Introduction to Databases

Paul Fodor

CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

Introduction

- What is a Database?
  - Collection of data central to some enterprise
  - A database is persistent
- What is a Database Management System (DBMS)?
  - A program that manages a database:
    - Supports creation and maintenance of databases
    - Supports a high-level query language (e.g. SQL)

Introduction

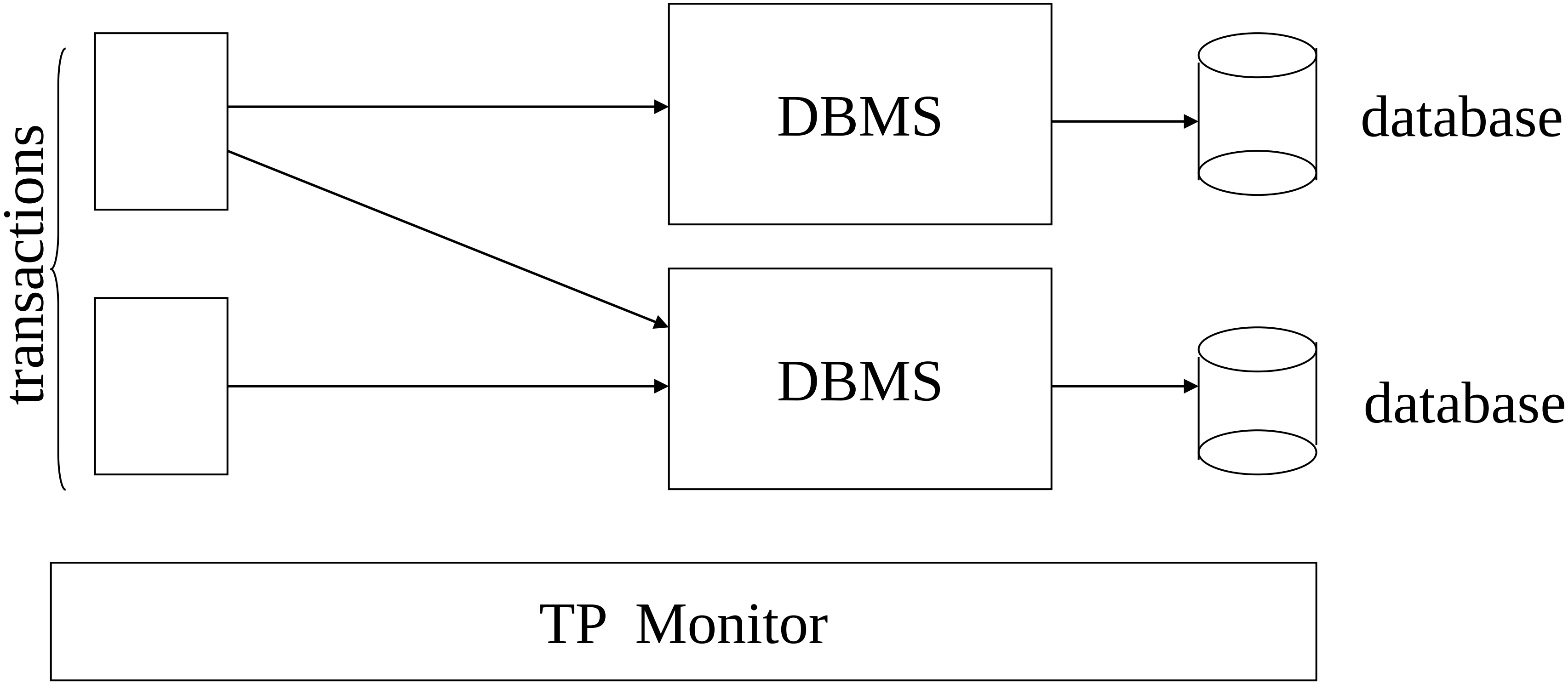
- What is a Transaction?
  - When an event in the real world changes the state of the enterprise, a "*transaction*" is executed to cause the corresponding change in the database state
    - A transaction is an application program with special properties (i.e., ACID=Atomicity, Consistency, Isolation, Durability) to guarantee it maintains database correctness

Introduction

- **On-line Transaction Processing** (OLTP)
  - Day-to-day handling of transactions that result from enterprise operation
  - Maintains correspondence between database state and enterprise state

Introduction

- Transaction execution is controlled by a Transaction Processing (TP) monitor



Introduction

- **On-line Analytic Processing** (OLAP)
  - Analysis of information in a database for the purpose of making management decisions
  - Analyzes historical data (terabytes) using complex queries
  - **Summarizes the data and makes forecasts!**
  - Example: it answers operational questions like "*What are the average sales of cars, by region and by year?*"

Introduction

- **Data Warehouse -** repository of historical data generated from OLTP or other sources
- **Data Mining** - use of warehouse data to *discover* relationships (discovers hidden patterns in data) that might influence enterprise strategy

Introduction

- Example: Supermarket:
  - OLTP
    - For the event of buying 1 milk and 1 box of diapers, the OLTP will **update** the database to reflect that event
  - OLAP
    - Last winter in all stores in northeast, how many customers bought milk and diapers together?
  - Data Mining
    - Are there any interesting combinations of products that customers frequently bought together?

Introduction

- Database Systems Requirements:
  - **High Availability**: always on-line = must be operational while the app is functioning
  - **Low Response Time**
  - **High Reliability**: correctly tracks state, does not lose data, controlled concurrency
  - **High Throughput**: many users and many transactions/sec

Introduction

- **Long Lifetime**: complex systems are not easily replaced
  - Must be designed so they can be easily extended as the needs of the enterprise change
- **Security**: sensitive information must be carefully protected since system is accessible to many users
  - Authentication
  - Encryption
  - Authorization

Introduction

- Roles in Design, Implementation, and Maintenance of a DBMS and TPS:
  - **System Analyst** - specifies system using input from customer; provides complete description of functionality from customer's and user's point of view
  - **Database Designer** - specifies structure of data (schema) that will be stored in database
  - **Application Programmer** - implements application programs (transactions) that access data and support enterprise rules
  - **Database Administrator** - maintains database once system is operational: space allocation, performance optimization, database security
  - **System Administrator** - maintains transaction processing system: monitors interconnection of HW and SW modules, deals with failures and congestion

