INFO20003 Exam notes

[**Diagram**](#_5agnkb71fxdn) **5**

[**Conceptual to logical**](#_6e4mp0a2rpsn) **8**

[**Relational algebra**](#_of5yfndzfq1j) **10**

[**SQL commands**](#_96kv6b28fgvx) **10**

[**Storage indexing**](#_q9c3ktfrrgv) **15**

[**Query calculations**](#_uu8hba8s7uvr) **17**

[**Normalisation**](#_k70vy6duo2iy) **19**

[**Transactions**](#_jikxk5z22go2) **23**

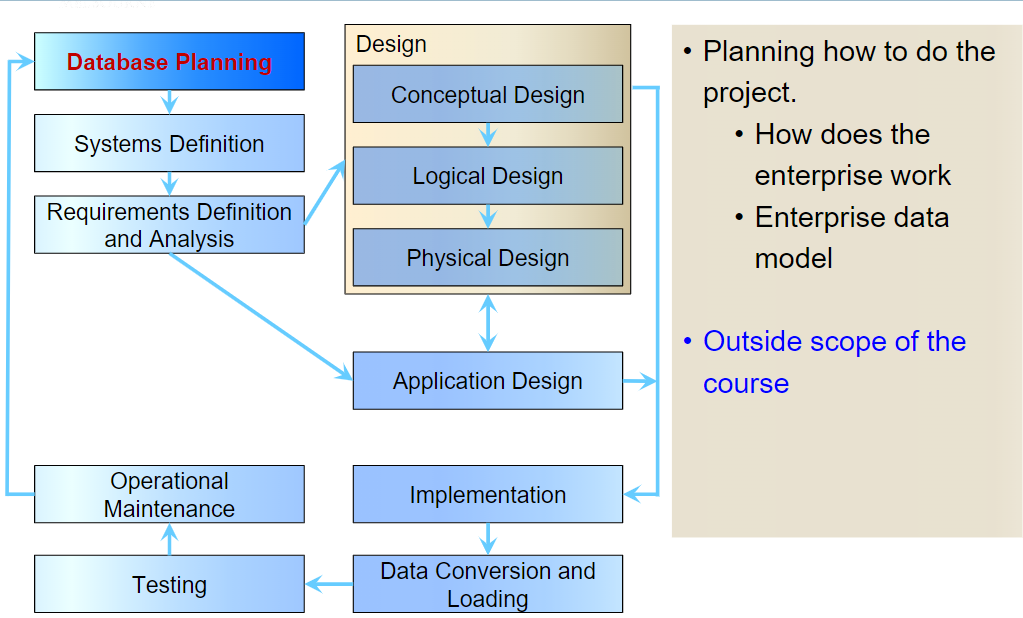
[**Types of failure**](#_jnlj8ajqfsyw) **25**

[**Types of backup**](#_u588mef5cess) **26**

[**Dimensional modelling/ Data warehousing**](#_q4l6c6hraedm) **30**

[**No SQL**](#_ij38ld1m1hkm) **31**

[**Exam prep**](#_s4bzjra9z8vc) **35**



**Advantage of Database vs file sharing system**

* Program - data independence (Changing one would change the other)
* No duplicate data/ Minimal data redundancy
* Improved data consistency - less storage space
* Improved data sharing
* Short development time
* Less program maintenance - data structure can change without changing the app

**Database planning**

Planning how to do the project.

