IN3020&4020 – Database Systems (2025)
Lecture 28: Summary

12 May

Egor V. Kostylev
IFI, University of Oslo
egork@ifi.uio.no



Databases and Database Management Systems

Database:

- collection of related data—that is, known facts with implicit meaning—organised in some way (data model)
- *In this course*: relational databases (mostly), RDF knowledge graphs, etc.

Database management system (DBMS):

- a software that facilitates defining, constructing, manipulating, and sharing databases by various users and applications
- *In this course*: SQL engines (mostly), SPARQL engines, etc.

We often use 'database' and 'DBMS' interchangeably, as well as omit 'relational' (when this does not cause confusion, which is usually the case)

Inside the database ‘black box’

- What is apparent to us is that we write an SQL query, and the database does something and comes up with an answer
- But there is a lot that happens behind the scenes, inside the ‘black box’
- This course shows what happens behind the scenes in the ‘black box’, so that we understand the common challenges and what to do about them

Course syllabus

Three main parts:

- Intro and SQL recap ✓

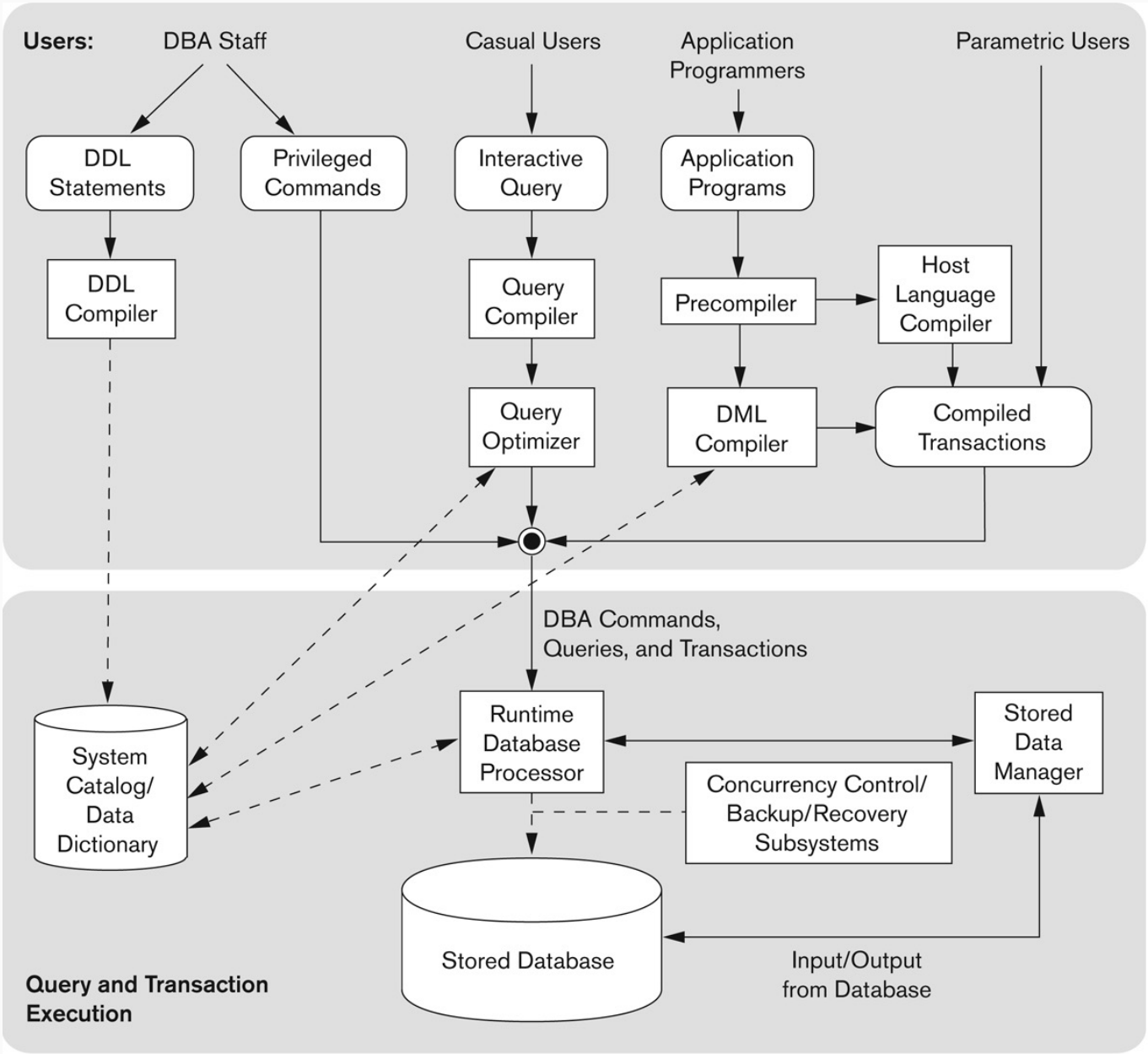
1. Query processing in relational databases ✓
   - relational algebra, indexes, query optimisation & evaluation, etc.
2. Transaction management in relational databases ✓
   - ACID, serialisation, concurrency control, recovery, etc.