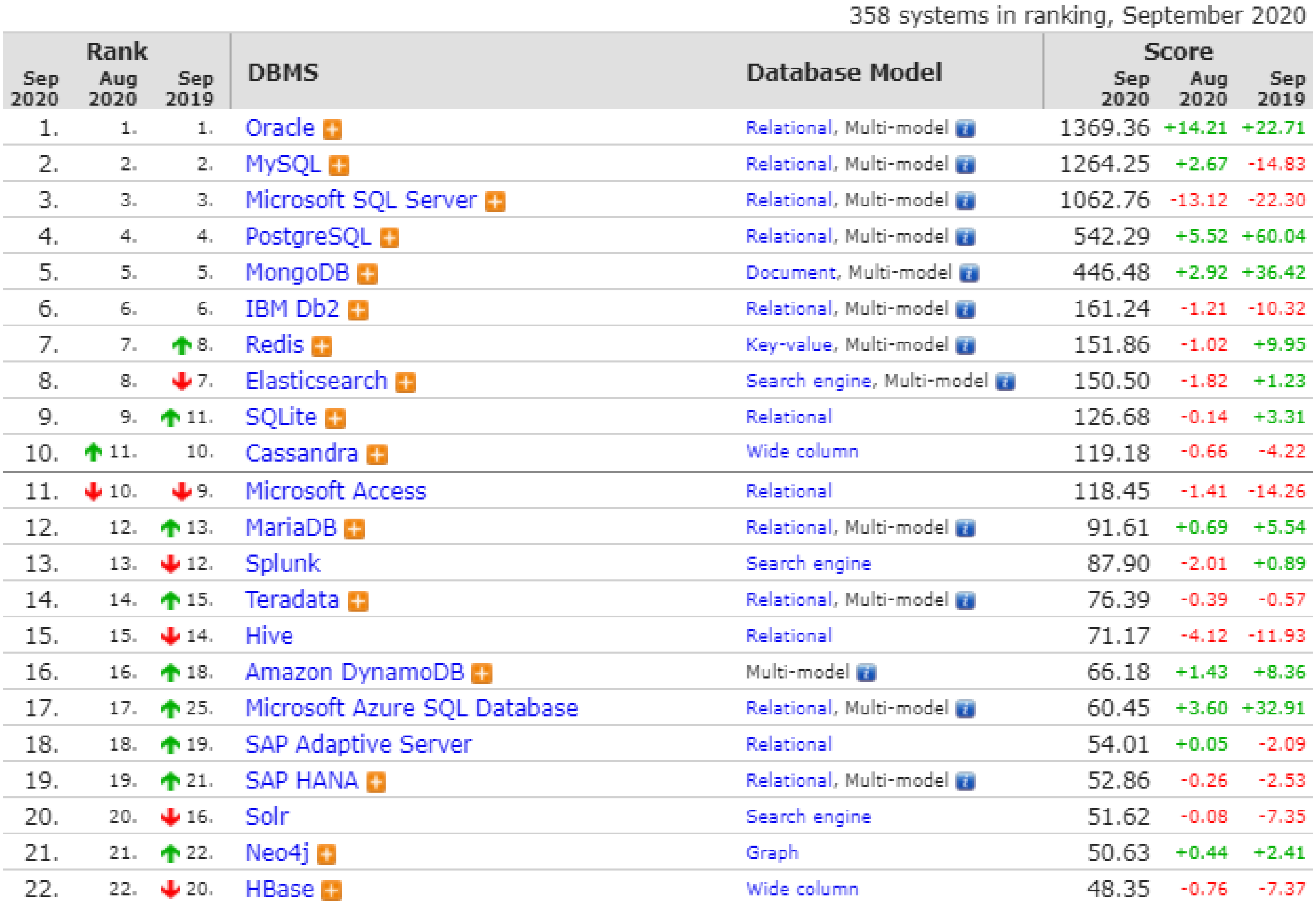
A Brief History of Database Systems

- Pre-relational era (1970's)
  - Hierarchical (IMS), Network (Codasyl)
  - Complex data structures and low-level query language
- Relational DBMSs (1980s)
  - Edgar F. Codd's relational model in 1970
  - Set of tuples (i.e., tables) as data model
  - Powerful high-level query language (SQL)
- Object-Oriented DBMSs (1990s)
  - Motivated by the "impedance mismatch" between RDBMS and OO PL
    - Data type differences
    - RDBMS does not have OO access levels, the notions of interfaces, inheritance, polymorphism
    - Lack of high level OO Query Language, Integrity Constraints and Transactions

A Brief History of Database Systems

- XML/JSON/NoSQL (2000s)
  - Triggered by the growth of the Web and by AJAX and REST
  - Native support of XML through ORDBMS extensions or native XML DBMS
  - Migration to key-value pairs and more complex JavaScript objects
  - Big data: datasets that grow so large (terabytes to petabytes) that they become awkward to work with traditional DBMS
  - MapReduce dominates on Web data analysis
  - Parallel DBMSs continue to push the scale of data
  - "NoSQL" (not only SQL) is fast growing
- Data Science (2010s)
  - Old: Data mining and OLAP (Online Analytical Processing), Big Data

