MANCHESTER 1824

The University of Manchester

H1

Introduction to Data Management

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 Handout 01 of 12]

The University of Manchester

Introduction to Data Management

Fundamentals of Databases

Alvaro A A Fernandes, SCS, UoM

[COMP23111 Handout 01 of 12]

Acknowledgements

- These slides are adaptations (mostly minor, but some major) of material authored and made available to instructors by **A. Silberschatz, H. F. Korth, and S. Sudarshan** to accompany their textbook **Database System Concepts, 6th Edition, McGraw-Hill, 2006, 978-0-07-352332-3**.
- Copyright remains with them, whom I thank.
- Any errors are my responsibility.

In This Handout

- How do we view data and how should we manage it?
- Where are DBMSs used?
- Why DBMSs? Why not use plain files?
- How do different levels of abstraction in DBMSs work?
- What is a (logical) data model?
- What are schemas and instances?
- What are the basic notions in relational data modelling?
- What different types of database language are needed and why?
- How are databases designed?
- How have DBMSs evolved over time?

MANCHESTER 1824

The University of Manchester

How do we view data and how should we manage it?

Data as an Asset

- Data is a crucial asset to modern organizations.
- Often as much as capital and human resources.
- The proper management of data as an asset is crucially dependent on database management systems (DBMSs).

Database Management Systems (DBMSs)

- DBMSs are intricate, powerful, highly-enabling software products.
- From a user viewpoint, they are best seen as a collection of services that enable cost-effective data management.
- From a computer scientist viewpoint, they are a convergence point of many advanced topics in computer science.

Database Management Systems (DBMSs)

- A DBMS manages information about a particular enterprise.
- It comprises:
  - A collection of interrelated data
  - A suite of software tools to interact with that data
  - A software environment that is both convenient and efficient to use

MANCHESTER 1824

The University of Manchester

Where are DBMSs used?

Applications :: Inside Organizations

- Online retailers
  - Order tracking, customized recommendations
- Manufacturing
  - Production, inventory, orders, supply chain
- Human resources
  - employee records, salaries, tax deductions

Applications :: Across Organizations

- Banking
  - Accounts, deposits, withdrawals
- Airlines
  - Reservations, schedules, rosters
- Sales
  - Customers, products, purchases

Example: A University Database

- Add new students, instructors, and courses
- Register students for courses, generate class rosters, control attendance
- Assign grades to students, compute grade point averages (GPA) and generate transcripts
- Charge the right fees, monitor fee payments

Example: A University Database

- Add new students, instructors, and courses
  - Relationships with Human Resources function
- Register students for courses, generate class rosters, control attendance
  - Relationships with Estates function
- Charge the right fees, monitor fee payments
  - Relationships with Finances function

Why DBMSs? Why not use plain files?

The Problem with Files

- In the early days, database applications were built directly on top of file systems.
- Over time, applications grew in complexity.
- So did the need to see all the data as conceptually integrated.
- Writing file/format-dependent programs proved inefficient and error-prone.

Drawbacks: Isolation, Redundancy, Inconsistency

- Data isolation:
  - Multiple files and formats
- Data redundancy and inconsistency:
  - Duplication of information in different files
- Difficulty in accessing data:
  - Need to write a new program to carry out each new task

Drawbacks: Lack of Integrity Enforcement

- Integrity enforcement:
  - Integrity constraints (e.g., `account balance > 0`) become buried in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

Drawbacks: Lack of Atomicity

- Atomicity of updates:
  - Failures may leave database in an inconsistent state with updates only partially carried out
  - Imagine a transfer of funds from one account to another...
  - It should either complete or be rolled back as if no sub-step had ever happened

Drawbacks: Lack of Concurrency Control

- Concurrent access by multiple users:
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies

Drawbacks: Lack of Concurrency Control

- Concurrent access by multiple users:
  - Imagine two users reading the balance of an account (say, 100) then withdrawing money (say, 50 one, 30 the other) at the same time...
  - Without control, this is a race condition: the final balance depends on the order in which withdrawals get processed against which previous balance

Drawbacks: Lack of Safety/Security Control

- Safety/security problems:
  - Hard to provide role-based authentication and authorization
  - Hard to provide programmatic access just to some, not all, the data
  - Crashes can leave the system in an inconsistent state

DBMS Services

- DBMSs offer efficient and easy-to-use services to solve all the above:
  - Location/format independence
  - Integration without visible redundancy
  - Programs use uniform logical representation
  - Different views by different programs are cleanly supported