- Database security ✓

3. NoSQL DBMS ✓
   - knowledge graphs, column-based, etc.

- Database research trends (for IN4020) ✓

Example DBMS architecture



$\frac{1}{2}$. Intro and Recap

Classic (set) relational data model

A relation is a table of values

Tutorials:

ID	*site*	*tutorial*	*topic*
1	w3schools	SQL_2003STD	Databases
2	w3schools	HTML_5	WebDev
3	w3schools	CSS_3	WebDev
4	w3resource	SQL_2003STD	Databases
5	w3resource	MySQL	Databases

- Terminology: relation name, attribute names (or just attributes), signature, relation schema, tuples (records), etc.
- Each attribute has a domain (integers, strings, etc.)
- Tuples are unordered and unique (i.e., not identical)
- Integrity constraints (primary keys, foreign keys, etc.)
  (basically for modelling, but can be used in optimisation)

SQL data model

SQL is based on the relational model, but there are some differences between its data model and the relational model:

1. NULLs
2. repeated and ordered tuples in output relations

Bag relational model:

same as the set model,

tuples are (still) unordered but may have repetitions (not unique)

SQL

SQL is a query language for relational databases

- (Arguably) a major reason for the success of relational databases
- Is a practical rendering of relational algebra (with differences)
- Usually reads as 'sequel' for historical reasons
- Now popularly known as 'Structured Query Language'
- Regularly updated ANSI&ISO standards since 1986 (newest is SQL:2023)
- Many SQL-based DBMS systems (Oracle, MS SQL Server, MySQL, PostgreSQL)

SQL components

SQL has several sub-languages:

- Data Definition Language (DDL): language for relation schemas and constraints (`CREATE`, `ALTER`, `DROP`, etc.)
- Data Manipulation Language (DML): language for populating relational databases (`INSERT`, `DELETE`, etc.)
- **Data Query Language (DQL)**: language for querying relational databases (`SELECT`)

  `SELECT` <attribute list>
  `FROM` <table>
  [`WHERE` <condition>]

- Data Control Language (DCL): language for access and user management (`GRANT`, `REVOKE`)
- ...

Part 1. Query processing in relational databases

SQL query processor in action

```
SELECT DISTINCT title FROM StarsIn WHERE starName IN
  (SELECT name FROM MovieStar WHERE birthDate LIKE '%1960');
```

$\downarrow$ parsing

(parse tree)

$\downarrow$ preprocessing

$$\pi_{title}(\sigma_{starName=name}(\mathbf{StarIn} \times \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar}))))$$

$\downarrow$ optimisation

$$\pi_{title}(\mathbf{StarIn} \bowtie_{starName=name} \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar})))$$

$\downarrow$ evaluation over tables (a lot of things)

(Result)

Our set (Codd-like) Relational Algebra

- Domain: Relations
- Core operators (independent):
  - (set) union $\cup$
  - (set) difference $\setminus$
  - selection σ_C
  - projection π_{Atts}
  - Cartesian product $\times$ (a.k.a. cross product, cross join)
  - renaming ϱ_N (technical, often not mentioned)
- Some unary, some binary, usually not always applicable
- Some parametrised: families of operators rather than single
- Expressible operators:
  - (set) intersection $\cap$
  - other joins: natural $\bowtie$, theta- $\bowtie_\theta$, semi- $\ltimes$ & $\rtimes$, etc.
  - division $\div$
  - ...

RA Algebraic Laws

The power of RA is largely in its rich set of algebraic laws

- allows to convert complex queries to equivalent but simpler ones (optimisation before evaluating them)
- 'simple' here means not 'short' or 'neat-looking', but 'expected to be evaluated quicker'

- there are (too) many laws, and we considered only most prominent examples, such as

  union is commutative: $R \cup S = S \cup R$

  selection can be often pushed through natural join:

  $\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$ if C mentions only attributes of R

Algebra for Full SQL

What is missing in our core RA to cover all SQL functionality?

1. Value invention (`SELECT R.A+5 FROM R`)
   - new (non-expressible) operation generalised projection $\pi_N(R)$
2. NULLs (in particular, for `OUTER JOIN`s)
   - extended conditions and various types of join operations
3. Recursion (`WITH RECURSIVE`) not covered
4. Bag relations and aggregates (`SELECT SUM(R.A) FROM R`)
   - bag algebra with bag relations as the domain
   - extending (σ, $\bowtie$ etc.) and re-defining ($\cup^b$, π^b, etc.) set RA operations
   - adding new operations (dup. elimination δ, grouping+aggr. γ)
   - algebraic lows (sometimes different from the set case)
5. Ordered relations (`ORDER BY`, `LIMIT`) not covered

... ...

Indexes Intro

An index on an attribute A (in a relation R):
a data structure that facilitates (i.e., speeds up) finding the tuples in R with a certain value of A

Attribute A is called the search key

It is possible to have an index on every attribute,
but there is a trade-off:

Indexes Intro

An index on an attribute A (in a relation R):
a data structure that facilitates (i.e., speeds up) finding the tuples in R with a certain value of A

Attribute A is called the search key

It is possible to have an index on every attribute,
but there is a trade-off:

More indexes for the same relation means

- faster search
- slower updates & more storage

Usually, DBMSs automatically create an index on every primary key

Indexes on other attributes can be created manually

Types of Indexes

One-level indexes:

- Primary Index (dense or sparse)
  - the tuples are sorted (physically) on the search key
  - at most one tuple for a search-key value (e.g., primary key)
- Cluster Index (dense or sparse)
  - the tuples are sorted (physically) on the search key
  - any number of tuples for a search-key value
- Secondary Index (dense only)
  - the tuples are not sorted on the search key
  - may be used in addition to other indexes on the same relation

