Introduction to Database Systems (2)

SWEN304 / SWEN435

Trimester 1, 2024

Lecturer: Kevin Shedlock

Engineering and Computer Science

VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

Outline

- Data models
- Schemas and Instances
- The Three-Schema Architecture
- Program - Data Independence
- Data manipulation languages
  - Navigational
  - Declarative
- Reading: Chapter 2 of the textbook

Data Models

A Data Model: a set of concepts to describe the structure of a database, the operations for manipulating these structures, and certain constraints that the database should obey

A data model is a mathematical abstraction used:

- to make an approximate representation (an *abstraction*) of a real system, or;
- to model a database of this real system

Categories of Data Models

1. Conceptual (high-level, semantic) data models:
   - Provide concepts close to the way users perceive data
   - Entity-based or object-based data models

2. Physical (low-level, internal) data models:
   - Concepts describe how data is stored in the computer

3. Implementation (Representational) data models
   - Provide concepts that fall *between* the above two: balances user views with some computer storage details
   - Relational data model: used in many commercial products (DB2, ORACLE, SQL Server, …)
   - Legacy data models: network model, hierarchical model

Data Models

A fundamental characteristic of the database is that it provides some form of data abstraction (details of data organization; storage; and, essential features for it to work with data.

Constructs are used to define the database structure

Constraints are statements about values and relationships that **must hold** between data

Operations specify database *retrievals* and *updates* by referring to the constructs of the data model

A Database Instance

STUDENT			
Id	**Lname**	**Fname**	**Major**
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300132	Smith	Susan	COMP

COURSE			
Course_id	**Cname**	**Points**	**Dept**
COMP302	DB sys	15	Engineering
COMP301	softEng	20	Engineering
MATH214	DisMat	15	Mathematics

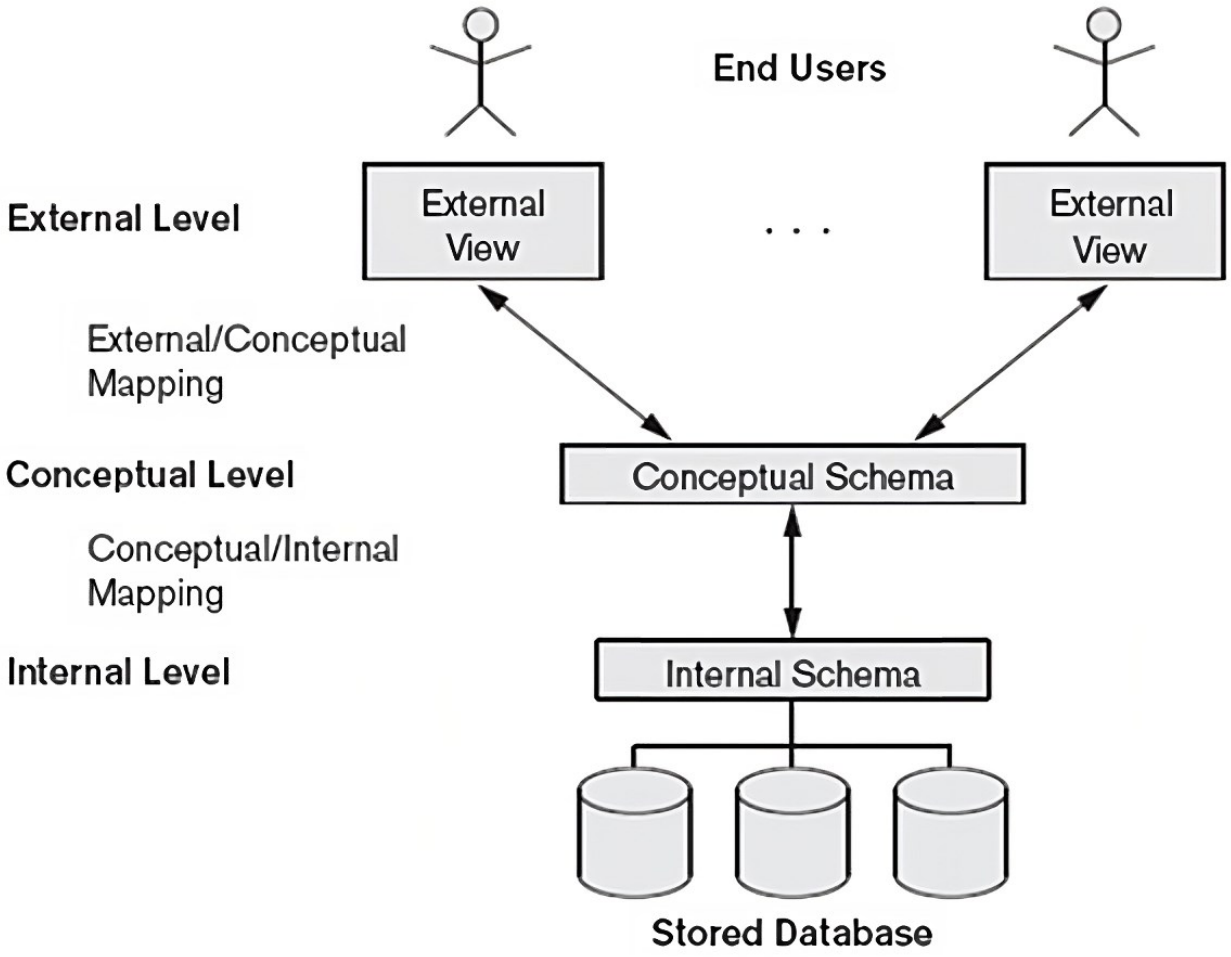
GRADES		
Id	**Course_id**	**Grade**
300111	COMP302	A+
300111	COMP301	A
300111	MATH214	A
300121	COMP301	B
300132	COMP301	C
300121	COMP302	B+
300132	COMP302	C+