How does the enterprise work

Enterprise data model

Eg: Customer, staff, account, deals

**Systems definition**

Specifying scope and boundaries

• Users

• Application areas

• How does the system interfere with other organizational systems

**Requirement Definition and Analysis**

Collection and analysis of requirements for the new system

Collection of client's requirements for the database, results in a set of 'business rules' which the database should fulfil

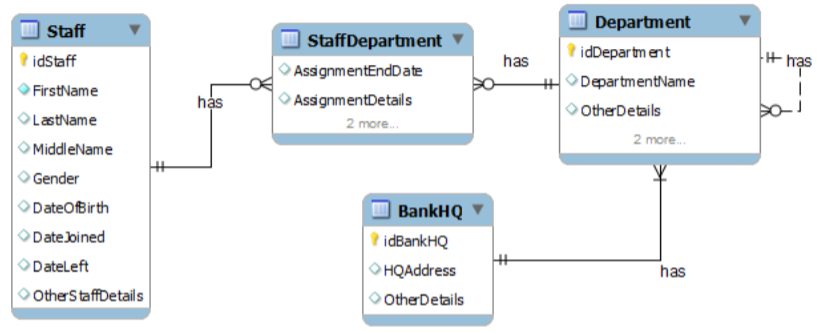
**Conceptual design**

• Construction of a model of the data used in the database – independent of all physical considerations

• Data Models

• ER Diagrams

• Model of the DATA and how data is related; the model is general, and no 'Database Model' has been applied

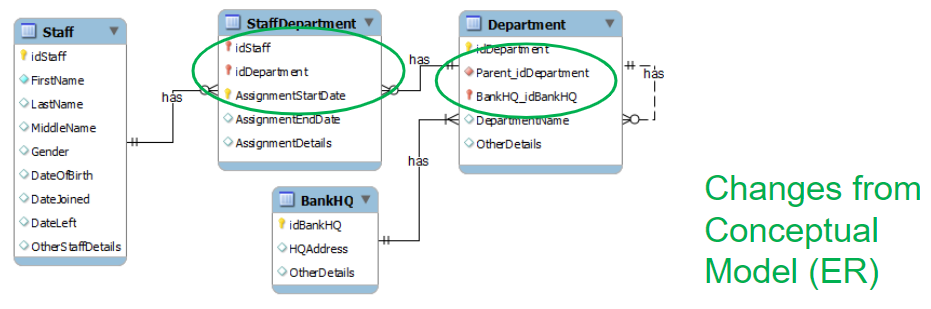


**Logical design**

• Construction of a (relational) model of the data based on the conceptual design

• Independent of a specific database and other physical considerations

• Model of how data will be stored, designed with a specific 'Database Model' in mind but NOT a specific database/DBMS



**Physical design**

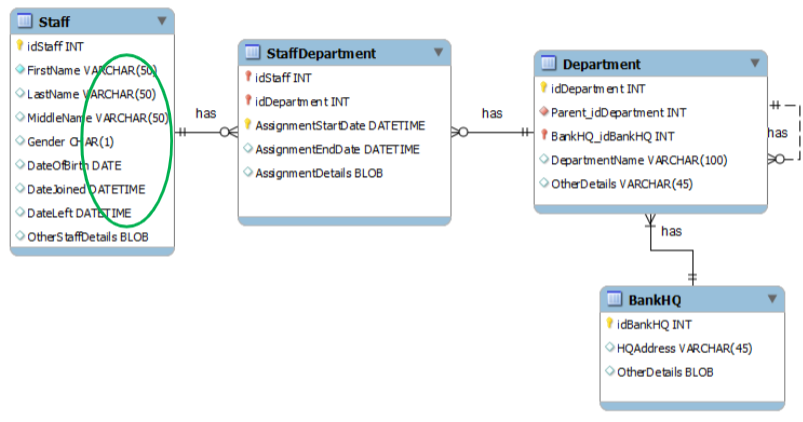
• A description of the implementation of the logical design – for a specific DBMS (Choose a DBMS (MySQL, MongoDB)

• Model containing implementation-level detail, targeted at a specific DBMS

• Describes:

• Basic relations (data types)

• File organisation

• Index

**Application design**

Done in conjunction with design

Design of the interface and application programs that use and process the database