Multi-level indexes:

- Static
- Dynamic (based on B(+)-trees)
- Hash-based

Multi-dimensional indexes (one index on several attributes)

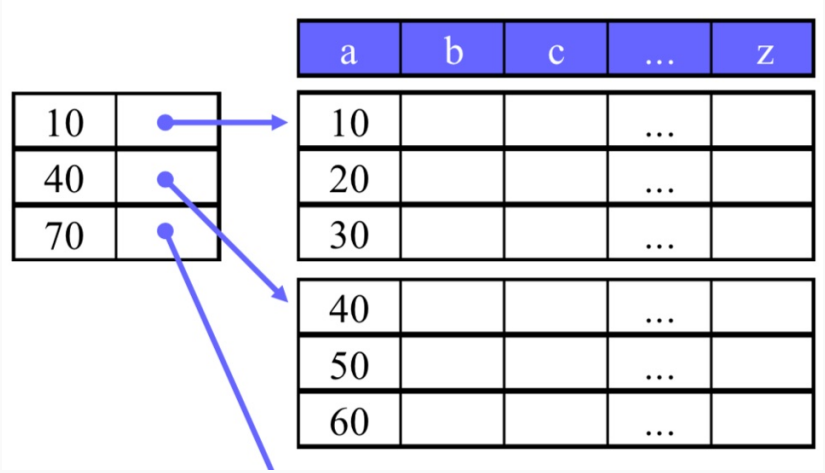
Primary Sparse Index

Primary Sparse Index on the primary key attribute A in a relation R stored as sorted by A:

- sorted table with two "attributes":

  primary key value, pointer to the tuple with this value
- only the first tuple in each block of R has a pair in the index

Example:



Benefit: index itself is also stored in secondary memory,

but with much fewer blocks

SQL (DQL) query processor in action

```
SELECT DISTINCT title FROM StarsIn WHERE starName IN
  (SELECT name FROM MovieStar WHERE birthDate LIKE '%1960');
```

$\wr$ parsing (standard parsing tools)

(parse tree)

$\wr$ compiliation (direct, expt. subqueries)

$$\pi_{title}(\sigma_{starName=name}(\mathbf{StarIn} \times \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar}))))$$

$\wr$ optimisation (the art)

$$\pi_{title}(\mathbf{StarIn} \bowtie_{starName=name} \pi_{name}(\sigma_{birthDate\ LIKE\ '\%1960'}(\mathbf{MovieStar})))$$

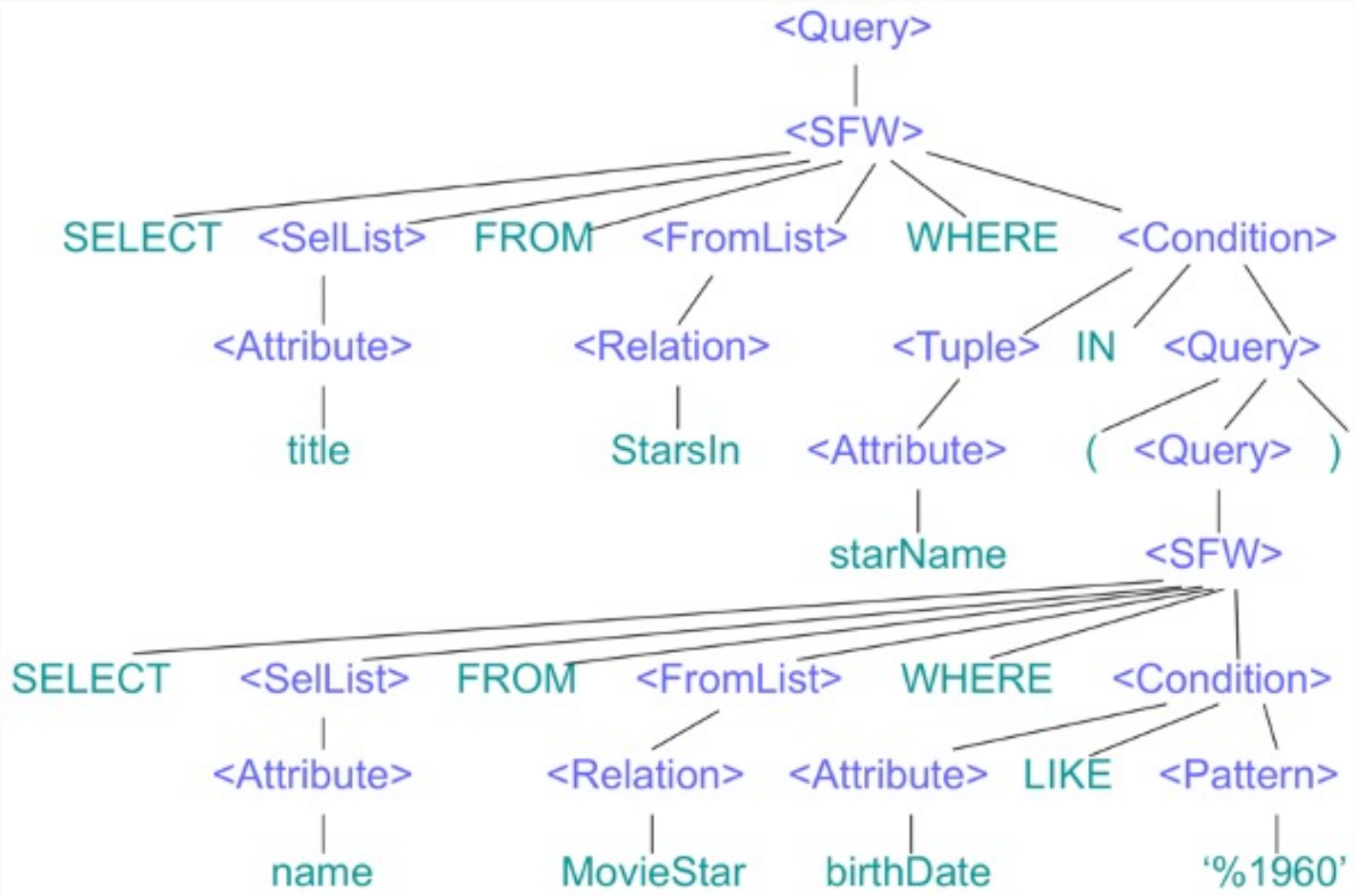
$\wr$ evaluation over tables (algorithms)

