

CS2102: Database Systems

Lecture 1 — Introduction & Relational Model

Overview

- **Why Database Management Systems (DBMS)?**

- Challenges for data-intensive applications
- From file-based data management to DBMS
- Core concepts of DBMS (transactions, data abstraction)

- **Relational Database Model**

- Motivation & history
- Core concepts: relation, domain, schema, etc.
- Integrity constraints

Common Challenges for Data-Intensive Applications

- Fast access to information
in huge volumes of data
→ **Efficiency**
- "All-or-nothing" changes to data
(e.g. bank transfer: debit + credit)
→ **Transactions**
- Parallel access and changes to data
→ **Data Integrity**

VISA

5,000 tps*

aMADEUS

(global travel booking platform)

100,000 tps*

 **Alibaba.com**[®]
(on Singles Day 2020)

544,000 tps*

Common Challenges for Data-Intensive Applications

- Fast and reliable handling of failures
(e.g., HDD/SDD/system crash, power outage, network disruption)

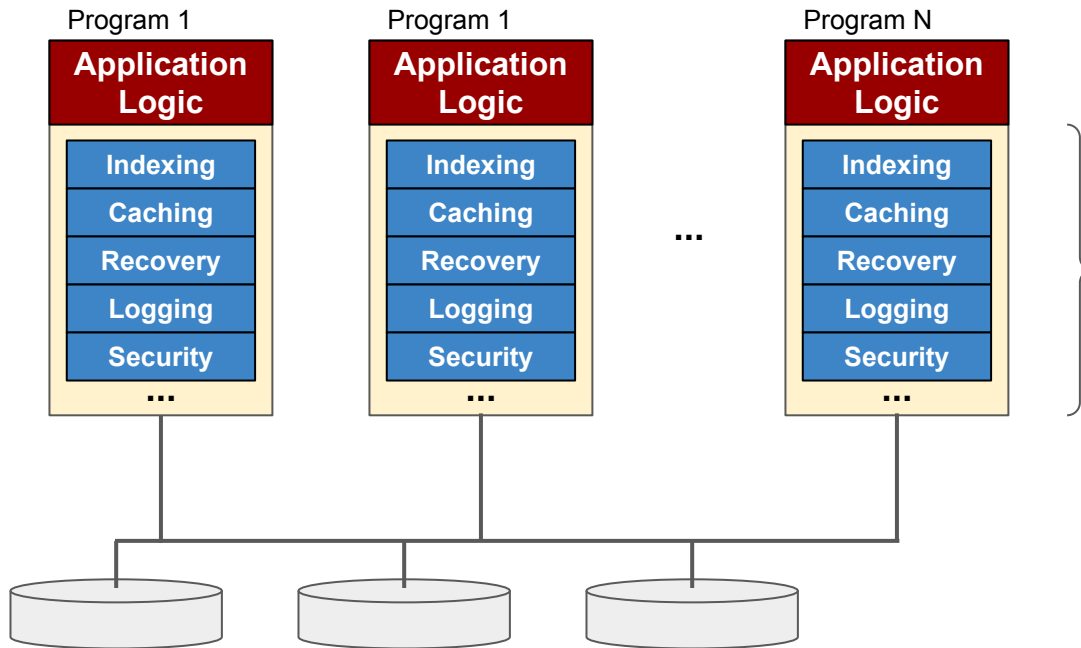
→ Recovery

- Fine-grained data access rights

→ Security

| Only HR & Management | | | | | |
|----------------------|-------|--------|-------|------------|--------|
| EmpID | Name | Office | Phone | DOB | Salary |
| 1 | Alice | 02-05 | 4520 | 10-08-1988 | 7,500 |
| 2 | Bob | 02-10 | 4530 | 06-11-2001 | 4,800 |
| 3 | Carol | 01-06 | 4540 | 25-02-1995 | 5,500 |
| All employees | | | | | |

