INFO20003 Tutorial 12

starting ~ 2.20 pm

Today's tutorial

- NoSQL
- CAP Theorem
- Revision

non-relational

Key Concepts:

- What are NoSQL databases?
- Types of NoSQL database
  - Graph databases
  - Key-value stores
  - Column-family stores
  - Document stores

value = JSON, XML

What are NoSQL databases?

A NoSQL database, also referred to as a **non-relational database**, is an approach that provides a facility to store and retrieve data in formats **other than tabular form**.

NoSQL databases **do not depend** on any particular **structure** such as tables, rows, columns or schemas to organize data; instead, they use a **more flexible model.**

NoSQL databases are good for dealing with **big data**

With the rapid evolution in the nature of data, the needs of next-generation data storage and analysis, and requirements of intensive but flexible data analysis using distributed systems, cloud computing and high-performance computing (HPC), **traditional relational databases** are **unable** to **meet performance**, **scalability** and **flexibility** requirements.

Examples of unstructured but exponentially-growing data include **chat data**, messaging, large objects such as videos and images and many types of business documents. **[Big Data]**

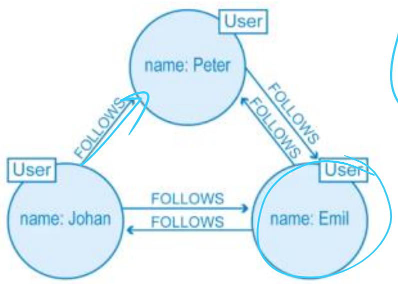
Types of NoSQL database
There are four main categories of NoSQL databases:

Graph databases
Graph databases are based on graph theory and utilize the concept of a graph to store, connect and query data.

In a graph database, **nodes** are equivalent to **rows** or records in the relational database and represent **entities** such as accounts, **people**, items etc.

Nodes are **linked** together by **edges**. Edges connect nodes and resemble the relational **relationship between tables**. Both nodes and edges can have properties associated with them.

A well-known example of a graph database is **neo4j**, used by Airbnb, Microsoft, IBM, eBay and Walmart.



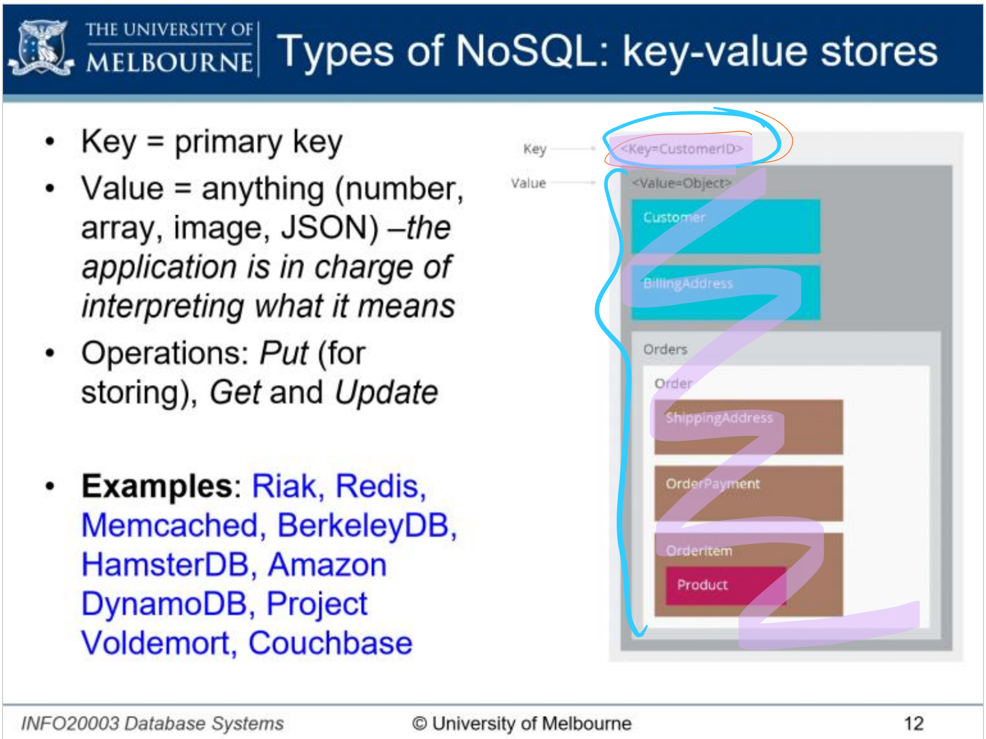
Graph databases are much faster at answering queries than relational databases (would require lots of self-joins)

