Principles of Database Systems (CS307)

Lecture 11: Physical Storage System; Trigger

Ran Cheng

Department of Computer Science and Engineering

Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult, Dr. Yuxin Ma and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

Physical Storage System

Physical Storage System

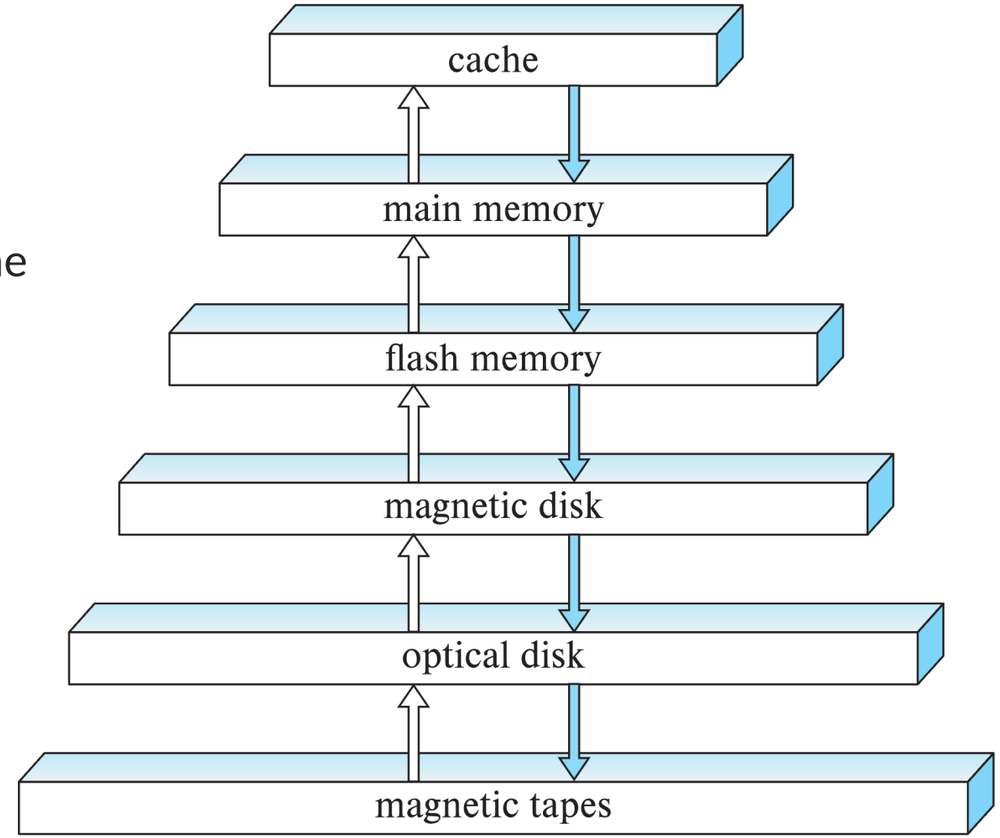
- The hardware where data is recorded

Classification of Physical Storage Media

- Can differentiate storage into:
  - Volatile (易失, 易挥发) storage: loses contents when power is switched off
  - Non-volatile storage:
    - Contents persist even when power is switched off.
    - Includes secondary and tertiary storage, as well as batter-backed up main-memory.
- Factors affecting choice of storage media include
  - Speed with which data can be accessed
  - Cost per unit of data
  - Reliability

Storage Hierarchy

- Primary storage
  - Fastest media but volatile (cache, main memory).
- Secondary storage
  - Next level in hierarchy, non-volatile, moderately fast access time
    - … also called on-line storage
  - E.g., flash memory, magnetic disks
- Tertiary storage
  - Lowest level in hierarchy, non-volatile, slow access time
    - … also called off-line storage and used for archival storage
  - e.g., magnetic tape, optical storage
    - Magnetic tape
      - Sequential access, 1 to 12 TB capacity
      - A few drives with many tapes
      - Juke boxes with petabytes (1000's of TB) of storage