File-Based Data Management

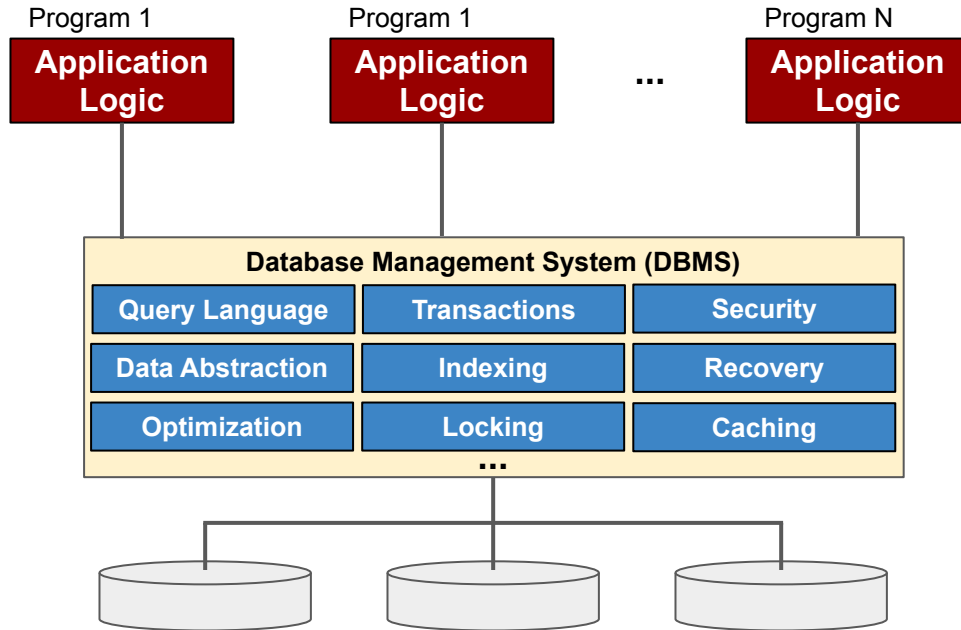


- Complex, low-level code
- Often similar requirements across different programs

→ Problems / Challenges:

- High development effort
- Long development times
- Higher risk of (critical) errors

Data Management with DBMS



- Complex, low-level code moved from application logic to DBMS
- DBMS = set of universal and powerful functionalities for data management

→ Benefits:

- Faster application development
- Increased productivity
- Higher stability / less errors

Separating "Files Only" from DBMS: Transactions

- **Transaction**

- Finite sequence of database operations (reads and/or writes)
- Smallest logical unit of work from from an application perspective


- Each transaction T has the following properties:

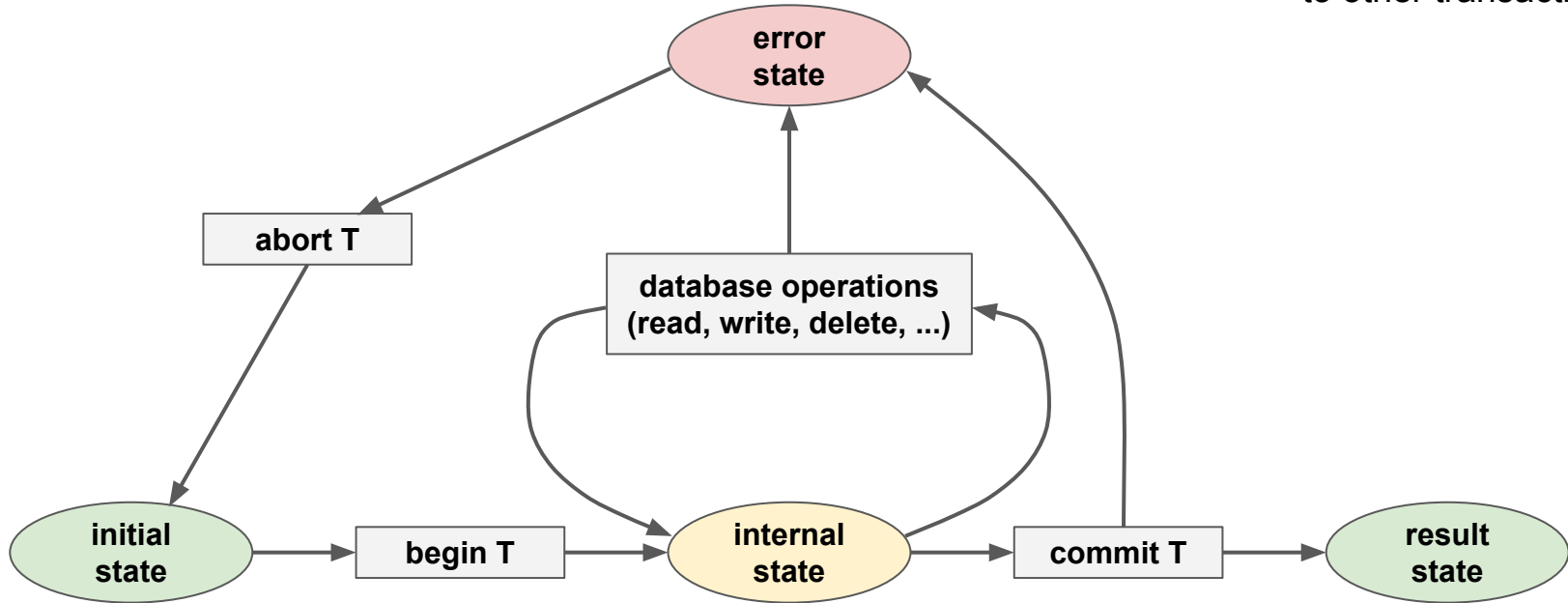
- **Atomicity**: either all effects of T are reflected in the database or none ("all or nothing")
- **Consistency**: the execution of T guarantees to yield a correct state of the database
- **Isolation**: the execution of T is isolated from the effects of concurrent transactions
- **Durability**: after the commit of T, its effects are permanent even in case of failures