Key-value stores
Key-value stores are the **most flexible** NoSQL databases, and also the **least structured,** using a simple key-value structure to organize data.

There is **no schema** and the data **values** can be of any **data type**.
Key: Value
Similar to a dictionary structure, the **key** should be a **unique** identifier to allow retrieval of the associated value. The key can theoretically be anything, but certain limitations can be imposed by the DBMS such as the key size and key type to achieve better performance. Usually keys are strings.

The **value** however can be **anything**, such as images, long text, videos, binary data, lists, JSON data, numbers, etc.

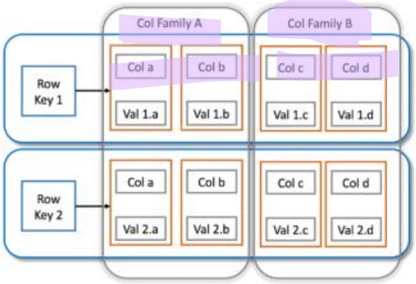
Examples of key-value stores include Berkeley DB, Aerospike and Redis. Key-value stores are highly flexible and support massive scalability.

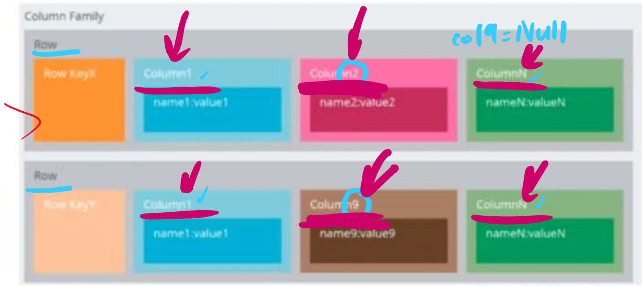


Column-family stores
Column-family stores, also referred to as wide-column stores or extensible record stores, are a type of key-value database.

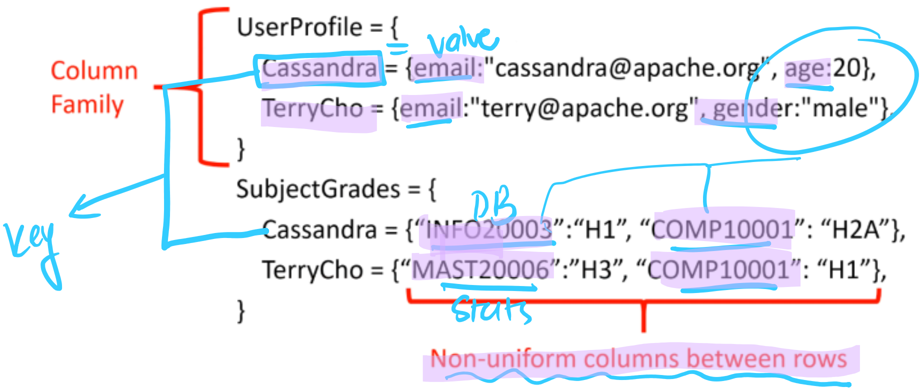
Column-family DBs, **like relational databases**, use tables, rows and columns but for each record the column names, their format and record keys can greatly **vary**, hence resulting in a **schema-free structure**. This enables organizations to store and query semi-structured data. Columns are created for each row instead of being predefined for a table. **(Each row can have a different set of columns associated with it)**

Cassandra and Hbase are two important NoSQL wide-column databases used by Facebook in the past to handle messaging and inbox search. Other enterprises using wide column stores are Netflix, Twitter and Reddit.





