Database Performance Issues

- Load/Batch Job Delays
- Reporting Delays
- Decision Making Delays
- Business Analytics Delays
- Customer Intelligence Delays
- Planning and Forecasting Delays
- Key Performance Metrics Delays

- Advantages overweigh drawbacks
  - Need for better response from the db
  - Efficient use of resources
  - Experience and knowledge to do it correctly
- **DB performance**
  - Indexing
- **Tuning Oracle**
  - Installation
  - Memory
  - I/O
  - CPU

Indexing

- A means of increasing database performance of queries
  - Conceptually similar to book indexing
  - Speed of operations is increased
  - Data retrieval done better & faster
  - ***Shortcut*** for the DB to find the records that match some search criteria

Indexing

- Data is stored in blocks
  - Atomic disk operations to access them
- Search problem
  - Records can be sorted on just one field
    - If the records are not sorted, a linear search requires *N/2* block accesses
    - If field is a non-key (not unique), the entire table space is searched at *N* block accesses

Indexing: a way of sorting

- Solution
  - Indexing permits the sorting of data on multiple fields
    - *Log N* block accesses (Binary Search)
    - Once a higher value is found, the rest needs not be searched
  - Creating an index on a field in a table creates another data structure which contains:
    - Value of the field
    - Pointer to the corresponding record
    - This index structure is then sorted
      - Allows Binary Search to be performed on it

Indexing: Disadvantages (I)

- The index structures occupy disk space
  - Correct selection of fields to index
    - File systems size limits the number of indexes to be created

Example:

- 5M records, block size 1MB, record length of 54b
  - need of almost 300,000 blocks
  - ☺ approx 19 block accesses to find a record

Indexing: Disadvantages (II)

- Indexes have to be updated when the corresponding data is changed
  - For static databases where heavy **reporting** is needed, *more* indexes are required to support the **read-only** queries
  - For systems with numerous **transactions** that **modify** the data, *fewer* indexes provide higher data rates delivered

Indexing

- Heuristics
  - Short index (lower disk work intensity)
  - Columns targeted for indexing must have high selectivity (cardinality/#rows

    **Cardinality = uniqueness of data in a column*
  - Small percentage of duplicated values
  - Covering queries
    - (composite index - using more than one column)
- **Careful analysis, benchmarking, and testing**

Tuning Oracle

- DBA responsible for optimizing the performance of Oracle software
  - also application developers
  - hardware experts
- 30% of SDLC dedicated to performance issues
- Constant monitoring
  - Third-party monitoring product

Installation Rules (I)

- *Readme file* up-to-date info
- Enough disk space up front
- Allocate an extra of 20% for the installation process (requirements are lower than what is optimal)
- O/S Level Privileges
  - DBA overly rich in privileges
  - Oracle must own its directory structure

Installation Rules (II)

- SHARED_POOL_SIZE set to ***twice*** the suggested default
- File Structure
  - Recommended not to merge multiple physical drives into one large logical drive
  - Allocate an entire device to hold Oracle data files (no partitioning)
  - One directory point should point to one physical device
  - Lay out large tablespaces into small manageable sections (limits imposed by backup devices and O/S)

Installation Rules (III)

- DB creation issues
  - **Maxdatafiles** set as highly as the O/S permits
  - **Redo Logs** must be mirrored, as they are a single point of failure
  - Minimal tablespace configuration: SYSTEM, ROLLBACK, TEMP, TOOLS, USERS* (small by default, ability to pre-allocate upfront if more space needed)
  - **Control files** (recovery info and integrity maintenance)

Memory (I)

- Maximize the requests satisfied in memory vs. performing I/O ops
  - Hits vs. misses - DB buffer cache
    - *Query table (xyz)* shows the effect of adding buffers:
      - 500 buffers -> 1200 hits
      - 1500 buffers -> 6700 hits
- Background processes to support DB activity
    - PMON – process recovery
    - SMON – instance recovery
    - DBRW – writes info from buffers to db
    - LGWR – info from redo log buffer to online redo logs
    - CKPT – responsible for header updates (takes work away from LGWR) set to true if more than 20 db files
    - ARCH – copies redo logs
- **Trace Files** (info about user sessions)

Memory (II)