Te Taupanga University of Performing Arts (TTUPA)

Three-Schema Architecture

- Internal Schemas:
  - describes physical storage and access structures, e.g. indexes, hashing algorithms, pointers
  - Access path to database; complete details of database
- Conceptual Schema:
  - Provides entire description of database
  - abstract, integrated description of all data independent from the implementation
  - mapped to physical schema
- External Schemas:
  - perspective of a user/application accessing the database
  - describes restructured parts of the database (views) used for a particular application
  - mapped to the logical schema

Three-Schema Architecture



(Elmasri & Navathe, 2011)

Schemas and Instances

- A database schema (or intension):
  the *abstract* description of a database
  - includes the descriptions of the **database structure** and the **constraints** that should hold in the database.
  - is "fairly" static
  - given in the context of a particular data model and a corresponding data definition language

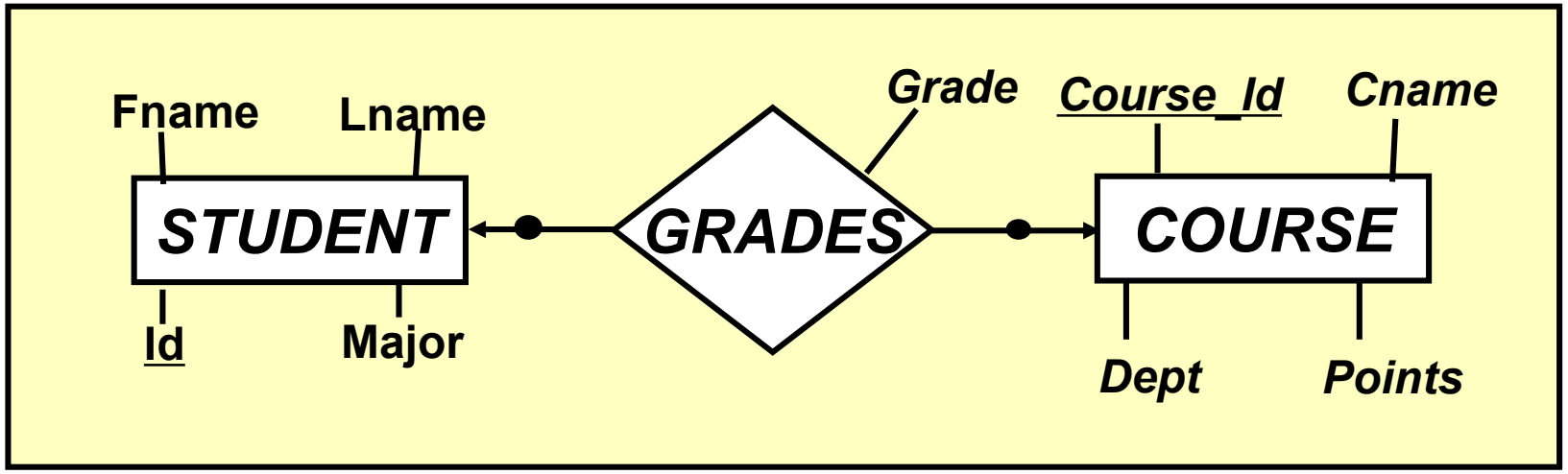
RELATIONAL Schema

STUDENT (Id, Lname, Fname, Major),
GRADES (Id, Course_id, Grade),
COURSE (Cname, Course_id, Points, Dept)

Schemas and Instances

- A schema diagram: an illustrative display of (most aspects of) a database schema
- A schema construct: a **component** of the schema or an object within the schema
  - e.g. STUDENT, COURSE

Enhanced Entity Relationship (EER) Schema Diagram



Schemas and Instances (continued)

- A database instance (or extension/state): the actual data in a database at a *specific moment in time*
  - produced by populating (loading) schema description with data
  - corresponds to schema structure, and
  - satisfies all schema constraints

- A database instance should reflect changes in the real system's state (i.e. via updates)