- Break up our Relational DB
- Can almost think of each Col Family as a relational db
  - but not as rigidly defined as relational db since
- Rows don't need to store the same columns!
  - e.g. Col 2 and 9
- Useful in lots of cases
  - Especially if you have lots of columns (big data scenario), you can split them up into belonging to different column families so that you know
    - e.g. Col9 is always in Col Family A
    - we only need to check through A and we don't need to look into any of the others



- age, gender diff columns
- Key (name) links their values across column families
- this is also why column-family stores are key-value databases

Document stores

Basically key-value databases, but the value = structured document

Document stores typically store data in **JSON**, XML or BSON documents (where JSON is by far the most dominant).

- But can also be other types of documents

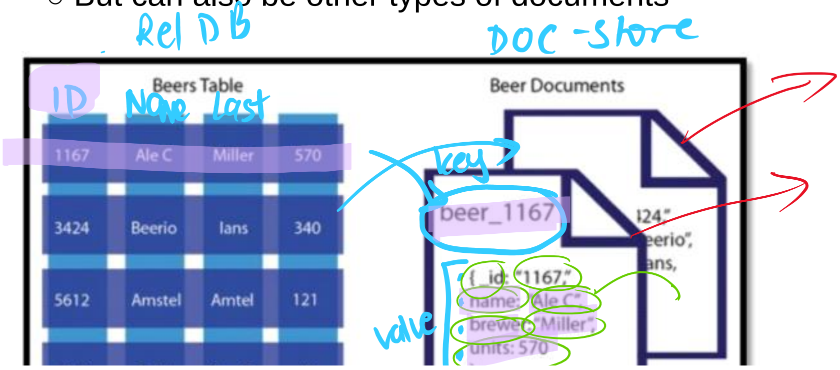


Figure S1: A comparison between a relational Beers table and a document store containing the same data. Adapted from

- each row becomes a separate document
- each document can have a different set of columns

}

Types of NoSQL: document databases

THE UNIVERSITY OF MELBOURNE

- Similar to a key-value store except that the document is "examinable" by the databases, so its *content* can be queried, and parts of it updated
- Document = JSON file
- **Examples:** MongoDB, CouchDB, Terrastore, OrientDB, RavenDB

<Key=CustomerID>

```
{
 "customerid": "fc986e48ca6"
 "customer":
 {
 "firstname": "Pramod",
 "lastname": "Sadalage",
 "company": "ThoughtWorks",
 "likes": [ "Biking","Photography" ]
 }
 "billingaddress":
 { "state": "AK",
   "city": "DILLINGHAM",
   "type": "R"
   }
}
```

INFO20003 Database Systems © University of Melbourne 13

The resulting documents are **independent components** which can be **distributed** more **easily**.

In addition, the storage **does not** require compliance with a **set schema**. Instead, each document can have its **own structure and schema,** allowing greater **agility** and **flexibility** as the business, website or app evolves.

Even though there is **no fixed schema**, it is still possible to create **indexes** within documents. If indexing features associated with document databases are used to their fullest, they can provide **fast** and **efficient querying** of data.

MongoDB is an example of a document store

Group work

Exercise:

Choosing a NoSQL database

Libraries store information about their collections in their catalogue.

Match each of the following statements to the type of NoSQL database that would be best for storing that library's data. Select from the four types of NoSQL database discussed previously.

col form

a. In one library, items are catalogued by author, title and publisher, as well as any number of other fields chosen by the cataloguer, such as physical description, subject codes and notes. doc - store

b. In another library, each catalogue record is stored in the MARC format (Figure 1), a coded text format that contains all the catalogue information for a particular item.

c. A public library wishes to store cover photos of all its items, which might be in JPEG, PNG or key-val

b. In another library, each catalogue record is stored in the MARC format (Figure 1), a coded text format that contains all the catalogue information for a particular ~~item.~~

c. A public library wishes to store cover photos of all its items, which might be in JPEG, PNG or PDF format, or stored as a URL. → key-val

d. A university library wishes to keep track of which published academic papers reference each other in order to help researchers measure their metrics. → graph

```
LEADER 00000nam  22000001  4500
008    730220s1955    ilu      b    00000 eng
019    55007351
050 0  QA276.5|b.R3
082    311.22
110 20 Rand Corporation.
245 12 A million random digits|bwith 100,000 normal deviates.
260 0  Glencoe, Ill.,|bFree Press|c[1955]
300    xxv, 400, 200 p.|c28 cm.
504    Bibliography: p. xxiv-xxv.
650  0 Numbers, Random.
984    |cMS T 519 R152
```

Figure 1: An example of a MARC record. MARC is a very old format that predates NoSQL, JSON and even XML by several decades, yet it remains the industry standard in library data systems.

