L13: Transactions and Data on the Enterprise Level

DSAN 6300/PPOL 6810: Relational and Semi-Structured Databases and SQL Programming

Irina Vayndiner

November 27 and 30, 2023

GEORGETOWN UNIVERSITY

Logistics

- Last Lecture today. Congrats!
- What is left?
  - Q4 (Database Optimization) is due on **Tue, 11/28**
  - Lab (Database Optimization) is due **Mon**, **11/27**
  - HW04 (mini-project) 130 points is due **Wed, 12/6**
    - If you submit by **12/4**, you will get 3 extra (bonus) points ☺
  - If need help: attend OHs
- Test: **Tue, 12/5** (All sections) on Zoom 10:30am-1pm
  - No early or late dates!
  - Covers full course materials, including lectures, labs, HWs and quizzes (not Mongo)
  - Reminder: Reload University database ahead of time
  - Open books, **required** to be on video, diff zoom link
  - Two parts:
    - Quiz (~100 points) (up to 45min)
    - SQL Programming test (~150 points)

Agenda for today

- Lecture
  - Transactions
  - Data at the Enterprise Level
  - SQL Injection
- “Virtual Lab”: Interview questions
- Fill in the evaluation form (15min)
  - Thu class only

TRANSACTIONS

Transaction Concept

- A **transaction** is a *unit* of program execution that accesses and possibly updates various data items.
- E.g., steps of transaction to transfer $50 from account A to account B
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Transaction ACID Properties

To preserve the integrity of data the database system must ensure:

- **Atomicity.** Either all operations of the transaction are properly reflected in the database or none are.
- **Consistency.** Execution of a transaction in isolation preserves the consistency of the database.
- **Isolation.** Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions.
- **Durability.** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Example of Fund Transfer

- Transaction to transfer $50 from account A to account B:
  1. **read**(*A*)
  2. *A* := *A* – 50
  3. **write**(*A*)
  4. **read**(*B*)
  5. *B* := *B* + 50
  6. **write**(*B*)
- **Atomicity requirement**
  - If the transaction fails after step 3 and before step 6, money will be "lost" leading to an inconsistent database state
    - Failure could be due to software or hardware
  - The system should ensure that updates of a partially executed transaction are not reflected in the database
- **Durability requirement**
  - Once the user has been notified that the transaction has completed (i.e., the transfer of the $50 has taken place), the updates to the database by the transaction must persist even if there are software or hardware failures.

Example of Fund Transfer (continued)

- **Consistency requirement** in above example:
  - The sum of A and B is unchanged by the execution of the transaction
- In general, consistency requirements include
  - Explicitly specified integrity constraints such as primary keys and foreign keys
  - Implicit integrity constraints
    - E.g., sum of balances of all accounts, minus sum of loan amounts must equal value of cash-in-hand
  - A transaction must leave a consistent database.
    - During transaction execution the database may be temporarily inconsistent **but**
    - When the transaction completes successfully the database must be consistent
  - Erroneous transaction logic can lead to inconsistency

Example of Fund Transfer (continued)

- **Isolation requirement** — if between the steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database (the sum $A + B$ will be less than it should be).

T1	T2
1. **read**(A)	
2. $A := A - 50$	
3. **write**(A)	
	read(A), read(B), print(A+B)
4. **read**(B)	
5. $B := B + 50$	
6. **write**(B)	

- Isolation can be ensured trivially by running transactions **serially**
  - That is, one after the other.
- However, executing multiple transactions concurrently has significant benefits, and is a requirement for production

Concurrent Executions

- Multiple transactions are allowed to run concurrently in the system. Advantages are:
  - **Increased processor and disk utilization**, leading to better transaction *throughput*
    - E.g., one transaction can be using the CPU while another is reading from or writing to the disk
  - **Reduced average response time** for transactions: short transactions need not wait behind long ones.
- **Concurrency control schemes** – mechanisms to achieve isolation
  - That is, to control the interaction among the concurrent transactions in order to prevent them from destroying the consistency of the database

Schedules

- **Schedule** – a sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions
  - Must preserve the order in which the instructions appear in each individual transaction.
- A transaction that successfully completes its execution will have a commit instructions as the last statement
  - By default transaction assumed to execute commit instruction as its last step
- A transaction that fails to successfully complete its execution will have an rollback instruction as the last statement

Schedule 1

- Let T_1 transfer \$50 from *A* to *B*, and T_2 transfer 10% of the A balance from *A* to *B.*
- A serial schedule 1 in which T_1 is followed by T_2 :

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Schedule 2

- A **serial** schedule 2 where T_2 is followed by T_1

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Schedule 3

- Let T_1 and T_2 be the transactions defined previously. The following schedule 3 is not a serial schedule, but it is *equivalent* to Schedule 1

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

- In Schedules 1, 2 and 3, the sum A + B is preserved.

Schedule 4

- The following concurrent schedule **does not** preserve the value of ($A + B$).

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Serializability

- Each transaction by itself preserves database consistency.
- Thus, serial execution of a set of transactions preserves database consistency.
- A concurrent schedule is **serializable** if it is equivalent to a serial schedule.