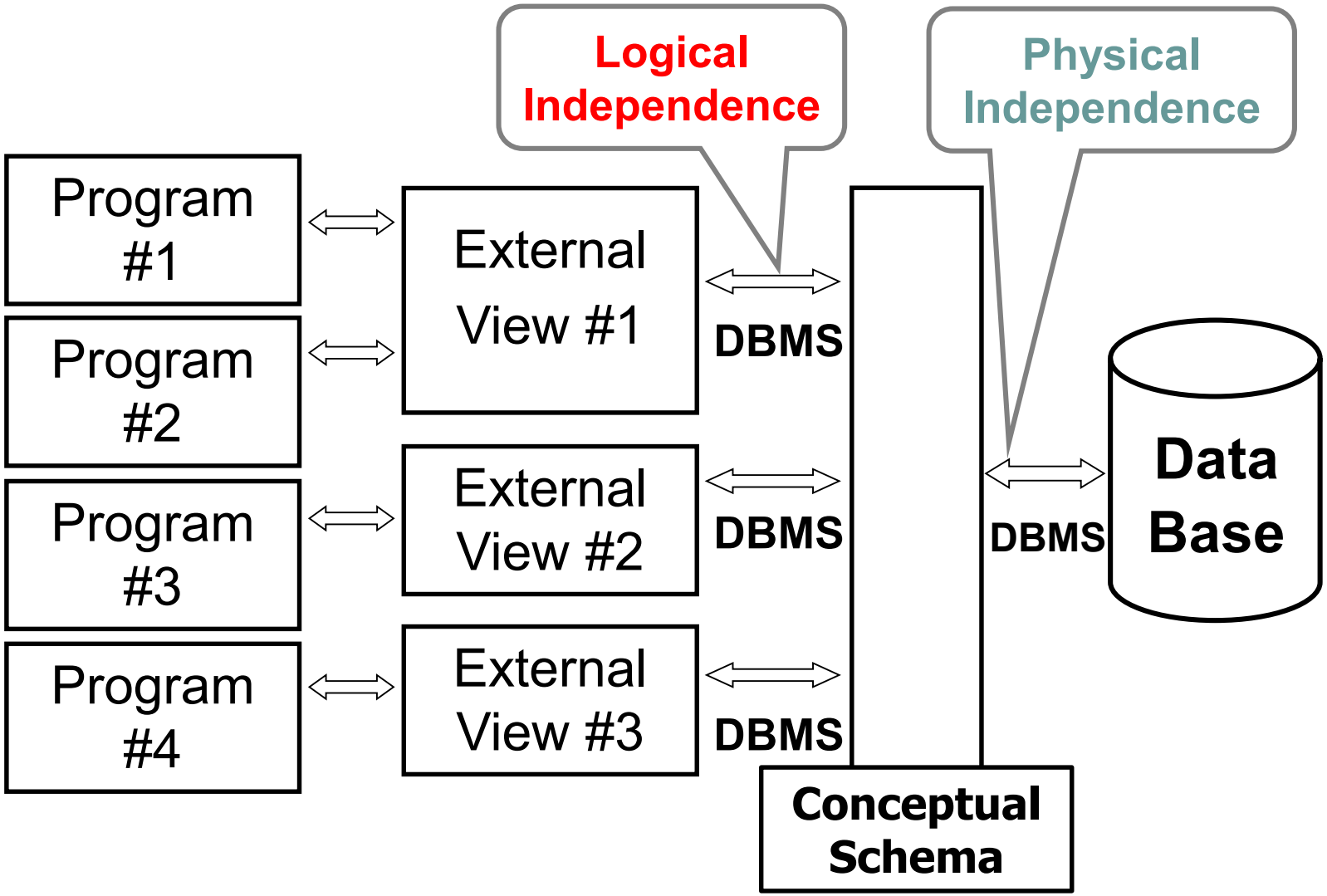
Data Independence

- In the traditional approach, information about a file's physical structure was embedded into programs
- The same was true for the first (pre-relational) database systems
- Changing the file structure induced changes in programs
- This is called program-data dependence
- To avoid this, we introduced procedures for **physical and logical data independence**

Logical Data Independence

- The external presentation/manipulation of the data is almost independent of the conceptual organization
- Changes to the conceptual/logical schema should not affect the external schema
- Only the mapping needs to be changed
- An application program should only see the external schema *(unfortunately, computationally hard to achieve – view update problem)*