Exercise:
Choosing a NoSQL database
Libraries store information about their collections in their catalogue.
Match each of the following statements to the type of NoSQL database that would be best for storing that library's data. Select from the four types of NoSQL database discussed previously.

*a. In one library, items are catalogued by author, title and publisher, as well as any number of other **fields** chosen by the cataloguer, such as physical description, subject codes and notes.*

A **column-family database** would be the best choice. Each row in a column-family table may have a different set of columns associated with it.

b. In another library, each catalogue record is stored in the MARC format (Figure 1), a coded text format that contains all the catalogue information for a particular item.

A **document store** would be best suited to this task. Normally, document stores use a modern data interchange format such as JSON or XML, but industry-specific structured data formats such as MARC can be used with specialised document store systems.

c. A public library wishes to store cover photos of all its items, which might be in JPEG, PNG or PDF format, or stored as a URL.

Key-value stores can store any kind of data. Each document in a document store should be made up of structured data – **images are not structured data** in the same way as JSON, so a document store is a poor choice.

d. A university library wishes to keep track of which published academic papers reference each other in order to help researchers measure their metrics.

By storing **papers** as **nodes** and **references** as **edges** joining the nodes, a graph database can efficiently capture, and answer complex queries about, the relationships between papers.

Key Concepts:

- Advantages of NoSQL
- The CAP theorem

<u>**Advantages of NoSQL**</u>

There are **four** key advantages offered by NoSQL databases as compared to relational databases:

- **Flexible modelling**
- **Scalability**
- **Performance** fast
- **High availability**

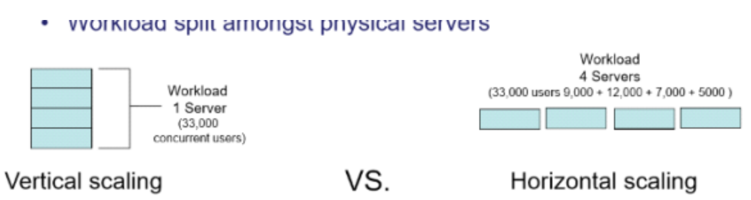
Flexible modelling

Instead of relying on a fixed schema, data types, row size and column names, NoSQL facilitates the implementation of **flexible data models**, making it more suited to coping with **less structured** data sources such as crowdsourced data (like big data sources e.g. chat messages, social media feeds etc).

Scalability

Capacity in a NoSQL database can be added and removed quickly using a **horizontal scale-out method** (adding inexpensive servers and connecting them to a database cluster).

As a result, the **cost** and complexity associated with **scaling up** a **relational** database into a **distributed database** are **avoided**.



Relational Databases

- Are not good with clustered servers. usually all on one server
- So you have to do **vertical scaling**
  - Buy more memory, add more computational power
  - "beef up" server
  - But this is expensive and we are also limited on how much we can beef it up

NoSQL Databases
- Run very well on distributed databases
- **Horizontal scaling** on DDs is very easy to do
- Can just add more machines, split the workload between them
- Very cheap and easy

Performance

By achieving seamless scalability using the horizontal scale-out method, the enterprises can manage efficient reads, writes and storage of the data items when handling big data.

distributed database

Companies like LinkedIn, Facebook and Google have users around the world; therefore, they deploy data centres in different parts of the world and **partition** their users so that all of their users experience the **fewest possible hops** by being routed to the **closest data centre**.

High performance = users can access databases very fast

High availability

With many businesses' customer and user engagement taking place mostly or entirely online, the availability of any application is a major concern for enterprises.