Locks

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
  1. **exclusive** *(X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  2. **shared** *(S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.
- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Locks Compatibility

- **Lock-compatibility matrix**

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item
- But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

Transaction with Locks

T_1: **lock-X**(*B*);
read (*B*);
B:=*B*-50;
write (*B*);
unlock(*B*);
lock-X(*A*);
read (*A*);
A:=*A*+50;
write (*A*);
unlock(*A*);

T_2: **lock-S**(*A*);
read (*A*);
unlock(*A*);
lock-S(*B*);
read (*B*);
unlock(*B*);
display(*A*+*B*)

- Examples of a transaction performing locking
  - T_1 – transfer funds; T_2 - display the total

- By itself locking is *not sufficient* to guarantee serializability

Schedule With Locks

- Still does not work right!

T_1	T_2	concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B)		
$B := B - 50$		
write(B)		
unlock(B)		
	lock-S(A)	
		grant-S(A, T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display($A + B$)	
lock-X(A)		
		grant-X(A, T_1)
read(A)		
$A := A + 50$		
write(A)		
unlock(A)		

Unlocking Delay

- We move unlock(s) to the very end of transactions, no inconsistency is possible, but delays can be introduced

T_3: **lock-X***(B)*;
read *(B)*;
B:=*B*-50;
write (*B)*;
lock-X*(A)*;
read *(A)*;
A:=*A*+50;
write (*A)*;
unlock*(B)*;
unlock*(A)*;

T_4: **lock-S***(A)*;
read *(A)*;
lock-S*(B)*;
read *(B)*;
display*(A+B)*
unlock*(B)*;
unlock*(A)*;

Locking Protocols

- In general, to enforce serialization of transactions we need a **locking protocol** - a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce serialization by restricting the set of possible schedules.

- But it has a serious issue – possible deadlock

Deadlock

- Look again to the unlocking delay example

T_3: **lock-X**(*B*);
read (*B*);
B:=*B*-50;
write (*B*);
lock-X(*A*);
read (*A*);
A:=*A*+50;
write (*A*);
unlock(*B*);
unlock(*A*);

T_4: **lock-S**(*A*);
read (*A*);
lock-S(*B*);
read (*B*);
display(*A*+*B*)
unlock(*B*);
unlock(*A*);

- Neither T_3 nor T_4 can make progress — executing **lock-S**(*B*) causes T_4 to wait for T_3 to release its lock on *B*, while executing **lock-X**(*A*) causes T_3 to wait for T_4 to release its lock on *A*.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

Deadlock (Cont.)

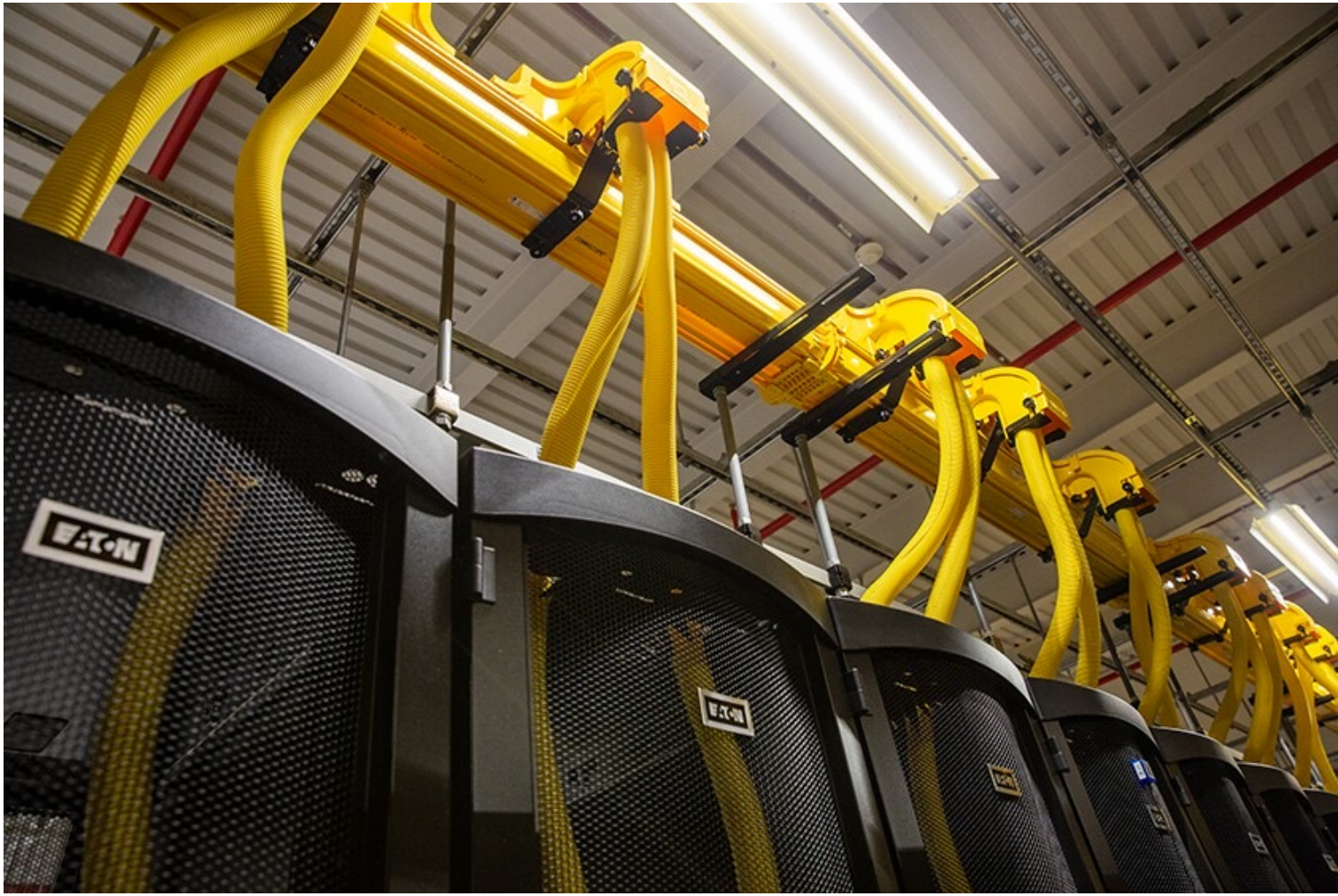
- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- **Starvation** is also possible if concurrency control manager is badly designed. For example:
  - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

