CS2102

AY21/22 SEM 1

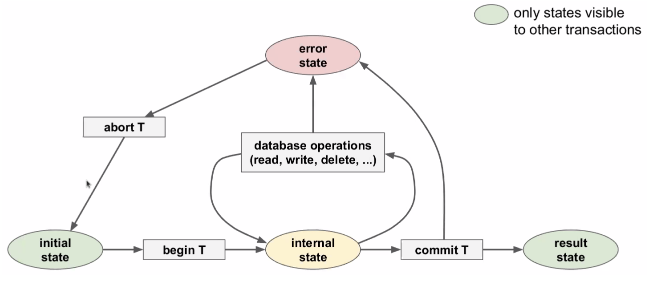
github/jovyntls

01. DBMS: DATABASE MANAGEMENT SYSTEMS

- set of universal and powerful **functionalities** for data management
- **database system**: DBMS (functionality) supporting several databases
  - DBS = DMBS + n*DB
- **data model**: framework to specify the structure of a DB
- **schema**: describes the DB structure using concepts provided by the data model
- **schema instance**: content of a DB at a particular time

Transactions

- **transaction**, T: a finite sequence of database operations
  - smallest logical unit of work from an application perspective
- guarantees the **ACID** properties



only states visible to other transactions
error state
abort T
database operations (read, write, delete, ...)
initial state
begin T
internal state
commit T
result state

ACID properties

1. **Atomicity** → either all effects of T are reflected in the database, or none
2. **Consistency** → the execution of T guarantees to yield a *correct state* of the DB
3. **Isolation** → execution of T is *isolated* from the effects of concurrent transactions
4. **Durability** → after the commit of T, its effects are *permanent* in case of failures

Serial vs Concurrent Execution

Serial Execution

- ✓ *correct* final result
- × less (unoptimised) resource utilisation; low throughput

Serializability

- Requirement for Concurrent Execution: **serializable transaction execution**
  - (concurrent execution of a set of transactions is) **serializable** → execution is equivalent to some serial execution of the same set of transactions
  - **equivalent** → they have the same *effect* on the data

Core tasks of DBMS

- Support *concurrent executions* of transactions - to optimise performance
- enforce *serializability* of concurrent executions - to ensure integrity of data

01-1. RELATIONAL MODEL

- **relation schema** → defines a relation
  - specifies the **attributes** (columns) and data constraints
  - **data constraints** → limits the kind of data you can put into the database
- **relational database schema** → set of relation schemas + data constraints
  - TableName(col_1, col_2, col_3) with dom(col_1) = {x, y, z}, ...
- **relational database** → collection of tables
- **domain** → a set of *atomic* values
  - domain of attribute A_i, $dom(A_i)$ = set of possible values for A_i
  - for each value v of attribute A_i, $v \in dom(A_i)$ or $v =$ `null`
    - `null`: special value indicating that v is not known or specified
  - e.g. dom(course) = {cs2102, cs2030, cs2040}
- **relation** → a set of *tuples*
  - $R(A_1, A_2, \ldots, A_n)$: relation schema with name R and n attributes $A_1, A_2, \ldots, A_n$
  - each instance of schema R is a relation which is a subset of $\{(a_1, a_2, \ldots, a_n) \mid a_i \in dom(A_i) \cup \{null\}\}$

01-2. ENSURING DATA INTEGRITY

- **integrity constraint** → condition that restricts what constitutes valid data
  - DBMS will check that tables only ever contain valid data
- **structural** → (integrity) inherent to the data model
- 3 main strucutral integrity constraints of the Relation Model
  1. Domain constraints
  2. Key constraints
  3. Foreign key constraints

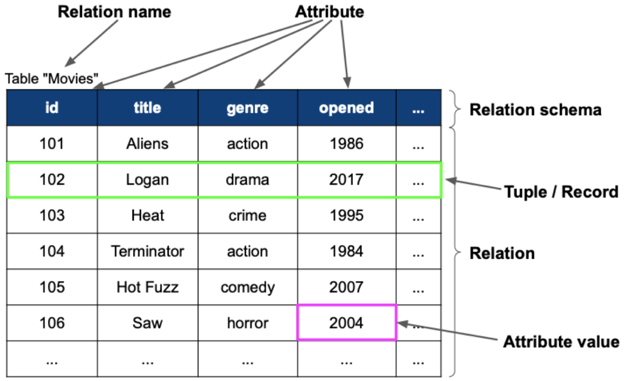
Key Constraints

- **superkey** → subset of attributes that *uniquely* identifies a tuple in a relation
  - e.g. {id, title}
- **key** → superkey that is also **minimal**
  - no proper subset of the key is a superkey
  - e.g. {id}
- **candidate keys** → set of all keys for a relation
- **primary key** → selected candidate key for a relation
  - *cannot* be `null` ⇒ **entity integrity constraint**

Foreign Key Constraints

- **foreign key** → subset of attributes of relation A if it refers to the *primary key* in a relation B.
- each foreign key in a relation must:
  1. appear as a **primary key** in the referenced relation, OR:
  2. be a `null` value

01-3. SUMMARY



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	...
...	...	...	...	...