Constant availability (24/7) is a **challenge** for **relational** databases, since they are physically implemented on a **single server** or on a cluster with a shared storage.

In contrast, NoSQL databases are typically stored in **partitions** and they **divide data** across multiple database instances without any shared resources. The automatic failover means that **if nodes fail**, the **database can continue** its read and write operations on a different node.

The CAP theorem

The CAP theorem has three key components:
What are they?

Consistency

All the servers hosting the database will have the **same data**, so that anyone accessing the data will get the **same copy** regardless of which server is answering the query.

(This is a different thing to “consistency” in the context of the ACID principles, which refers to data integrity e.g. showing the correct bank acc balance.)

Availability

The system will **always respond** to a request even if it is **not** the **latest** data or **consistent** across the system.

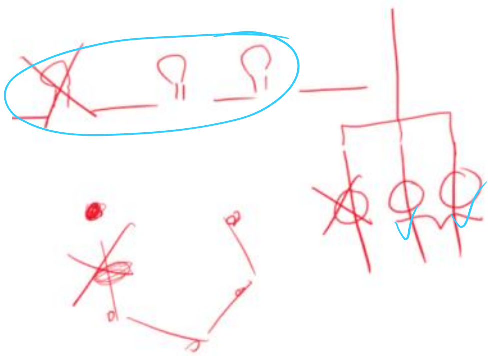
Partition tolerance

A “partition” in the context of the CAP theorem refers to a **disruption** in network access so that some servers are unable to access other servers. **The system continues to operate as a whole even if individual servers fail or can’t be reached.**

Fault Tolerance even if lost connection between nodes
('Automatic Failover' discussed previously)

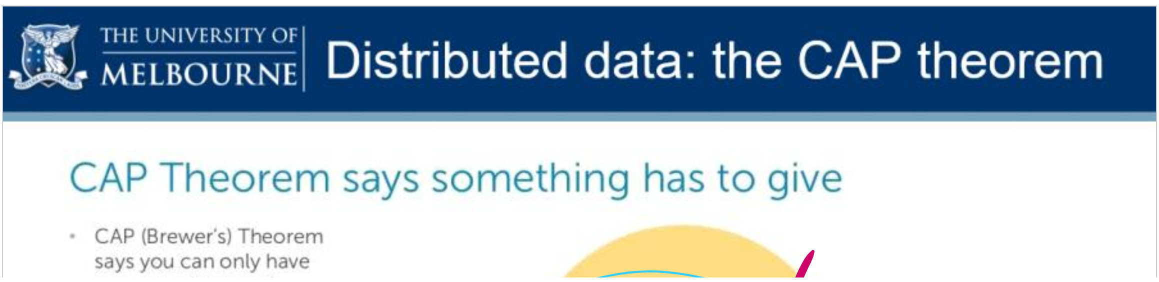
A partition is a communications break within a distributed system—a **lost** or temporarily delayed **connection** between two **nodes**.

