Disclaimer

This document summarizes the lecture slides of *Database Systems & Information Modelling* and thus is for the use of students enrolled in INFO90002 only.

This document is distributed on an “as is” basis, without warranties or conditions of any kind, either express or implied. No further support will be provided for this document.

Week 1

File Processing

Problems with flat-files

1. Data access routines must be programmed in detail
2. Each program must include full detail of data structure
3. Multiple users cannot simultaneously access data
4. Multiple copies of data - not centrally managed

Advantages of relational db over file processing

1. Data independence
2. Minimize redundancy
3. Improved consistency
4. Improved sharing
5. Reduced program maintenance
6. Better application development productivity
7. Improved data quality
8. Standards
9. Ad-hoc data queries

Database life cycle

1. Design the database
2. Implement the database
    - Data Definition Language (`CREATE`, `DROP`, `ALTER`, `RENAME`)
3. Data access / programming
    - Data Manipulation Language (`SELECT`, `INSERT`, `UPDATE`, `DELETE`)
4. Data administration
    - Data Control Language (`GRANT`, `REVOKE`)

Week 2

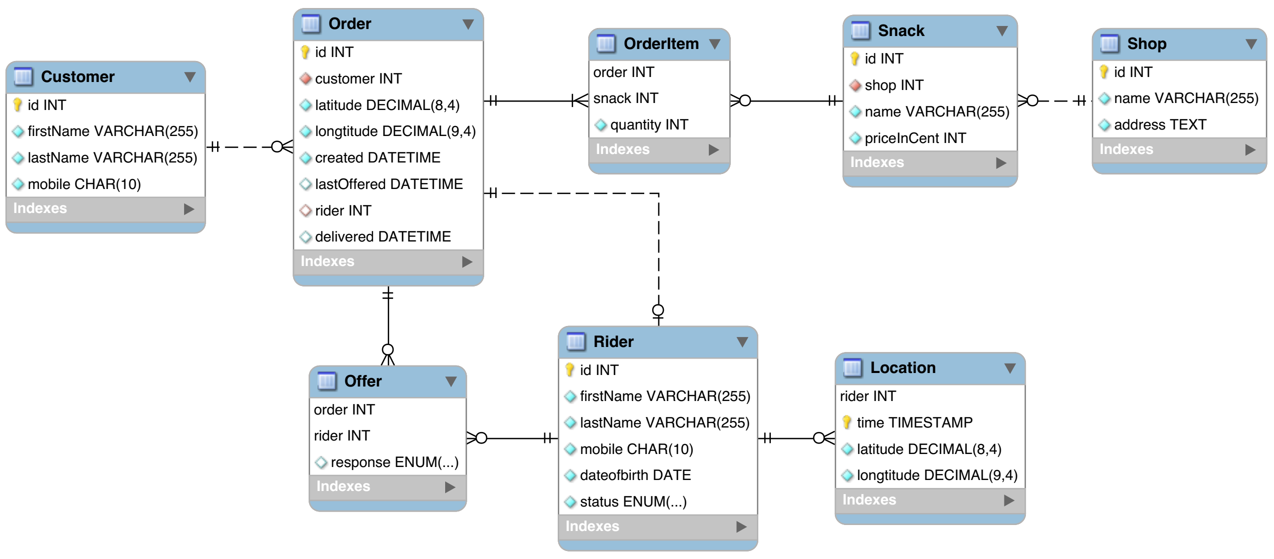
Database Development life cycle

1. Planning
2. Definition
3. Requirement Analysis
4. Design
5. Implementation
6. Data loading
7. Testing
8. Maintenance

Design

1. Conceptual
   - High-level, first-pass model of entities and their connections
   - Typically omits attributes
   - Could potentially be implemented in a non-relation database
   - Thus can include many-to-many relationships, repeating groups, composite attributes
2. Logical
   - Builds on the conceptual design
   - Designing now for a relational database
   - Includes columns and keys
   - Specific vendor and other physical considerations
3. Physical
   - Implements the logical design for a specific DBMS
   - Describes:
     (a) Base tables
     (b) Data types
     (c) Indexes
     (d) Integrity constraints
     (e) File organization
     (f) Security measures