- SGA (system global area)
  - Data and control info particular to an Oracle instance
  - # of buffers dedicated to the cache
  - # of bytes allocated to the shared SQL area
- How much memory is enough?
  - O/S, buffers, coexisting software, Oracle db, etc
  - Roughly 3 times that calculated for the support of the Oracle systems
- Shared pool (library & dictionary)
  - ***v$ibrarycache*** dictionary (sql statements)
  - info pertaining to segments (indexes, tables)

Memory (III)

- Multithreaded Server
  - Server work is done on behalf of the user by a dedicated process (shadow)
  - Pool of server processes to be shared by users for
  - Edited in the MTS Initialization Parameter File
  - main memory conservation
- **SORT_AREA_SIZE** parameter
  - allocation of chunks of memory for sorting activities
  - 512K (10g) default, DBA can increase it though
  - if more than 25% of sort requests require disk space (using ***v$sysstat***), increase is necessary

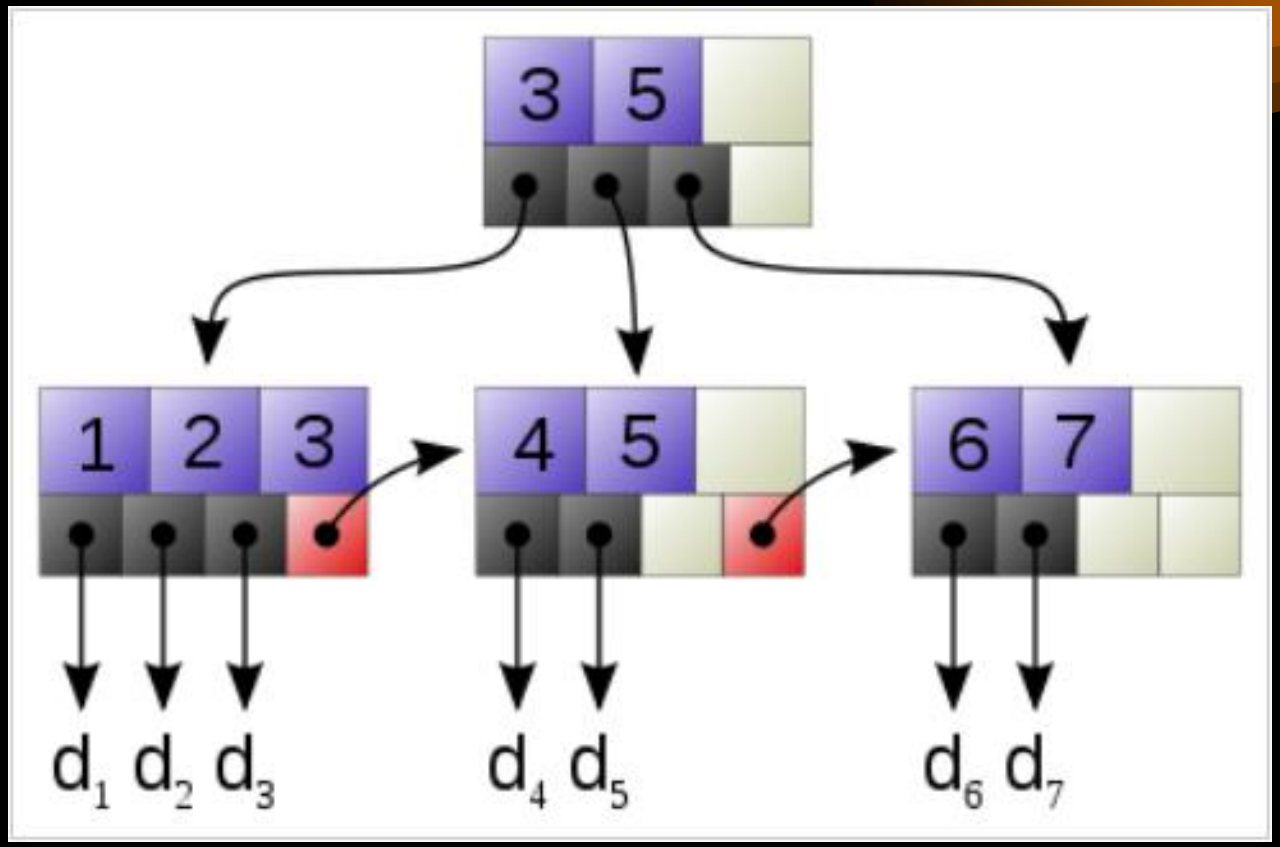
I/O (I)