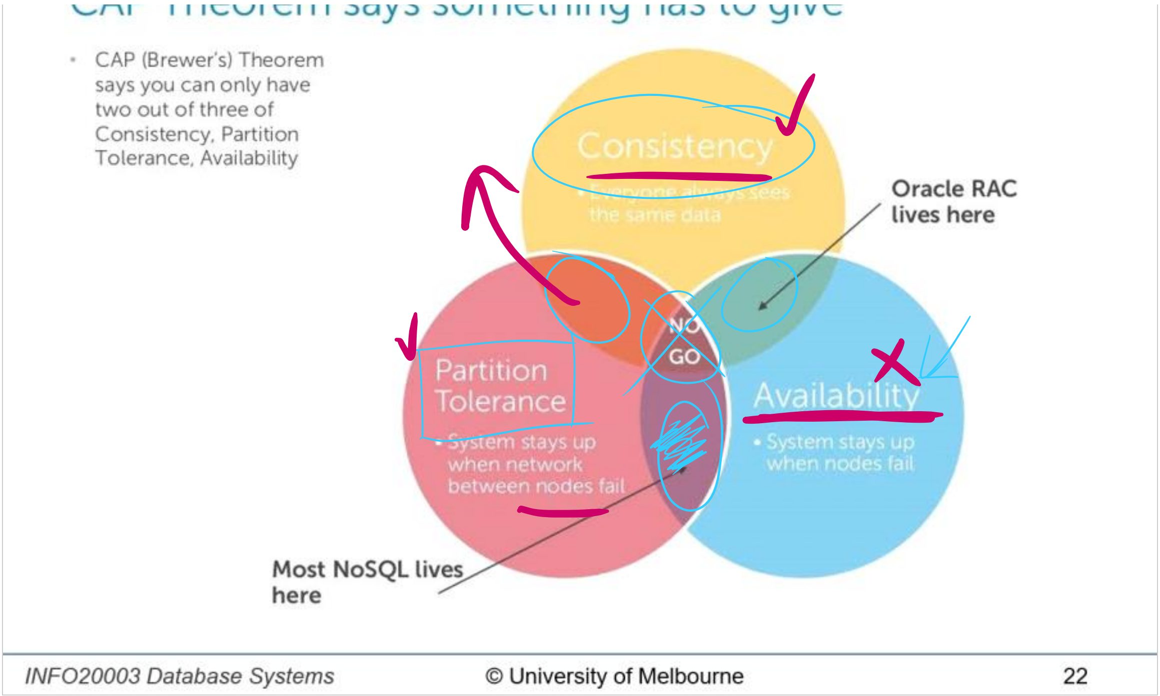
Partition tolerance means that the cluster must continue to **work despite any number of communication breakdowns between nodes in the system.**



Partition Tolerance - Analogy with Series vs Parallel Lightbulb
- What happens when a lightbulb fails in a
  - Series vs Parallel Circuit?
  - Does rest of circuit still continue to work?

The **CAP theorem** states that, at any given point in time, a system can achieve **two out of three** principles, while it is theoretically impossible to achieve three at the same time.





In case of **NoSQL** databases, the choice is between **AP** or **CP**, as the biggest **advantage** of **NoSQL** databases is partition tolerance when compared to relational DBMSs:

AP:
The database **always answers,** but possibly with **outdated** or wrong data, hence ensuring **availability** instead of consistency.

On systems that allow reads before updating all the nodes, high availability is achieved. Such systems eventually achieve consistency as well.

For example, Google and Facebook enforce **eventual consistency** such that different servers might have inconsistent views depending on how many servers are updated at a given time. Maybe the likes on a fb post are different depending on which part of the world you're in.

CP:
The database **stops all the operations until the latest copy of data is available on all nodes**. On systems that **lock** all the nodes before allowing reads, high consistency is achieved.

Such systems become available after the consistency is achieved.

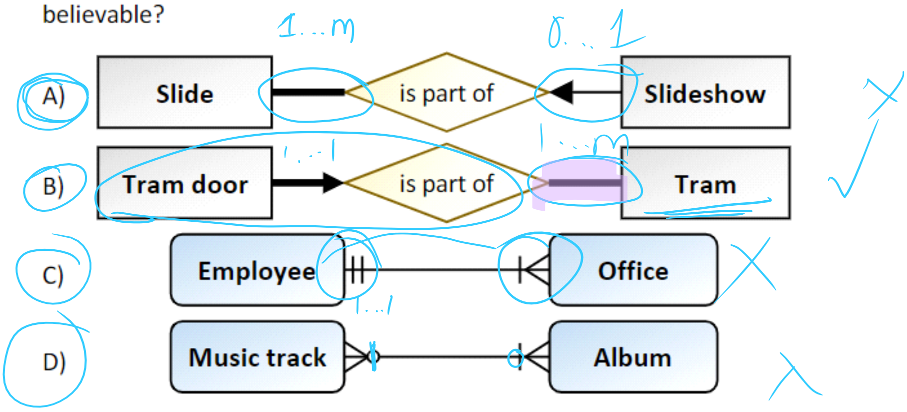
Most NoSQL databases choose **AP** over CP to ensure continuous availability and eventual consistency.