Data Modeling



SQL

Assignment #2 (2017 S2)

```
SELECT
    postcode, COUNT(id) AS 'number of stores'
FROM
    Store
GROUP BY postcode
HAVING COUNT(id) > 1
```

```
SELECT
    PaymentMethod.name AS method,
    SUM((price + tax) * quantity) AS amount
FROM
    PurchaseItem
        INNER JOIN
    Purchase ON PurchaseItem.purchase = Purchase.id
        INNER JOIN
    Product ON PurchaseItem.product = Product.id
        INNER JOIN
    PaymentMethod ON Purchase.paymethod = PaymentMethod.id
GROUP BY PaymentMethod.name
```

```
SELECT
    store, customer, COUNT(id) AS times
FROM
    Purchase
GROUP BY store, customer
```

```
SELECT ROUND(
    (
        SELECT
            COUNT(Purchase.id)
        FROM
            Purchase
        INNER JOIN
            Customer ON Purchase.customer = Customer.id
        WHERE
            YEAR(dateofbirth) < 1990
    ) / (
        SELECT
            COUNT(id)
        FROM
            Purchase
    ) * 100, 1
) AS percentage
```

```
SELECT
    firstName, lastName, Customer.postcode AS postcode
FROM
    Customer
        INNER JOIN
    Purchase ON Purchase.customer = Customer.id
        INNER JOIN
    Store ON Purchase.store = Store.id
WHERE
    Customer.postcode = Store.postcode
```

Functions

1. AVG
   STD
2. CONCAT
3. COUNT
   SUM
4. CURDATE
   NOW
   TIMESTAMPDIFF
5. DATE_FORMAT
6. FORMAT
   ROUND
7. IF
   IFNULL
   ISNULL
8. LENGTH

Week 6 Normalization

- A theoretical foundation for the relational model

- Application of a series of rules that improve DB design
- Last phase of logical design
- Normalization rules are designed to
  - Prevent data redundancy
  - Prevent update anomalies
- Normalization is biased towards optimizing `UPDATE`s
  - i.e. assuming non-key columns are updated frequently
  - thus slowing down `SELECT`s because of the need to join tables
  - thus designers sometimes "denormalize" some tables to improve performance
- Every attribute must be functionally dependent on "the key, the whole key, and nothing but the key".

Dependency

- Partial functional dependency
  - a functional dependency of one or more non-key attributes upon **part (but not all)** of the primary key.
- Transitive dependency
  - a functional dependency between two (or more) non-key attributes

Hierarchy of Normal Forms

- First Normal Form
  - Any **multivalued** attributes and **repeating groups** have been removed
- Second Normal Form
  - Any **partial** functional dependencies have been removed
    i.e. all non-key attributes are identified by the **whole** key
- Third Normal Form
  - Any **transitive** dependencies have been removed
    i.e. non-key attributes are identified by **only** the **primary** key

What if we did not normalize?

- Repetition of data
  - individual facts are stored many times.
- Update anomalies
  - to change a fact, we must change it many times.
- Deletion anomalies
  - information about entity A is stored inside entity B.
    If we delete all B rows, we lose our record of A.
- Insertion anomalies
  - to record a new A, we must insert a B

Week 6 Physical Database Design

Aims of physical design

- Purpose: translate the logical description of data into the technical specifications for storing and retrieving data on disk
- Goal: create a design for storing data that will provide good performance and ensure database integrity, recoverability and security

Physical Design Process

Inputs

- normalized data model
- attribute definitions
- volume estimates
- response time expectations
- data security needs
- backup / recovery needs
- integrity expectations
- DBMS used

Decisions

- attribute data types
- physical record descriptions (does not always match logical design)
- file organization
- indexes
- query optimization

Data integrity

- Default value
  - assumed value if no explicit value given
- Range control
  - allowable value limitations (constraints or validation rules)
- Null value control
  - allowing or prohibiting empty columns (nullable / mandatory)
- Referential integrity
  - checks values (and null value allowances) of foreign-key (i.e. domain of FK)

Indexing

- Index: a separate file that contains pointers to table rows, to allow fast retrieval
  - similar to an index in a book
- PKs and FKs are automatically indexed
- when to use indexes ⚡
  - use on larger tables
  - on columns which are frequently in `WHERE` clause
  - or in `ORDER BY` and `GROUP BY` commands
  - if column has >100 distinct values; but not if <30 values
- limit the use of indexes for volatile databases
  - volatile = data are frequently updated
  - indexes need to be updated when table data are changed