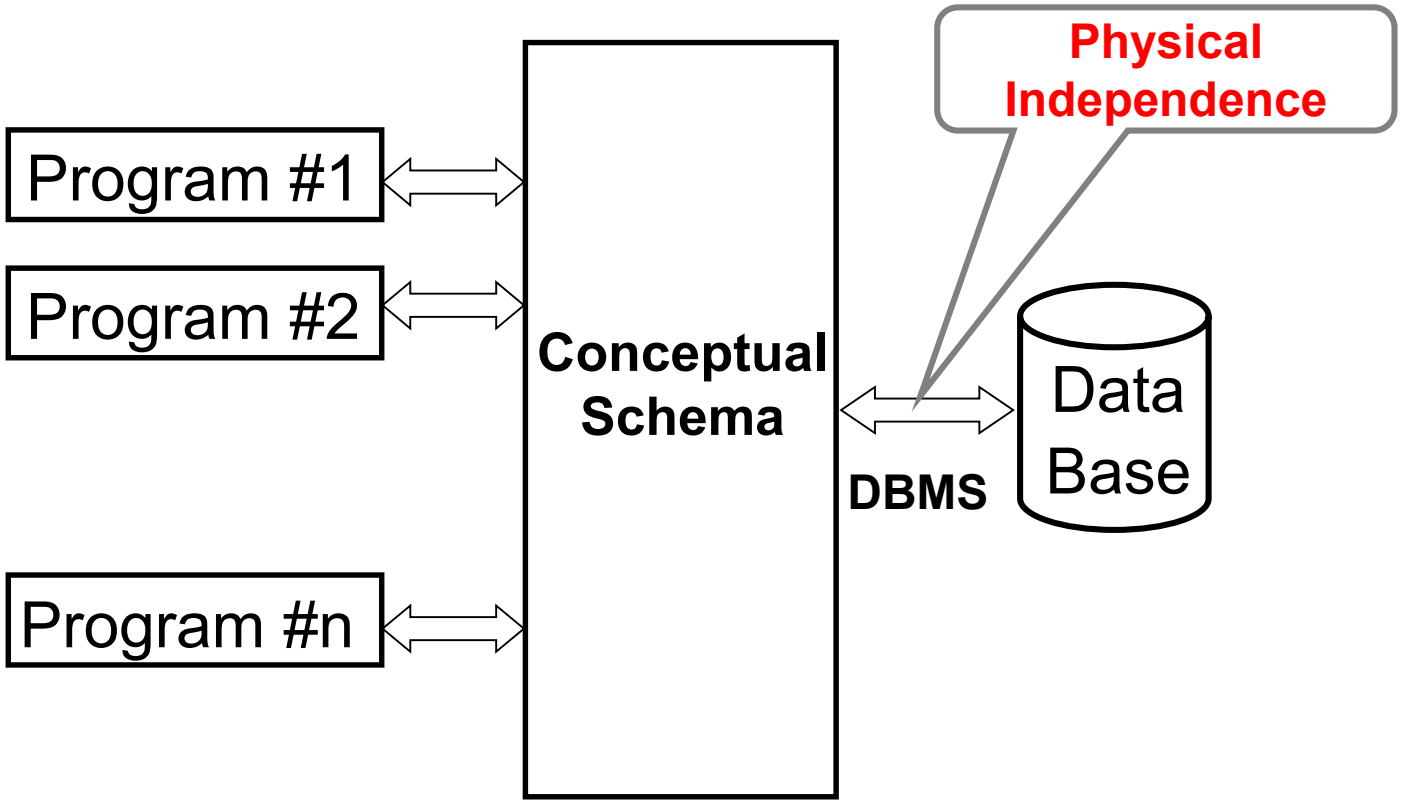
Logical Data Independence



Physical Data Independence

- The physical organisation of the data is almost independent of the conceptual organisation
- Changes to physical schema have no implications on the logical/conceptual layer
- Abstracts from the realisation of the DBMS storage organisation: allows reasoning about the data.
- Allow physical optimisation/tuning