- Separate tablespaces for heavily accessed tables and their indexes and place them on separate disks
- Knowing how data is to be accessed by end users
- Rollback segments
  - Must store enough information
  - Info about concurrent actions
  - Rollback segments must not be used in the system tablespace (b/c of extension needs)
- Allocate at least one tablespace exclusively used for temporary segments

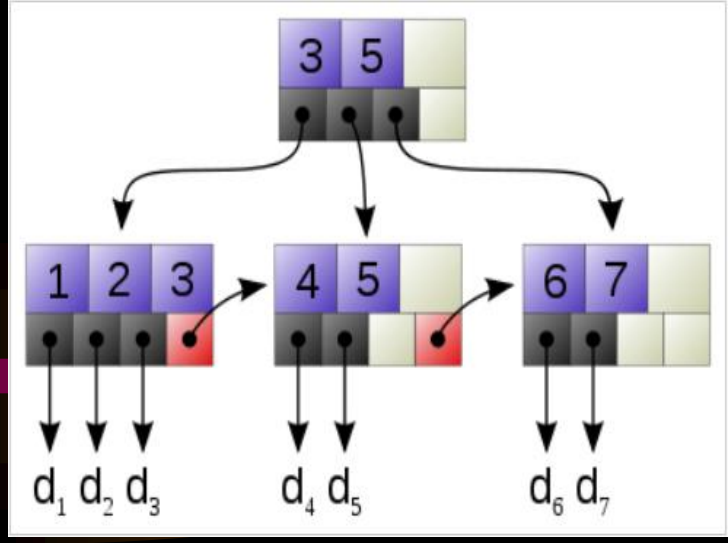
I/O (II)

- Redo logs must reside on a disk that has a low rate of reads and writes
- “Hot Spots” – files within Oracle db that are most heavily read or written to
  - Command ***monitor fileio*** to see the content of the pool
  - Make sure they are on separate disks

Oracle Indexing: B-Tree)



I/O (III)



- The root node
  - contains node pointers to intermediate nodes
- Branch node
  - contains pointers to other branch nodes (intermediate) nodes or leaf nodes
- Leaf node
  - contains index items and horizontal pointers to other leaf nodes.

I/O (IV)

- block size **b**
- **h** levels of entry
  - Maximum # of records stored

    $n = b^h$
  - Insert, find, remove operations

    $O(\log_b n)$

    in worst case

I/O (V)

- B-trees used to avoid large sorting operations
- Columns with high ***cardinality***
  - minimum 10%
    - If the table has 1000 rows then the column to index should have at least 100 different values

CPU (I)

- Maximizing CPU power
  - Allocate as much memory as possible to the shared pool and database buffers
  - Allocate more than default amount of memory to do sorting (sorts not requiring I/O use much less CPU)
  - Minimize I/O to free up CPU
  - On multi-CPU machines, make sure that one process per CPU deals with copying entries in the Redo Log Buffers

CPU (II)

- Maximize availability by distributing the load over the business day and night
- Backups
  - Schedule
  - 11:30 – 1:00PM tend to be most quiet
- Up an running 24h a day
  - Precious time is lost when db has to close for backup

Transaction Management

Goal: The ACID properties

- **Atomicity**: Either all actions are carried out, or none are
- **Consistency**: If each transaction is consistent, and the database is initially consistent, then it is left consistent
- **Isolation**: Transactions are isolated, or protected, from the effects of other scheduled transactions
- **Durability**: If a transactions completes successfully, then its effects persist

Concurrency in a DBMS

- Users submit transactions, and can think of each transaction as executing by itself.
  - Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
  - Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some ICs, depending on the ICs declared in CREATE TABLE statements.
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- *Issues:* Effect of *interleaving* transactions, and *crashes*.

Atomicity

- A transaction can
  - **Commit** after completing its actions, or
  - **Abort** because of
    - Internal DBMS decision: restart
    - System crash: power, disk failure, …
    - Unexpected situation: unable to access disk, data value...
- A transaction interrupted in the middle could leave the database inconsistent
- DBMS needs to remove the effects of partial transactions to ensure **atomicity**: either all a transaction's actions are performed or none

Atomicity cont.