Stay updated

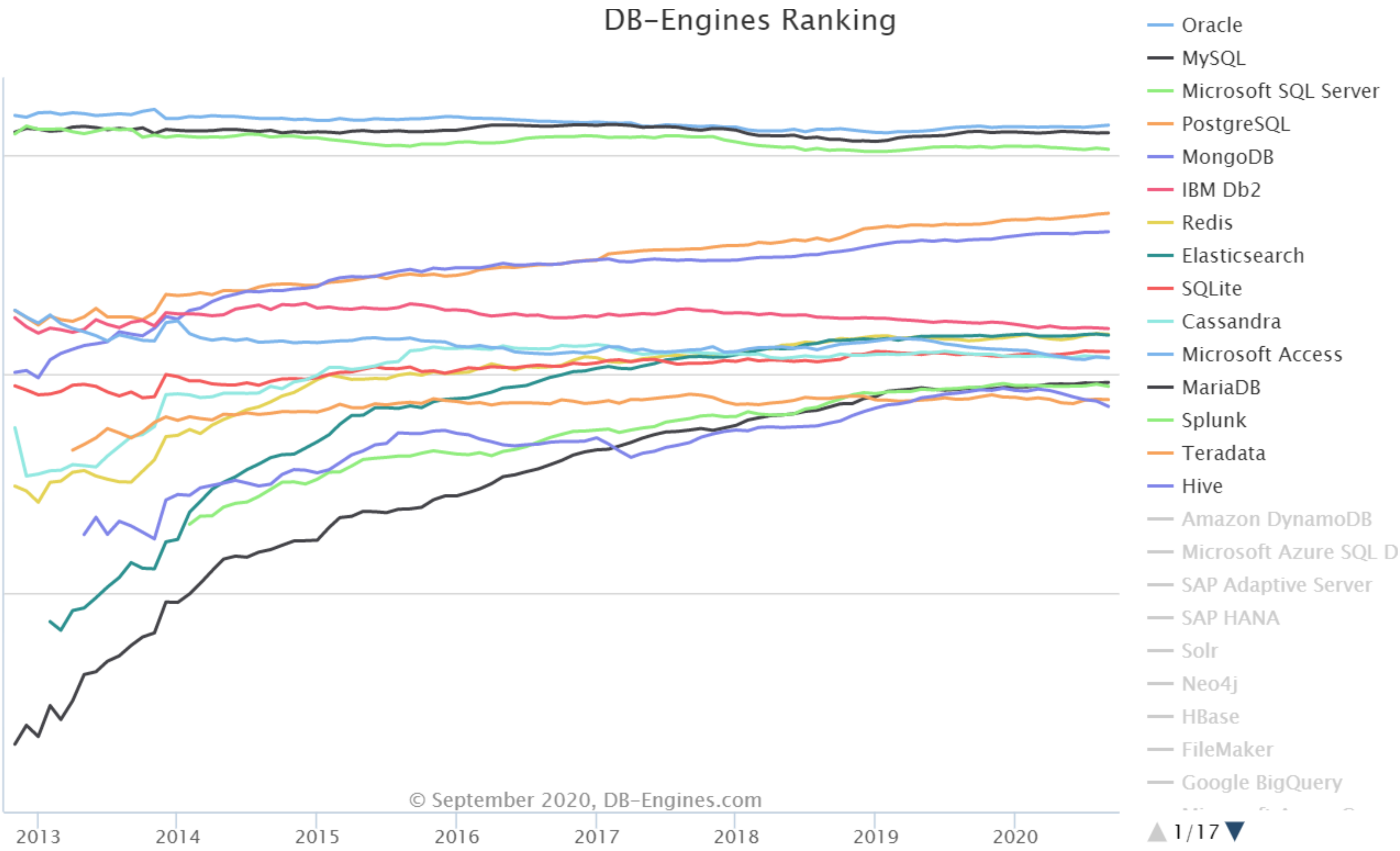


358 systems in ranking, September 2020

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22
11.	↓ 10.	↓ 9.	Microsoft Access	Relational	118.45	-1.41	-14.26
12.	12.	↑ 13.	MariaDB	Relational, Multi-model	91.61	+0.69	+5.54
13.	13.	↓ 12.	Splunk	Search engine	87.90	-2.01	+0.89
14.	14.	↑ 15.	Teradata	Relational, Multi-model	76.39	-0.39	-0.57
15.	15.	↓ 14.	Hive	Relational	71.17	-4.12	-11.93
16.	16.	↑ 18.	Amazon DynamoDB	Multi-model	66.18	+1.43	+8.36
17.	17.	↑ 25.	Microsoft Azure SQL Database	Relational, Multi-model	60.45	+3.60	+32.91
18.	18.	↑ 19.	SAP Adaptive Server	Relational	54.01	+0.05	-2.09
19.	19.	↑ 21.	SAP HANA	Relational, Multi-model	52.86	-0.26	-2.53
20.	20.	↓ 16.	Solr	Search engine	51.62	-0.08	-7.35
21.	21.	↑ 22.	Neo4j	Graph	50.63	+0.44	+2.41
22.	22.	↓ 20.	HBase	Wide column	48.35	-0.76	-7.37

http://db-engines.com/en/ranking

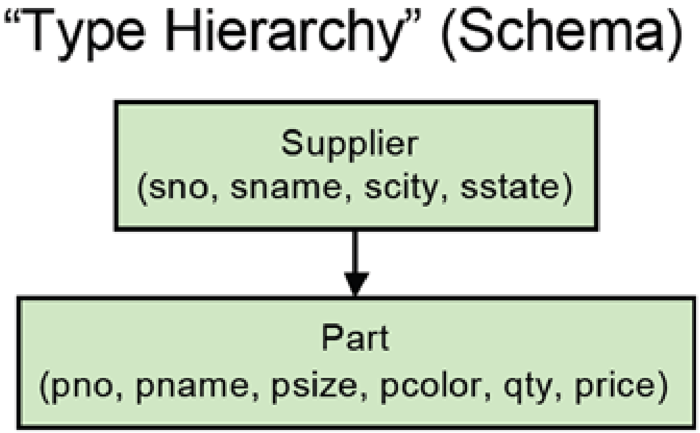
Stay updated



http://db-engines.com/en/ranking

Evolution of DBMS

- Hierarchical model (~1968):
  - record types arranged as a hierarchy


“Type Hierarchy” (Schema)

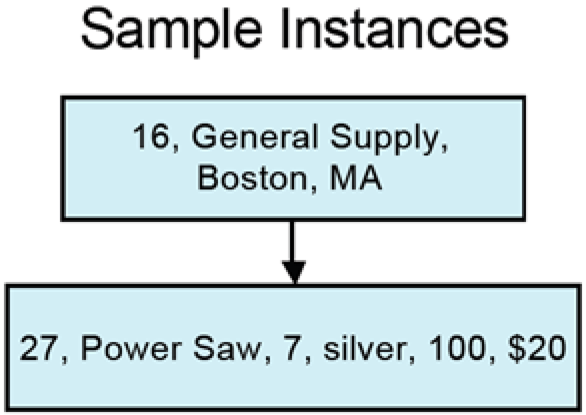
Supplier
(sno, sname, scity, sstate)