→ **ACID** properties of transactions

Transition Graph of a Transaction T

if anything goes wrong, the initial state will remain and the result state will not have anything

 only states visible to other transactions



Transactions — Example: Update Bank Account Balance

- Very simple transaction

Transaction `update(X, amount)`

```
begin:
    read(X)
    X = X + amount
    write(X)
commit
```

- Assume 2 transactions

(initial balance B: 1,000)

- $T_1(B, 500)$
- $T_2(B, 100)$

Serial execution of T_1 and T_2

| $T_1(B, 500)$ | $T_2(B, 100)$ |
|-----------------|-----------------|
| begin | |
| read(B) | |
| $B = B + 500$ | |
| write(B) | |
| commit | |
| | begin |
| | read(B) |
| | $B = B + 100$ |
| | write(B) |
| | commit |

- + Correct final result (by definition)
- Less resource utilization and low throughput

Concurrent Execution — Common Problems

| $T_1(B, 500)$ | $T_2(B, 100)$ |
|---------------|---------------|
| begin | |
| read(B) | |
| $B = B + 500$ | |
| | begin |
| | read(B) |
| | $B = B + 100$ |
| write(B) | |
| commit | |
| | write(B) |
| | commit |

Final balance $B = 1,100$
(effect of T_1 overwritten)

→ Lost Update

| $T_1(B, 100)$ | $T_2(B, 500)$ |
|---------------|---------------|
| begin | |
| read(B) | |
| $B = B + 500$ | |
| write(B) | |
| | begin |
| | read(B) |
| | $B = B + 100$ |
| | write(B) |
| | commit |
| abort | |

Final balance $B = 1,600$
(when it should be 1,100)

→ Dirty Read

| $T_1(B, 100)$ | $T_2(B, 500)$ |
|---------------|---------------|
| begin | |
| read(B) | |
| | begin |
| | read(B) |
| | $B = B + 500$ |
| | write(B) |
| | commit |
| read(B) | |
| ... | |

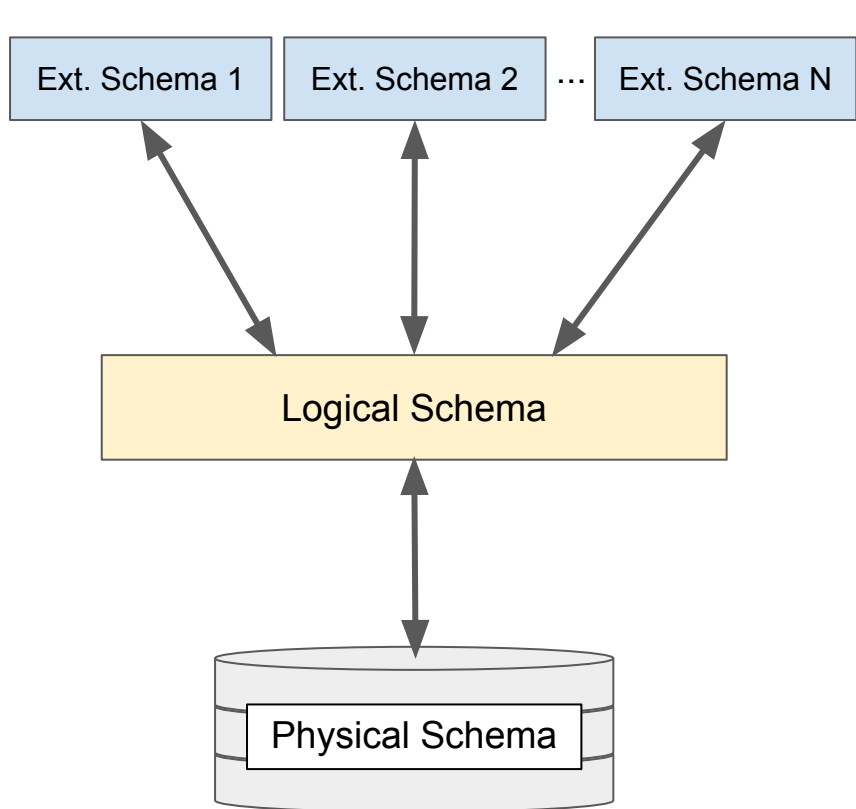
Balance B is retrieved twice
but the values differ

→ Unrepeatable Read

Requirement for Concurrent Transactions: Serializability

- Serializable transaction execution
 - A concurrent execution of a set of transactions is **serializable** if this execution is equivalent to some serial execution of the same set of transactions
 - Two executions are equivalent if they have the same effect on the data
- Core tasks of DBMS
 - **Support concurrent executions** of transactions to optimize performance
 - **Enforce serializability** of concurrent executions to ensure integrity of data