- A DBMS ensures atomicity by **undoing** the actions of partial transactions
- To enable this, the DBMS maintains a record, called a **log**, of all writes to the database
- The component of a DBMS responsible for this is called the **recovery manager**

Consistency

- Users are responsible for ensuring transaction consistency
  - when run to completion against a consistent database instance, the transaction leaves the database consistent
- For example, consistency criterion that my inter-account-transfer transaction does not change the total amount of money in the accounts!
- **Database consistency** is the property that every transaction sees a consistent database instance. It follows from transaction atomicity, isolation and transaction consistency...

Isolation

- Guarantee that even though transactions may be interleaved, the net effect is identical to executing the transactions **serially**
- For example, if transactions T1 and T2 are executed concurrently, the net effect is equivalent to executing
  - T1 followed by T2, **or**
  - T2 followed by T1
- NOTE: The DBMS provides **no guarantee of effective order of execution**

Durability

- DBMS uses the log to ensure durability
- If the system crashed before the changes made by a completed transaction are written to disk, the log is used to remember and restore these changes when the system is restarted
- Again, this is handled by the recovery manager

Transactions and Schedules

- A transaction is seen by the DBMS as a series, or list, of actions
  - Includes read and write of objects
  - We'll write this as R(o) and W(o) (sometimes $R_T(o)$ and $W_T(o)$)
- For example

  T1: [R(a), W(a), R(c), W(c)]

  T2: [R(b), W(b)]
- In addition, a transaction should specify as its final action either **commit**, or **abort**

Schedules

- A **schedule** is a list of actions from a set of transactions
  - A well-formed schedule is one where the actions of a particular transaction T are in the same order as they appear in T
- For example
  - $[R_{T1}(a), W_{T1}(a), R_{T2}(b), W_{T2}(b), R_{T1}(c), W_{T1}(c)]$ is a well-formed schedule
  - $[R_{T1}(c), W_{T1}(c), R_{T2}(b), W_{T2}(b), R_{T1}(a), W_{T1}(a)]$ is not a well-formed schedule

Schedules cont.

- A **complete schedule** is one that contains an abort or commit action for every transaction that occurs in the schedule
- A **serial schedule** is one where the actions of different transactions are not interleaved

Serializability

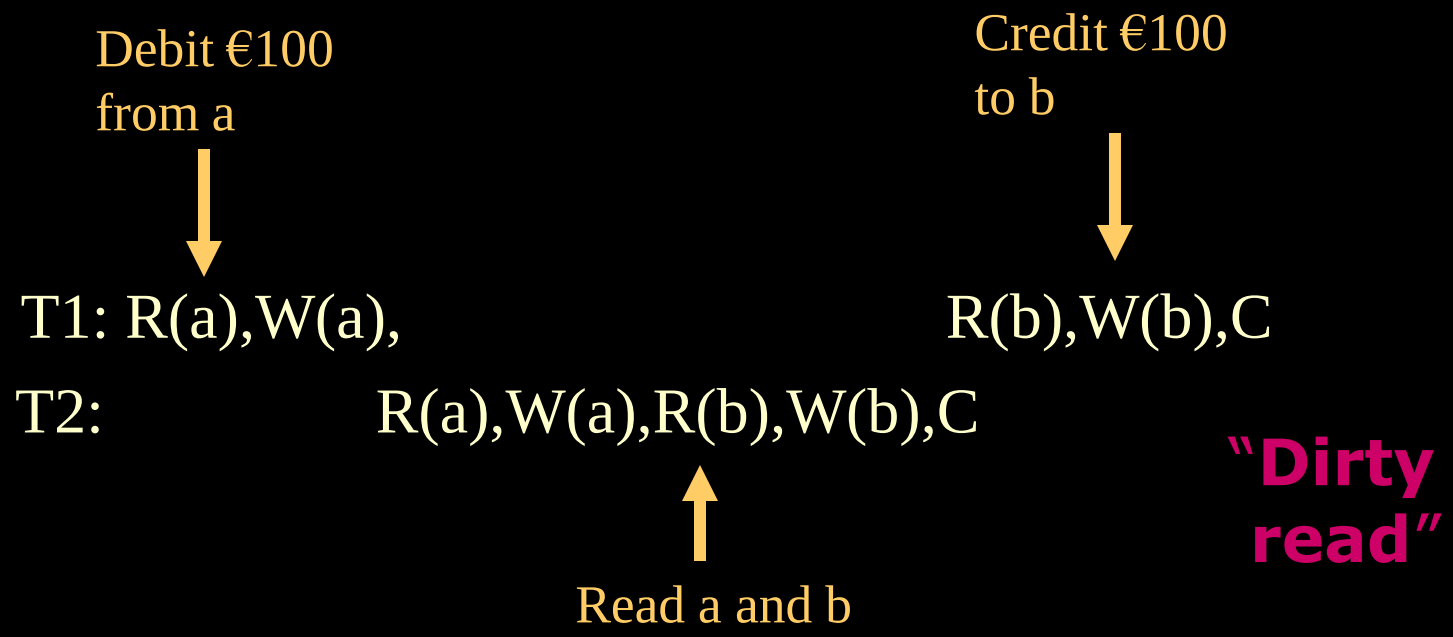
- A **serializable schedule** is a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule
- NOTE:
  - All different results assumed to be acceptable
  - It's more complicated when we have transactions that abort
  - We'll assume that all 'side-effects' of a transaction are written to the database