Part
(pno, pname, psize, pcolor, qty, price)

a supplier sells parts

a part is sold by only 1 supplier

- each type has a single parent

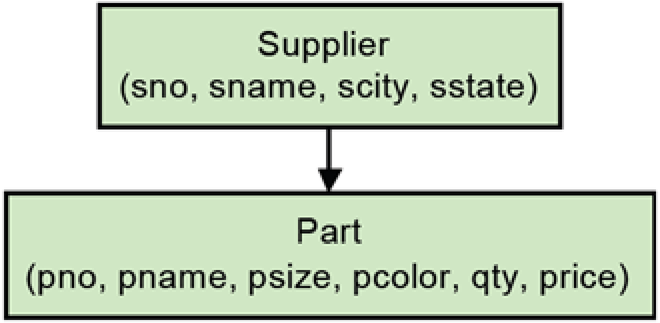

Sample Instances

16, General Supply,
Boston, MA

27, Power Saw, 7, silver, 100, $20

Evolution of DBMS

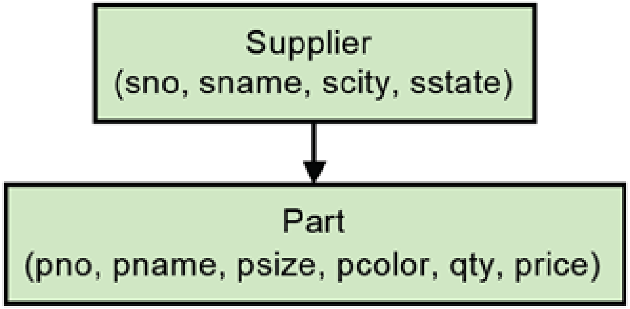
- Hierarchical model problems:
  - Existence of child data depends on parent data
    - What if there is a part that is not sold by any supplier?
    - If there is no parent, we cannot have the child.


Supplier
(sno, sname, scity, sstate)
Part
(pno, pname, psize, pcolor, qty, price)

Evolution of DBMS

- Hierarchical model queries:
  - DL/1 programming language: "*record-at-a-time*" language: the programmer constructs an algorithm for solving a query and IMS executes it
    - Example pseudocode: get all suppliers of parts that are red

```
Until-no-more{
    get next Supplier
    Until-no-more{
     get next Part
     check (color=red)
    }
}
```


Supplier
(sno, sname, scity, sstate)
Part
(pno, pname, psize, pcolor, qty, price)

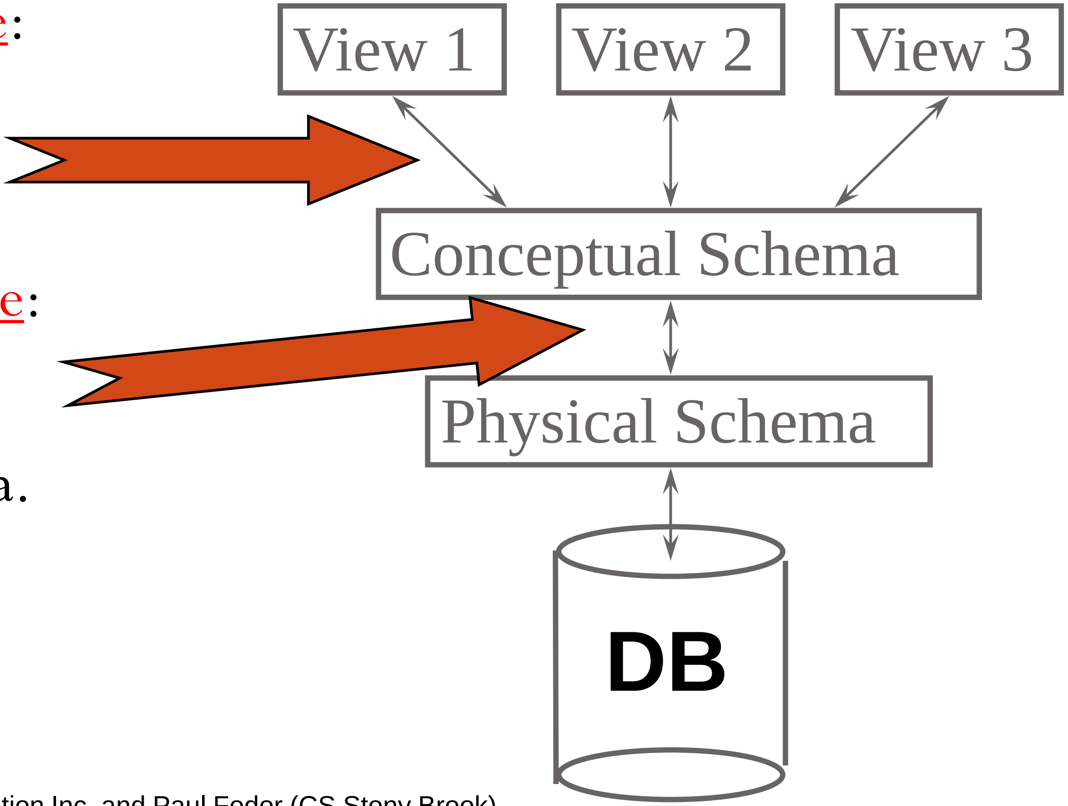
Evolution of DBMS

- Hierarchical model Storage:
  - Different underlying storage = different restrictions on commands: heavy coupling between storage format used (sequential/B-tree/hashed) and client application
    - Different sets of data = different optimization opportunities
      - even if the optimization is programmed by the programmer

Evolution of DBMS

- **Data Independence:**
  - A Simple Idea: Applications should be insulated from how data is structured and stored

- Logical data independence: protection from changes in the logical structure of data.
- Physical data independence: protection from changes in the physical structure of data.



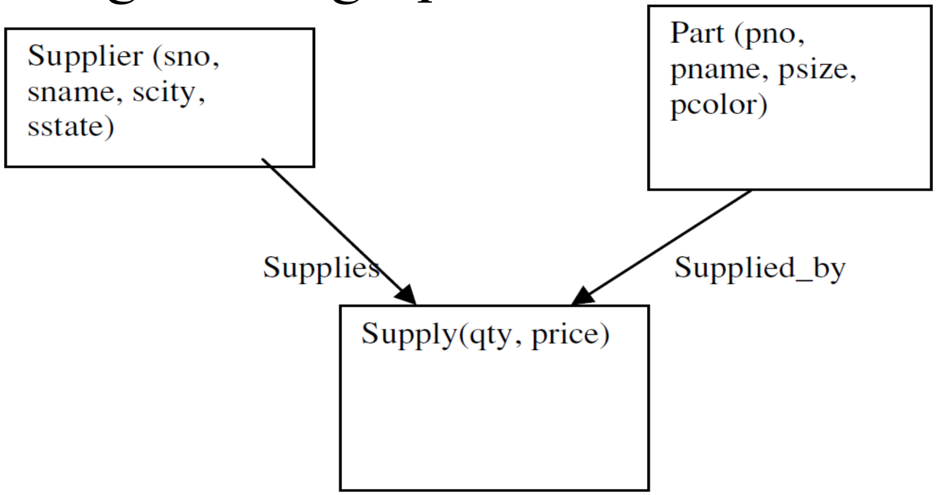
(c) Pearson Education Inc. and Paul Fodor (CS Stony Brook)