3-Tier Architecture of DBMS — Levels of Data Abstraction

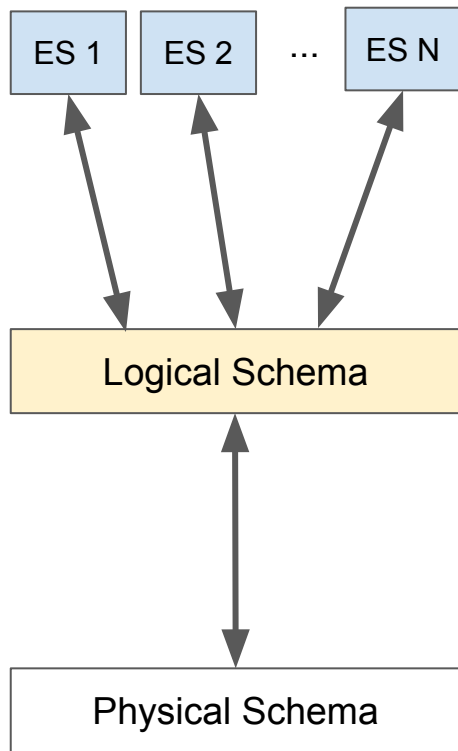


- User or group-specific view on the data

- Logical organization of data → **data model** (e.g., relations/tables, objects, graphs)
- Unified representation of all data
- Support of **physical data independence** and **logical data independence**

- Organization of data on disk and in memory
- Database as collection of fields, arrays, records, files, pages, etc.

Data Independence



Logical data independence

- Ability to change logical schema without affecting external schemas (e.g., adding/deleting/updating attributes, changing data types, changing data model)

Physical data independence

- Representation of data independent from physical scheme
- Physical schema can be changed without affecting logical schema (e.g., creating indexes, new caching strategies, different storage devices)

Study of DBMS — Scope of CS2102

- **Database design**

- How to model the data requirements
- How to organize data using a DBMS

Topics covered in CS2102

Relation Model
ER Model
Schema Refinement

- **Database programming**

- How to create, query and update a database
- How to specify data constraints
- How to use SQL in applications

Relational Algebra
SQL

- **DBMS implementation**

- How to build a DBMS? (covered in CS3223 & CS4224)

Describing Data in a DBMS

- **Data Model**

- Set concepts for describing the data
- Framework to specify structure of a DB

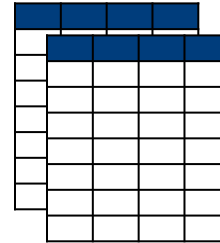
- **Schema**

- Description of the structure of a DB using the concepts provided by the data model

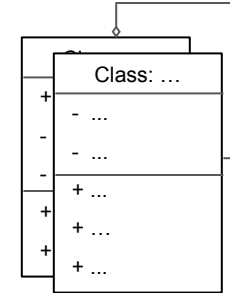
- **Schema Instance**

- Content of a DB at a particular time

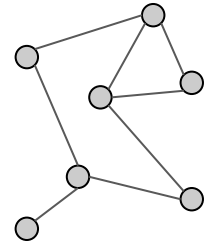
Tables



Objects



Graphs



Employees (id: integer, name: text, dob: date, salary: numeric)

Table "Employees"

| ID | Name | DOB | Salary |
|----|-------|------------|--------|
| 1 | Alice | 10-08-1988 | 7,500 |
| 2 | Bob | 06-11-2001 | 4,800 |
| 3 | Carol | 25-02-1995 | 5,500 |

Overview

- **Why Database Management Systems (DBMS)?**
 - Challenges for data-intensive applications
 - From file-based data management to DBMS
 - Core concepts of DBMS (transactions, data abstraction)
- **Relational Database Model**
 - Motivation & history
 - Core concepts: relation, domain, schema, etc.
 - Integrity constraints

Timeline of DBMS (Regarding the Supported Data Model)

- "Historical" models

- Hierarchical model
- Network model

- **Relation Model**

(early: prototypes 1970+, commercial products: 1980+)

- Commercial RDBMS
- Open-source RDBMS

- **Object-oriented model**

- Native OO model (e.g., Objectstore, 1988)
- Object-relational model
(now supported by most RDBMS)

- **More recent development**

- NoSQL models, in-memory DBMS
(e.g., Cassandra, 2008; MongoDB, 2009; Redis, 2009)