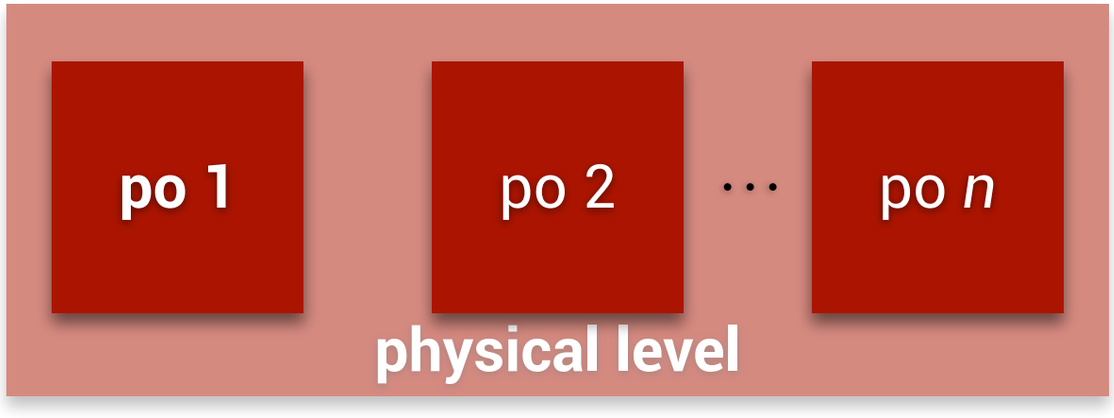
DBMS Services

- DBMSs offer efficient and easy-to-use services to solve all the above:
  - Concurrent updates are atomic, consistent, isolated and durable
  - Fault-resilience, and seamless recovery
  - Role-based authentication and authorization

How do different levels of abstraction in DBMSs work?

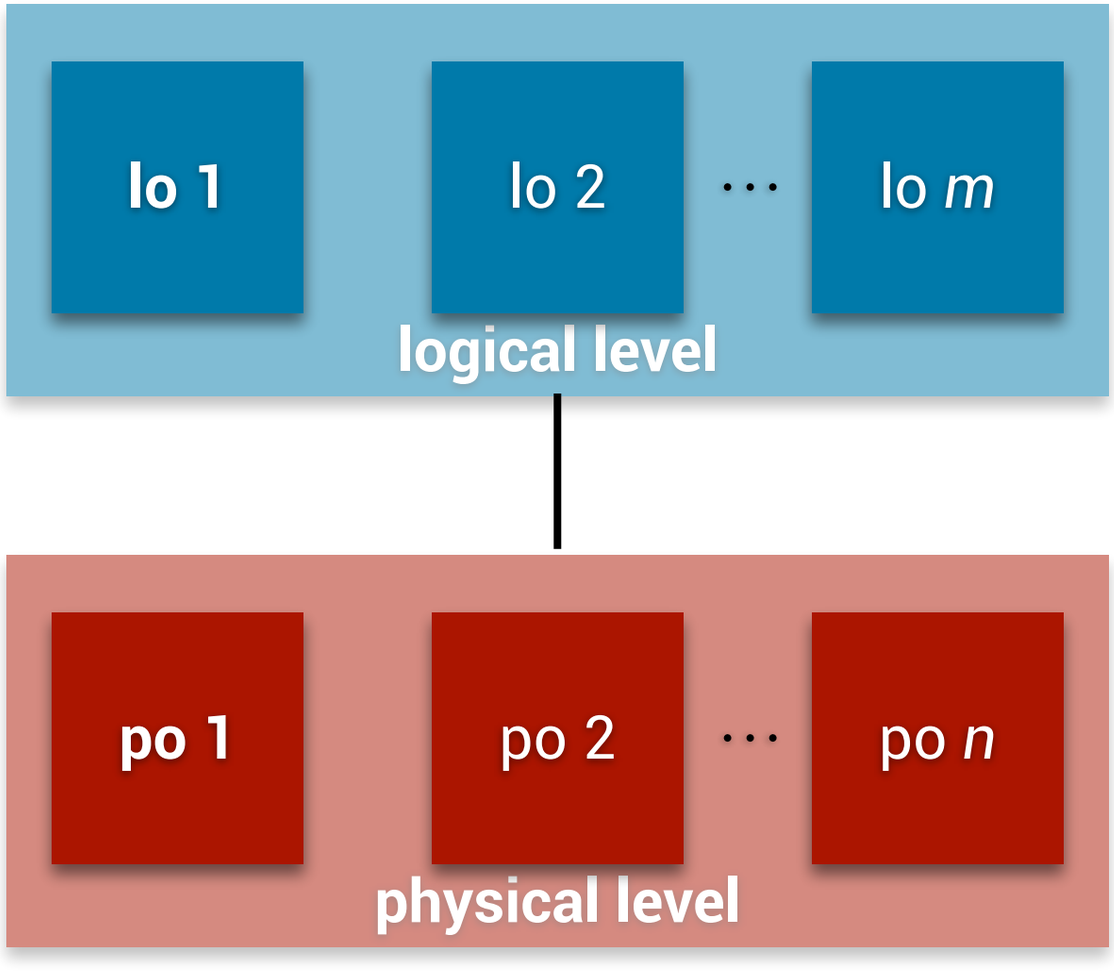
Levels of Abstraction in DBMSs: Physical