Evolution of DBMS

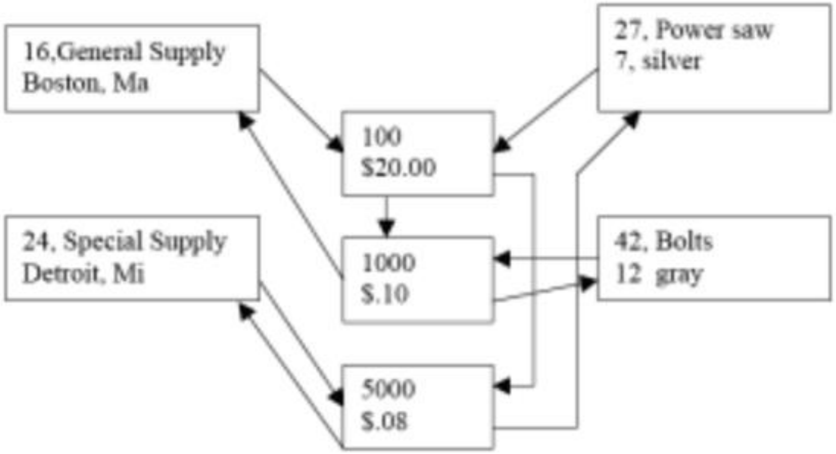
- Logical data independence:
  - changes to logical structure should not require changes at the application level (ideally)
    - in general, should not require expensive changes to apps
  - Impossible to achieve in the hierarchical model, where:
    - trees are difficult to reorganize
    - the record-at-a-time language delegates the optimization to the programmer

Evolution of DBMS

- Graph / Network model (CODASYL 1969):
  - Schema arranged in a graph model


Supplier (sno, sname, scity, sstate)
Part (pno, pname, psize, pcolor)
Supplies
Supplied_by
Supply(qty, price)

  - Instance:

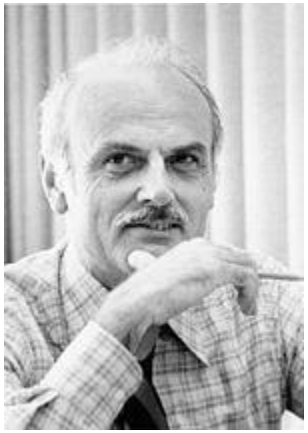


On evolution of DBMS

- Graph / Network model (CODASYL 1969):
  - Improvement to the hierarchical model:
    - entities can exist without their parents
  - Limitations:
    - still using the record-at-a-time DML language
    - still no physical data independence
    - more difficult to program against a graph than a tree
    - graphs are more complex: the whole graph must be loaded at once (trees could be loaded individually)

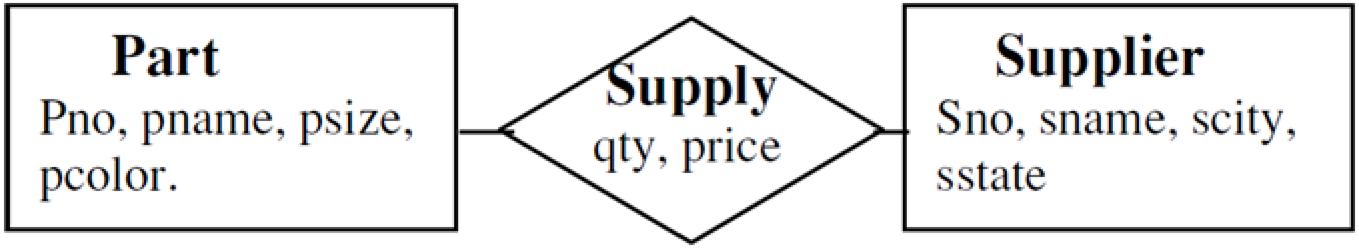
Evolution of DBMS

- Relational model (1970)
  - Ted Codd was motivated by the heavy maintenance required by the applications
  - Ideas:
    - data stored in tables/relations <- simple intuitive model
    - high level set oriented data model Data Manipulation Language (DML) for adding (inserting), deleting, and modifying (updating) data in a database.
    - underlying physical storage is up to vendors of DBMS



Evolution of DBMS

- Entity Relationship (mid 1970s)
  - Proposed by Peter Chen
  - Relationships with attributes and multiplicities



Part
Pno, pname, psize, pcolor.