02. RELATIONAL ALGEBRA

- **algebra** → mathematical system of operands and operators
  - **operands**: variables or values from which new values can be constructed
  - **operators**: symbols denoting procedures that construct new values from given values
- **relation algebra** → procedural query language
  - **operands**: relations or variables representing relations
  - **operators**: transform one or more input relations into one output relation

Closure Property

- **closure** → relations are *closed* under relational algebra
  - all input operands and outputs of all operators are *relations*
  - the output of one operator can serve as input for subsequent operators
- allows for nesting of relational operators ⇒ **relational algebra expressions**

02-1. BASIC OPERATORS

UNARY OPERATORS

Selection, σ_c

- $\sigma_c(R)$ → selects all tuples from a relation R (i.e. rows from a table) that satisfy condition c.
  - for each tuple $t \in R, t \in \sigma_c(R) \iff c$ evaluates to true on t
  - input and output relation have the same schema
- **selection condition** →
  - a *boolean expression* of one of the following forms:
    - constant selection - attribute **op** constant
    - attribute selection - attribute$_1$ **op** attribute$_2$
    - expr$_1 \wedge$ expr$_2$; expr$_1 \vee$ expr$_2$; item $\neg$ expr ; (expr)
  - with **op** $\in \{=, <>, <, \leq, \geq, >\}$
    - **operator precedence**: (), **op**, ¬, ∧, ∨
  - handling `null` values
    - comparison operation with `null` ⇒ **unknown**
    - arithmetic operation with `null` ⇒ `null`

Projection, π_ℓ

- $\pi_\ell(R)$ → projects all attributes of a given **relation** specified in list ℓ
  - *relation* = set of tuples ⇒ duplicates removed from output relation!
  - **order** of attributes matters!
  - i.e. projects all columns of a table specified in list ℓ

Renaming, ρ_ℓ

- $\rho_\ell(R)$ → renames the attributes of a relation R
  R is a relation with schema $R(A_1, A_2, \ldots, A_n)$
- 2 possible formats for ℓ
  - ℓ is the new *schema* in terms of the new attribute names
    - $\ell = (B_1, B_2, \ldots, B_n); B_i = A_i$ if attribute A_i does not get renamed
  - ℓ is a list of attribute renamings of the form: $B_i \leftarrow A_i, \ldots, B_k \leftarrow A_k$
    - each renaming $B_j \leftarrow A_j$ renames attribute A_j to attribute B_j
    - order of renaming doesn't matter

SET OPERATORS

- **union** → $R \cup S$ returns a relation with all tuples that are in both R **or** S
- **intersection** → $R \cap S$ ... all tuples that are in both R **and** S
- **set difference** → $R - S$ ... all the tuples that are in R **but not in** S

! requirement for all set operators: R and S must be **union-compatible**

Union Compatibility

- two relations R and S are **union-compatible** → if
  - R and S have the same number of attributes and
  - the corresponding attributes have the *same or compatible domains*
  - BUT R and S do not have to use the same attribute names

CROSS PRODUCT

- **cross product** → combines two relations R and S by forming all pairs of tuples from the two relations
  - given two relations $R(A, B, C)$ and $S(X, Y)$, $R \times S$ returns a relation with schema (A, B, C, X, Y) defined as
    $R \times S = \{(a, b, c, x, y) \mid (a, b, c) \in R, (x, y) \in S\}$
- **size** of cross product $= |R| * |S|$

02-2. JOIN OPERATORS

Inner Joins θ-join

- eliminate all tuples that do not satisfy a matching criteria (i.e. **attribute selection**)

θ-join

- the θ-join $R \bowtie_\theta S$ of two relations R and S is defined as

$$R \bowtie_\theta S = \sigma_\theta(R \times S)$$

Equi Join $\bowtie$

- special case of θ-join defined over the **equality** operator ($=$) only

Natural Join $\bowtie$

- the **natural join** $\rightarrow$ (of two relations R and S) is defined as
  $R \bowtie S = \pi_\ell(R \bowtie_c \rho_{b_i \leftarrow a_i, \dots, b_k \leftarrow a_k}(S))$
  - $A = \{a_i, \dots, a_k\}$ is the set of attributes that R and S have in common
  - $c = ((a_i = b_i) \wedge \cdots \wedge (a_k = b_k))$
  - $\ell =$ list of all attributes of R + list of all attributes in S that are **not in** A
- performed over all attributes that R and S have in common
  - no explicit matching criteria has to be specified
- output relation contains the common attributes of R and S only *once*

Outer Joins

- **dangling tuples** $\rightarrow$ tuples in R or S that do not match with tuples in the other relation
  - ***dangle***$(R \bowtie_\theta S) \rightarrow$ set of dangling tuples in R wrt to $R \bowtie_\theta S$
    - *dangle*$(R \bowtie_\theta S) \subseteq R$
  - always removed by inner joins, kept by outer joins
  - missing attribute values are padded with `null`
- $null(R) \rightarrow n$-component **tuple** of `null` values where n is the number of attributes of R

Definitions

- **left outer join** $\rightarrow R ⟕_\theta S = R \bowtie_\theta S \cup (\textit{dangle}(R \bowtie_\theta S) \times \{\textit{null}(S)\})$
- **right outer join** $\rightarrow R ⟖_\theta S = R \bowtie_\theta S \cup (\{\textit{null}(R)\} \times \textit{dangle}(S \bowtie_\theta R))$
- **full outer join** $\rightarrow R ⟗_\theta S$
  $= R \bowtie_\theta S \cup (\textit{dangle}(R \bowtie_\theta S) \times \{\textit{null}(S)\}) \cup (\{\textit{null}(R)\} \times \textit{dangle}(S \bowtie_\theta R))$

Natural Outer Joins

- only equality operator is used for the join condition
- join is performed over all attributes that R and S have in common
- output relation contains the common attributes of R and S only once

SUMMARY: RELATIONAL MODEL

Term	Description
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
tuple	roles of a table
database schema	set of relation schemas
database	set of relations / tables
key	minimal set of attributes uniquely identifying 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