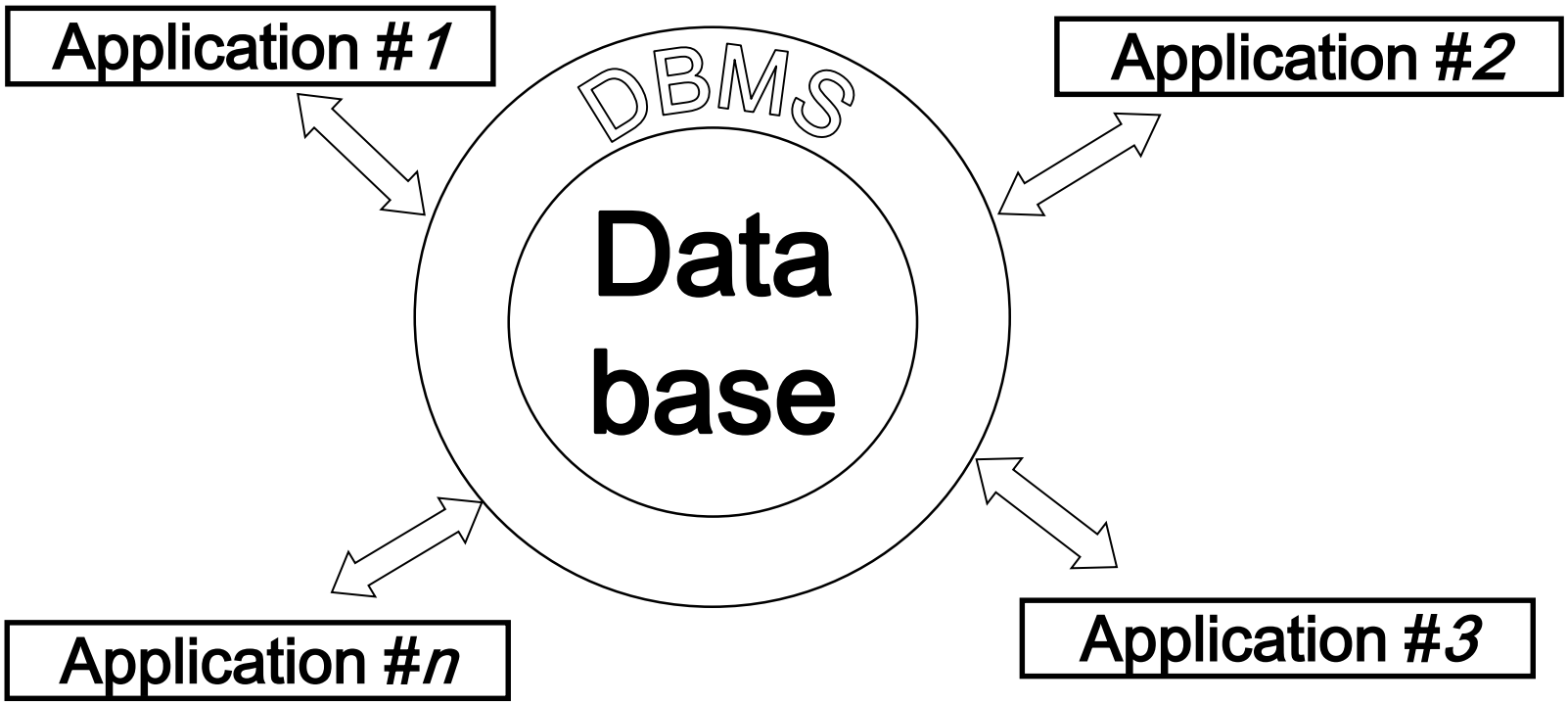
Physical Data Independence



Universe of TTUPA

- TTUPA: Some small part of the real world that we want to model
- Not to be confused with the *Unicorns of Discourse…*

Database Approach FOR TTUPA



Data Organization Approaches

- Suppose TTUPA functions (e.g. HR, Sales, Payroll Accounting) are provided by several applications

- There are two characteristic approaches to data organisation and storage in such an IT system:
  1. Traditional (file) approach: each application contains **all** necessary data, organised into files
  2. Database approach: all applications use a common database (no one application owns it)

Database Languages and Interfaces

- It is common to distinguish language according to its specific purpose:
  - A data definition language (DDL) is used to **define** the database schemata
  - A data manipulation language (DML) is used to **update** data in the database (insert, delete, modify)
  - A query language is used to **access** the data in the database and to **retrieve** data

Data Manipulation Language (DML)

- Used to retrieve, insert, delete, and modify data
- Always selects a (relatively) small part of a database and transfers it from disk to memory

- A DML can be:
  - Either Navigational, or
  - Declarative

Navigational DML

- A navigational DML:
  - Procedural (has loops, branching conditions),
  - Selects records one at a time
  - Programmer explicitly utilises information about the database's physical organisation to "navigate" through the database
  - Programmer defines WHAT and HOW
  - Any issues with this?

Declarative DML

- A declarative DML:
  - Non procedural
  - Set oriented (selects all data that matches given conditions)
  - Search conditions are defined according to abstract database representation; programs are completely independent of the DB's physical organization
- A programmer (or even a casual user) has to define just **WHAT**

- Pros/cons?