(Result)

Parsing: Example

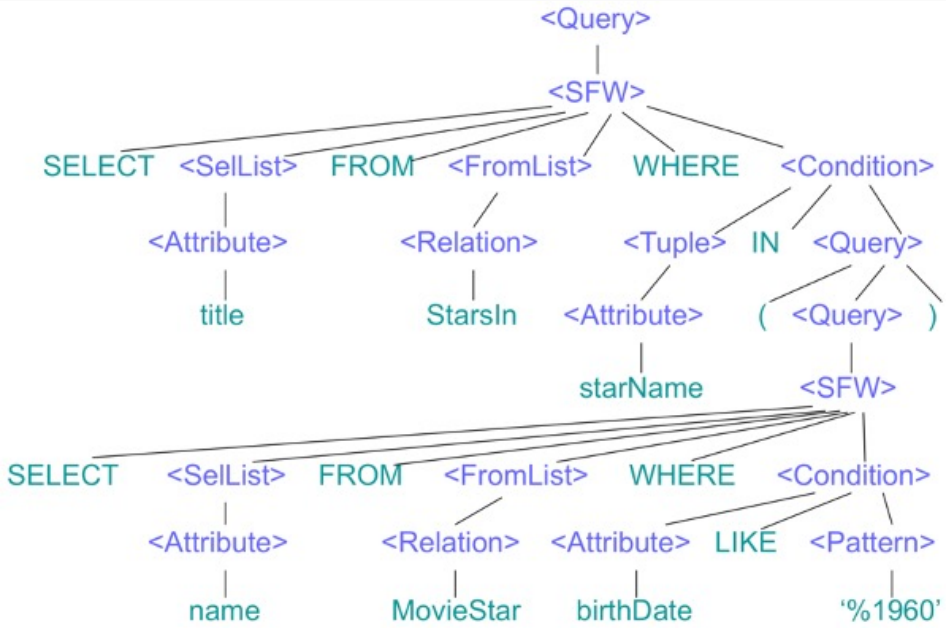
```
SELECT title FROM StarsIn WHERE starName IN
  ( SELECT name FROM MovieStar WHERE birthDate LIKE '%1960' );
```

parses to

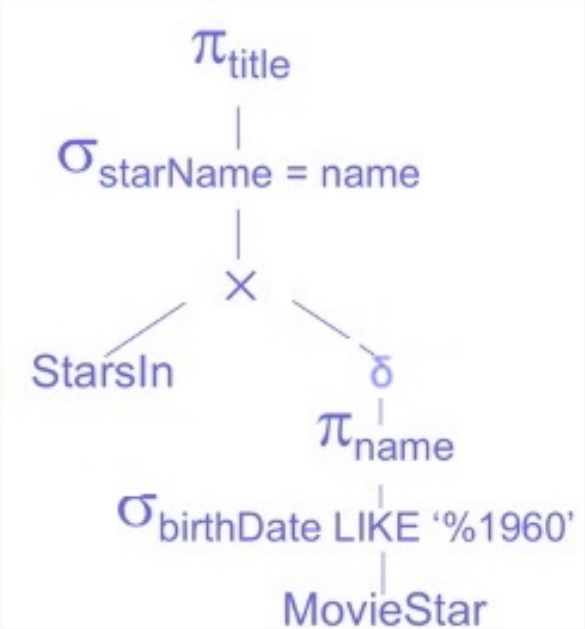


Query compilation: Example

Converts (i.e., 'compiles') the parse tree to an RA expression, called initial (logical) query plan:







Optimisation: Pragmatic approach overview

Given a (logical) query plan (RA expression) Expr, DBMSs pragmatically find another query plan Expr' such that

- Expr and Expr' give the same answer
- Expr' is estimated to be quicker than other candidates
- finding candidate Expr' and estimation is quick

This approach is based on two components:

1. good cost function f for query plans that is quick to compute and reflects evaluation time (i.e., $f(Expr') \leq f(Expr)$ in most cases means that Expr' is quicker to evaluate than Expr)
2. good heuristic (usually based on RA algebraic laws) that allows to quickly construct (one or several) candidate Expr' with (much) better cost