Anomalies with Interleaved Execution

- Two actions on the same data object **conflict** if at least one of them is a write
- We'll now consider three ways in which a schedule involving two consistency-preserving transactions can leave a consistent database inconsistent

WR conflicts

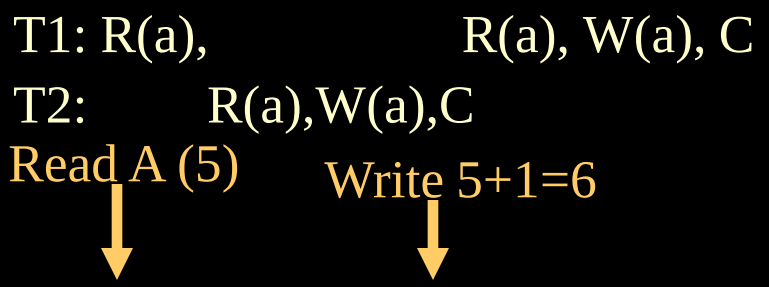
- Transaction T2 reads a database object that has been modified by T1 which has not committed



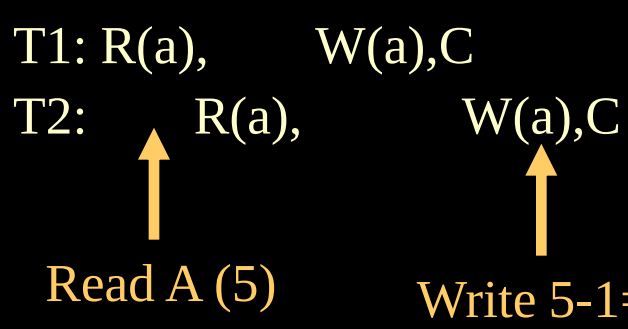
RW conflicts

- Transaction T2 could change the value of an object that has been read by a transaction T1, while T1 is still in progress

"Unrepeatable Read"



A is 4



WW conflicts

- Transaction T2 could overwrite the value of an object which has already been modified by T1, while T1 is still in progress

  T1: [W(Brit), W(gmb)] "Set both salaries at £1m"

  T2: [W(gmb), W(Brit)] "Set both salaries at $1m"

But:

"Blind Write"

T1: W(Brit), W(gmb)

T2: W(gmb), W(Brit)

gmb gets £1m
Brit gets $1m

Serializability and aborts

- Things are more complicated when transactions can abort

Deduct €100 from a

T1:R(a), W(a), **Abort**

T2: R(a),W(a),R(b),W(b),C

Add 6% interest to a and b

Can't undo T2
It's committed

Strict two-phase locking

- DBMS enforces the following locking protocol:
  - Each transaction must obtain an S (**shared**) lock before reading, and an X (**exclusive**) lock before writing
  - All locks held by a transaction are released when the transaction completes
  - If a transaction holds an X lock on an object, no other transaction can get a lock (S or X) on that object
- Strict 2PL allows only serializable schedules