Deadlock Handling

- **Deadlock prevention** protocols ensure that the system will *never* enter into a deadlock state. Some prevention strategies:
  - Require that each transaction locks all its data items before it begins execution (pre-declaration).
  - Impose partial ordering of all data items and require that a transaction can lock data items only in the order specified by the partial order (graph-based protocol).

Data Management for Enterprise

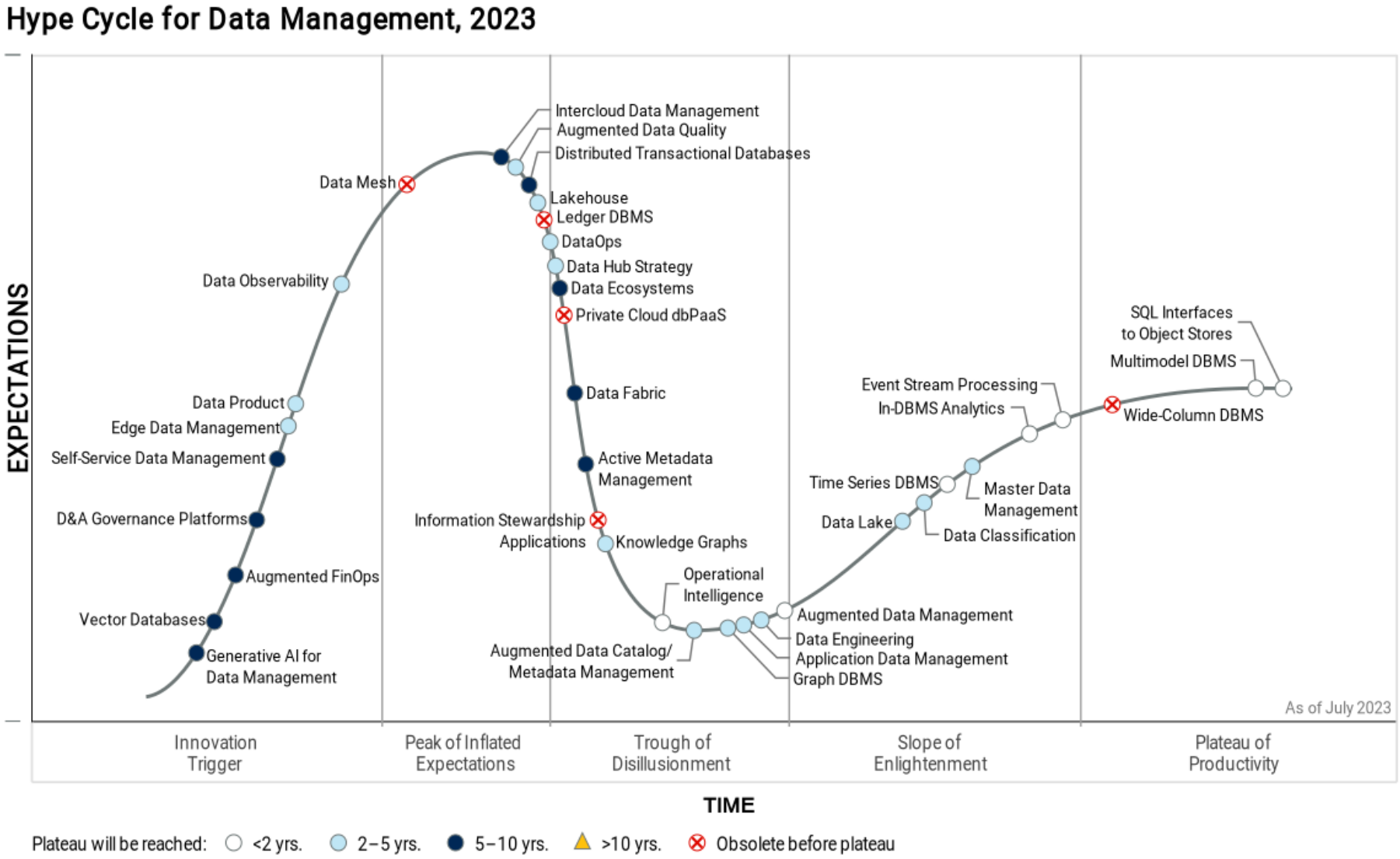
The Data Center Capital of the World Is in Virginia