<u>Revision</u>

▾ Practice on your own

- Modelling Louis Vuitton Moet Hennessy
- Modelling Bridgman Art Library
- Modelling Parliament
- Modelling Swim school
- SQL: Golf Study
- Practice for exam (booklet)

2. Assuming that common sense applies, which **one** of the following ER model fragments is most believable?



$\pi(\sigma(\quad))$

Relational algebra

5. The following tables are part of a database for a role-playing game:

   player (<u>playerid</u>, playername, experience)
   playeritem (<u>playerid</u> FK, <u>itemid</u> FK, quantity)
   item (<u>itemid</u>, itemname, value, weight, colour)

   Write relational algebra expressions to answer the following questions:

   a. List all the possible experience levels of players.
   b. Show all information about gold-coloured items with a value of 20 coins.
   c. List the names of players who have bought a "Vine Knife".
   d. List the names and values of items bought by players with experience level "Adventurer" who are not named "coulter".
   e. Find names which are shared by at least one player and at least one item.

π_a - projection | SELECT ...

σ - selection | WHERE

$\bowtie$ _ NJ

a) $\pi_{experience}(player)$

$\pi_{itemname}(player \bowtie_{pname = item\ name} item)$

b) $\sigma_{val=20 \wedge col='gold'}(item)$

c π (σ (player $\bowtie$ player item $\bowtie$ item))

$c\ \pi_{pname}(\sigma_{itemname = 'vine...'}(player \bowtie playeritem \bowtie item))$

Compound Operator: Intersection

- In addition to the 5 basic operators, there are several additional "Compound Operators"
  - These add no computational power to the language, but are useful shorthands
  - Can be expressed solely with the basic operations
- **Intersection** retains rows that appear in *both* relations
- Intersection takes two input relations, which must be *union-compatible*
- Q: How to express it using basic operators?

$$R \cap S = R - (R - S)$$

Normalisation

13. Consider the following relation:

MealsOrdered (OrderID, CustomerID, CustomerName, DishID, DishName, DishPrice)

It has the following functional dependencies:

OrderID → CustomerID, CustomerName
CustomerID → CustomerName
DishID → DishName, DishPrice

a. Label these functional dependencies as partial, transitive or neither.
b. Normalise this relation to second normal form. Write down the functional dependencies that remain.
c. Normalise this relation to third normal form.

Indexes and query cost

9. Consider the following table:

   Employee (<u>EmployeeID</u>, Name, Age, DepartmentID)

   Suppose the following three queries are executed frequently on this table:

   i.
   ```
   SELECT Name
   FROM Employee
   WHERE Age > 30 AND DepartmentID = 5;
   ```
   ii.
   ```
   SELECT EmployeeID, Name
   FROM Employee
   WHERE Age = 65;
   ```
   iii.
   ```
   SELECT Name, Age
   FROM Employee
   WHERE DepartmentID = 7;
   ```

   Out of the above queries, which ones (if any) could potentially make use of:

   a. A clustered B-tree index on Age
   b. A hash index on Age
   c. A hash index on DepartmentID
   d. A clustered B-tree index on (DepartmentID, EmployeeID)
   e. A hash index on (EmployeeID, DepartmentID)
   f. An unclustered B-tree index on (DepartmentID, Age)

5. a. $\pi_{\text{experience}}(\text{player})$
   b. $\sigma_{\text{colour = 'gold'} \wedge \text{value = 20}}(\text{item})$
   c. $\pi_{\text{playername}}(\sigma_{\text{itemname = 'Vine Knife'}}(\text{player} \bowtie \text{playeritem} \bowtie \text{item}))$
   d. $\pi_{\text{itemname, value}}(\sigma_{\text{experience = 'Adventurer'} \wedge \text{name != 'coulter'}}(\text{player} \bowtie \text{playeritem} \bowtie \text{item}))$
   e. $\pi_{\text{playername}}(\text{player}) \cap \pi_{\text{itemname}}(\text{item})$

9. a. i, ii
   b. ii
   c. i, iii
   d. i, iii
   e. None
   f. i, iii