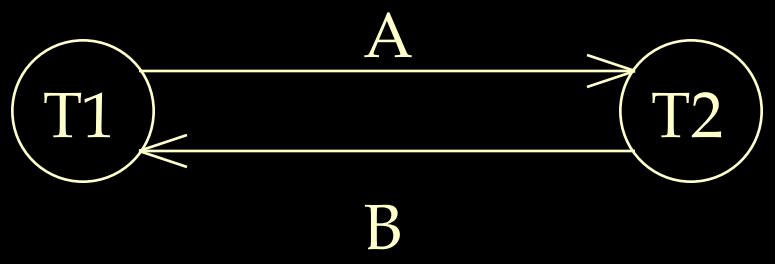
More refined locks

- Some updates that seem at first sight to require a **write (X)** lock, can be given something weaker
  - Example: Consider a seat count object in a flights database
  - There are two transactions that wish to book a flight – get **X** lock on seat count
  - *Does it matter in what order they decrement the count?*
    - They are **commutative actions!**
    - Do they need a write lock?

Example

- A schedule that is not conflict serializable:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	



Dependency graph

- The cycle in the graph reveals the problem. The output of T1 depends on T2, and vice-versa.

Aborting

- If a transaction Ti is aborted, then all actions must be undone
  - Also, if Tj reads object last written by Ti, then Tj must be aborted!
- Most systems avoid **cascading aborts** by releasing locks only at commit time (strict protocols)
  - If Ti writes an object, then Tj can only read this after Ti finishes
- In order to undo changes, the DBMS maintains a **log** which records every write

Two-Phase Locking to Ensure Serializability

- Defines how transactions acquire and relinquish locks
- Guarantees serializability, but it does not prevent deadlocks
  - *Growing phase*, in which a transaction acquires all the required locks without unlocking any data
  - *Shrinking phase*, in which a transaction releases all locks and cannot obtain any new lock

Two-Phase Locking to Ensure Serializability

- Governed by the following rules:
  - Two transactions cannot have conflicting locks
  - No unlock operation can precede a lock operation in the same transaction
  - No data are affected until all locks are obtained—that is, until the transaction is in its locked point

Wait/Die and Wound/Wait Schemes

- Wait/die
  - Older transaction waits and the younger is rolled back and rescheduled
- Wound/wait
  - Older transaction rolls back the younger transaction and reschedules it
- In the situation where a transaction is requests multiple locks, each lock request has an associated time-out value. If the lock is not granted before the time-out expires, the transaction is rolled back

The log

- The following facts are recorded in the log
  - "Ti writes an object": store new and old values
  - "Ti commits/aborts": store just a record
- Log records are chained together by transaction id, so it's easy to undo a specific transaction
- Log is often duplexed and archived on stable storage (it's important!)

Recovering From a Crash

- There are 3 phases in the *Aries* recovery algorithm:
  - *Analysis*: Scan the log forward (from the most recent *checkpoint*) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - *Redo*: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - *Undo*: The writes of all Xacts that were active at the crash are undone (by restoring the *before value* of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)

Connection to Normalization

- The more redundancy in a database, the more locking is required for (update) transactions.
  - Extreme case: so much redundancy that all update transactions are forced to execute serially.
- In general, less redundancy allows for greater concurrency and greater transaction throughput.

The Fundamental Tradeoff of Database Performance Tuning

- De-normalized data can often result in faster query response
- Normalized data leads to better transaction throughput

Yes, indexing data can speed up transactions, but this just proves the point --- an index IS redundant data. General rule of thumb: indexing will slow down transactions!

What is more important in your database--- query response or transaction throughput?