De-normalization

- Normalization
  - Removes data redundancy
  - Solves `INSERT`, `UPDATE` and `DELETE` anomalies
  - makes it easier to maintain information in an consistent state
- However
  - It leads to more tables in the database
  - Often these need to be joined back together during `SELECT`s which is expensive to do
  - sometimes we decide to de-normalize
- De-normalization opportunities
  - database speeds are unacceptable
  - there are going to be very few `INSERT`s, `UPDATE`s or `DELETE`s
  - there are going to be lots of `SELECT`s that involve the joining of tables
- Examples
  - one-to-one relationship
  - many-to-many relationship
  - reference data / lookup table

Week 7 Databases in Applications

Procedural programming languages can do:

1. Sequence (several steps performed in order)
2. Iteration (loops)
3. Control flow (conditions, decisions)
4. User interface (accept input and present output for users)

Stored Procedures & Triggers

1. Advantages
    - Complied SQL statements
    - Faster code execution
    - Reduced network traffic
    - Improved security and data integrity
    - Business logic under control of DBA
    - Thinner clients
2. Disadvantages
    - Harder to write code
    - Proprietary language

System Architecture

- Presentation logic $\leftrightarrow$ User interface
    - input (keyboard, touchscreen, voice)
    - output (large screen, printer, phone, ATM)
- Business logic $\leftrightarrow$ Application Software
    - input and command handling
    - enforcement of business rules
- Storage logic $\leftrightarrow$ Data Storage
    - persistent storage of data
    - enforcement of data integrity

Two-tier architecture

- Advantages
    - Clients and server share processing load
    - Good data integrity since data is all processed centrally
    - Stored procedures allow some business rules to be implemented on the database server
- Disadvantages
    - Presentation, data model, business logic are intertwined at client
    - If DB schema changes, all clients break
    - Updates need to be deployed to all clients
    - DB connect for every client, thus difficult to scale
    - Difficult to implement beyond the organization (to customers)

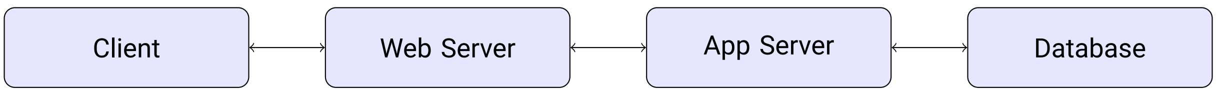
Three-tier architecture

- Advantages
  - Scalability
  - Technological flexibility (can change business logic easily)
  - Can swap out any single component fairly easily
  - Long-term cost deduction
  - Improved security - customer machine dose presentation only
- Disadvantages
  - High short-term cost
  - Tools and training
  - Complex to design
  - Variable standards

Three-tier web-based architecture

- Browser handles presentation logic
- Browser talks to web server via simple, standard protocol
- Business logic and data storage handled on servers
- Pros
  - Everyone has a browser
  - No need to install and maintain client software
  - widely supported simple protocol
  - opens up the possibility of global access to database
- Cons
  - Even more complexity in the middle-tier
  - Simple standards $\rightarrow$ hard to make complex application
  - Global access $\rightarrow$ potentially security nightmare

Four-tier architecture



Security

- Network environment creates complex security issues
- Security can be enforced at different tiers
  - application password security
    - for allowing access to the application software
  - database-level password security
    - for determining access privileges to tables
  - secure client / server communication
    - via encryption

Week 7 Web Apps

Why create web applications?

1. Web browsers are ubiquitous
2. No need to install client software for external customers
3. Simple communication protocol
4. Platform and operating system independent
5. Reduction in development time and cost
6. Has enabled eGov, eBusiness, eCommerce, B2B, C2C

Problems with old-style web apps

- Placing "raw" SQL inside PHP/HTML files
  - Mixes presentation, business logic, database
  - Hard to maintain when things change
  - Want separation of concerns e.g. MVC
- Lots of reinvention of wheels
  - each dev writes their own solution to common features
  - e.g. login security, presentation templates, database access
- Increasing variety of clients e.g. phones and tablets
  - Manually program for different platforms

Web Services

- The WWW allows humans to access databases
- Web Services allow computer to access database
- 2 major approaches: SOAP and REST
  - Simple Object Access Protocol
  - REpresentational State Transfer
- structured data usually returned in XML or JSON format
- REST nouns are resources, addressed via URIs
- REST verbs correspond to DML statements
- `GET`(select), `POST`(insert), `PUT`(update), `DELETE`(delete)