- Physical level
  - How a record (e.g., three strings and an integer) is stored as a physical object



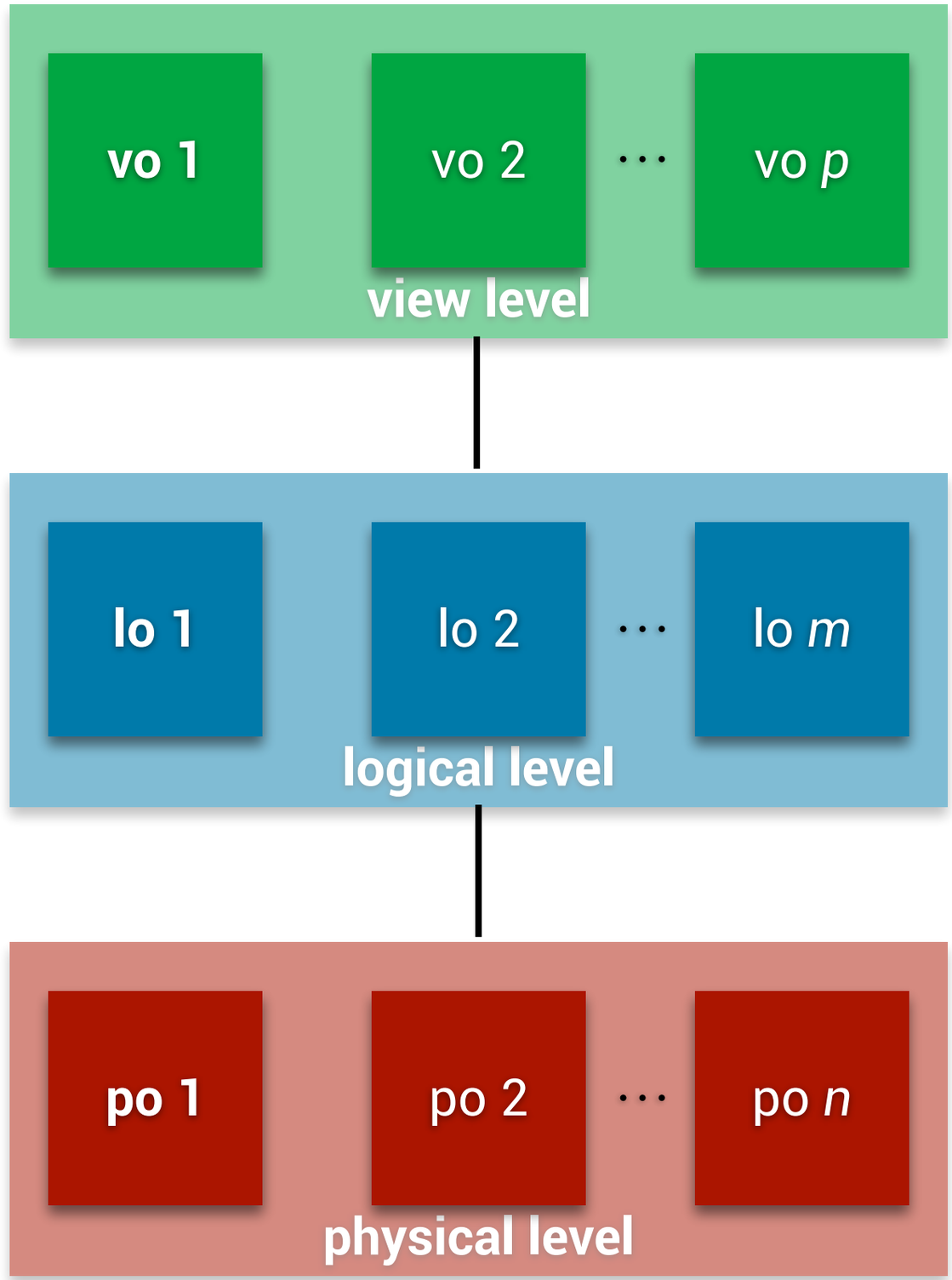
Levels of Abstraction in DBMSs: Logical

- Logical level
  - How representations of real-world concepts (e.g., a customer) are stored in database
  - Along with their relationships (e.g., which products each customer has bought)