**Implementation**

• The physical realisation of the database

• Implementation of the design

• Code which, when run in a specific DBMS, creates a functional database

**Data conversion and loading**

• Transfer existing data into the database

• Conversion from old systems

• Non trivial task

**Testing**

• Running the database to find errors in the design / setup (both at a physical level and at a logical level)

• Performance

• Robustness

• Recoverability

• Adaptability

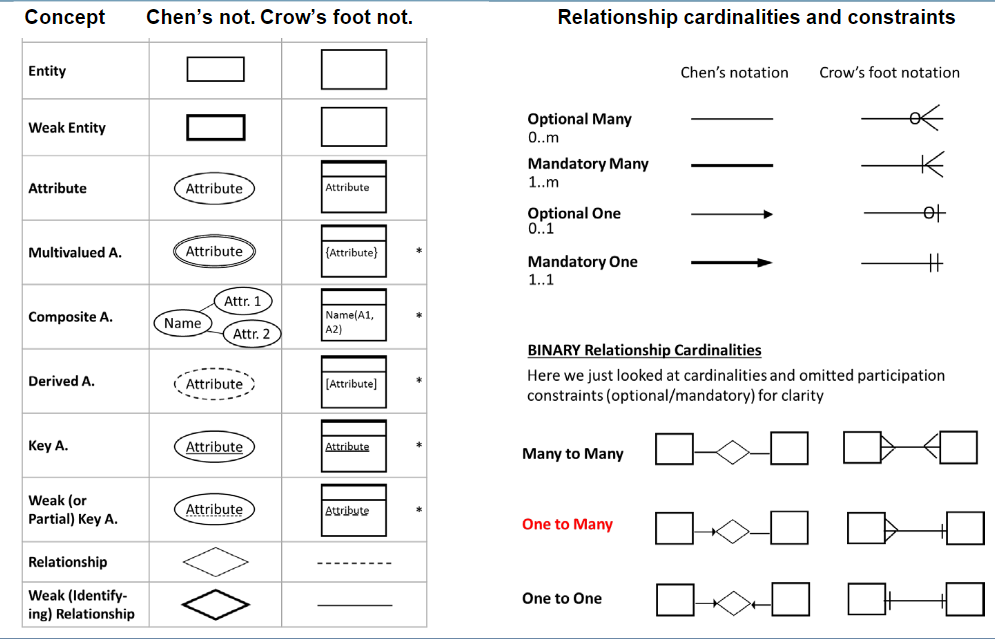
**Operational maintenance**

• The process of monitoring and maintaining the database system following its commissioning

• Handling new requirements

• Handling changes to requirements

## Diagram



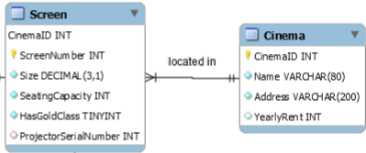
Multivalued attribute can be phone number, where a person can have multiple phone numbers

Remember for weak entities, there has to be a weak relationship and a weak key.

For every relationship including unary relationships, arrows always point to the relation



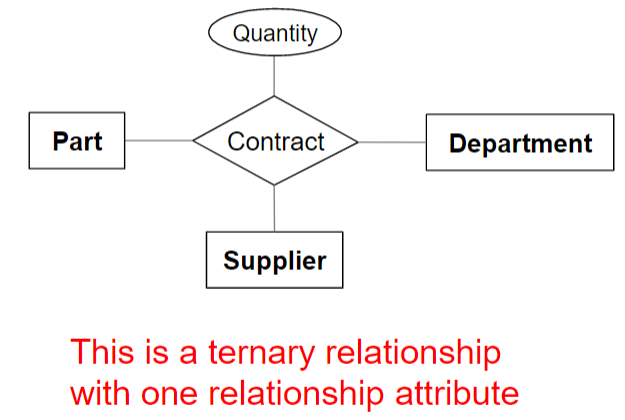
Screen can only belong to 1 cinema. Cinema has more than 1 screens



For chen’s notation, the arrow points towards the relationship.