Week 8 Transactions and Concurrency

Transactions solve two problems:

1. users need the ability to define a unit of work
2. concurrent access to data by more than one user or program

Transaction Properties (ACID) ⚡

1. Atomicity
   - A transaction is treated as a single, indivisible, logical unit of work. All operations in a transaction must be completed; if not, then the transaction is aborted.
2. Consistency
   - Constraints that hold before a transaction must also hold after it.
   - (multiple users accessing the same data see the same value)
3. Isolation
   - Changes made during execution of a transaction cannot be seen by other transactions until this one is completed.
4. Durability
   - When a transaction is complete, the changes made to the database are permanent, even if the system fails.

Problems caused by concurrent access

1. lost update ⚡
   - the time line diagram
2. uncommitted data
   - Uncommitted data occurs when two transactions execute concurrently and the first is rolled back after the second has already accessed the uncommitted data.
3. inconsistent retrieval
   - Occurs when one transaction calculates some aggregate functions over a set of data, while other transactions are updating the data.
     - Some data may be read after they are changed and some before they are changed, yielding inconsistent results.

Concurrency control methods

1. locking
2. time stamping
3. optimistic method

Lock

- Guarantees exclusive use of a data item to a current transaction
- Required to prevent another transaction from reading inconsistent data
- Lock manager
  - Responsible for assigning and policing the locks used by the transactions
- Lock Granularity
  1. Database-level lock

    - Good for batch processing but unsuitable for multi-user DBMSs
    - SQLite, Access
  2. Table-level lock
    - Can cause bottlenecks, even if transactions want to access different parts of the table and would not interfere with each other
    - Not suitable for highly multi-user DBMSs
  3. Page-level lock
    - An entire disk page is locked (a table can span several pages and each page can contain several rows of one or more tables)
    - Not commonly used now
  4. Row-level lock
    - Improves data availability but with high overhead (each row has a lock that must be read and written to)
    - Currently the most popular approach (MySQL, Oracle)
  5. Field-level lock
    - Allow concurrent transactions to access the same row, as long as they access different attributes within that row
    - Most flexible lock but requires an extremely high level of overhead
    - Not commonly used

- Types of Locks
  - binary locks
  - exclusive and shared locks

Week 8 Distributed Databases

- Distributed database
  - A single logical database physically spreads across multiple computers in multiple locations connected by a data communications link
  - Appears to users as though it is one database
- Decentralized database
  - a collection of independent databases which are not networked together as one logical database
  - appears to users as though many databases

Advantages

1. good fit for geographically distributed organizations and users
2. data located near site with greatest demand
3. faster data access (to local data)
4. faster data processing
    - workload spread out across many machines
5. allow modular growth
    - add new servers as load increases

6. increased reliability and availability
    - less danger of single-point failure
7. supports database recovery
    - data is replicated across multiple sites

Disadvantages

1. Complexity of management and control
    - db or application must stitch together data across sites
2. Data integrity
    - additional exposure to improper updating
3. Security
    - many server sites $\rightarrow$ more chance of breach
4. Lack of standards
    - different DBMS vendors use different protocols
5. Increased training costs
    - more complex IT infrastructure
6. Increased storage requirement
    - if there are multiple copies of data

Trade-offs

- Availability vs Consistency
- Synchronous updates vs Asynchronous updates

Objectives

Local Transparency

1. A user (or program) using data need not know the location of the data in the network of DBMS
2. Requests to retrieve or update data from any site are automatically forwarded by the system to the site or sites related to the processing request
3. All data in the network appears as a single logical database stored at one site to the users
4. A single query can join data from tables in multiple sites

Local Autonomy

1. Users can administer their local database
    - control local data
    - administer security
    - log transactions
    - recover when local failures occur
    - provide full access to local data
2. Being able to operate locally when connections to other databases fail

Distribution Options ⚡

1. Data replication
   - data concept across sites
2. Horizontal partitioning
   - Tables rows distributed across sites
3. Vertical partitioning
   - Tables columns distributed across sites
4. Combinations of the above

Replication

This is currently popular as a way of achieving high availability for global systems. Most NoSQL databases offer replication.