Levels of Abstraction in DBMSs: View

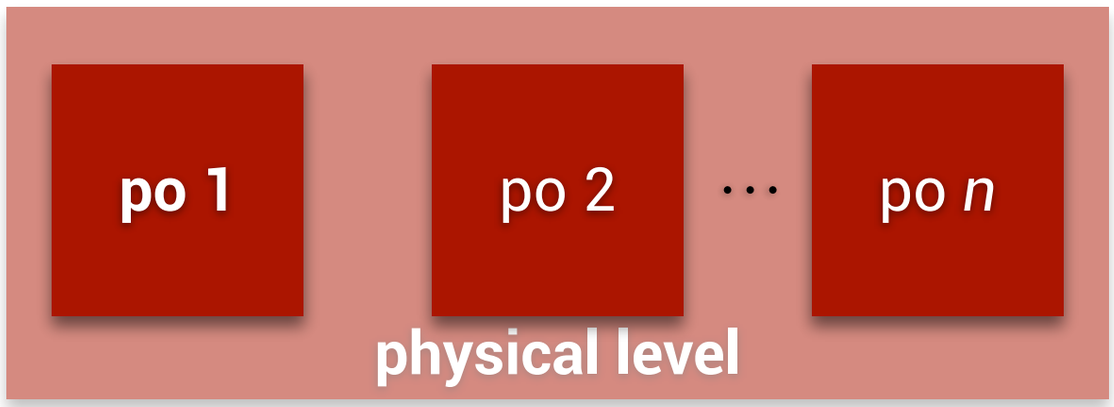
- View level
  - How applications and user interfaces retrieve the data they need
  - Hides data type details
  - Hides information (such as an employee's salary), e.g., for security/ privacy purposes



Capturing Abstraction: Records

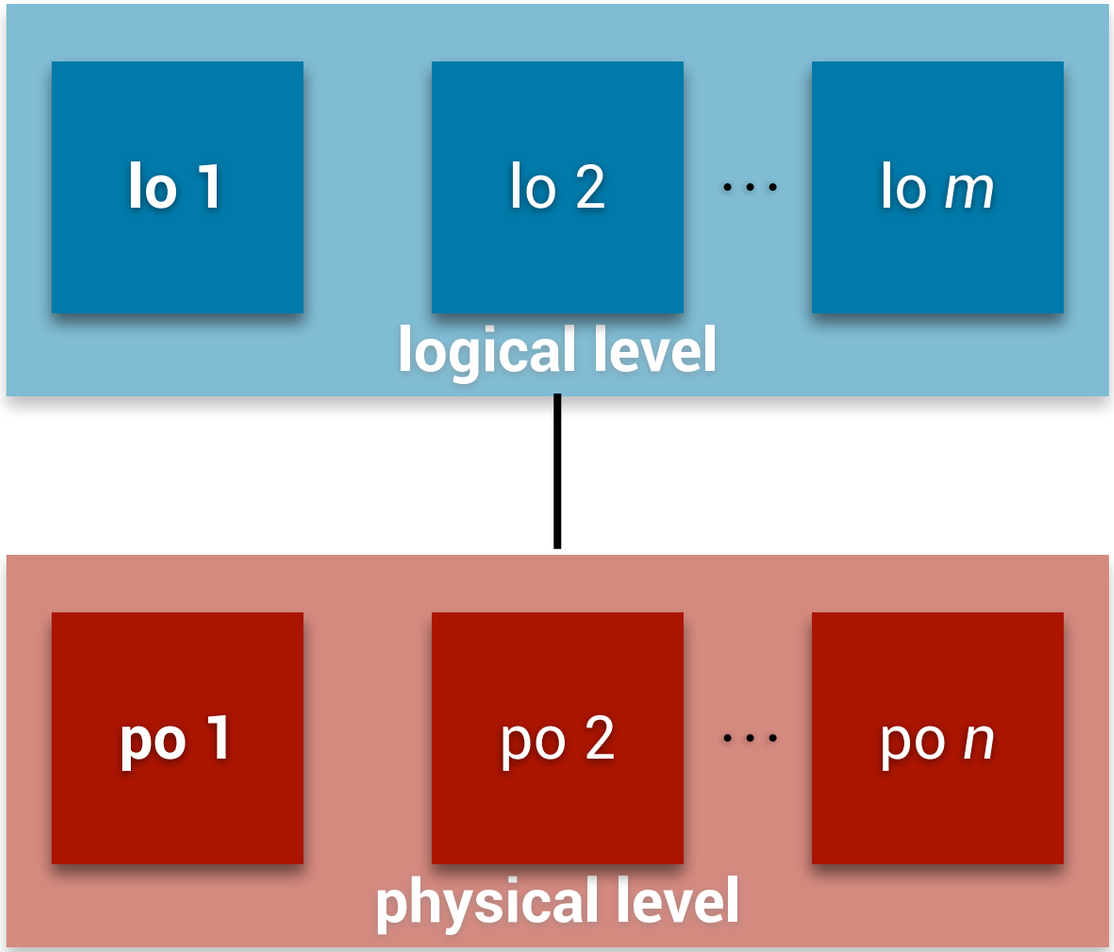
- Physical level
  - Physical objects are, conceptually, records
  - They are instances of data types defined in programs

```
type instructor =
    record ID : string;
    name : string;
    dept_name : string;
    salary : integer;
end;
```