Distributed Database

- A distributed database system consists of loosely coupled sites that share no physical component
- Database systems that run on each site are independent of each other
- Transactions may access data at one or more sites

Homogeneous Distributed Databases

- In a homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - Appears to user as a single system
- In a heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

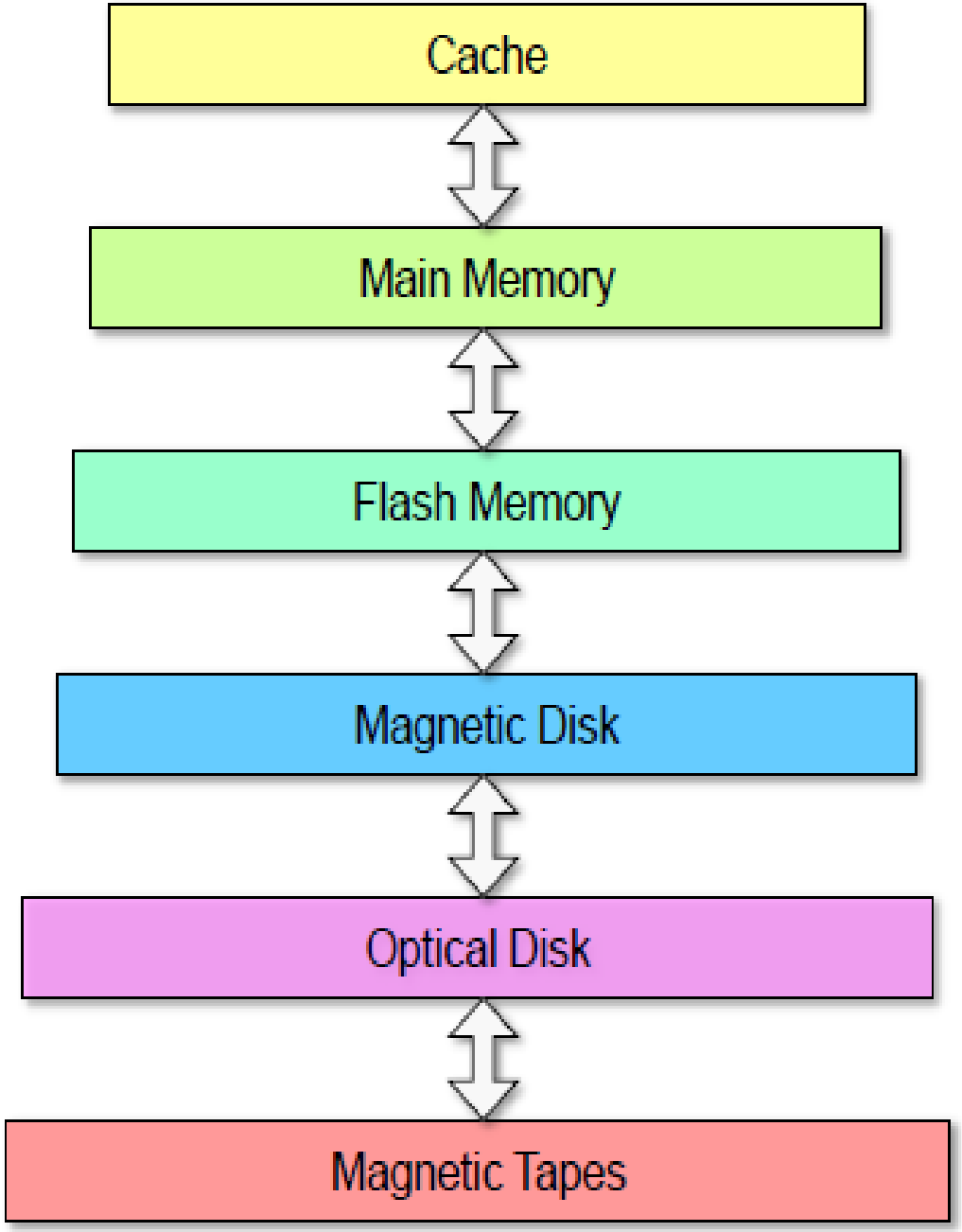
Distributed Transactions

- Transaction may access data at several sites.
- Each site has a local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Storage Management in Database

- **primary storage:** Fastest media but volatile (cache, main memory).
- **secondary storage:** next level in hierarchy, non-volatile, moderately fast access time
  - also called **on-line storage**
  - E.g. flash memory, magnetic disks
- **tertiary storage:** lowest level in hierarchy, non-volatile, slow access time
  - also called **off-line storage**
  - E.g. magnetic tape, optical storage

Storage Device Hierarchy



User Interface for Database

- DB supports *three* primary views for working with forms:
  - *Form Design View*, where the controls and layout can be modified, but the table data is not displayed
  - *Form View*, where controls linked to table fields display the field's value for the current record
  - *Datasheet View*, where you can work directly with the table used by the form

SQL Connectivity Issues

- Basically, when you failed to connect to SQL Server, the issue could be:
  1) Network issue,
  2) SQL Server configuration issue.
  3) Firewall issue,
  4) Client driver issue,
  5) Application configuration issue.
  6) Authentication and logon issue.