Advantages

1. High reliability due to redundant copies
2. Fast access to local data
3. May avoid complicated distributed integrity routines
   - if replicated data is refreshed at scheduled intervals
4. Decouples nodes
   - transactions proceed even if some nodes are down
5. Reduced network traffic at prime time
   - if updates can be delayed

Disadvantages

1. Need more storage space
2. Integrity
   - can retrieve incorrect data if updates have not arrived
3. Takes time for update operations
   - high tolerance for out-of-date data may be required
   - updates may cause performance problems for busy nodes
4. Network communication capabilities
   - updates place heavy demands on telecommunications
5. Better for non-volatile (read-only) data

Partitioning

Advantages

- data stored close to where it is used → efficiency
- local access optimization → better performance
- only relevant data is stored locally → security
- horizontal partitioning
  - unions across partitions → ease of query
- vertical partitioning
  - combining data across partitions is more difficult because it requires joins

Disadvantages

- accessing data across partitions
  - inconsistent access speed
- no data replication
  - backup vulnerability

Week 9 Database Administration

Data & Database Administration

- Data Administrator (management role)
  - data policies, procedures and standards
  - planning
  - data conflict resolution
  - managing info repository (data dictionary)
  - internal marketing
  - similar to Chief Data Officer
- Database Administrator (technical role)
  - analyze and design DB
  - select DBMS / tools / vendor
  - install and upgrade DMBS
  - tune DBMS performance
  - manage security, privacy, integrity
  - backup and recovery

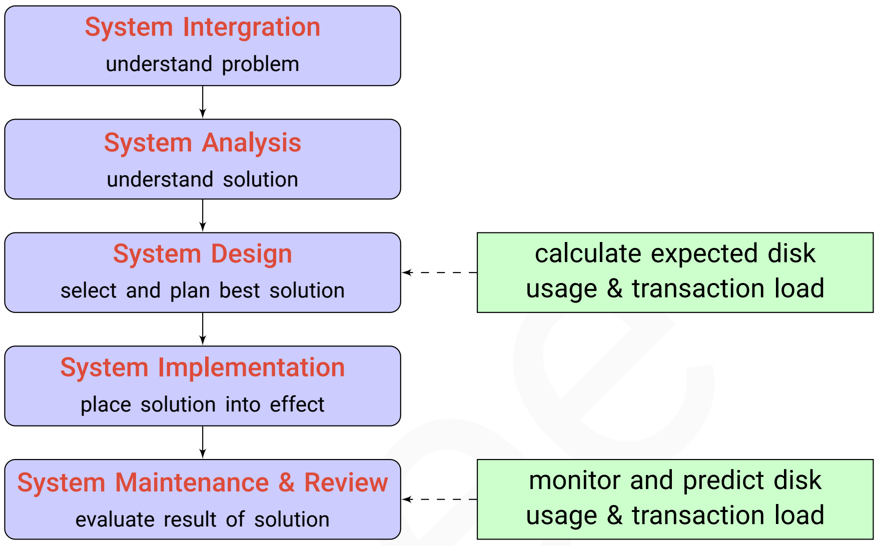
DBMS Architecture

Buffer Pool

Capacity Planning

When implementing a database, need to consider:

1. disk space requirement
2. transaction throughput
3. at go-live and throughout the life of the system


System Intergration
understand problem
System Analysis
understand solution
System Design
select and plan best solution
System Implementation
place solution into effect
System Maintenance & Review
evaluate result of solution
calculate expected disk usage & transaction load
monitor and predict disk usage & transaction load

- Estimate disk space requirement
  - calculate row widths
  - calculate growth of tables
  - other disk space requirement
    * control files
    * data dictionary
    * indexes
    * undo area
    * sort area
    * redo logs
- Estimate transaction load
  - speed and availability requirement

Database Performance

- caching data in memory
- placement of datafiles across disc drives
- database replication and server clustering
- use of fast storage such as SSD

- use of indexes to speed up searches and joins
- good choice of data types (especially PKs)
- good program logic
- good query execution plans

Security

Threats to databases

- Loss of integrity
  - keep data consistent
  - free of errors or anomalies
- Loss of availability
  - must be available to authorized users for authorized purposes
- Loss of confidentiality
  - must be protected against unauthorized access