There are nearly 300 data centers in Northern Virginia, scattered across Loudoun, Fairfax and Prince William counties. It's the largest concentration of data centers in the world.(Photographs by David Kidd)

https://www.governing.com/infrastructure/the-data-center-capital-of-the-world-is-in-virginia

Gartner Hype Cycle for Data Management



Source: Gartner
ID: 450207

Gartner

Hype Cycle Phases

Phase	Definition
Innovation Trigger	A breakthrough, public demonstration, product launch or other event generates significant press and industry interest.
Peak of Inflated Expectations	During this phase of overenthusiasm and unrealistic projections, a flurry of well-publicized activity by technology leaders results in some successes, but more failures, as the technology is pushed to its limits. The only enterprises making money are conference organizers and magazine publishers.
Trough of Disillusionment	Because the technology does not live up to its overinflated expectations, it rapidly becomes unfashionable. Media interest wanes, except for a few cautionary tales.
Slope of Enlightenment	Focused experimentation and solid hard work by an increasingly diverse range of organizations lead to a true understanding of the technology's applicability, risks and benefits. Commercial off-the-shelf methodologies and tools ease the development process.
Plateau of Productivity	The real-world benefits of the technology are demonstrated and accepted. Tools and methodologies are increasingly stable as they enter their second and third generations. Growing numbers of organizations feel comfortable with the reduced level of risk; the rapid growth phase of adoption begins. Approximately 20% of the technology's target audience has adopted or is adopting the technology as it enters this phase.
Years to Mainstream Adoption	The time required for the technology to reach the Plateau of Productivity.

Source: Gartner (July 2020)

Logical Data Warehouse

- The logical data warehouse (LDW) is a data management architecture that combines multiple physical analytics engines into a logically integrated whole.
- Data and analytics leaders can use the LDW to cover the full range of modern analytic requirements within a logically unified system, rather than a single physical server.
  - It is effective for on-premises as well as cloud deployments.
  - Use the LDW architecture as a template to ensure that all data requirements, processing and users can be satisfied within that architecture.
  - Use the minimum number of data servers to satisfy the maximum number of requirements while reducing redundancy.
  - Useful for faster exploration of new data assets.

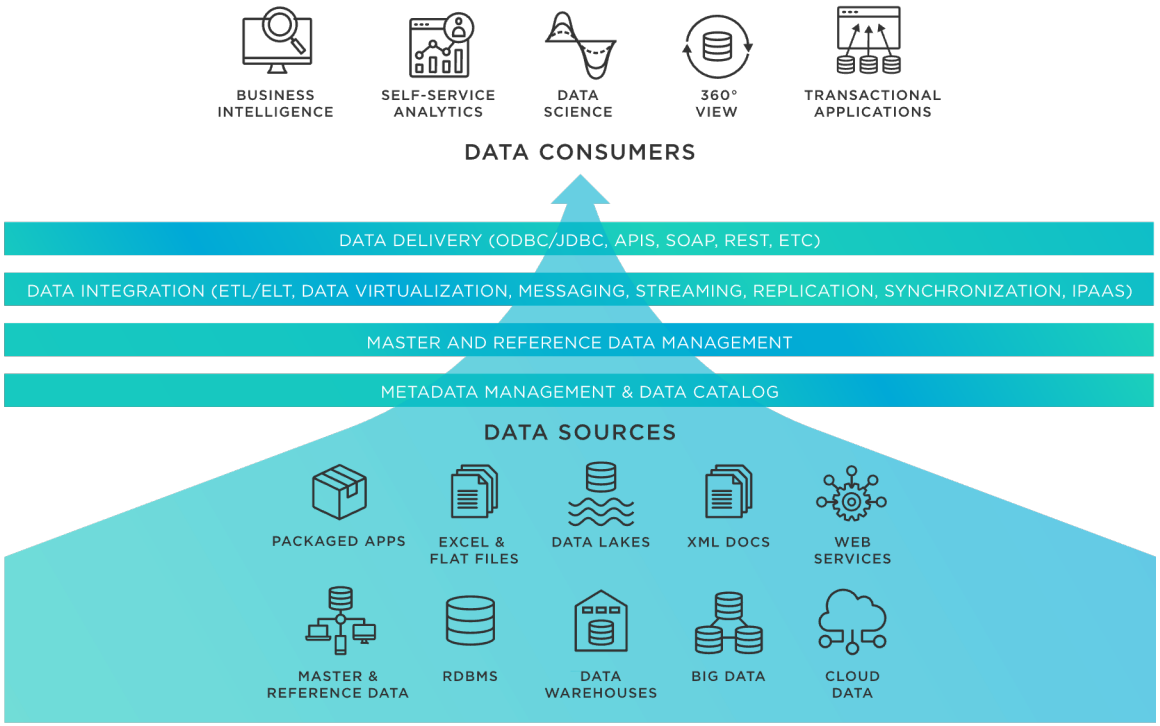
Logical Data Warehouse (continued)