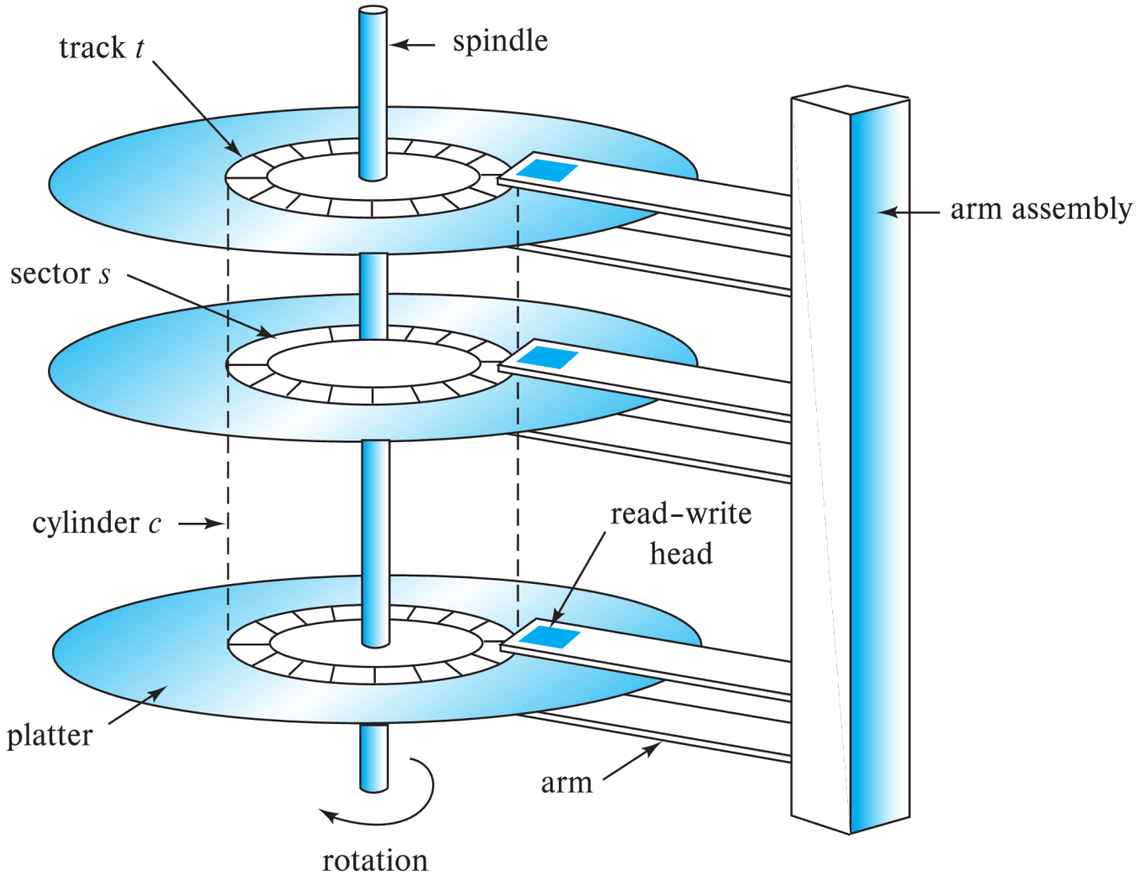


Storage Interfaces

- Disk interface standards families
  - SATA (Serial Advanced Technology Attachment)
    - SATA 3 supports data transfer speeds of up to 6 gigabits/sec
  - SAS (Serial Attached SCSI)
    - SAS Version 3 supports 12 gigabits/sec
  - NVMe (Non-Volatile Memory Express) interface
    - Works with PCIe connectors to support lower latency and higher transfer rates
    - Supports data transfer rates of up to 24 gigabits/sec

- Disks usually connected directly to computer system, however…
  - In Storage Area Networks (SAN), a large number of disks are connected by a high-speed network to a number of servers
  - In Network Attached Storage (NAS) networked storage provides a file system interface using networked file system protocol, instead of providing a disk system interface

Magnetic Hard Disk Mechanism



Schematic diagram of magnetic disk drive

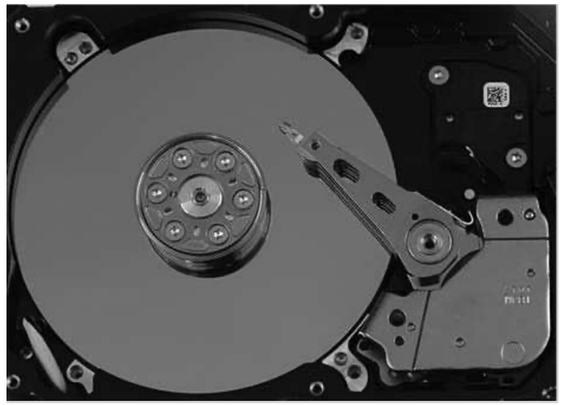


Photo of a magnetic disk drive

Magnetic Hard Disk Mechanism

- Read-write head
- Surface of platter divided into circular tracks
  - Over 50K-100K tracks per platter on typical hard disks
- Each track is divided into sectors.
  - A sector is the smallest unit of data that can be read or written.
  - Sector size typically 512 bytes (modern OS requires 4KB)
  - Typical sectors per track: 500 to 1000 (on inner tracks) to 1000 to 2000 (on outer tracks)
- To read/write a sector
  - Disk arm swings to position head on right track
  - Platter spins continually; data is read/written as sector passes under head
- Head-disk assemblies
  - Multiple disk platters on a single spindle (1 to 5 usually)
  - One head per platter, mounted on a common arm.
- Cylinder i consists of ith track of all the platters