Countermeasures

- Access control
  - grant different types of discretionary privileges to users
    * account level
    * table level
  - views are an important discretionary authorization mechanism
    * hiding the database structure
    * hiding some data (i.e. columns in tables)
- Encryption
  - Particular tables or columns may be encrypted to:
    * protect sensitive data (e.g. password) when they are transmitted over a network
      · prevents interception by third party
    * encrypt data in the database (e.g. credit card numbers)
      · provides some protection in case of unauthorized access
  - Data is encoded using an algorithm
    * authorized users are given keys to decipher data

SQL Injection ϟ

- a technique used to exploit web applications that use user input within database queries
- malicious code is entered in to a data entry filed in such a way
- How to prevent
  - sanitize user inputs
  - pass inputs as parameters to a stored procedure, rather than directly building the SQL string in the code.

SQL Injection Prevention ϟ

- Primary defences
  - prepared statements (parameterized queries)
  - stored procedures (both mean SQL is no longer dynamic)
- Additional defences
  - Principle of least privilege (do not give application accounts DBA privileges)
  - White list input validation (check input is from a list of acceptable values)

Backup & Recovery

- A backup is simply an extra copy of your data
  - however there are several types of backup
- If data becomes corrupted or deleted, it can be restored from the backup copy
- A backup and recovery strategy is need to plan how data is backed up and how it will be recovered

Protect data from

1. human error
2. hardware or software malfunction
   - bug in application
   - hard drives
   - CPU
   - memory
3. malicious activity
   - security compromise
     - server, database, application
4. natural or man made disasters
   - consider the scale of the damage
5. government regulation
   - historical archiving rules

Types of backups

1. Physical vs Logical ϟ
2. On-line (hot) vs Off-line (cold)
3. Full vs Incremental

Backup Policy

1. backup strategy is usually a combination of full and incremental backups
2. conduct backups when database load is low
3. if you replicate the database, use the mirror database for backups to negate any performance concerns with the main database
4. you need to test hat you can restore from the backup

Other Methods

1. server replication
2. server cluster
3. RAID (Redundant Array of Independent Disks)
4. off site solutions (enables disaster recovery)
    - backup tapes transported to underground vault
    - remote mirror database maintained via replication
    - backup to cloud

Week 10 NoSQL

Relational databases

- Pros
  - flexible, suits any data model
  - can integrate multiple applications via shared data store
  - standard within and between organizations
  - standard interface language SQL
  - ad-hoc queries, across and within data aggregates
  - fast, reliable, concurrent, consistent
- Cons
  - object-relational impedance mismatch
  - not good with big data
  - not good with clustered / replicated servers

NoSQL Databases

- Features
  - does not use relational data model or SQL language
  - designed to run on distributed servers
  - most are open-source
  - built for modern web
  - schema-less (though there may be implicit schema)

    * e.g. MongoDB ⚡
  - eventually consistent

- Goals
  - to improve programmer productivity (object-relation impedance mismatch)
  - to handle larger data volume and throughput (big data)

Aggregate-oriented Databases

- Key-value
  - Redis, Memcached
- Document
  - the content can be queried and partially updated
  - MongoDB
- Column-family
  - Each row can store a different set of columns.
  - Columns rather than rows are stored together on disk.
  - faster analysis by column - not OLTP (on-line transaction processing)
  - BigTable (Google), HBase (Apache), DynamoDB (Amazon)
- Pros
  - entire aggregate of data is stored together
  - efficient storage on clusters / distributed databases
- Cons
  - hard to analyze across subfields of aggregates
  - e.g. sum over products instead of orders

Graph Databases

- A graph is a node-and-arc network
- Social graphs (e.g. friendship relational DB)
- Graphs are difficult to program in relational DB
- A graph DB stores entities and their relationships
- Graph queries deduce knowledge from the graph

CAP Theorem

- Consistency - Availability - Partition tolerance
- In the presence of a partition, one is then left with two options: consistency or availability.
  - When choosing consistency over availability, the system will return an error or a time-out if particular information cannot be guaranteed to be up-to-date due to network partitioning.
  - When choosing availability over consistency, the system will always process the query and try to return the most recent available version of the information, even if it cannot guarantee it is up-to-date due to network partitioning.