- Requirements:
  - Support the LDW architecture: either as a controlling hub, or as a participating component.
  - Ability to use multiple processing engines and techniques, queries that can span multiple server types, integrated metadata and easy ways to transfer data within the architecture.
  - Evaluate the benefits of these in how they can support more requirements with faster and more agile development, thus maximizing benefits delivered.
- Measures:
  - Data lakes often optimizing cost per terabyte
  - Data warehouses and marts often for optimizing cost per query.
  - The blended costs and benefits will deliver maximum return on investment.

- Sample Vendors: Amazon Web Services; Cloudera; Databricks; Denodo; IBM; Microsoft; Oracle; Snowflake; Teradata; VMware (Pivotal)

Data Fabric

- Industry definition: A data fabric is an entire data ecosystem that connects disparate data sources and infrastructure types across a choice of endpoints (on premises, in the cloud, or hybrid environments) via standard interfaces (APIs), to accelerate digital transformation
- Part of moving to data-centric approach
- Consists of: Data Sources, Data Preparation, Data Management, Data Virtualization, and Data Consumption



Snowflake: Modern Cloud Data Warehouse

- "Accelerate your analytics with the data platform built to enable the modern cloud data warehouse"
- Founded 2012, IPO in 2020
- SW as a Service, aka "The *Data* Cloud"
- Cloud Storage
  - Automates Data Warehouse administration and maintenance
  - Runs on AWS S3, MS Azure, Google Cloud
- ANSI SQL compatible, with support for semi-structured data
  - Robust support for JSON-based functions
  - Supports UDFs
- Optimized connectors for BI and Analytics tools
  - Snowsight, built-in visualization UI for Snowflake
- More info:
  - https://www.snowflake.com

Snowflake: support for semi-structured data

- Provides native support for semi-structured data
  - Flexible schema data types for loading semi-structured data without transformation
  - Data Types
    - OBJECT: to represent collection of key-value pairs
    - ARRAY**:** to represent arrays of arbitrary size
      - E.g., multiple phone numbers
    - VARIANT: universal type that can store values of any other type, including object and array, up to max 16MB compressed
      - Stored as separate physical columns
      - Semi-structured data is often loaded into variant column
    - Alternatively, COPY INTO table statement with data transformation, one can extract selected columns from a staged data files into separate table columns
  - **SQL Statement**
    - **Create** or **replace** table *test_semi_structured* (var variant, arr array, obj object);
  - For more info: https://docs.snowflake.com/en/sql-reference/data-types-semistructured.html

Distributed Transactional Databases

- Problem statement: in the event of a network interruption, a database has to choose between availability and data integrity & consistency
- Definition: A distributed transactional database is a database that allows for write transactions to be performed on any of a distributed set of database instance nodes.
  - These databases have the ability to accept writes from a geographically distributed set of nodes while maintaining data integrity and consistency.
- How it is done: Recent combinations of hardware and software now allow these implementations
  - Some distributed transactional systems do not allow complete transparency for transactional activity, requiring some compromises in design and implementation.
  - Ensure developers understand the consequences of lost data integrity, both near- and long term effects.

Distributed Transactional Databases (continued)

- Business case: Geographically consistent distributed transactions
  - Systems such as global trading applications
  - Expanded use of transactional systems
- Combine some of the backend features of NoSQL databases with the front-end features of relational databases.
  - The result allows similar scalability to NoSQL but with some of the more robust features of relational databases.
- Sample Vendors: CockroachDB; FaunaDB; FoundationDB; Google (Cloud Spanner); NuoDB; RavenDB, YugabyteDB

Example: CockroachDB

- Database that describes itself as "almost impossible" to take down
  - Combining relational data with “limitless”, elastic cloud scale, “bulletproof resilience”
- Founded in 2015 by ex-Google employees
  - 2019: from open-source to proprietary, free for community use
  - Motto: “Scale Fast. Survive Anything. Thrive Everywhere”
- Main Features
  - Transactional, ACID-compliant
  - Single instance can scale from a single laptop to thousands of servers
  - Stores copies of data in multiple locations in order to deliver speedy access.
    - Geo-partitioning so you can tie data to a location and ensure peak performance for local and global deployments.
  - Designed to run in the cloud and be resilient to failures
    - “It automatically distributes data and workload demand. Break free from manual sharding and complex workarounds”

CockroachDB (continued)

