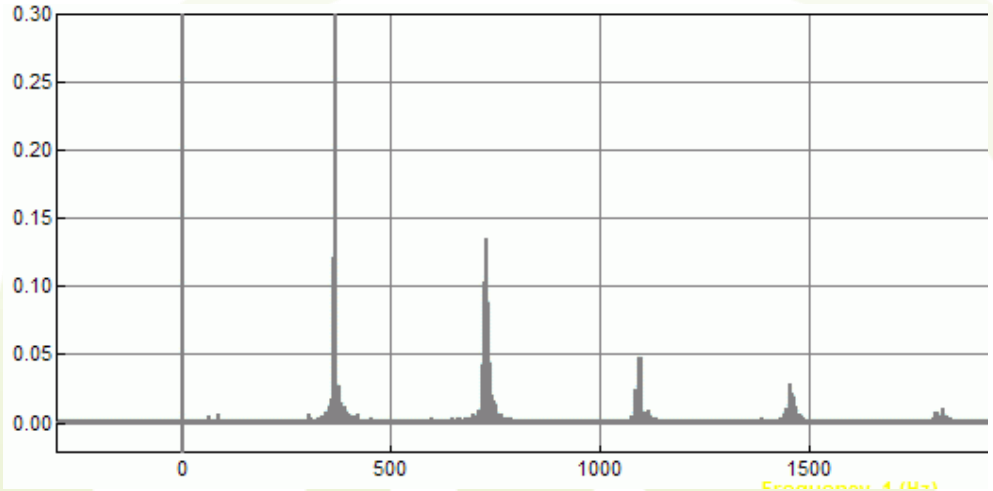


Multimedia Databases

Coming up next

- **Harmonies** in music allow to recognize and compare **melodies**
  - **Query by Humming and 'sounds like' search**