Magnetic Hard Disk Mechanism

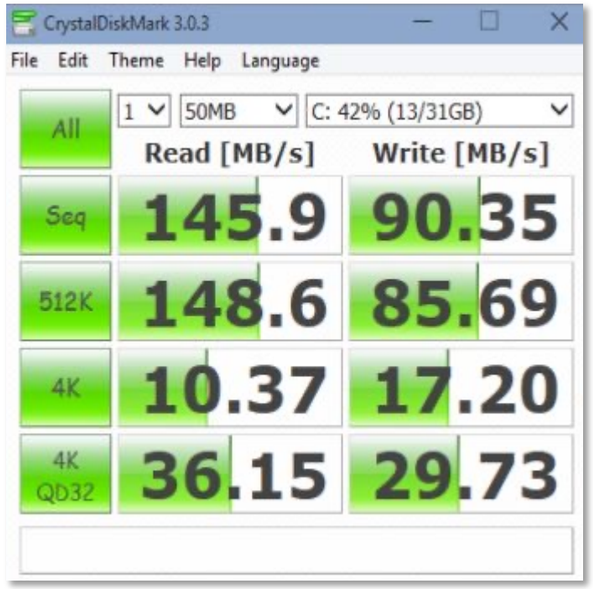
- Disk controller（从指令到物理动作）
  - Interfaces between the computer system and the disk drive hardware.
    - 令：Accepts high-level commands to read or write a sector
    - 动：Initiates actions such as moving the disk arm to the right track and actually reading or writing the data
    - 读：Computes and attaches checksums（校验和） to each sector to verify that data is read back correctly
      - If data is corrupted, with very high probability stored checksum won't match recomputed checksum
    - 写：Ensures successful writing by reading back sector after writing it
    - 换：Performs remapping of bad sectors

Performance Measures of Disks

- Access time: The time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - Seek time – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - Rotational latency – time it takes for the sector to be accessed to appear under the head.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
    - Average latency is 1/2 of the above latency.
  - Overall latency is 5 to 20 msec depending on disk model
- Data-transfer rate: The rate at which data can be retrieved from or stored to the disk.
  - 25 to 200 MB per second max rate, lower for inner tracks

Performance Measures of Disks

- Disk block is a logical unit for storage allocation and retrieval
  - 4 to 16 kilobytes typically
    - Smaller blocks: more transfers from disk
    - Larger blocks: more space wasted due to partially filled blocks
- Sequential access pattern
  - Successive requests are for successive disk blocks
  - Disk seek required only for first block
- Random access pattern
  - Successive requests are for blocks that can be anywhere on disk
  - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks
- I/O operations per second (IOPS)
  - Number of random block reads that a disk can support per second
  - 50 to 200 IOPS on current generation magnetic disks



Performance Measures of Disks

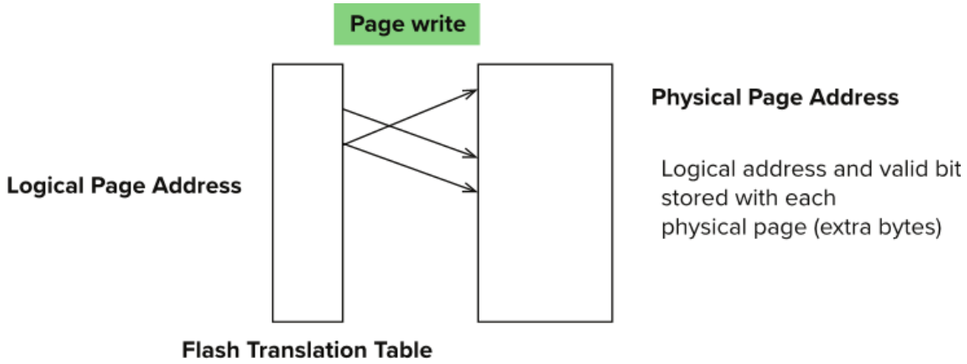
- Mean time to failure (MTTF) – the average time the disk is expected to run continuously without any failure.
  - Typically, 3 to 5 years
  - Probability of failure of new disks is quite low, corresponding to a “theoretical MTTF” of 500,000 to 1,200,000 hours for a new disk
    - E.g., an MTTF of 1,200,000 hours for a new disk means that given 1000 relatively new disks, on an average one will fail every 1200 hours
  - MTTF decreases as disk ages