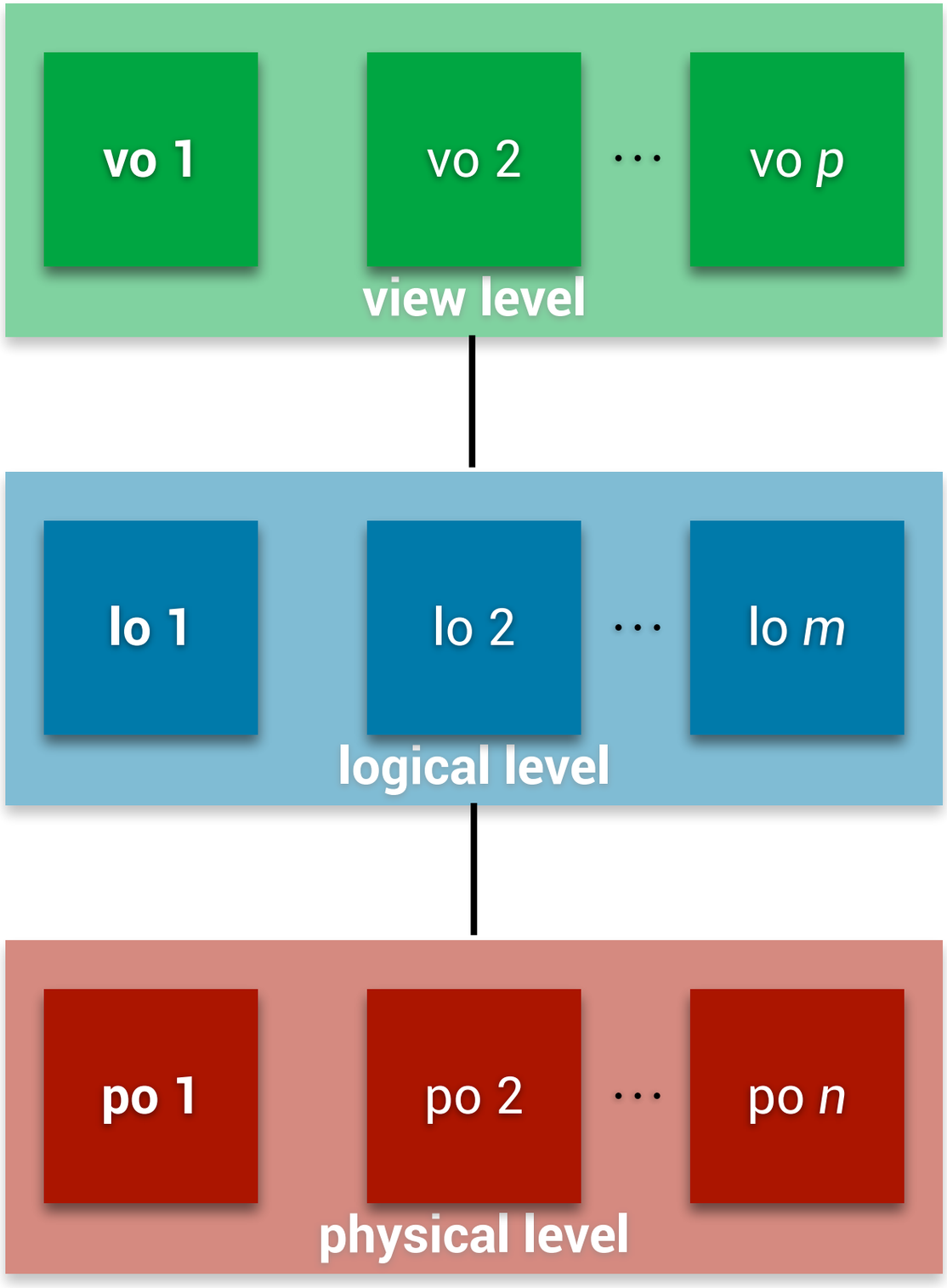
Capturing Abstraction: Relations

- Logical level
  - Enabled by a data model, i.e., a programming-language independent representation of data
  - In the case of the relational model, the logical objects are tables