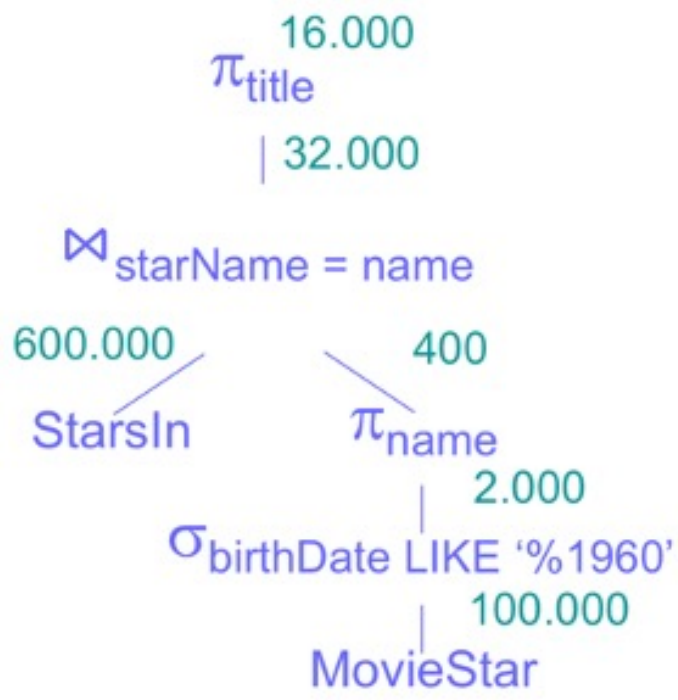
Our cost function

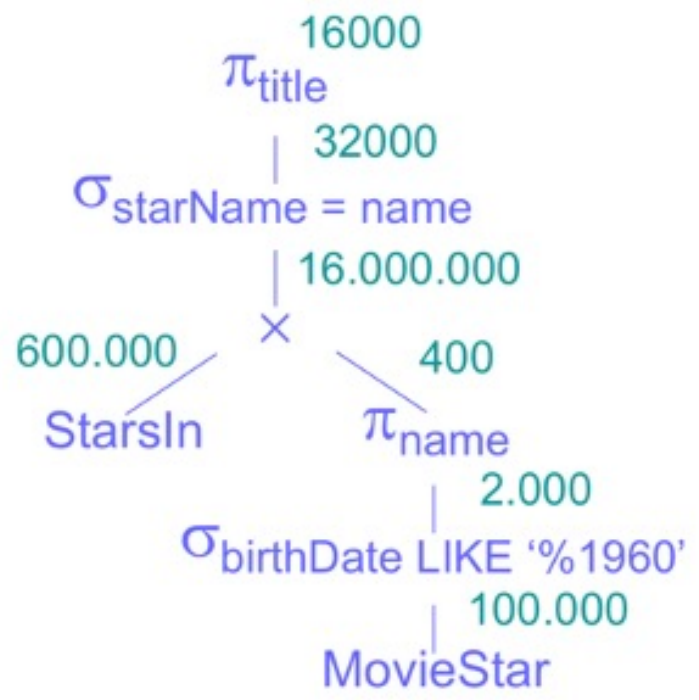
We take as our cost function:
an estimate of the sum of sizes (in bytes) of the intermediate results of the plan (i.e., the RA expression)

- estimate, because exact may be difficult to compute (our estimate is computed as described below)
- intermediate, because the inputs and the outputs are the same
- reflects the most important thing above: the number of disk I/O (number of used blocks is roughly proportional to the size of a relation)
- compositional: knowing estimates for immediate sub-plans, we can quickly compute the estimate for the plan (e.g., if we know the size estimates of Expr and Expr', we can estimate the size of Expr $\bowtie$ Expr')
- quick to compute (requires to maintain statistics)

Essentially, we assume that each operation is executed in negligible time and in isolation, and all the time is spent in reading all arguments (in whole) of each operation from the disk and writing the result back

Example





Our cost of the left plan: 2,000+400+32,000 = 34,400

Our cost of the right plan: 2,000+400+16,000,000+32,000 = 16,034,400

Reflects our intuition: good cost function (at least for this case)

Common heuristic rules

The algebraic laws that (almost always) decrease our cost:

- Push selections as far down as possible
- If the selection condition consists of several parts (AND or OR), split into multiple selections and push each one as far down the tree as possible
- Push projections as far down as possible
- Combine selections and joins (especially, Cartesian products) to appropriate theta-joins
- Apply idempotence as much as possible

- Choose as good join order as possible
  (most important and most difficult)

Algorithms for Executing RA Operations

Classes based on approach:

- Sort-based: always work,
  usually use N-Phase Multiway Merge Sort
- Hash-based: relies on a good hash function,
  usually use Hash Partitioning
- Index-based: relies on indexes

Classes based on memory requirements:

- One-pass: data fits into the main memory,
  read once
- Two-pass: data does not fit into the memory,
  but two passes are enough
- N-pass, $N \geq 3$: recursive generalisation to more memory,
  several passes (mostly omitted below)

Part 2. Transaction management in relational databases

Transaction idea

Transaction (informally):

- program that forms a logical unit of database processing
- includes one or more operations

  (e.g., `begin`, `read`(X), `write`(X), variable updates, `commit`, `abort`)
- boundaries can be specified by begin and end statements
- an application program may have several transactions

  (even parallel)

Different transactions access (reads, writes) the same database,

but have their own processors (main memory)