Flash Storage

- NOR flash vs NAND flash
- NAND flash
  - Used widely for storage, cheaper than NOR flash
  - Requires page-at-a-time read (page: 512 bytes to 4 KB)
    - 20 to 100 microseconds for a page read
    - Not much difference between sequential and random read
  - Page can only be written once
    - Must be erased to allow rewrite
- Solid state disks (SSD)
  - Use standard block-oriented disk interfaces, but store data on multiple flash storage devices internally
  - Transfer rate of up to 500 MB/sec using SATA, and up to 3 GB/sec using NVMe PCIe

Flash Storage

- Erase happens in units of erase block
  - Takes 2 to 5 millisecs
  - Erase block typically 256 KB to 1 MB (128 to 256 pages)
- Remapping of logical page addresses to physical page addresses avoids waiting for erase
- Flash translation table tracks mapping
  - Also stored in a label field of flash page
  - Remapping carried out by flash translation layer



- After 100,000 to 1,000,000 erases, erase block becomes unreliable and cannot be used
  - Wear leveling

SSD Performance Metrics

- Random reads/writes per second
  - Typical 4 KB reads: 10,000 reads per second (10,000 IOPS)
  - Typical 4KB writes: 40,000 IOPS
  - SSDs support parallel reads
    - Typical 4KB reads:
      - 100,000 IOPS with 32 requests in parallel (QD-32) on SATA
      - 350,000 IOPS with QD-32 on NVMe PCIe
    - Typical 4KB writes:
      - 100,000 IOPS with QD-32, even higher on some models
- Data transfer rate for sequential reads/writes
  - 400 MB/sec for SATA3, 2 to 3 GB/sec using NVMe PCIe

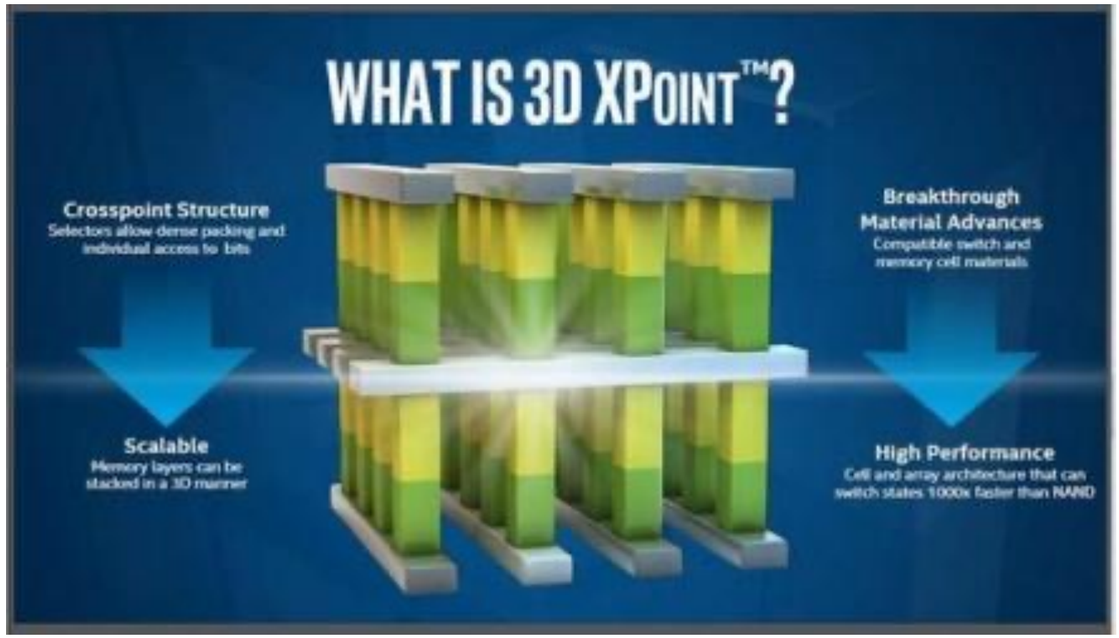
- Hybrid disks: Combine small amount of flash cache with larger magnetic disk