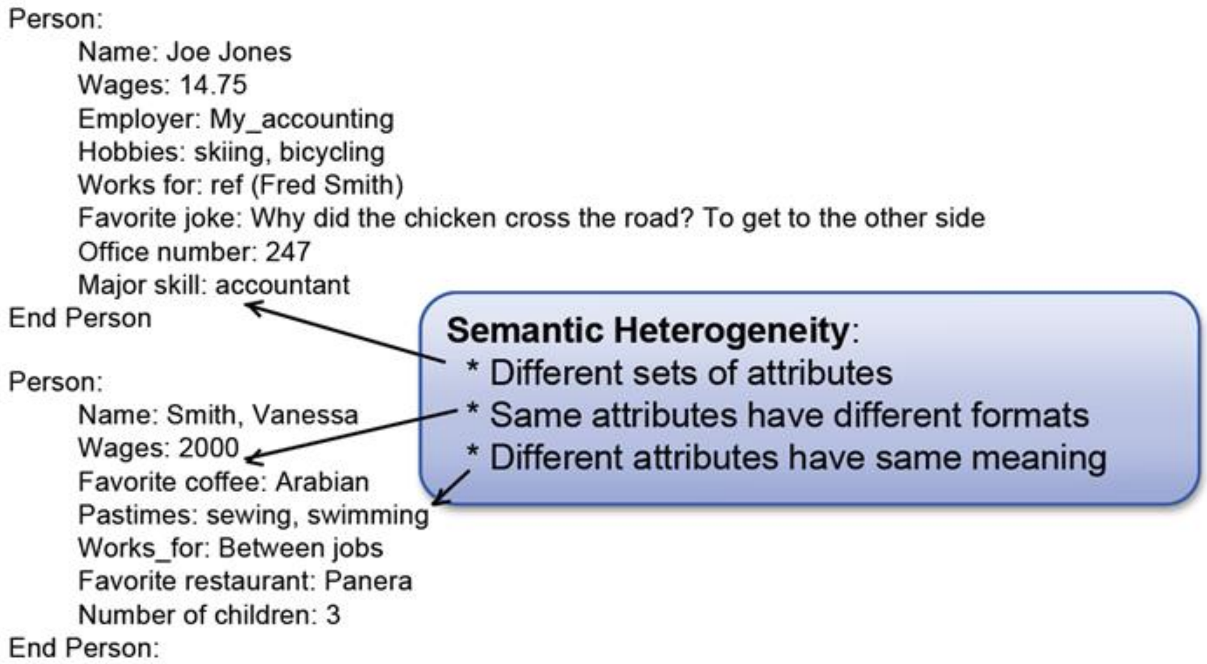
Supply
qty, price

Supplier
Sno, sname, scity, sstate

  - As a physical model: never caught on (little benefit)
  - As a conceptual model: widely used for database schema design because it offers a methodology for creating initial tables and some normalization on E-R models can be done automatically

Evolution of DBMS

- Semi-structured era (~2000+)
  - Schema Evolution OR Schema "later": data is self describing



```
Person:
        Name: Joe Jones
        Wages: 14.75
        Employer: My_accounting
        Hobbies: skiing, bicycling
        Works for: ref (Fred Smith)
        Favorite joke: Why did the chicken cross the road? To get to the other side
        Office number: 247
        Major skill: accountant
End Person

Person:
        Name: Smith, Vanessa
        Wages: 2000
        Favorite coffee: Arabian
        Pastimes: sewing, swimming
        Works_for: Between jobs
        Favorite restaurant: Panera
        Number of children: 3
End Person:
```

Semantic Heterogeneity:
 * Different sets of attributes
 * Same attributes have different formats
 * Different attributes have same meaning

  - A Response to the growth of Web services and XML as a language (same for JSON as Javascript)
  - Complex graph-oriented data models

Evolution of DBMS

- Semi-structured era (~2000+)
  - Relational DBMS have heavy-weight mechanisms to change schema (ALTER)
  - XML and JSON as a data model:
    - records can be hierarchical
    - records can still reference to other records through paths (i.e., XPath)
    - schema can be defined "later" in DTDs and XMLSchema

What is a Data Model?

- All data is recorded as bits and bytes on a disk, but it is difficult to work with data at that level.
- It is convenient to view data at different *levels of abstraction*
  - Programmers prefer to work with data stored in *files.*
  - Database developers work with data in an abstract data model.
    - A data model is a mathematical representation of data
      - Examples:
        - relational model = tables;
        - semistructured model = trees/graphs.
      - Determines the operations on the data.
      - Determines the constraints on the language.

Physical Data Level

- *Physical schema* describes details of how data is stored: cylinders, tracks, indices etc.
- Early applications worked at this level – explicitly dealt with the details above (physical cylinders, tracks).
- **Problem:** Routines were hard-coded to deal with physical representation
  - Changes to data structure are difficult and very expensive to make.
  - Application code becomes complex since it must deal with details.
  - Rapid implementation of new features is impossible.

Conceptual Data Level

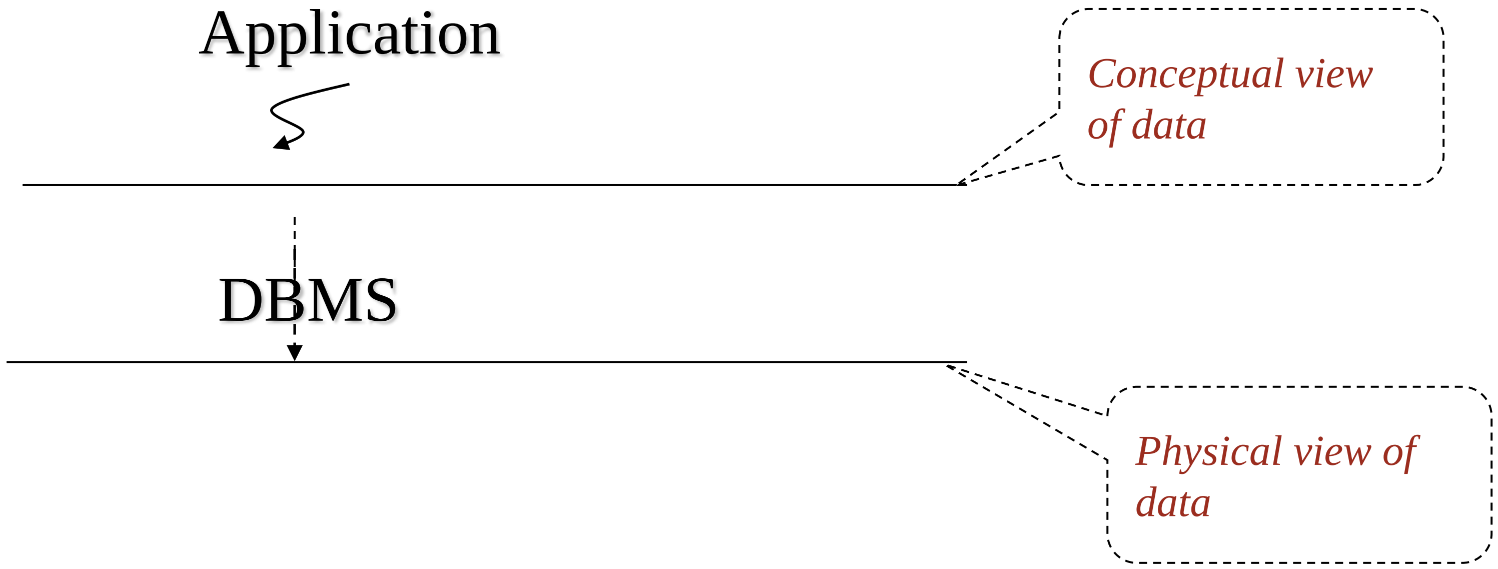
- The *Conceptual Data Level* hides the details of the physical data representation and instead describes data in terms of higher-level concepts that are closer to the way humans view it.
    - In the *relational model*, the conceptual schema presents data as a set of tables, e.g.:

Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)

- The DBMS maps from conceptual to physical schema automatically.