Levels of Abstraction in DBMSs: View

- View level
  - Enabled by the languages exposed by the data model
  - In the case of the relational model, view objects are named queries



Levels of Abstraction v. Models

- The physical level is the concern of a storage model
  - It specifies formats, records, files, clustering, indexes, compression, replication, redundancy
- The logical level is the concern of a logical or a conceptual data model
  - It specifies the concepts and relationships these establish with one another and has organization-wide scope
- The view level is the concern of a database application model
  - It specifies query-based views, retrieval queries, procedural abstractions, etc.

What is a (logical) data model?

Data Model

- A formal construct (giving rise to a collection of formalisms, then tools) for describing:
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Often (though not always) including special database languages for interaction

Data Model

- Each DBMS supports (i.e., is compliant with) one or more logical data models
- Ideally, when we design a database using a particular data model, there is a DBMS in which we can implement the design directly, with minimal effort

Widely-Used Data Models

- Relational
- Object-based:
  - Object-relational
  - Object-oriented
- Semistructured:
  - XML
  - RDF
- Document-based

Object-Relational

- The relational model (in detail, later) is flat and (typically) only supports atomic/ scalar values
- The object-relational model
  - Extends the relational model by including object-based notions and constructs to deal with added data types.
  - Allows attributes of tuples to have complex types, including non-atomic values such as nested relations.
  - Preserves relational foundations, in particular the declarative access to data, while extending modelling power.
  - Provides upward/forward compatibility with existing relational databases.

XML :: Extensible Markup Language

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language not a database language
- The ability to specify new tags, and to create nested tag structures made XML a great way to exchange data, not just documents
- XML has become the basis for all new generation data interchange formats.
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Older Data Models

- Some older models survive in legacy applications:
  - Network model
  - Hierarchical model

Newest Data Models

- The newest data models are document-based
- They are prevalent in modern, web-based applications
- Broadly, they are:
  - Key-value model
  - Column-family model
  - Graph-based model
- They are mostly APIs into massively-distributable data stores

What are schemas and instances?

Schemas v. Instances

- In data model terminology, we distinguish between a schema and the many instances that conform to it.

Schemas

- A schema defines the logical structure of the database
- Similar to types and variables in programming languages
- Customers, accounts, their properties, and the relationships between them (e.g., has, shares, etc.)

Instances

- An instance is the actual content of the database at a particular point in time
- Similar to the value of a variable at a point in the execution of a program
- Mary Quaint has the current account 015467 shared with her business partner Vivienne Eastwood.

Physical v. Logical Schemas

- Physical schema
  - Defines the design of database structures at the physical level
  - Records, files, indices, compression, etc.
- Logical schema
  - Defines the design of database structures at the logical level
  - Tables, columns, logical dependences, etc.

Physical Data Independence

- In general, interfaces between the various levels and components should be defined so that changes in some parts do not seriously influence others.

- In DBMSs, this principle is supported:
  - Application programs only depend on the logical schema
  - We can modify the physical schema without any changes needed to logical schema
  - Changes in physical location, format, structure, indexes, etc. do no propagate to application programs

MANCHESTER 1824

The University of Manchester

What different types of database language are needed and why?

Data Modelling Languages

- A well-defined logical data model comes with three or more languages:
  - A data definition language (DDL)
  - One or more query languages (QLs)
  - One or more data manipulation languages (DMLs)

Why Three or More Languages?

- The three or more languages enforce different versions of the principle of separation of concerns
  - Between defining and using
  - Between retrieving and manipulating
  - Between describing-what and telling-how
  - Between programmatic paradigms and environments

Why DDL + (QL+DML)?

- One should define the data without constraining how it could be used (i.e., retrieved, transformed, updated, etc.)
- Enterprises
  - Grow and shrink with more, or less, data being core to their business interests over time
  - New ways of presenting data and providing access to it often emerge unforeseen as disruptive technologies become mainstream
  - New forms of exploiting data assets are constantly emerging as business opportunities
- New data without changing programs, new programs without changing data.