Storage Class Memory: Magnetic Tapes

- Hold large volumes of data and provide high transfer rates
  - Few GB for DAT (Digital Audio Tape) format, 10-40 GB with DLT (Digital Linear Tape) format, 100 GB+ with Ultrium format, and 330 GB with Ampex helical scan format
  - Transfer rates from few to 10s of MB/s
- Tapes are cheap, but cost of drives is very high
- Very slow access time in comparison to magnetic and optical disks
  - limited to sequential access.
  - Some formats (Accelis) provide faster seek (10s of seconds) at cost of lower capacity
- Used mainly for backup, for storage of infrequently used information, and as an off-line medium for transferring information from one system to another.
- Tape jukeboxes used for very large capacity storage
  - Multiple petabytes (10^15 bytes)

Storage Class Memory: 3D-XPoint

- 3D-XPoint memory technology pioneered by Intel
- Available as Intel Optane
  - SSD interface shipped from 2017
    - Allows lower latency than flash SSDs
  - Non-volatile memory interface announced in 2018
    - Supports direct access to words, at speeds comparable to main-memory speeds



RAID

- RAID: Redundant Arrays of Independent Disks
  - Disk organization techniques that manage a large numbers of disks, providing a view of a single disk of
    - high capacity and high speed by using multiple disks in parallel,
    - high reliability by storing data redundantly, so that data can be recovered even if a disk fails
- The chance that some disk out of a set of N disks will fail is much higher than the chance that a specific single disk will fail.
  - e.g., a system with 100 disks, each with MTTF (Mean Time To Failure) of 100,000 hours (approx. 11 years), will have a system MTTF of 1000 hours (approx. 41 days)
  - Techniques for using redundancy to avoid data loss are critical with large numbers of disks

Improvement of Reliability via Redundancy

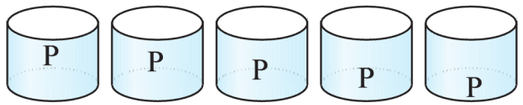
- Redundancy – store extra information that can be used to rebuild information lost in a disk failure
  - E.g., Mirroring (or shadowing)
    - Duplicate every disk. Logical disk consists of two physical disks.
    - Every write is carried out on both disks
      - Reads can take place from either disk
    - If one disk in a pair fails, data still available in the other
      - Data loss would occur only if a disk fails, and its mirror disk also fails before the system is repaired
      - Probability of combined event is very small
      - Except for dependent failure modes such as fire or building collapse or electrical power surges
  - Mean time to data loss depends on mean time to failure, and mean time to repair
    - E.g., MTTF of 100,000 hours, mean time to repair of 10 hours gives mean time to data loss of 500*106 hours (or 57,000 years) for a mirrored pair of disks (ignoring dependent failure modes)

RAID Levels

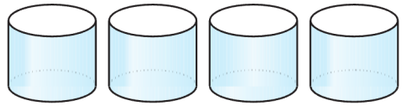
- Schemes to provide redundancy at lower cost by using disk striping combined with parity bits
  - Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics

  - RAID 0: Block striping (分段); non-redundant
  - RAID 1: Mirrored disks with block striping
  - RAID 10: Combination of striping and mirroring

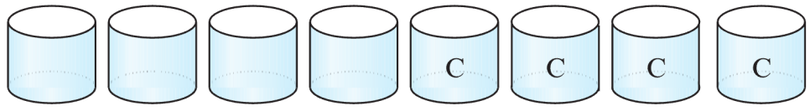
  - RAID 5: Block-interleaved distributed parity
  - RAID 6: P+Q Redundancy scheme



(c) RAID 5: block-interleaved distributed parity



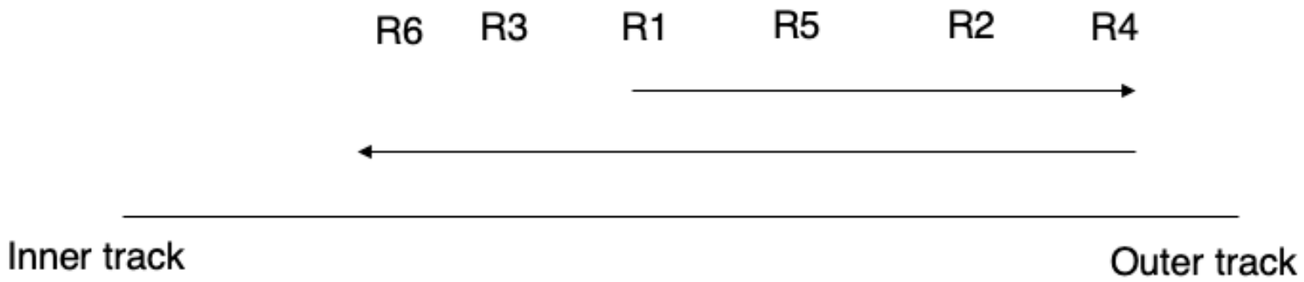
(a) RAID 0: nonredundant striping



(b) RAID 1: mirrored disks

Optimization of Disk-Block Access

- Buffering
  - In-memory buffer to cache disk blocks
- Read-ahead
  - Read extra blocks from a track in anticipation that they will be requested soon
- Disk-arm-scheduling algorithms
  - Re-order block requests so that disk arm movement is minimized
  - E.g., elevator algorithm