Access operations (`read`(X), `write`(X)) are the most relevant

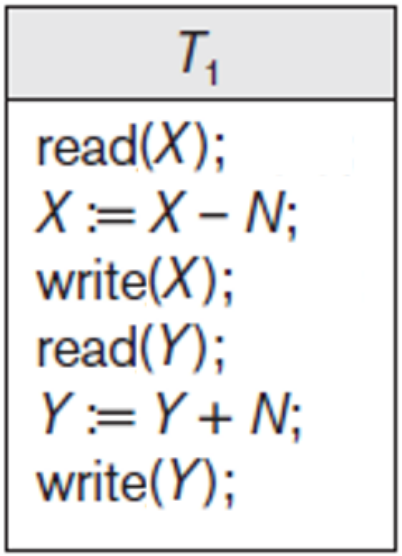
Main challenges: concurrency and recovery

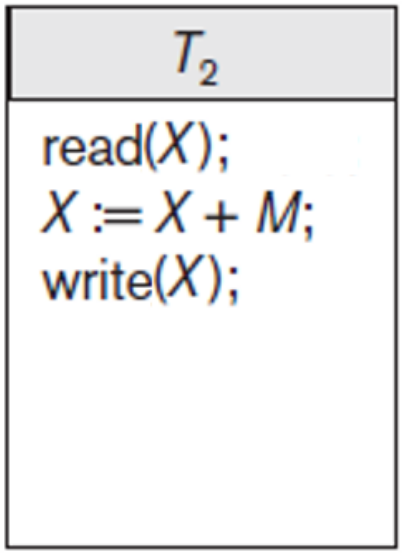
Example transactions

Let X and Y be the numbers of reserved seats in two flights

Transaction T_1 transfers N reservations from X to Y

Transaction T_2 reserves M seats in X





ACID properties

ACID:

- Atomicity:
  Transaction performed in its entirety or not at all
  (ensured by transaction recovery subsystem of a DBMS)
- Consistency:
  The database should always remain consistent
  (the responsibility of the transaction program)
- Isolation:
  Transaction should not interfere with other transactions
  (ensured by the concurrency control subsystem)
- Durability:
  Changes of committed transactions must persist
  (ensured by the recovery subsystem)

(some are often dropped in practice, formalised as isolation levels)

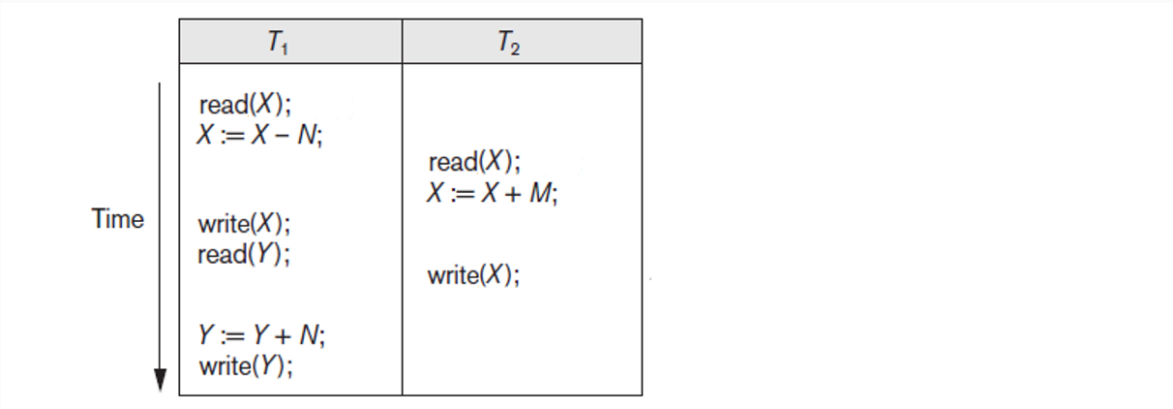
Schedules

Schedule (or history, execution plan) S of transactions $T_1, \ldots, T_n$: a (total) ordering of the operations of $T_1, \ldots, T_n$

- operations of different transitions can interleave
- operations of each T_i are in the same order as in T_i

Complete schedule: all $T_1, \ldots, T_n$ are committed or aborted

Example: S_a: $\mathtt{r}_1(X)$, $\mathtt{r}_2(X)$, $\mathtt{w}_1(X)$, $\mathtt{r}_1(Y)$, $\mathtt{w}_2(X)$, $\mathtt{w}_1(Y)$



Example (complete): S_b: $\mathtt{r}_1(X)$, $\mathtt{w}_1(X)$, $\mathtt{r}_2(X)$, $\mathtt{w}_2(X)$, $\mathtt{r}_1(Y)$, $\mathtt{a}_1$, $\mathtt{c}_2$

Classification schedules based on serialisability

For isolation (transaction should not interfere with other transactions), we need to be able to ensure that all transactions of a schedule are executed correctly

- Serial schedule: no transaction begins when there is an active (non-committed and non-aborted) transaction

- Serialisable schedule: equivalent (in some way) to a serial one
  - semantic equivalence (best, difficult to check)
  - conflict equivalence (based on a notion of a conflict)
  - ...