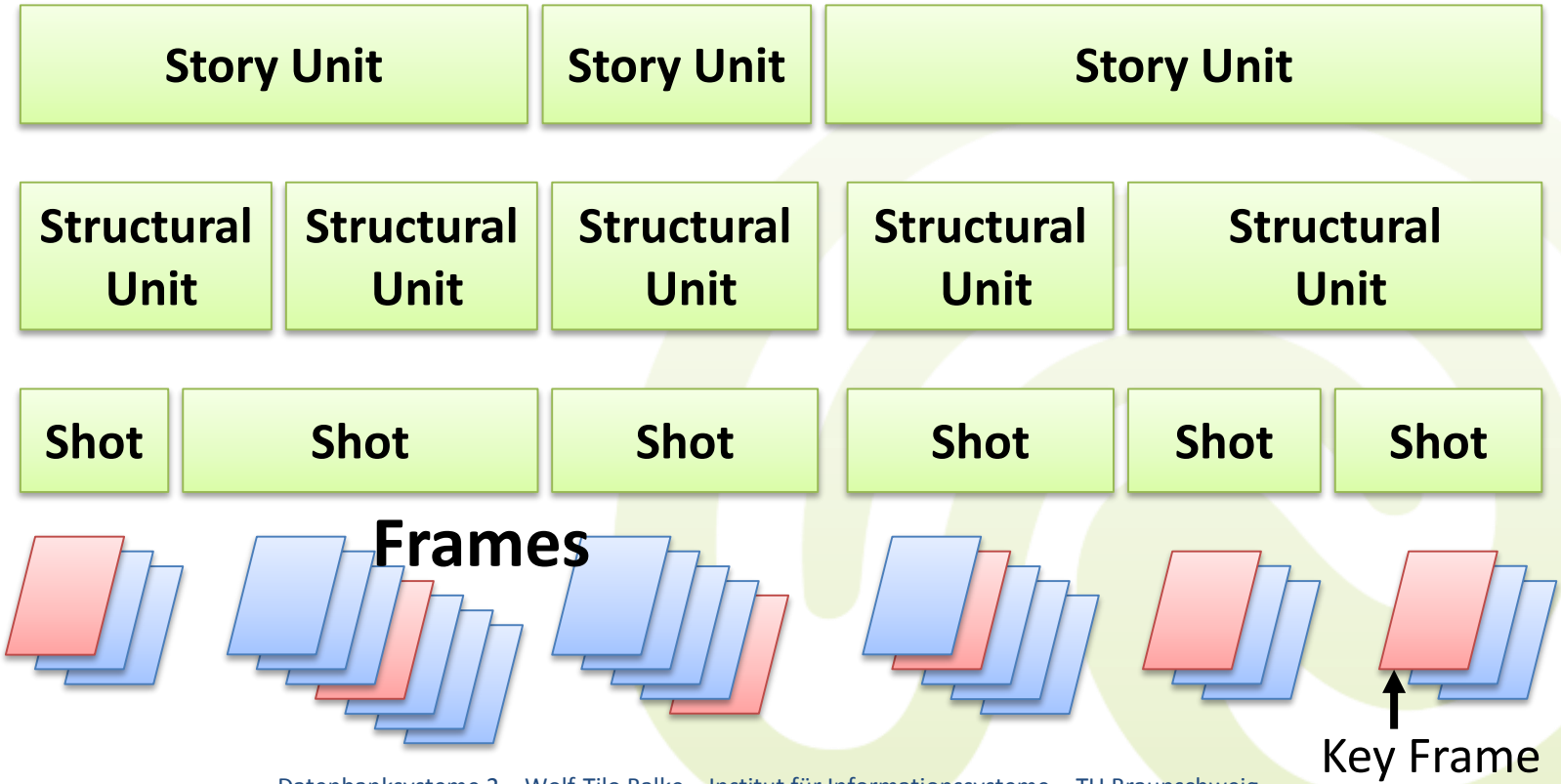




Multimedia Databases

Coming up next

- Videos **combine** many techniques used in audio and image retrieval and interleave them respecting the **structure of the video**



Multimedia Databases

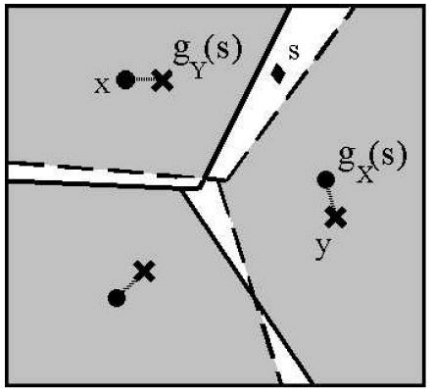
Coming up next

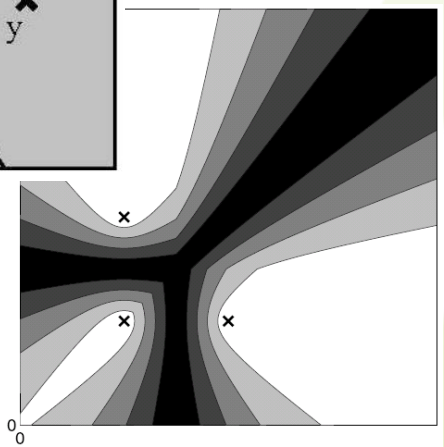
- The Video is broken down to its **shots** and the shots are individually compared for similarity
  - Efficient methods needed for **shot detection** and effective **video similarity** measures

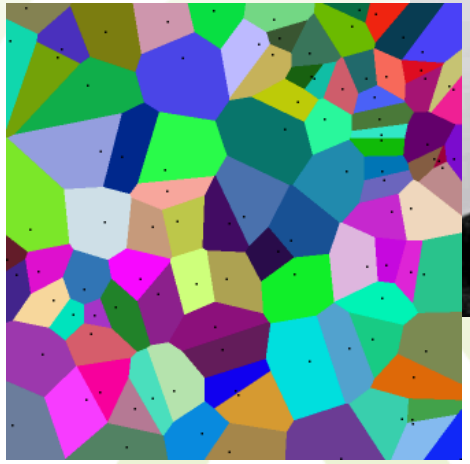
Multimedia Databases

Coming up next

- **Clustering techniques** allow to estimate the similarity of entire video sequences and movies
  - Very interesting, e.g., for finding movies of similar genre, or detecting copyright infringements