- CockroachDB combines the benefits of relational SQL—consistency and reliability—with those of NoSQL—easy scale and global reach.
  - CockroachDB uses "consistency" in both the sense of ACID semantics and the CAP theorem, BUT less formally than either definition.
    - Built on a transactional and strongly-consistent key-value store
- When to use:
  - Distributed or replicated OLTP
  - Multi-datacenter deployments
  - Multi-region deployments
  - Cloud migrations
  - Infrastructure initiatives built for the cloud
- When not to use
  - CockroachDB is not suitable for heavy analytics / OLAP.
- For more info: https://www.cockroachlabs.com/

Example: YugabyteDB

- Distributed SQL Database
  - Apache
  - Can upgrade to Enterprise
- *Yuga* in Sanskrit represents about 4.32 million years
- "Open source, cloud native relational DB for powering global, internet-scale apps."
- Main Features
  - PostgreSQL-compatible SQL
  - Distributed ACID transactions
  - Velocity/Low latency: "millions of transactions per second"
  - Deploy across regions and clouds with synchronous or multi-master replication.

YugabyteDB (contunied)

- When is YugabyteDB a good fit?
  - Fast-growing, cloud native applications with high availability and low latency.
- Common use cases include:
  - Distributed OLTP applications needing multi-region scalability with strong consistency and low latency.
    - E.g., User identity, Retail product catalog, Financial data services
  - Hybrid Transactional/Analytical Processing (HTAP), (aka Translytical) applications needing real-time analytics on transactional data.
    - E.g., User personalization, fraud detection, machine learning.
  - Streaming applications needing to efficiently ingest, analyze and store ever-growing data.
    - E.g., IoT sensor analytics, time series metrics, real-time monitoring.
- When is YugabyteDB not a good fit?
  - Not a good fit for traditional OLAP use cases that need ad-hoc analytics.
    - Use instead an OLAP store, e.g. data warehouse such as Snowflake.
- More info: https://docs.yugabyte.com/

DataOps

- DataOps is a collaborative data management practice focused on improving the communication, integration and automation of data flows between **data managers** and **data consumers** across an organization.
- DataOps uses technology to *orchestrate* and *automate* data delivery
  - Data Orchestration is the automation of data-driven processes from end-to-end, including preparing data, making decisions based on that data, and taking actions based on those decisions.
  - It's a process that often spans across many different systems, departments, and types of data.
  - DataOps focuses on changing the organizational **speed** in delivering data management and integration solutions to the enterprise.
- As a new practice, DataOps will be most successful on projects targeting a small scope with some level of executive sponsorship, primarily from the CDO or other top data and analytics leaders.
- Sample Vendors: Composable Analytics; DataKitchen; Delphix; Hitachi Vantara; IBM; Informatica; Nexla; Saagie; Unravel

Blockchain

- A **blockchain**, originally block chain, is a growing list of records, called *blocks*, that are linked using cryptography.
  - Each block contains a hash of the previous block, a timestamp, and transaction data
  - Anyone with access rights can historically trace a state change in data or an event belonging to any participant
  - Blocks of records shared by all participants in a peer-to-peer (P2P) network
- Will evolve over the next 10 years in varying degrees
- Opportunities in capabilities like identity portability, trustless interactions, smart contracts and new forms of value exchange

Ledger DBMS

- A ledger DBMS is an append-only, immutable DBMS with an embedded cryptographically verifiable audit trail.
- A ledger DBMS is useful for private and permissioned “blockchainlike” applications
  - data tampering detection
  - auditing
  - more manageable and easier to implement
- Designates which parties, other than the owner, have access to and may add to it.
- Impact: Today, many blockchain projects are forced to use public blockchain technologies when a DBMS would suffice.
- Sample vendors: The Amazon Quantum Ledger Database (QLDB), Oracle announced in its 20c version (“blockchain tables”)

Metadata

- A set of data that describes and gives information about other data.
  - In other words, it is "data about data".
- Many distinct types of metadata exist.
- **Descriptive** metadata is descriptive information about a resource.
  - It is used for discovery and identification. It can include elements such as title, abstract, author, and keywords.
- **Structural** metadata is metadata about containers of data and indicates how compound objects are put together
  - For example, how pages are ordered to form chapters. It describes the types, versions, relationships and other characteristics of digital materials.
- **Administrative** metadata is information to help manage a resource, like resource type, permissions, and when and how it was created.
- **Reference** metadata is information about the contents and quality of statistical data.
- **Statistical** metadata, also called process data, may describe processes that collect, process, or produce statistical data.

Metadata Management Solutions (MMS)

- Includes one or more of the following:
  - Metadata repositories, a data glossary, data lineage, impact analysis, rule management, semantic frameworks, and metadata ingestion and translation from different data sources.
- In most applications of IT today, the limiting factor is is the cost of consistency and coordination
  - Hardware can be obtained at commodity prices, and software infrastructure can be downloaded free of charge.
- MMSs are accelerating due to innovation generated by active metadata that leverages machine learning
  - Metadata will become a critical input to machine learning approaches for dynamic data management solutions

- Sample Vendors: Alation; Alex Solutions; ASG; Collibra; erwin; IBM; Infogix; Informatica; Oracle; SAP