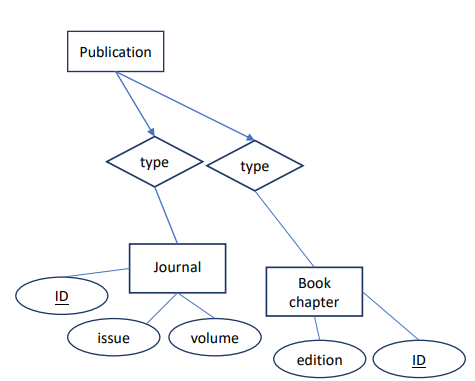
**Bold rectangle: weak entity** - cannot exist on its own. Must be drawn with a bold diamond and a bold arrow. Must have a weak key. Duplicate primary key over to the weak entity side

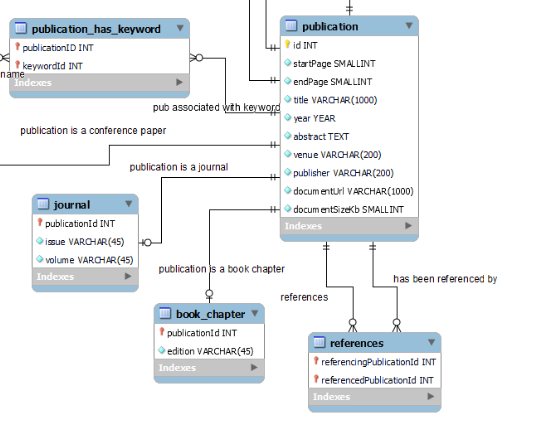