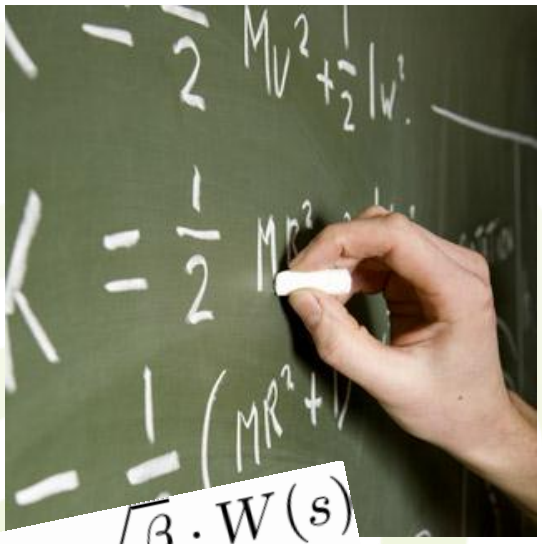




Multimedia Databases

Coming up next

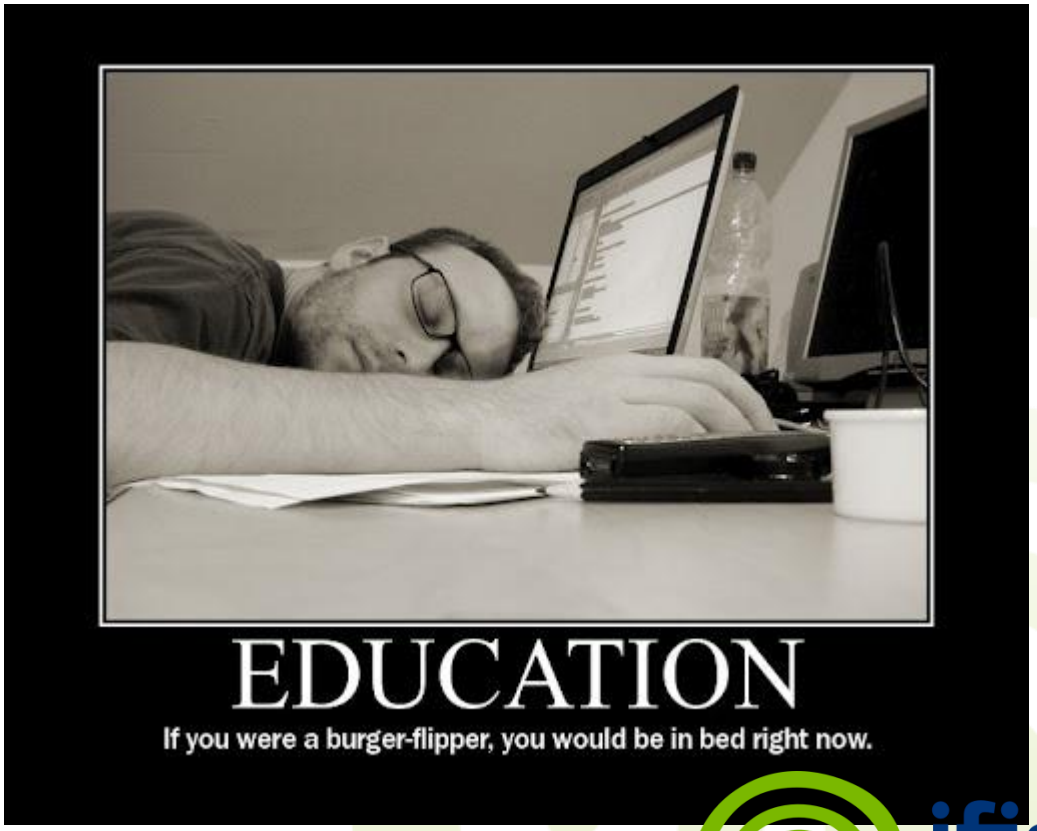
- In summary: a variety of mathematical techniques can be used for describing **media perception**
  - Fourier transform, wavelets, random field models, multi resolution analysis, etc.
  - They describe different aspects of images, audio, and video files



$$y_k = \sum_{l=0}^{n/2} a_l \cdot \cos\left(l \cdot 2\pi \cdot \frac{k}{n}\right) + \sum_{l=0}^{n/2} b_l \cdot \sin\left(l \cdot 2\pi \cdot \frac{k}{n}\right)$$

$$F(s) = \sum_{t \in N} \theta(t) \cdot F(s+t) + \sqrt{\beta} \cdot W(s)$$

See you next Semester



ifis
Institut für Informationssysteme
Technische Universität Braunschweig