Classification schedules based on recoverability

For durability (changes of committed transactions must persist), we need to be able to recover schedules from transaction and system failures

We classified transactions into a hierarchy with respect to recoverability:

- impossible to recover (we saw examples)
- possible to recover
  recoverable (defined formally, saw examples)
- easy to recover
  cascadeless, strict (defined formally, saw examples)

Concurrency Control: Protocols to ensure serialisability

1. Serialisability via two-phase locking
   - based on several types of locks (binary, shared/exclusive, with/without conversion)
   - each schedule has first growing then shrinking phase
   - may end up in a deadlock, several ways to avoid them

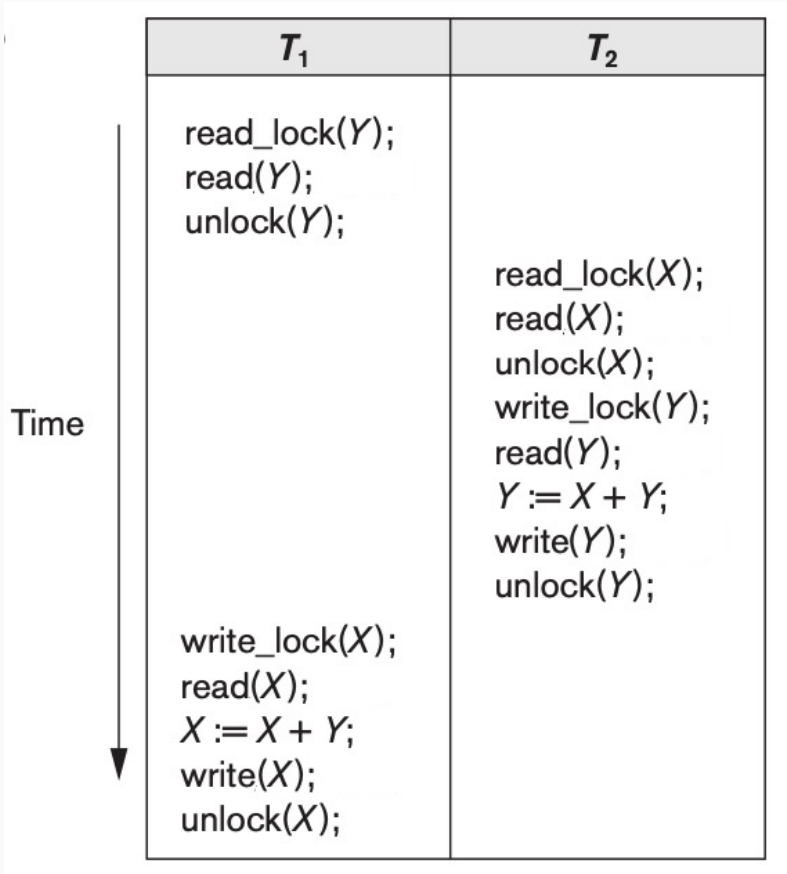
2. Serialisability via timestamps
   - use a "backbone" serial schedule, based on the timestamp of the transaction
   - a number of approaches (basic, strict, Thomas rule)

3. Multiversioning for concurrency control
   - several versions of the same item are kept
   - more disk memory, more concurrency
   - a number of approaches (timestamps, validation, etc.)

Two-phase locking: Negative example

T_1	T_2

read_lock(Y);
read(Y);
unlock(Y);
write_lock(X);
read(X);
$X := X + Y$;
write(X);
unlock(X);
	read_lock(X);
	read(X);
	unlock(X);
	write_lock(Y);
	read(Y);
	$Y := X + Y$;
	write(Y);
	unlock(Y);



T_1	T_2

read_lock(Y);
read(Y);
unlock(Y);
 	
write_lock(X);
read(X);
$X := X + Y$;
write(X);
unlock(X);
	
	read_lock(X);
	read(X);
	unlock(X);
	write_lock(Y);
	read(Y);
	$Y := X + Y$;
	write(Y);
	unlock(Y);

Transactions T_1 and T_2 (left) do not follow the two-phase locking protocol, so they allow for a non-serialisable schedule (right)

Recovery Protocols: Cache, Log, Checkpoints

1. Shadowing recovery protocols (NO-UNDO/NO-REDO)
   - essentially, no cache and no log is used

2. Deferred-update recovery protocols (NO-UNDO/REDO)
   - Do not physically update the database (on the disk)
     until after transaction commits
   - Undo (changing a piece of data back) is not needed
   - Redo (changing a piece of data again) may be needed

3. Immediate-update recovery protocols (UNDO/REDO)
   - Database may be updated by some operations of a transaction before it reaches commit point
   - Both undo and redo may be needed

4. ARIES recovery protocol (advanced UNDO/REDO)
   - remembering dirty pages instead of flashing on a checkpoint

$2\frac{1}{2}$. Database security

Database multi-user security mechanisms

DBMS must provide techniques to enable certain
users or user groups to access selected portions of a database

There are several types of security mechanisms; most fundamental:

- Discretionary security mechanisms
  Used to grant privileges to users to access certain data
  Commonly used in practice
- Mandatory security mechanisms
  Classify data pieces and users into various security classes
  Give access according to a corresponding security policy
  Used (and extended) in professional DBMSs (e.g., Oracle)

We considered such mechanisms for access control

3. NoSQL databases

NoSQL: Intro

New main-line types of applications (data science, social media)
impose new requirements upon the underlying systems

NoSQL (Not only SQL) serves these needs

Most NoSQL systems are distributed over several nodes

Main challenges:

- semi-structured data (no or little schema)
- high performance
- data availability and replication (several unreliable nodes)
- scalability (as opposed to immediate consistency)
- sharding and partitioning