Commercial systems*

ORACLE®



Open-source systems



* some vendors offer free versions with limited functionalities

RDBMS (still) Reign Supreme

| Rank | | | DBMS | Database Model | Score | | |
|----------|----------|----------|------------------------|------------------------------|----------|----------|----------|
| Jul 2021 | Jun 2021 | Jul 2020 | | | Jul 2021 | Jun 2021 | Jul 2020 |
| 1. | 1. | 1. | Oracle + | Relational, Multi-model ⓘ | 1262.66 | -8.28 | -77.59 |
| 2. | 2. | 2. | MySQL + | Relational, Multi-model ⓘ | 1228.38 | +0.52 | -40.13 |
| 3. | 3. | 3. | Microsoft SQL Server + | Relational, Multi-model ⓘ | 981.95 | -9.12 | -77.77 |
| 4. | 4. | 4. | PostgreSQL + | Relational, Multi-model ⓘ | 577.15 | +8.64 | +50.15 |
| 5. | 5. | 5. | MongoDB + | Document, Multi-model ⓘ | 496.16 | +7.95 | +52.68 |
| 6. | ↑ 7. | ↑ 8. | Redis + | Key-value, Multi-model ⓘ | 168.31 | +3.06 | +18.26 |
| 7. | ↓ 6. | ↓ 6. | IBM Db2 | Relational, Multi-model ⓘ | 165.15 | -1.88 | +1.99 |
| 8. | 8. | ↓ 7. | Elasticsearch + | Search engine, Multi-model ⓘ | 155.76 | +1.05 | +4.17 |
| 9. | 9. | 9. | SQLite + | Relational | 130.20 | -0.33 | +2.75 |
| 10. | ↑ 11. | 10. | Cassandra + | Wide column | 114.00 | -0.11 | -7.08 |

Source: <https://db-engines.com/en/ranking>

RDBMS (still) Reign Supreme

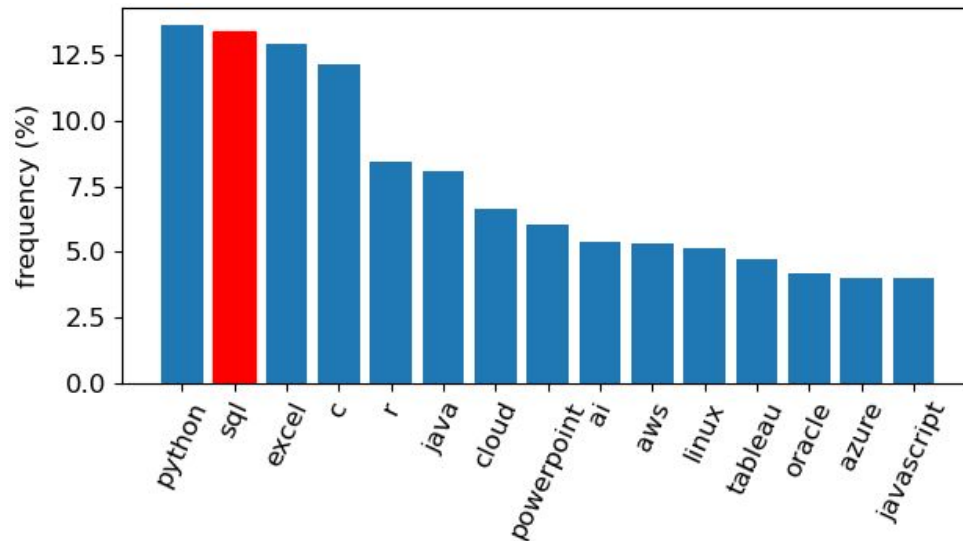
Java, SQL and Python are the most in-demand digital skills

Key Skill: SQL, Because Companies are Obsessed with Data

Want a Job in Data? Learn SQL.

- Analysis of job descriptions

- 15k+ job offers from JobStreet
(data analyst, data engineer, data scientist)
- Quick-&-dirty keyword extraction
- ...but check for yourself! :)



The Relational Model

- Proposed by Edgar F. Codd in 1970

- Basic concept: **relations**

(tables with rows and columns)

Table "Employees"

| id | name | dob | salary |
|----|-------|------------|--------|
| 1 | Alice | 10-08-1988 | 7,500 |
| 2 | Bob | 06-11-2001 | 4,800 |
| 3 | Carol | 25-02-1995 | 5,500 |

- Relation schema: definition of a relation

- Specifies attributes (columns) and data constraints (e.g., domain constraints)

Employees (id: **integer**, name: **text**, dob: **date**, salary: **numeric**)

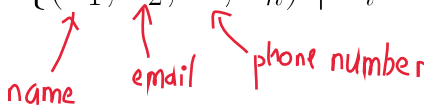
A Relational Model of Data for Large Shared Data Banks

E. F. Codd

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

The Relational Model

- **Domain** — set of atomic values (e.g., integer, numeric, text)
 - $dom(A_i)$ domain of attribute A_i = set of possible values of A_i
 - Each value v of attribute A_i : $v \in dom(A_i)$ or $v = null$
 - $null$ — special value indicating the v is not known or not specified
- **Relation** — set of tuples (or records)
 - $R(A_1, A_2, \dots, A_n)$ — relation schema with name R and n attributes A_1, A_2, \dots, A_n
 - Each instance of schema R is a relation which is a subset of
$$\{(a_1, a_2, \dots, a_n) \mid a_i \in dom(A_i) \cup \{null\}\}$$