Why QL+DML?

- Constraining the expressive power that is available allows the reduction in complexity to be efficiently used by the system.

- In particular, expressions whose evaluation only involves bounded iteration and is guaranteed not to effect changes in the state of the database (i.e., update the data) are guaranteed to terminate.

- Strictly speaking, a (pure) query language does not have bounded iteration and cannot update instances, hence query evaluation always terminates.

- One should describe the data we want without having to tell the DBMS how to go about obtaining it step-by-step etc.

Why $QL_1 + ... + QL_n$?

- It is possible (and very useful) to distinguish between declaratively-specified and procedurally-specified queries.
- A declaratively-specified query states what data is needed, it says nothing as to how to obtain it
- A procedurally-specified query describes how to obtain the data, it say which precise steps in which precise order
- If a declarative query language Q is proved to retrieve whatever a procedural query language Q' retrieves, and vice-versa, then we can write algorithms that map from one to the other
- We can also apply semantics-preserving rewrites, e.g., to simplify one expression E into an expression E' that is, all else being equal, more efficient to evaluate.

Why DML_1 + ... + DML_m?

- It is pragmatically desirable to have different ways of expressing computations on database instances that involve updates.

- For example, we may want to update entire row sets at once, or else one row at a time.

- We may also want to cause updates to be an automatic response by the DBMS to certain events, or else keep control of that with the application.

- We may also want have the application drive the updates, or else delegate them to the DBMS.

DDL :: Data Definition Language

- A language for specifying database schemas
- This involves capabilities for:
  - Adding, changing and deleting tables, as well as columns in tables
  - Declaring types for columns
  - Constraining the values a cell can take
  - Specifying constraint-based consequences of changes in a table, including their propagation to related tables

DDL :: Data Dictionary

- A DDL compiler stores the internal structures it generates in the DBMS's data dictionary
- The data dictionary contains metadata (i.e., data about data):
  - Schemas
  - Integrity constraints
    - Primary key: `ID` uniquely identifies each `instructor`
    - Referential integrity: a `dept_name` value in any `instructor` row must appear in the `department` table
  - Authentication and authorization information

QL+DML :: Query + Data Manipulation Language

- A language for accessing values, e.g.,
  - Selecting rows
  - Selecting columns
  - Combining columns from different tables
- A language for manipulating schema-compliant instances, e.g.,
  - Inserting rows
  - Deleting rows
  - Updating cell values

QL+DML :: Query + Data Manipulation Language

- In spite of the benefits of separation, QLs and DMLs are often bundled as one concrete language in practice

- Still often possible to conceive of different language subsets

- SQL, the most-widely used language for database interaction, bundles DDL, QL and DML capabilities

- In SQL, the language subsets are more or less distinguishable

SQL: History

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999, SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
- Oracle SQL deviates from the standard in many respects but has great clout, due to Oracle's market share.

MANCHESTER 1824

The University of Manchester

How are databases designed?

Methodological Phases

- The process of designing the general structure of the database:
- Logical Design is deciding on the database schema.
- Database design requires that we find a “good” collection of relation schemas.
  - Business decision: What attributes should we record in the database?
  - Computer Science decision: What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design is deciding on the physical layout of the database.

Quality Goals

- Is there any problem with this design?

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Quality Goals

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

- Is this better?
- Why?
- What is the fundamental difference between this design and the single-table one?

Design Theories: Normalization Approach

- First to be proposed
- Formal, axiomatic theory
- Specifies formal conditions for a schema design to meet some quality goals
- The axioms allow design transformations towards improving quality
- Directly implementable designs
- Requires specialist designers whilst being hard to grasp for users in validation