Navigational Versus Declarative DML

- Query: "Retrieve all Courses and Grades of the student with Id = 300111"

- Simplified navigational pseudo code:

```
Find record with Id= 300111 in GRADES
If successful then
    Do while there are courses connected to the student
        Find next course Id in GRADES
        Find corresponding Grade
        Find Course name in COURSE
    End do
Else
    Display error message
End if
```

Navigational Versus Declarative DML

- Query: "Retrieve all Courses and grades of the student with Id = 300111"

- A fully declarative program:

```
SELECT Course_id, Cname, Grade
FROM COURSE C, GRADES G
WHERE Id = 300111 AND G.Course_id = C.Course_id;
```

Database Users and Languages

- Database administrator (DBA):
  - DDL describes conceptual (or implementation) schema
  - View Definition Language (VDL) describes user views
  - Storage Definition Language (SDL) describes internal schema
- Casual end users:
  - Interactive declarative DML – mainly for database queries
- Naïve users:
  - Canned transaction programs (written by programmers)
- Programmers:
  - Interactive DML
  - DML embedded into general purpose programming language

Summary

- A **data model**: a set of concepts to describe the structure of a database, the operations for manipulating these structures, and certain constraints that the database should obey
- A **database instance** refers to the actual data stored in a database at a *particular moment in time*
- The **three schema architecture** (external, conceptual, and internal schemas): introduced to enable logical and physical data independence

- Advantages of the DB over traditional approach:
  - Program-data independence
  - Data consistency
  - DBMS controlled data sharing and recovery

Plan for the Next Lecture

- Introduction to the relational data model – motivations and basic ideas
- Basic terms/concepts of the relational data model
- Relational schemas and instances
- Constraints of the relational data model
- Reading:
  - Sections 5.1 and 5.2, Chapter 5 of the textbook

Set Notations

- We use set notation for formal definitions in this course
- A set is several things considered together as one thing

- There are two ways to specify a set:

1. $\{x_1, \ldots, x_n\}$, list all the elements in a set
   - {}, the empty set
   - {Wellington, Auckland, Christchurch}
   - {{*A*}, {*A*, *B*}, {*A*, *B*, *C*}}, a set of sets

2. $\{x|x \in \varphi\}$, describe the members that satisfy a property
   - $\{x|x \in \mathbb{N}\}$, the set of natural numbers
   - {students | all the students enrolled in SWEN304}

Set Notations

- If two sets have the same members, they are the same set
- If one set contains something that is not in the other set, then they are different
  - e.g. $\{1, 2, 3\} \neq \{1, 2, 4\}$
- The members in a set have no order
  - e.g. $\{1, 2\} = \{2, 1\}$
- Each element cannot be in the set more than once
  - E.g. $\{1, 1\} = \{1\}$

Set Operations

Membership: $x \in A$ if x is in set A
$x \notin A$ if x is not in set A

Equality: A and B are equal if they have the same members, A = B

Subset: If every member of A is in B, we write $A \subseteq B$; A is called a proper subset of B if $A \subseteq B$ and A and B are not equal

Set Operations

Union: $A \cup B$ for the set containing everything in A and B

e.g. {a,b,c} ∪ {a,c,d,e} = {a,b,c,d,e}

Intersection: $A \cap B$ for the set of elements that are in both A and B

e.g. {a,b,c} ∩ {a,c,d,e} = {a,c}

Difference: $A - B$ for the set of elements from A but not B

e.g. {a,b,c} – {a,c,d,e} = {b}

Examples of Set Operations

- Let A= {x, y, z}, B = {1, 2}

- Which of the following are correct?
  1. $\{y\} \subseteq A$
  2. $z \in A$
  3. $\{x, y\} \subseteq (A \cup B)$
  4. $x \in (A \cap B)$
  5. $x \in (A - \{y, z\})$

- $(\{x,1\} \cup A) \cap B = ?$

Cartesian Products of Sets

- A **n-tuple** is an ordered list of n elements
  - (Hui, 5657) is a 2-tuple, (x, y, 1, 2, 2) is a 5-tuple

- The **Cartesian product** operation takes an ordered list of sets and returns a set of tuples
- Cartesian product $D_1 \times \ldots \times D_n$ is the set of all possible combinations of values from the sets $D_1, \ldots, D_n$
- For example, D_1 ={Wellington, Auckland}, D_2 = {1, 2, 3}

$D_1 \times D_2$ ={(Wellington, 1), (Wellington, 2), (Wellington, 3), (Auckland, 1), (Auckland, 2), (Auckland, 3)}