Optimization of Disk-Block Access

- File organization
  - Allocate blocks of a file in as contiguous a manner as possible
  - Allocation in units of extents
  - Files may get fragmented
    - E.g., if free blocks on disk are scattered, and newly created file has its blocks scattered over the disk
    - Sequential access to a fragmented file results in increased disk arm movement
    - Some systems have utilities to defragment the file system, in order to speed up file access
- Non-volatile write buffers
  - Temporarily store the written data
    - … and immediately notifies the OS that writing is completed without errors
  - Write data into the disk when idle
    - … with some optimizations

Trigger

Trigger - Actions When Changing Tables

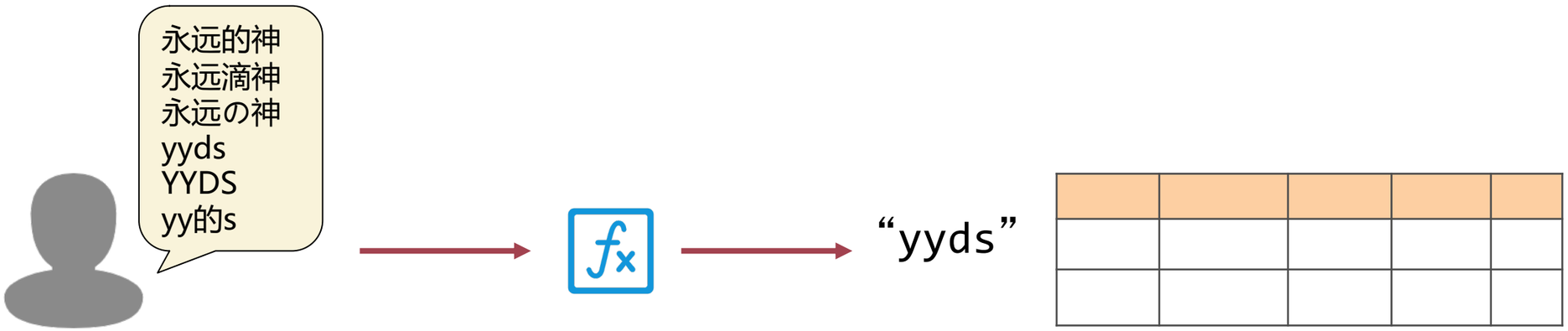
> A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed.
>
> -- Chapter 39, PostgreSQL Documentation
>
> A trigger is a statement that the system executes automatically as a side effect of a modification to the database.
>
> -- Chapter 5.3, Database System Concepts, 7th

- We can attach “actions” to a table
  - They will be executed automatically whenever the data in the table changes
- Purpose of using triggers
  - Validate data
  - Checking complex rules
  - Manage data redundancy

Purpose of Using Triggers

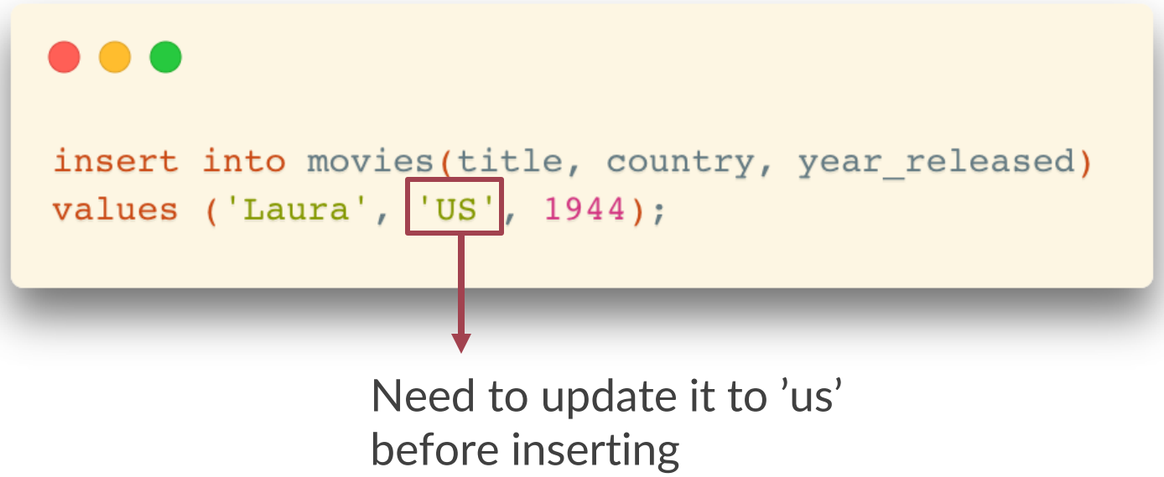
- Validate data
  - Some data are badly processed in programs before sending to the database
  - We need to validate such data before inserting them into the database