NoSQL: Categories

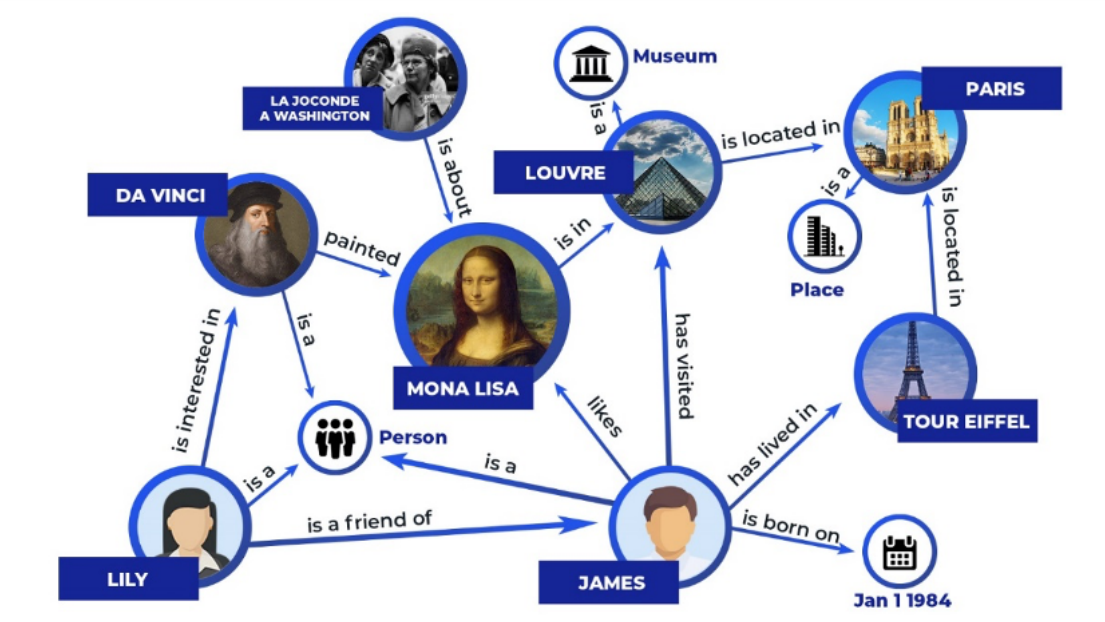
Categories of NoSQL systems

- Document-based systems: MongoDB, CouchDB, etc.
- Key-value stores: DynamoDB, Voldemort, etc.
- Column-based (or wide-column) systems: HBase, Cassandra, etc.
- Graph-based systems: RDF, Neo4j, etc.
- Hybrid systems: HADOOP, etc.
- (Object databases, XML databases)

Knowledge graphs

Knowledge graphs (KGs):

- annotated, usually directed, often multi- graphs
- studied in Knowledge Representation and Reasoning AI branch
- capture data (facts) and semantics (metadata, schema, rules)
- nodes connected by relationships
- Example: Google Knowledge Graph



RDF graphs (Semantic KGs)

RDF: a language that enable to make statements about resources

Example statement: "*John is father of Ann*"

RDF graph is a set of triples (here in Turtle-like syntax):

$$subject_1\ predicate_1\ object_1\ .\ subject_2\ predicate_2\ object_2\ .\ \ldots$$

- subject: resource (IRI) or blank node
- predicate: resource
- object: resource (or collection of), literal or blank node

Implicit logical knowledge in RDFS/OWL

SQL-like query language SPARQL

W3C standards for Linked Data and the Semantic Web

Neo4j (property graph) data model

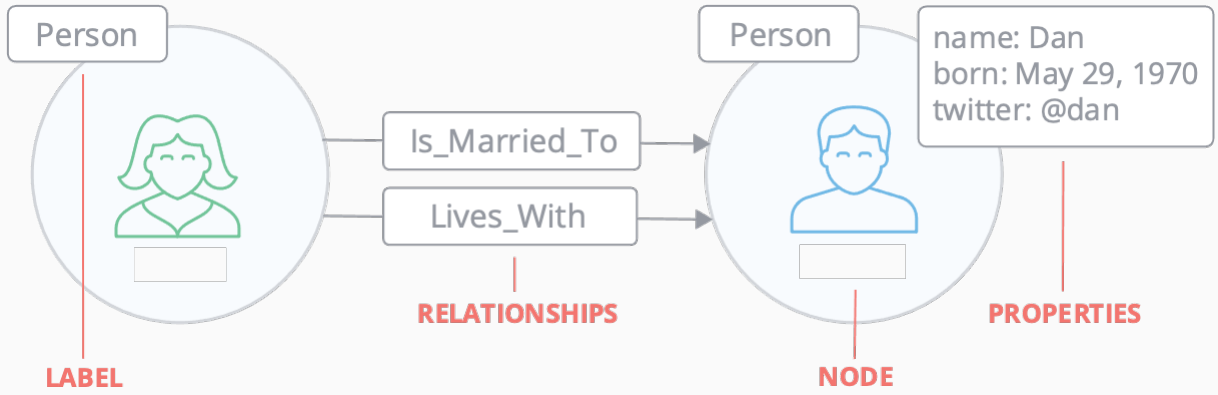
Node: can have any number (none, one or several) labels

Relationships:

- directed edges between two nodes
- two nodes can have several relationships between
- each relationship has exactly one type

Properties:

- key-value pairs
- can be attached to both nodes and relations (any number)



Cypher query language (pattern-based, not SQL-like)

Rich and efficient library for graph tasks (reachability, etc.)

The end

If anything is still unclear, get in touch

Good luck with the exam!

(We discuss 2024 exam tomorrow if you wish)