Data Catalog

- The **database catalog** of a database instance consists of metadata in which definitions of database objects such as base tables, views (virtual tables), synonyms, value ranges, indexes, users, and user groups are stored.
- It provides context to to locate a relevant dataset and understand what it means, in order to determine and extract value from it.
- The SQL standard specifies a uniform means to access the catalog, called the INFORMATION_SCHEMA, but not all databases follow this, even if they implement other aspects of the SQL standard.
  - For an example of database-specific metadata access methods, see Oracle metadata.
- Used to manage an inventory of heterogeneous and distributed data assets through the discovery, organization and description of the enterprise datasets.
- Sample Vendors: Alation; Collibra; IBM; Informatica

Future: Distributed Multi-Model Database

- Unified Multi-Model Database that can support variety of data
  - Can support document, graphs, key-value models, etc.
  - Users can plugin domain/application specific functions, types, and instances
  - The need stems from the fact that organizations do not have the resources to manage multiple databases.
  - Can store different data models to different users without additional complexity
  - Single backend that use data and query standards appropriate to each model
  - Seamless querying across all the supported data models
  - Indexing, parsing and processing standards appropriate to the data models

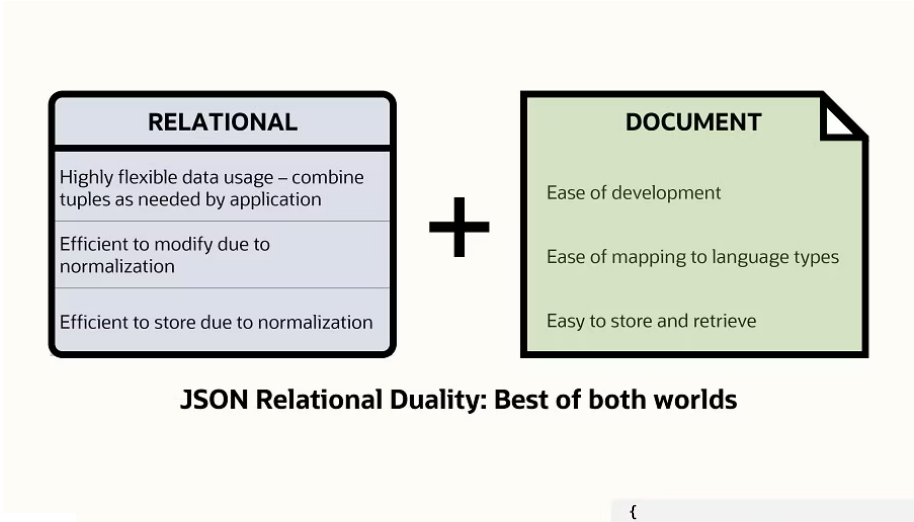
- Examples:
  - ArangoDB (open-source)
  - Amazon DynamoDB with Neptune graph capabilities
  - Azure Cosmos DB
  - SingleStore (former MemSql)

SingleStore Database

- Relational distributed database with support for JSON, graph, time-series
  - Both OLTP and OLAP
  - Public and private clouds
- History:
  - 2013 – MemSQL row-based, in-memory
  - Then added on-disk, columnar support
  - Oct 2020 – rebranded as SingleStore with the goal of achieving a universal storage format
- Database is distributed across many commodity machines
- Allows for real-time data ingest: from Kafka, Spark, Azure Blobs, AWS S3, etc.
- SingleStore can be downloaded for free
  - For Linux for systems up to 4 leaf nodes of 32 gigs RAM each

Oracle is catching up and fast!

- Oracle 23c: "**Unification of Document, Object, and Relational Models**"



Source: https://blogs.oracle.com/database/post/json-relational-duality-app-dev

Augmented Data Management

- Augmented data management refers to the application of AI and ML for optimization and improved operations for configuration, security and performance.
- They are also applied to create, manage and apply policy rules within the different products, such as metadata management, master data management, data integration, data quality and database management systems.
- Cost-based query optimization based on the collection of statistics across large numbers of deployed instances, especially in the cloud, has accelerated the pace of vendor delivery.
  - These solutions are being used not only to tune and optimize the use of the products themselves based on actual usage, including failures and poor performance, but also suggest and implement new designs, schemas and queries.
  - They can also infer the semantics and associations of the data in order to recommend structural improvements.

Augmented Data Management (continued)

- Potential Benefits in the following areas
  - Metadata management: Use AI/ML to evaluate metadata more rapidly, accurately and with reduced redundancy
  - Data integration: To automate the integration development process, by recommending or deploying repetitive integration flows.
  - MDM —AI/ML-driven configuration and optimization of record matching and merging algorithms as a part of their information quality and semantics capabilities.
  - Data quality — AI/ML used to extend data profiling, cleansing, linking, identifying and semantically reconciling master data in different data sources, to create and maintain “golden records.”
  - DBMS — In addition to enhancing cost-based query optimization, AI and ML are being used to automate many current manual management operations, including the management of configurations, elastic scaling, storage, indexes and partitions, and database tuning.