  - “On-the-fly” modification
    - Change the input directly when the input arrives



Purpose of Using Triggers

- Validate data
  - Example: insert a row in the movies table
    - In the JDBC program, an `insert` request is written like the following:



  - Although,
    - Such validation or transformation should be better handled by the application programs

Purpose of Using Triggers

- Check complex rules
  - Sometimes, the business rules are so complex that it cannot be checked via declarative constraints

Purpose of Using Triggers

- Manage data redundancy
  - Some redundancy issues may not be avoided by simply adding constraints

  - For example: We inserted the same movie but in different languages

```sql
-- US
insert into movies(title, country, year_released)
values ('The Matrix', 'us', 1999);

-- China (Mainland)
insert into movies(title, country, year_released)
values ('黑客帝国', 'us', 1999);

-- Hongkong
insert into movies(title, country, year_released
values ('22世紀殺人網絡', 'us', 1999);
```

It satisfies the constraint of uniqueness on (`title, country, year_released`)

- ... but they represent the same movie

Trigger Activation

- Two key points:
  - When to fire a trigger?
  - What (command) fires a trigger?

Trigger Activation

- When to fire a trigger?
  - In general: “During the change of data”
    - … but we need a detailed discussion

--- Note: “During the change” means `select` queries won’t fire a trigger.

Trigger Activation: When

- Example: Insert a set of rows with “`insert into select`”
  - One statement, multiple rows

```
insert into movies(title, country, year_released)
select titre, 'fr', annee
from films_francais;
```

Trigger Activation: When

- Example: Insert a set of rows with "`insert into select`"
  - One statement, multiple rows

```
insert into movies(title, country, year_released)
select titre, 'fr', annee
from films_francais;
```

  - Option 1: Fire a trigger only once for the statement
    - Before the first row is inserted, or after the last row is inserted
  - Option 2: Fire a trigger for each row
    - Before or after the row is inserted

Trigger Activation: When

- Different options between DBMS products

- Before statement
  - Before each row
  - After each row
- After statement

- Before statement
  - Before each row
  - After each row
- After statement

- Before statement
  - Before each row
  - After each row
- After statement

Trigger Activation: What

- What (command) fires a trigger?
  - `insert`
  - `update`
  - `delete`

Example of Triggers

- A (Toy) Example
  - For the following `people_1` table, count the number of movies when updating a person and save the result in the `num_movies` column

```sql
-- auto-generated definition
create table people_1
(
    peopleid   integer,
    first_name varchar(30),
    surname    varchar(30),
    born       integer,
    died       integer,
    gender     bpchar,
    num_movies integer
);
```

	peopleid	first_name	surname	born	died	gender	num_movies
1	13	Hiam	Abbass	1960	<null>	F	<null>
2	559	Aleksandr	Askoldov	1932	<null>	M	<null>
3	572	John	Astin	1930	<null>	M	<null>
4	585	Essence	Atkins	1972	<null>	F	<null>
5	598	Antonella	Attili	1963	<null>	F	<null>
6	611	Stéphane	Audran	1932	<null>	F	<null>
7	624	William	Austin	1884	1975	M	<null>
8	637	Tex	Avery	1908	1980	M	<null>
9	650	Dan	Aykroyd	1952	<null>	M	<null>
10	520	Zackary	Arthur	2006	<null>	M	<null>
11	533	Oscar	Asche	1871	1936	M	<null>
12	546	Elizabeth	Ashley	1939	<null>	F	<null>

Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

- Create a trigger

Name of the trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

- Create a trigger

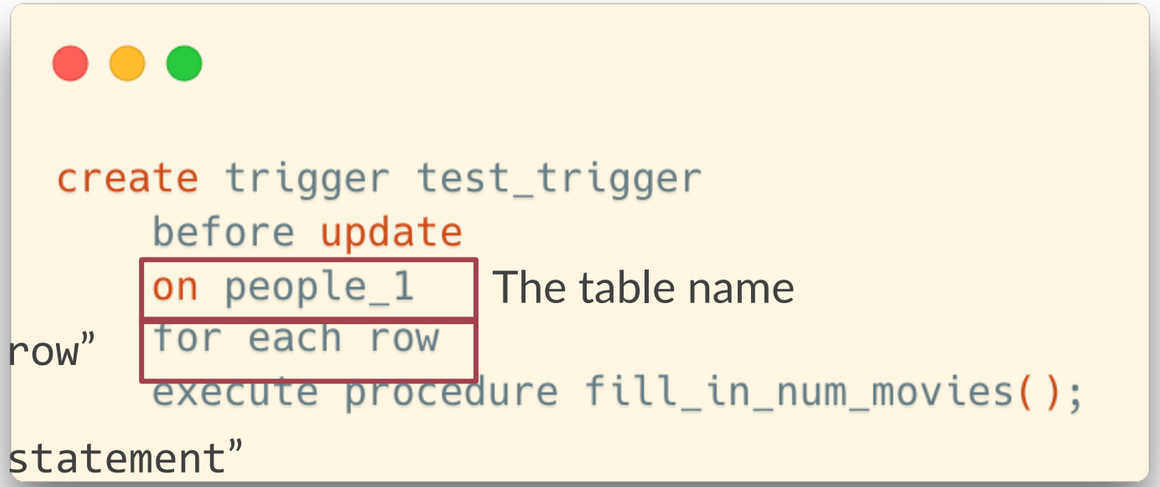
{ BEFORE | AFTER | INSTEAD OF } { event [OR ...] }

- Specify when the trigger will be executed
  - `before` | `after`
- … and on what operations the trigger will be executed
  - `insert [or update [or delete]]`

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

Example of Triggers

- Create a trigger



Example of Triggers

- Create a trigger

```
create trigger test_trigger
    before update
    on people_1
    for each row
    execute procedure fill_in_num_movies();
```

The actual procedure for the trigger

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

"trigger" is the return type

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

”new” and “old” are two internal variables that represents the row before and after the changes

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well

```
create or replace function fill_in_num_movies()
    returns trigger
as
$$
begin
    select count(distinct c.movieid)
    into new.num_movies
    from credits c
    where c.peopleid = new.peopleid;
    return new;
end;
$$ language plpgsql;
```

Remember to return the result which will be used in the update statement

Example of Triggers

- Create a trigger
  - Besides, a corresponding procedure should be created as well
    - Remember to create the procedure before creating the trigger

- Run test updates

```
-- create the procedure fill_in_num_movies() first

-- then, create the trigger

-- finally, we can run some test update statements
update people_1 set num_movies = 0 where people_1.peopleid <= 100;
```

Before and After Triggers

- Differences between before and after triggers
  - “Before” and “after” the operation is done (`insert`, `update`, `delete`)
  - If we want to update the incoming values in an `update` statement, the “before trigger” should be used since the incoming values have not been written to the table yet

Before and After Triggers

- Typical usage scenarios for trigger settings
  - Modify input on the fly
    - before insert / update
    - for each row
  - Check complex rules
    - before insert / update / delete
    - for each row
  - Manage data redundancy
    - after insert / update / delete
    - for each row

Example: Auditing

- One good example of managing some data redundancy is keeping an audit trail
  - It won't do anything for people who steal data
    - (remember that `select` cannot fire a trigger – although with the big products you can trace all queries)
  - … but it may be useful for checking people who modify data that they aren't supposed to modify

Example: Auditing

- Trace the insertions and updates to employees in a company

```
create table company(
   id int primary key     not null,
   name           text    not null,
   age            int     not null,
   address        char(50),
   salary         real
);


create table audit(
   emp_id int not null,
   change_type char(1) not null,
   change_date text not null
);
```

Example: Auditing

- Trace the insertions and updates to employees in a company

```
create trigger audit_trigger
    after insert or update
    on company
    for each row
execute procedure auditlogfunc();


create or replace function auditlogfunc() returns trigger as
$example_table$
begin
    insert into audit(emp_id, change_type, change_date)
    values (new.id,
            case
                when tg_op = 'UPDATE' then 'U'
                when tg_op = 'INSERT' then 'I'
                else 'X'
            end,
            current_timestamp);
    return new;
end ;
$example_table$ language plpgsql;
```

Example: Auditing

- Trace the insertions and updates to employees in a company

```
insert into company (id, name, age, address, salary)
values (2, 'Mike', 35, 'Arizona', 30000.00);
```

company

	id	name	age	address	salary
1	2	Mike	35	Arizona	30000

audit

	emp_id	change_type	change_date
1	2	I	2022-04-25 18:37:35.515151+00