Supplementary for Weeks 3 & 4

COMP1041 DBI AY2019-20 Spring Semester

Outline

- This document is purposed to provide further insights, explanations, clarifications, and disambiguation to the points talked in the lecture slides, includcing the topics of
  - ACID
  - Data model and corresponding course planning
  - Declarative vs. Procedural programming
  - Cartesian product
  - Grouping

ACID (Atomicity, Consistency, Isolation, Durability)

- ACID is the fundamental database property. You cannot call a system as database if it cannot hold those properties.
- **Atomicity**
  - A meaningful database operation/action, formally called a transaction, cannot be further divided into meaningful sub-operations.
  - E.g., in a banking db system, the action of fund transferring can be treated as a transaction. It is atomic. It can be divided into two sub-operations: debit the amount from a user, and credit the same amount to another. But neither of the two sub-operations can be called a transaction, because the bank's operating logic cannot allow debiting a user and credit to nowhere, or vice versa. For a db, each of its transaction is strictly atomic.
  - Clarification: there was a neglected point in my video recording about the scope of atomicity – it not only refers to the indivisibility of a data item, but also refers to the indivisibility of transactions.

ACID (Atomicity, Consistency, Isolation, Durability)

- **Consistency**
  - The database should remain invariant at all the time. That is, any data processed by the system should follow the same rule at all the time.
  - E.g., the referential integrity should always be followed when defining foreign-primary keys relationship.
- **Isolation**
  - Concurrent operations by multiple users should guarantee the correctness of the database transactions.
- **Durability**
  - When writing the results to the db, we should ensure that the results won't go away, even in the situation of power failure or system crash.

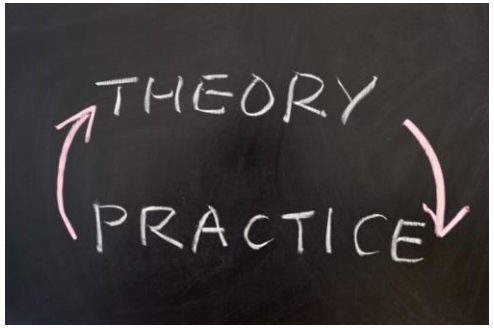
Data model

- When modeling something, you are in fact thinking of how to describe it, mathematically or some other ways.
- When modeling a database, one may start from many abstraction layers, from low to high:
  - Physical layer, e.g., how and where to represent a data (bit lengths, in cache or memory or hard disk, file format, etc.)
  - Logical layer, e.g., Using a relation/table to represent an entity? What are the available operations that can be done on one or more relations? How are different relations related to each other? How to optimize the relation representations?
  - Application/user/view layer, e.g., how to best represent the needed data to the users?

Topics covered/planned in the Database part of DBI

- Intro to DB (W3)
- Relational Algebra (W4)
- SQL programming (W5-7)
- Entity-Relationship model (W8)
- Normalization (W9)
- Revision (W10)

Theoretical: Using a relation/table to represent an entity? What are the available operations that can be done on one or more relations? How are different relations related to each other? How to optimize the relation representations?



Practical: SQL programming practices. Have fun!

Declarative vs. Procedural Programming

- Procedural programming
  - The programmer describes the what s/he wants step-by-step.
  - Examples are C and other legacy languages like Java, MIPS, PHP, etc.
- Declarative programming
  - The programmer describes the results or goal s/he desires, but don't have to care about how it is achieved (blackbox).
  - Examples are SQL, Java (also), yacc, lex, HDL, etc.
- Note that there is no clear classification between these two styles. Many languages exhibit both. E.g., SQL is mostly declarative, but also procedural.

Cartesian Product

- For two sets, the Cartesian product just enumerates all the combinations of the corresponding elements in the two sets in a pair-wise manner.
- E.g., set1 = {a1, a2}, set2 = {b1, b2}, then set1 × set2 = {(a1, b1), (a1, b2), (a2, b1), (a2, b2)}.

- E.g., s1 = {(a1, a2), (b1, b2)}, s2 = {(c1, c2), (d1, d2)}, then s1 × s2 = { ((a1, a2), (c1, c2)), ((a1, a2), (d1, d2)), ((b1, b2), (c1, c2)), ((b1, b2), (d1, d2)) }

Cartesian Product

- E.g., s1 = {(a1, a2), (b1, b2)}, s2 = {(c1, c2), (d1, d2)}

s1.attr1	s1.attr2
a1	a2
b1	b2

×

s2.attr1	s2.attr2
c1	c2
d1	d2

s1.attr1	s1.attr2	s2.attr1	s2.attr2
a1	a2	c1	c2
a1	a2	d1	d2
b1	b2	c1	c2
b1	b2	d1	d2

Division

- Integer division, 7 ÷ 3
  - Semantic: how many ‘multiples’ of 3 exist in 7? There are 2.
- Relational division, A ÷ B
  - Semantic: which/how many clientNo are attached to both PG4 and PG36?
  - One clientNo in the result represents a ‘multiple’ of PG4 and PG36.

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36
CR76	PG36

A

propertyNo
PG4
PG36

B

clientNo
CR76
CR56

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

ρ_R(branchNo, myCount, mySum) branchNo COUNT staffNo, SUM salary (Staff)

branchNo	myCount	mySum
B003	3	54000
B005	2	39000
B007	1	9000