Example

- Relational schema: Modules(course, mc, exam) with

- $\text{dom}(\text{course}) = \{\text{cs2102}, \text{cs3223}, \text{cs4221}\}$
- $\text{dom}(\text{mc}) = \{2, 4\}$
- $\text{dom}(\text{exam}) = \{\text{yes}, \text{no}\}$

- Each instance of "Modules" is a subset of

$\{\text{cs2102}, \text{cs3223}, \text{cs4221}, \text{null}\} \times \{2, 4, \text{null}\} \times \{\text{yes}, \text{no}, \text{null}\}$

max. 36 tuples {

| course | mc | exam |
|--------|------|------|
| cs2102 | 2 | yes |
| cs2102 | 2 | no |
| cs2102 | 4 | yes |
| cs2102 | 4 | no |
| cs3223 | 2 | yes |
| ... | ... | ... |
| null | 4 | no |
| null | null | no |
| null | null | null |

Quick Quiz

duplicates are not allowed in the database



The Relational Model

- **Relational database schema** — set of relation schemas + data constraints

Movies (id: **integer**, title: **text**, genre: **text**, opened: **date**)

Cast (movie_id: **integer**, actor_id: **integer**, role: **text**)

Actors (id: **integer**, name: **text**, dob: **date**)

- **Relational database** — collection of tables

Table "Movies"

| id | title | genre | opened |
|-----|------------|--------|--------|
| 101 | Aliens | action | 1986 |
| 102 | Logan | drama | 2017 |
| 103 | Heat | crime | 1995 |
| 104 | Terminator | action | 1984 |
| ... | ... | ... | |

Table "Cast"

| movie_id | actor_id | role |
|----------|----------|----------------|
| 101 | 20 | Ellen Ripley |
| 101 | 23 | Private Hudson |
| 101 | 54 | Corporal Hicks |
| 102 | 21 | Logan |
| 104 | 23 | Punk Leader |
| ... | ... | ... |

Table "Actors"

| id | name | dob |
|-----|------------------|------------|
| 20 | Sigourney Weaver | 08-10-1949 |
| 21 | Hugh Jackman | 12-10-1968 |
| 22 | Tom Hanks | 09-07-1956 |
| 23 | Bill Paxton | 17-05-1955 |
| ... | ... | ... |

Challenge: Ensuring Data Integrity

- The definition $R(A_1, A_2, \dots, A_n) \subseteq \{(a_1, a_2, \dots, a_n) \mid a_i \in \text{dom}(A_i) \cup \{\text{null}\}\}$ allows:

Table "Movies"

| id | title | genre | opened |
|-----|------------|--------|--------|
| 101 | Aliens | action | 1986 |
| 101 | Logan | drama | 2017 |
| 103 | Heat | crime | 1995 |
| 104 | Terminator | action | 1984 |

Table "Cast"

| movie_id | actor_id | role |
|----------|----------|----------------|
| 101 | 20 | Ellen Ripley |
| 101 | 23 | Private Hudson |
| 101 | 54 | Corporal Hicks |
| 102 | 21 | Logan |
| abc | 23 | Punk Leader |

Table "Actors"

| id | name | dob |
|------|------------------|------------|
| 20 | Sigourney Weaver | 08-10-2049 |
| 21 | Hugh Jackman | 12-10-1968 |
| null | Tom Hanks | 09-07-1956 |
| 23 | Bill Paxton | 17-05-1955 |

Can we tell the DBMS what are valid tuples and attribute values?

➔ Integrity Constraints

Integrity Constraints

- **Integrity Constraint** — condition that restricts what constitutes valid data

- DBMS checks that tables only ever contain valid data → data integrity

- 3 main **structural integrity constraints** of the Relation Model

("structural" = inherent to the data model, independent from the application)

- Domain constraints (e.g., cannot store a string in a integer column)
- Key constraints
- Foreign key constraints

- **General constraints**

- Depend on the specific application
- Covered in later lectures (keyword: triggers)

Table "Cast"

| movie_id | actor_id | role |
|----------------|----------|----------------|
| 101 | 20 | Ellen Ripley |
| 101 | 23 | Private Hudson |
| 102 | 21 | Logan |
| abc | 23 | Punk Leader |

not allowed

Key Constraints

- **Superkey** — subset of attributes that uniquely identifies a tuple in a relation
 - e.g., {id, title}, {id, title, opened}
- **Key** — superkey that is also minimal, i.e., no proper subset of the key is a superkey
 - e.g., {id} (maybe: {title}, {opened})
- **Candidate keys** — set of all keys for a relation
- **Primary key** — selected candidate key for a relation
 - Important: values of primary key attributes cannot be *null* (**entity integrity constraint**)