Augmented Data Management (continued)

- Oracle example:
  - Oracle Autonomous Database, can run on prem and in Oracle Cloud
  - "Machine learning–driven automated tuning, scaling, and patching"
- Needed:
  - Verify the validity of the automated functionality
  - Audit the results:
    - as with any new functionality, there is the risk of introducing errors and reduced performance.
  - Model and measure the benefits realized from the resources that will be released for other functions.
- Benefit Rating: High
- Sample Vendors: Amazon Web Services; Cinchy; CluedIn; IBM; Informatica; Microsoft; Oracle; SAP; SnapLogic; Teradata

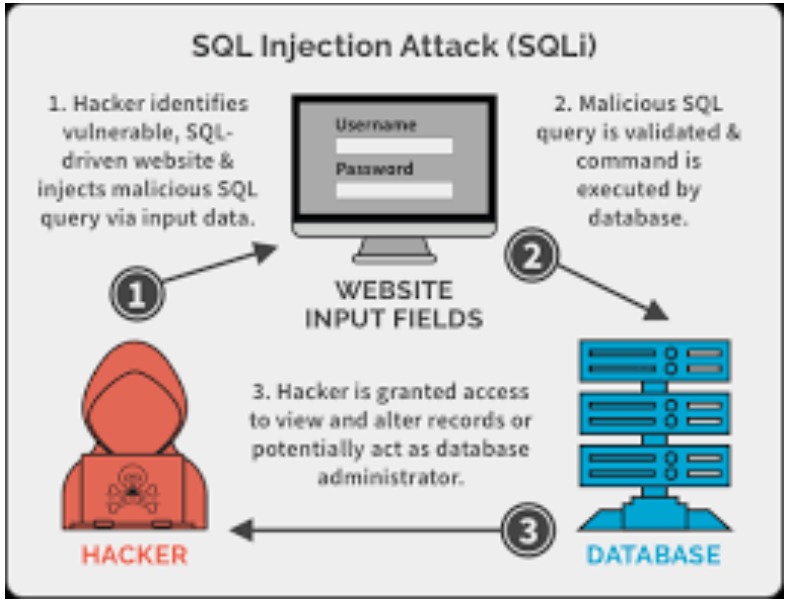
Generative AI for Data Management

- Definition
  - Generative AI technologies can generate new derived versions of content, strategies, designs and methods by learning from large repositories of original source content
- GenAI transforms data management activities through natural language interfaces, making data management activities more widely accessible.
  - Gen AI products, e.g. Azure OpenAI or Google Codey
  - Make self-service data management activities accessible to much larger audiences.
  - But: GenAI will be constrained by data and analytics governance, regulatory, and data security considerations, and will require human supervision.
- Vector Databases
  - When customers adopt generative AI models, vector databases store the embeddings that result from the model training.
    - The database can do a similarity search, which matches a prompt (the question) with specific or similar vector embedding.
    - More info: https://learn.microsoft.com/en-us/semantic-kernel/memories/vector-db

SQL Injection (SQLi)

- **Definition**

- SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database.
- It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.



SQL Injection: Some Examples

1. **/*! MYSQL Special SQL */**

/*! select count(*) from advisor */
looks like commented up but instead executes

2. SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'

This could log you as admin user, because rest of the SQL query (after comment ---) will be ignored.

3. select * FROM advisor -- where s_id =”12345”

you get all info from the table instead of just few rows (allows to explore the database)

SELECT * **FROM** users **WHERE** username=‘--' **OR true**

Looks like we looking for 1 student only but will get the full table

SQL Injection Links for additional reading

- https://www.imperva.com/learn/application-security/sql-injection-sqli/
- https://portswigger.net/web-security/sql-injection
- https://www.sqlshack.com/learn-sql-sql-injection/

“Virtual” Lab: Some sample interview questions

- I think you are ready now for the most challenging questions!

- Not a complete list!
  - https://www.codewars.com/post/top-sql-interview-questions-and-answers-in-2022
  - https://www.edureka.co/blog/interview-questions/sql-interview-questions#subsetsofsql
  - https://www.interviewbit.com/sql-interview-questions/
  - https://www.softwaretestinghelp.com/database-interview-questions/
  - https://www.toptal.com/sql/interview-questions
  - https://www.onlineinterviewquestions.com/amazon-rds-interview-questions/
  - https://www.techbeamers.com/sql-query-questions-answers-for-practice/
  - https://www.geeksforgeeks.org/sql-interview-questions/

MySQL for Developers

- https://dev.mysql.com
  - Create and Use MySQL Document Store
  - Import JSON documents into MySQL
  - Create a stored proc in MySQL
  - More on Date/Time Functions
  - MySQL Workbench Forum
  - EXPLAIN INTO and EXPLAIN FOR SCHEMA in MySQL
    - Allows to get back just the query cost, if needed
    - Explain for allows to run explain in another database
  - Etc.

Where do I find even more Challenging Queries to practice?

- Practicing to write challenging queries
  - Leetcode: https://leetcode.com/problemset/database/
  - HackerRank: https://www.hackerrank.com/domains/sql

- Book:
  - Joe Celko's "SQL for Smarties: Advanced SQL Programming"
    - The Morgan Kaufmann Series in Data Management Systems
    - 5th Edition

Please fill in Class feedback form

- 15min for Thu, 11/30, class