How to Troubleshoot Connecting to the SQL Server Database Engine

- Gathering Information about the Instance of SQL Server
- Enable Protocols
- Testing TCP/IP Connectivity
- Testing a Local Connection
- Opening a Port in the Firewall
- Testing the Connection

Database Access from Smartphone

- Consider SQLite or consider a mobile nosql platform like couchdb and just use JavaScript to manage it. All depends on your requirements.
- A smartphone or tablet is not a good development environment: lack of tools, lack of ergonomics for entering lots of text, lack of enough screen size etc.
- A laptop provides all you need in terms of the above, and you can use XAMPP or WAMP to test your scripts. Both include Apache, PHP and MySQL.

- There are some applications for android to make your smartphone as a local webserver.
  - lighttpd server v1.4.35 (SSL)
  - nginx v1.7.3 (SSL)
  - PHP v5.6.2 (SSL)
  - MySQL v5.6.19
  - msmtp v1.4.32
  - Web Interface v1.2
- Or to make your pc a local webserver you can use
  - Wamp for windows
  - Xammp for windows

Database on Cloud

- Cloud SQL is a **fully-managed database** service that makes it easy to set up, maintain, manage, and administer your relational PostgreSQL BETA and MySQL databases in the cloud. **Cloud SQL offers high performance, scalability, and convenience**. Hosted on Google Cloud Platform, Cloud SQL **provides a database infrastructure for applications running** anywhere.
- Cloud SQL is perfect for Wordpress sites, e-commerce applications, CRM tools, geospatial applications, and any other application that is compatible with MySQL or PostgreSQL BETA.

- Cloud SQL is easy to use. It doesn't require any software installation. It automates all your backups, replication, patches, and updates - while ensuring greater than 99.95% availability, anywhere in the world.
- Cloud SQL delivers high performance and scalability with up to 10TB of storage capacity, 40,000 IOPS, and 416GB of RAM per instance.
- asily configure replication and backups to protect your data. Go further by enabling automatic failover to make your database highly available (HA). Your data is automatically encrypted and Cloud SQL is SSAE 16, ISO 27001, PCI DSS v3.0, and HIPAA compliant.
- Cloud SQL offers per-second billing, automatic sustained use discounts, and instance sizes to fit any budget. Database instances are easy to stop and start. There is no up-front commitment, and with sustained use discounts, you'll automatically get discounted prices for databases that run continuously.

CLOUD SQL FEATURES

1. Scalability
2. High Performance
3. Integrated
4. Fully Managed
5. Security
6. Standard APIs
7. Availability Protection
8. Partnerships & Integrations

Data Warehouse and Data Mining

- A data warehouse is a copy of transaction data specifically structured for querying, analysis and reporting.
- Data integration from across and even outside the organization.
- Data mining is the process of using raw data to infer important business relationships.
- Data in the DW is integrated and stable.

- Data mining automates the process of locating and extracting the hidden patterns and knowledge
- “Searching for new knowledge”
- Data Warehouse, “Collection of Information gathered from multiple sources, stored under unified schema, at a single site & mainly intended for decision support applications.
- A subject oriented, integrated, nonvolatile, time-variant, collection of data in support of management’s decision.

Information Storage and Retrieval

- Gathering information from a source(s) based on an information need usually from a query
  - *Major assumption - that the information need can be specified*
  - *Broad definition of information*
  - *Most methods are automated - scaling*
- Sources of information
  - Other people
  - Archived information (libraries, maps, etc.)
  - Radio, TV, etc.
  - Web (search engines)
  - Natural settings

Information retrieval is more than just web search

Information Retrieval vs.

- Information retrieval (IR) is the activity or process of obtaining information resources relevant to an information need from a collection of information resources.
- Data mining is the process that attempts to discover patterns in large data sets.
- Information extraction (IE) is the task of automatically extracting structured information from unstructured and/or semi-structured machine-readable documents