**Ternary relationship**

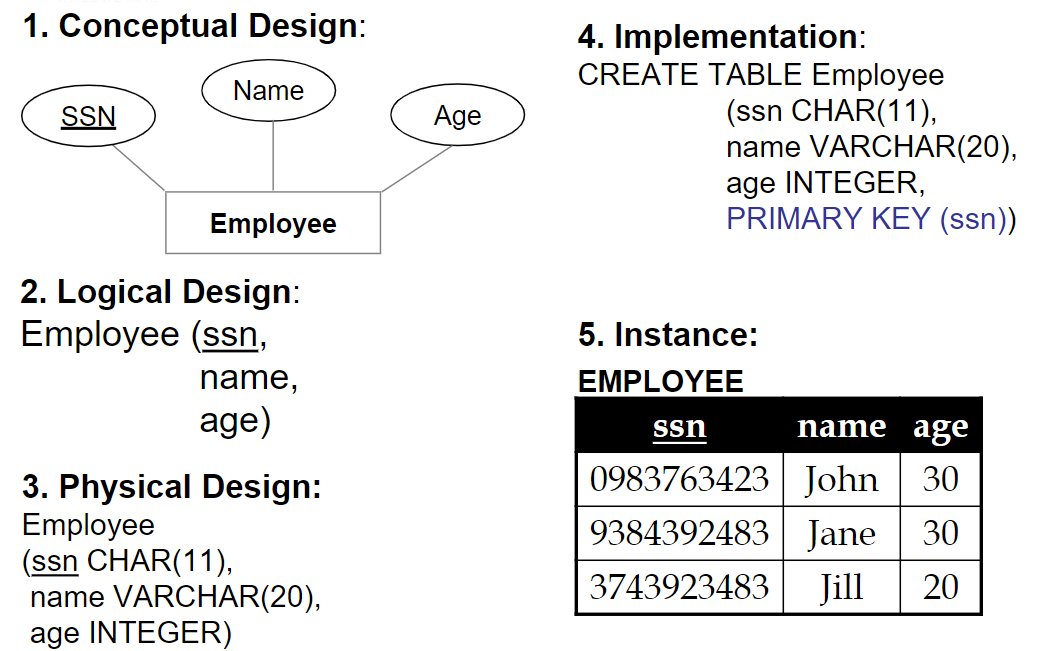


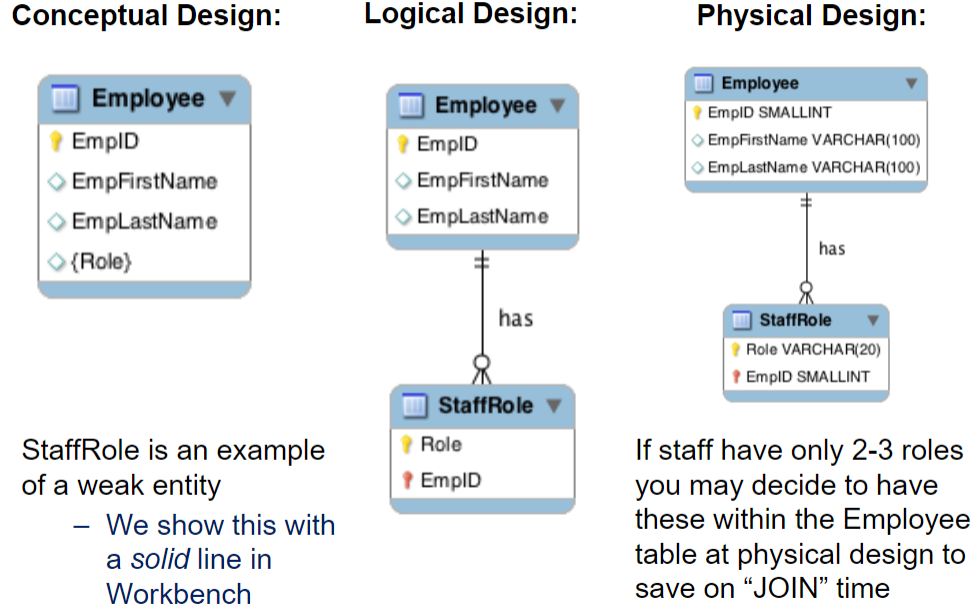
How to represent type/ enum entity:

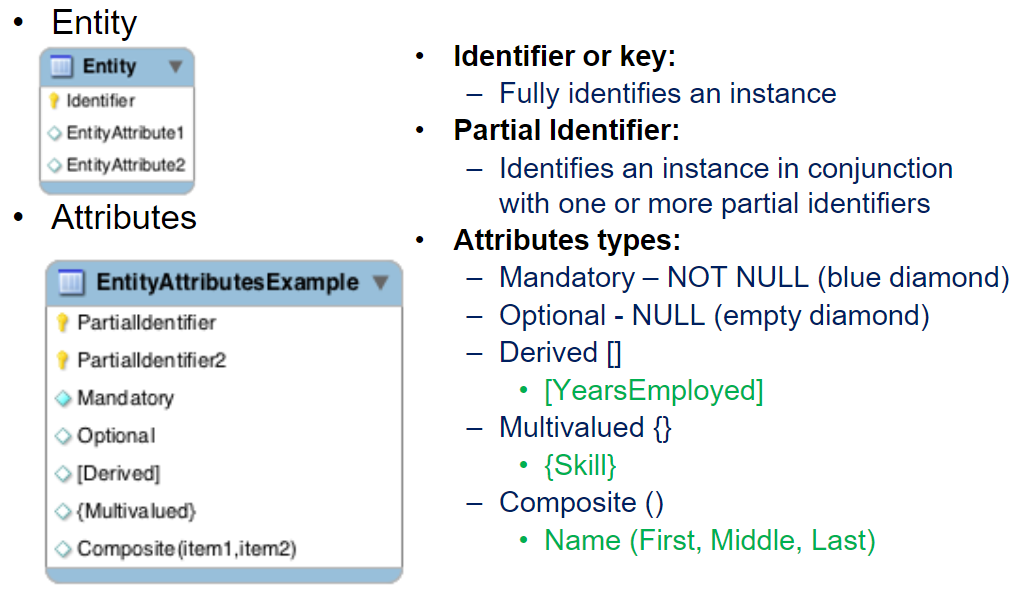
Below shows a lookup table. ID can be placed under publication otherwise.