Conceptual Data Level

- Physical schema can be changed without changing application:
  - The DBMS would change mapping from conceptual to physical transparently
  - This property is referred to as ***physical data independence***



External Data Level

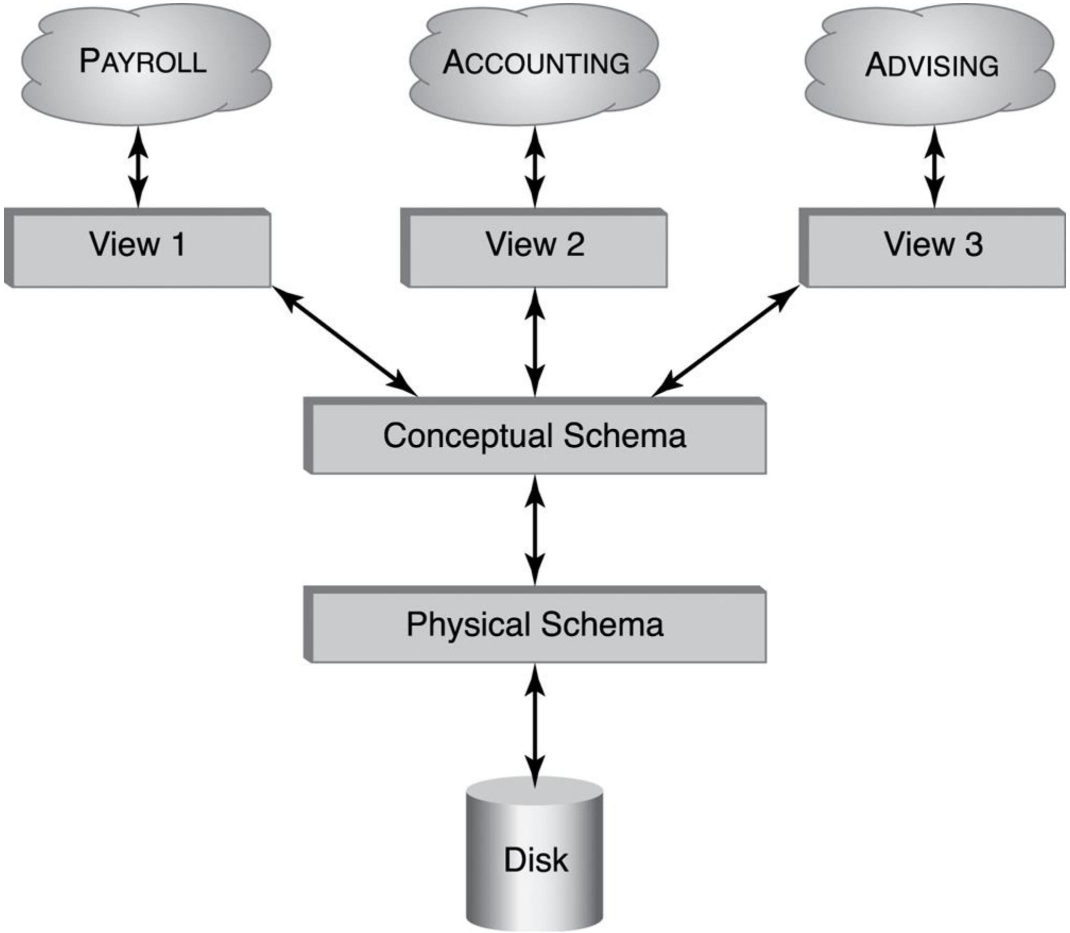
- The *External Data Level* customize the conceptual schema to the needs of various classes of users (it also plays a role in database security).
    - In the relational model, the *external schema* also presents data as a set of relations.
    - An external schema specifies a *view* of the data in terms of the conceptual level.
- The external schema looks and feels like a conceptual schema, and both are defined in essentially the same way in modern DBMSs.
    - There might be several external schemas (i.e., views on the conceptual schema), usually one per user category.

External Data Level

- It is tailored to the needs of a particular category of users.
  - Portions of stored data should not be seen by some users:
    - Students should not see their colleagues data (HWs, IDs, grades).
    - Faculty should not see billing data.
  - Information that can be derived from stored data might be viewed as if it were stored:
    - GPA not stored, but calculated when needed.

External Data Level

- Applications are written in terms of external schemas.
- A view is computed when accessed (not stored).
- Translation from external to conceptual done automatically by DBMS at run time.



(c) Pearson Education Inc. and Paul Fodor (CS Stony Brook)

External Data Level

- Conceptual schema can be changed without changing application (referred to as *conceptual data independence*):
  - Only the mapping from external to conceptual must be changed.

Data Model

- A *data model* consists of a set of concepts and languages for describing:
  - The conceptual and external schema
    - *Data definition language (DDL)*
    - Including the integrity constraints and domains
  - The operations on data
    - *Data manipulation language (DML)*
      - *inserts, deletes, updates, transactions.*
  - The directives that influence the physical schema (affects performance, not semantics)
      - *Storage definition language* (SDL)

Transactions

- Many enterprises use databases to store information about their state
  - *E.g.*, balances of all depositors
- The occurrence of a real-world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way
  - *E.g.*, balance must be updated when you withdraw
- A *transaction* is a program that accesses the database in response to real-world events

Transactions

- Transactions are not just ordinary programs
- Additional requirements are placed on transactions (and particularly their execution environment) that go beyond the requirements placed on ordinary programs.
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability

ACID properties

Atomicity

- The system must ensure that the transaction either runs to completion (i.e., *commits*) **or**, if it does not complete, has no effect at all (as if it had never been started) (i.e., *aborts*).
  - This is not true of ordinary programs. A hardware or software failure could leave files partially updated.
  - The TP monitor has the responsibility of ensuring that whatever partial changes the transaction has made to the database are undone (i.e., *rolled back*)

Integrity Constraints

- Or Consistency Constraints.
- Rules of the enterprise generally **limit the occurrence of certain real-world events**.
  - Student cannot register for a course if current number of registrants = maximum allowed
- Correspondingly, **allowable database states are restricted**.
  - All database states must have:

    current_registrations <= *maximum_registrations*
- These limitations are expressed as *integrity constraints*, which are **assertions** that **must be satisfied by the database state**.