Table "Movies"

| id | title | genre | opened |
|-----|------------|--------|--------|
| 101 | Aliens | action | 1986 |
| 102 | Logan | drama | 2017 |
| 103 | Heat | crime | 1995 |
| 104 | Terminator | action | 1984 |

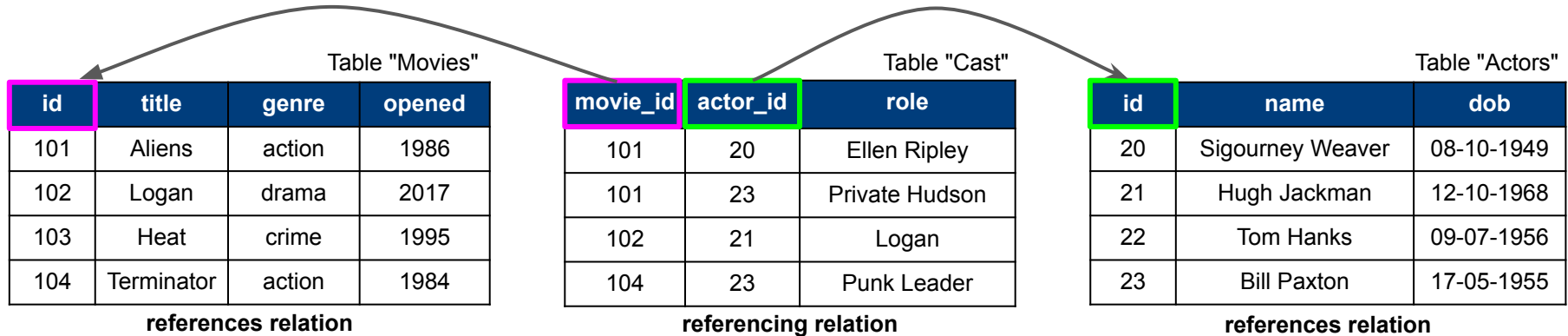
Movies (id: **integer**, title: **text**, genre: **text**, opened: **date**)

Quick Quiz



Foreign Key Constraints (also: referential integrity constraints)

- **Foreign key** — subset of attributes of relation A if it refers to the primary key in a relation B



- Requirement: each foreign key in referencing relation must
 - appear as primary key in referenced relation OR
 - be a *null* value

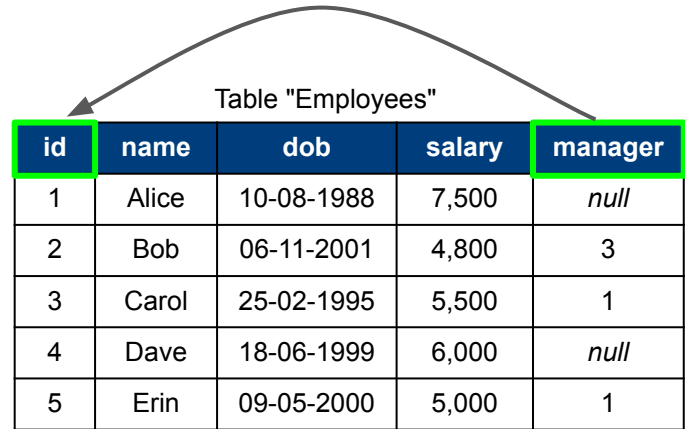
Foreign Key Constraints

- Referencing & referenced relation can be the same relation
 - Example: each employee has at most one manager

can come from the same table

Table "Employees"

| id | name | dob | salary | manager |
|----|-------|------------|--------|-------------|
| 1 | Alice | 10-08-1988 | 7,500 | <i>null</i> |
| 2 | Bob | 06-11-2001 | 4,800 | 3 |
| 3 | Carol | 25-02-1995 | 5,500 | 1 |
| 4 | Dave | 18-06-1999 | 6,000 | <i>null</i> |
| 5 | Erin | 09-05-2000 | 5,000 | 1 |



- A relation can be referencing and referenced relation for different relations

Table "Genre"