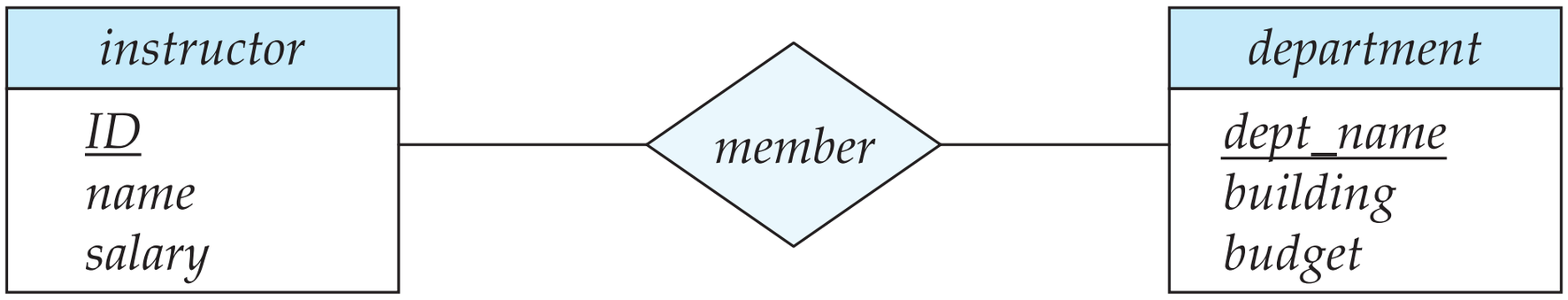
Design Theories: Conceptual Approach

- Requires no specialist designers whilst being easier to grasp for users in validation
- Good design (judged by normalization theory) follows as a natural consequence
- Not directly implementable, but conceptual model can be mapped to a normalized logical-level design

Design Theories: Conceptual Modelling

- One popular conceptual model is the entity-relationship model
- Models an enterprise as a collection of entities and relationships
- Uses a diagrammatic approach
  - Entity: a “thing” or “object” in the enterprise that is distinguishable from other objects, represented as a rectangle.
  - Relationship: an association among several entities, represented as a lozenge.
  - Entities and relationships are described by a set of attributes, represented as circles (or else listed inside the entity rectangle)

Design Theories: Conceptual =/= Logical



- (This diagrammatic notation is less common. We'll use the classical one.)

- Where did `dept_name` in `instructor` go?

MANCHESTER 1824

The University of Manchester

How have DBMSs evolved over time?

History of DBMSs :: 1950s- early1960s

- Data processing using magnetic tapes for storage
- Tapes provided only sequential access
- Punched cards for input

History of DBMSs :: Late 1960s-1970s

- Hard disks allowed direct access to data
- Network and hierarchical data models in widespread use
- Ted Codd defines the relational data model
  - Would win the ACM Turing Award for this work
  - IBM Research begins System R prototype
- UC Berkeley begins Ingres prototype
- High-performance (for the time) transaction processing

History of DBMSs :: 1980s

- Research relational prototypes evolve into commercial systems
  - SQL becomes industrial standard
- Parallel and distributed database systems
- Object-oriented database systems

History of DBMSs :: 1990s

- Object-relational pushes object-oriented into niche areas
- Large decision support and data-mining applications
- Large multi-terabyte data warehouses
- Emergence of Web commerce

History of DBMSs :: Early 2000s

- XML and XQuery standards
- Automated database administration
- Continuous queries over high-volume data streams: Aurora, NiagaraCQ, STREAM, StreamBase, etc.
- Columnar stores: C-Store, Vertica, etc.
- Data appliances: Netezza, DatAllegro, etc.

History of DBMSs :: Late 2000s

- Later 2000s:
  - Giant data storage systems
  - Google BigTable, Yahoo PNuts, Amazon, etc.
  - MapReduce takes over the world of analytical processing

History of DBMSs :: Early 2010s

- Early 2010s:
  - Big data becomes BIG^{BIG}!
  - NoSQL systems grow in popularity: MongoDB, CouchDB, Cassandra, Dynamo, Riak, etc.
  - NewSQL strikes back: VoltDB, H-Store, NuoDB, etc.
  - Web of Data launches: SPARQL over RDF data

MANCHESTER 1824

The University of Manchester

Summary

DBMSs, Applications, Abstraction, Data Models

- A DBMS is a complex software artefact whose components are designed to deliver services to applications that enable data to be managed as a crucial organization asset.
- They are used for all kinds of application, but, in particular, transactional ones. They offer unrivalled efficiency and robustness while retaining ease of use. They are superior to plain files because they allow the separation of concerns between different user views of the data.
- Different levels of abstraction in data management work to separate concerns.
- Data models, schemas and instances enable a conceptual and logical view of the data.
- The relational approach to data modelling centres on the use of regular tables within, and between, which integrity constraints are maintained.

Languages, SQL, Design, History

- DBMS languages often have a definition, a manipulation and query sublanguage. This again allows for separation of concerns.
- SQL is the dominant, concrete, language used for DBMS interaction.
- There are different ways to design a database, but the most widely-adopted starts with a conceptual model and through a sequence of transformations derives a DBMS-specific model without loss of organization-wide semantics.
- DBMSs have evolved over time towards more uniformity and elegance without loss of efficiency.
- This is now changing under pressure from new, analytical needs.

In the Next Handout

- We'll start to have a deeper look at the relational model that underpins the most widely-deployed DBMSs today.
- We'll also begin to look at how SQL implements the relational languages by looking at its DDL and DML facilities.