Consistency

- A transaction must access and update the database in such a way that it preserves all database integrity constraints.
- The transaction designer must ensure that:

  IF the database is in a state that satisfies all integrity constraints when execution of a transaction is started

  THEN when the transaction completes:

  - All integrity constraints are once again satisfied (constraints can be violated in intermediate states)

Consistency

- Examples:
  - The database contains the Id of each student:
    - IC1: The Id of each student must be unique.
  - The database contains a list of prerequisites for each course and, for each student, a list of completed courses.
    - IC2: A student cannot register for a course without having taken all prerequisite courses.

Consistency

- The database contains the maximum number of students allowed to take each course and the number of students who are currently registered for each course.
  - IC3: The number of students registered for each course cannot be greater than the maximum number allowed for that course.

Consistency

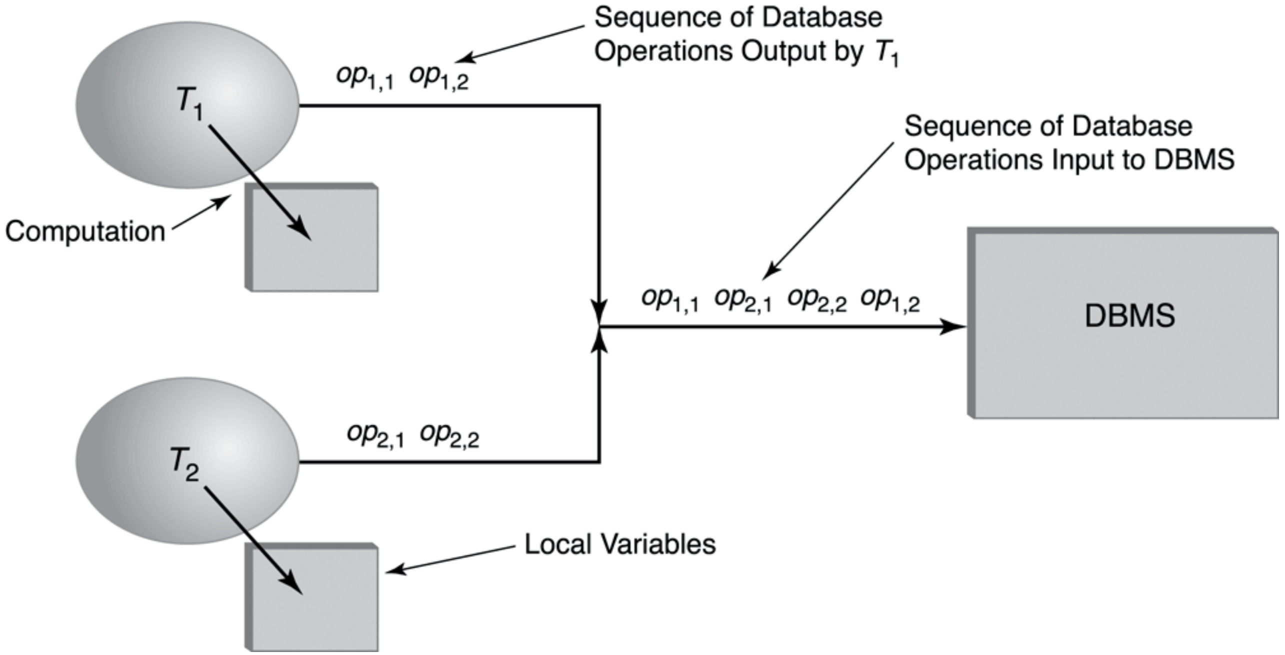
- It might be possible to determine the number of students registered for a course from the database in two ways: the number is stored as a count in the course, and computed from the information describing each student by counting the number of student records that indicate that the student is registered for (or enrolled in) the course
  - IC4: the two determinations must yield the same result.

Isolation

- A set of transactions is executed *sequentially*, or *serially*, if one transaction in the set is executed to completion before another is started.
  - If all transactions are consistent and the database is initially in a consistent state, serial execution maintains consistency.
  - But serial execution is *inadequate* from a performance perspective

Isolation

- *Concurrent execution* is when multiple transactions are executed simultaneously.
  - different transactions are effectively interleaved in time


Sequence of Database Operations Output by T_1
$op_{1,1}$ $op_{1,2}$
T_1
Computation
Sequence of Database Operations Input to DBMS
$op_{1,1}$ $op_{2,1}$ $op_{2,2}$ $op_{1,2}$
DBMS
T_2
$op_{2,1}$ $op_{2,2}$
Local Variables

Isolation

- Concurrent (interleaved) execution of a set of transactions offers performance benefits, but <u>might not be correct</u>.
- **Example**: Two students execute the course registration transaction at about the same time (*cur_reg* is the number of current registrants)

T_1:	read(*cur_reg* : 29)			write(*cur_reg* : 30)
T_2:		read(*cur_reg* : 29)	write(*cur_reg* : 30)	

--->*time*

Result: Database state no longer corresponds to real-world state, integrity constraint violated.

Lost update: one of the increments has been lost.

Isolation

- *Isolation* = Even though transactions are executed concurrently, the overall effect of the schedule must be the same as if the transactions had executed serially in some order.
- The effect of concurrently executing a set of transactions must be the same as if they had executed serially in some order
  - The execution is thus *not* serial, but *serializable*

Isolation

- Serializable execution has better performance than serial, but performance might still be inadequate.
  - Database systems offer several isolation levels with different performance characteristics (but some guarantee correctness only for certain kinds of transactions – not in general)

Durability

- The system must ensure that once the transaction commits, its effects remain in the database even if the computer, or the medium on which the database is stored, subsequently crashes.
  - Example: if a student successfully registers for a course, he/she expects the system to remember that he/she is registered even if the system later crashes.

ACID Properties

- The transaction monitor is responsible for ensuring atomicity, (the requested level of) isolation and durability.
  - Hence it provides the abstraction of failure-free, non-concurrent environment, greatly simplifying the task of the transaction designer.
- The transaction designer is responsible for ensuring the consistency of each transaction, but doesn't need to worry about concurrency (because of isolation) and system failures (because of atomicity and durability).

Summary: ACID Properties

- Atomicity = Each transaction is executed completely or not at all.
- Consistency = Each transaction maintains database consistency.
- Isolation = The concurrent execution of a set of transactions has the same effect as some serial execution of that set.
- Durability = The effects of committed transactions are permanently recorded in the database.