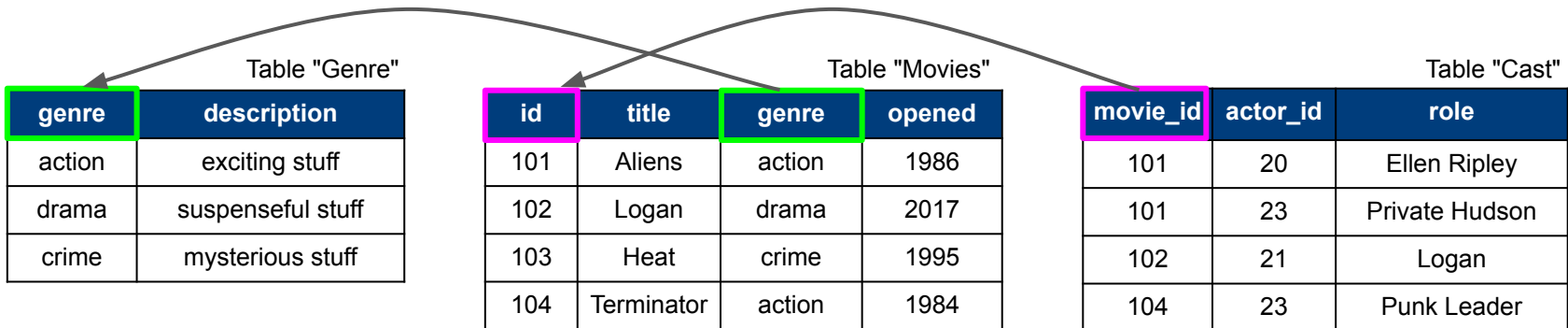
| genre | description |
|--------|-------------------|
| action | exciting stuff |
| drama | suspenseful stuff |
| crime | mysterious stuff |

Table "Movies"

| id | title | genre | opened |
|-----|------------|--------|--------|
| 101 | Aliens | action | 1986 |
| 102 | Logan | drama | 2017 |
| 103 | Heat | crime | 1995 |
| 104 | Terminator | action | 1984 |

Table "Cast"

| movie_id | actor_id | role |
|----------|----------|----------------|
| 101 | 20 | Ellen Ripley |
| 101 | 23 | Private Hudson |
| 102 | 21 | Logan |
| 104 | 23 | Punk Leader |



Quick Quiz



Integrity Constraints

- Limitations

- Structural integrity constraints do cover application-independent constraints
(e.g., limiting the domain to valid values)
- Not covered: application-dependent constraints derived from deeper semantics of the data

- Practical considerations

- Integrity constraints are optional, not mandatory
(but they allow pushing checks from the application into the DBMS)
- Integrity constraints may affect performance*
(checking constraints require additional processing steps)

Table "Actors"

| id | name | dob |
|----|------------------|------------|
| 20 | Sigourney Weaver | 08-10-2049 |
| 21 | Hugh Jackman | 12-10-1968 |
| 22 | Tom Hanks | 09-07-1956 |
| 23 | Bill Paxton | 17-05-1955 |

***Sidenote:** Key constraints typically involve the creation indexes which can significantly boost query performance!

Relational Model — Cheat Sheet

Diagram illustrating the Relational Model structure:

Relation name: Table "Movies"

Attribute: id, title, genre, opened, ...

| id | title | genre | opened | ... |
|-----|------------|--------|--------|-----|
| 101 | Aliens | action | 1986 | ... |
| 102 | Logan | drama | 2017 | ... |
| 103 | Heat | crime | 1995 | ... |
| 104 | Terminator | action | 1984 | ... |
| 105 | Hot Fuzz | comedy | 2007 | ... |
| 106 | Saw | horror | 2004 | ... |
| ... | ... | ... | ... | ... |

Relation schema: The structure of the relation, including the attributes (id, title, genre, opened, ...).

Tuple / Record: A single row of data, such as the row for Logan (id: 102, title: Logan, genre: drama, opened: 2017).

Relation: The entire set of tuples (records) in the relation.

Attribute value: A specific value for an attribute, such as the value 2004 for the attribute opened in the row for Saw.

Relational Model — Cheat Sheet

| Term | Description (informal) |
|------------------------|---|
| attribute | Column of a table |
| domain | Set of possible values for an attribute |
| attribute value | Element of a domain |
| relation schema | Set of attributes (with their data types + relation name) |
| relation | Set of tuples |
| tupel | Row of a table |
| database schema | Set of relation schemas |
| database | Set of relations / tables |

Relational Model — Cheat Sheet

| Term | Description (informal) |
|------------------------|--|
| key | Minimal set of attributes that uniquely identify a tuple in a relation |
| primary key | Selected key (in case of multiple candidate keys) |
| foreign key | Set of attributes that is a key in referenced relation |
| prime attribute | Attribute of a key |

- Terminology: DB. vs DBS vs. DBMS

$$\text{DBS} = \text{DBMS} + n \cdot \text{DB} \quad (n > 0)$$

Summary

- Advantages of DBMS for large-scale data management (compared to "files only")
 - Transactions with ACID properties to guarantee integrity of the data
 - Levels of abstraction for data independence
- Relational Model
 - Unified representation of all data as tables (relations)
 - (Structural) integrity constraints to specify restrictions on what constitutes correct/valid data
- Outlook for next lecture: Relational Algebra
 - Formal method to process and query relations
 - Theoretical underpinning for query languages such as SQL