BASE

- Basically Available
  - This constraint states that the system does guarantee the availability of the data: there will be a response to any request. But data may be in an inconsistent or changing state.
- Soft state
  - The state of the system could change overtime - even during times without input there may be changes going on due to eventual consistency.
- Eventual consistency
  - The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it needs to, sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

Week 12 Database Trends

Top Vendors

- Commercial
  - Oracle, Microsoft SQL Server, IBM DB2
  - top 3 own $\frac{3}{4}$ of market
- Open-source
  - MySQL, MariaDB, PostgreSQL, MongoDB

Big Data

- Difficult to process high volume and velocity of data using traditional database management and processing tools
  - Data velocity: batch $\rightarrow$ periodic $\rightarrow$ real-time
- Challenges include capture, storage, search, sharing, analysis and visualization.
- Relational database emerged 40 years ago when data was generated within organizations, mostly by employees, at a human scale, and often manually entered.
- This scenario still exists. In situations like this, relational may continue to dominate.
- But technical innovations mean data is generated by: social media, global website click streams, machinery, sensors, scientific instruments. (data variety)
- Volume and velocity of data is too high for SQL databases.

Solution

1. "Bigger servers" gives diminishing returns
2. Therefore spread data across many small servers
3. But relational not good with distributed data
4. So these companies devised new databases that are designed from the ground up to run across multiple servers

Problems

- in its infancy
  - tools for using and analyzing big data, as well as standards, are still being developed
- Need the right talent, technology and structure of workflows to optimize the use of big data
- Requires expensive professionals

Cloud Storage

Advantages

- Scalability
- Reduced administrative burden
  - simplifies setting up

Potential Issues

- Data security
  - provider may not have fully integrated security structure
  - may need to resort to encryption
- Legal frameworks
  - need to (continue to) conform to laws governing use of data
- Large movements of data between your site and cloud
  - during initial setup
  - during ongoing integration with local data

Object-oriented Application Software

Relational vs Object-oriented

- History
  - relational DB rose to prominence in the 1980s
  - object-oriented programming rose later in the 1990
- Mismatch
  - OO allows more complex data structures
  - complex objects must be normalized into relational database
- How can an OO program persist data in a database?
  - relational database (simple, powerful, but not OO)
  - object-relational databases (complex data types)
  - object-oriented databases (did not become popular)
  - object-relational mapping software (currently popular)
  - NoSQL (becoming popular in some niches)

Object-oriented DBMS

- Direct storage of object-oriented data
- Offers benefits of OO and Relational
  - directly persist data
  - offers performance and security of relational DB
- But did not become popular in the marketplace
  - often tied to particular OO programming language
  - lack of compatibility between different OODBMSs
  - lack of standard ad hoc query language
  - RDBMSs are used to integrate applications

NewSQL

- Features
  - like NoSQL, but offers ACID transactions
  - suited where consistency is important, e.g. financial data
  - offers traditional relational features like SQL, transaction
  - but scales horizontally like NoSQL systems
  - distributed, automatic sharding

Hadoop

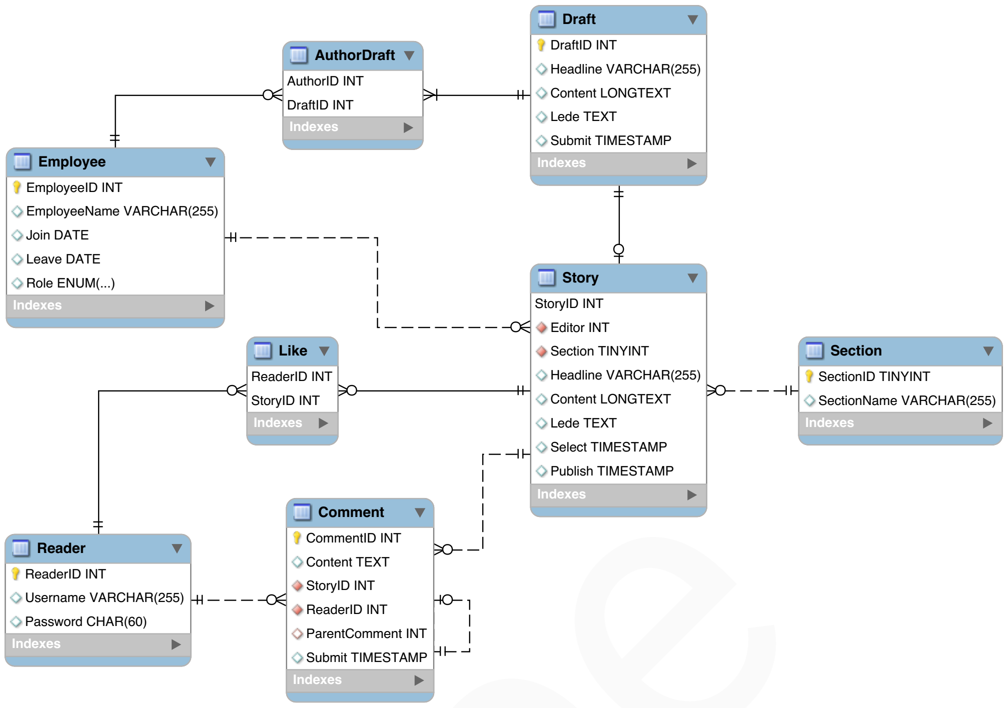
- for data-processing, not operational database
- Hadoop = framework of software utilities including:
  - HDFS (Hadoop Database File System): for distributed processing across a cluster of many computers
  - HBase: database for data storage
  - MapReduce: for processing data across the cluster

In-memory database

- data are stored in memory and processed there
- disk is for long-term persistence
- primary motive is speed of storage, transactions and analytics
- access speed of main memory is much faster than disk
- SAP HANA is the best-known example
- support OLTP and OLAP

2016 S1

Q1



Q2 SQL

A) ambiguous

Which tenants have paid us the most rent? List the top 5 payers, and how much rent each has paid in total.

```
SELECT
    CONCAT(givenName, ' ', surname), SUM(rent) AS totalPayment
FROM
    Tenant
        INNER JOIN
    Tenancy ON Tenant.id = Tenancy.tenant
WHERE
    dateEnd <= CURDATE()
GROUP BY givenName , surname
ORDER BY SUM(rent) DESC
LIMIT 5
```

B)

What is the longest given name among our tenants? If several names tie for first place, list them all.

```
SELECT
    givenName
FROM
    Tenant
WHERE
    LENGTH(givenName) = (
        SELECT
            MAX(LENGTH(givenName))
        FROM
            Tenant
    )
```

C)

List the addresses of the house and apartments which only have one bedroom.

```
SELECT
    CONCAT(
        IF(ISNULL(unitnum), '', CONCAT(unitnum, ' ')),
        street,
        ' ',
        suburb
    )
FROM
    Property
        LEFT JOIN
    Apartment ON Property.id = Apartment.id
        LEFT JOIN
    House ON Property.id = House.id
WHERE
    Apartment.numberBed = 1
    OR House.numberBed = 1
```

D)

List the names of tenants who have rented at least two different types of properties.

```
SELECT
    CONCAT(givenName, ' ', surname)
FROM
    Tenant
        INNER JOIN
    Tenancy ON Tenant.id = Tenancy.tenant
        INNER JOIN
    Property ON Tenancy.property = Property.id
GROUP BY Tenant.id
HAVING COUNT(DISTINCT propertyType) >= 2
```

Q3 Normalization

STUDENTMARKS(studentid, givenName, surname, (assignmentId, assignmentTitle, mark, dateSubmitted))

1NF

STUDENTMARKS(studentid, givenName, surname)
SUBMISSION(*studentid*, assignmentId, assignmentTitle, mark, dateSubmitted)

2NF

STUDENTMARKS(studentid, givenName, surname)
SUBMISSION(*studentid*, *assignmentId*, mark, dateSubmitted)
ASSIGNMENT(assignmentId, assignmentTitle)

Q4 Physical Design

1. `Tenant.id` $\longrightarrow$ `INT`
2. `Tenant.surname` $\longrightarrow$ `VARCHAR(255)`
3. `Tenant.phone` $\longrightarrow$ `CHAR(10)`
4. `Tenancy.tenant` $\longrightarrow$ `INT`
5. `Tenancy.dateEnd` $\longrightarrow$ `DATE`
6. `PropertyType.id` $\longrightarrow$ `CHAR(1)`
7. `Property.unitnum` $\longrightarrow$ `VARCHAR(255)`
8. `Property.postcode` $\longrightarrow$ `CHAR(4)`
9. `Property.rent` $\longrightarrow$ `MEDIUMINT`
10. `Appartment.numBedrooms` $\longrightarrow$ `TINYINT`