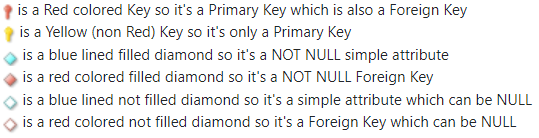












## Conceptual to logical

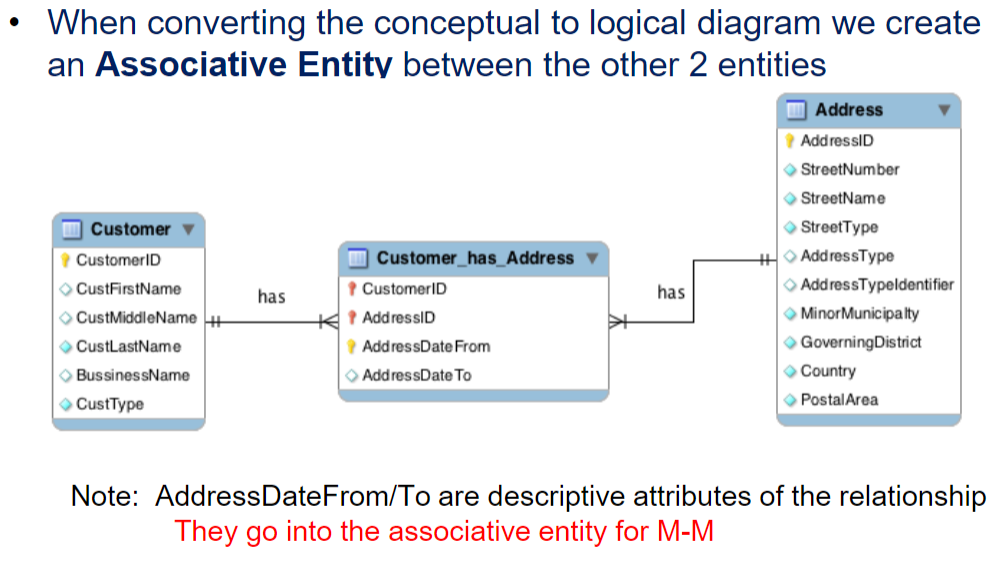
**Tasks checklist (from conceptual to logical):**

1. Flatten composite and multi-valued attributes

* Multi-value attributes can become another table
* 

If a customer has many phone numbers, they can store multiple phone numbers by having another table called phone numbers which has two columns, customer id and phone number.

2. Resolve many-many relationships

* Create an associative entity
* 

3. Resolve one-many relationships

* Primary key on the one side becomes a foreign key on the many side

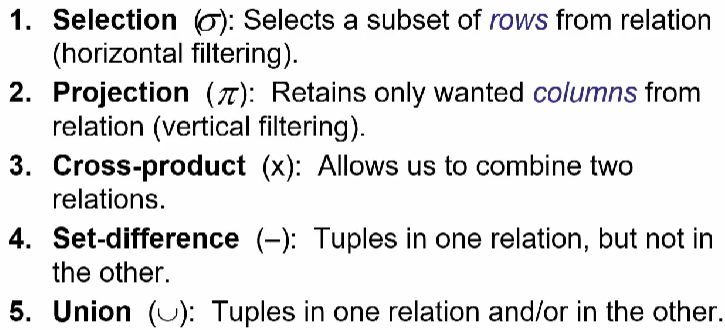
4. Resolve one-one relationships

* One-to-one relationships are resolved by adding a foreign key on either table, giving preference to the table that has mandatory participation in the relationship if there is only one.

5. Derived attributes disappear (Because they can be get using sql queries)

## Relational algebra

Basic operations:

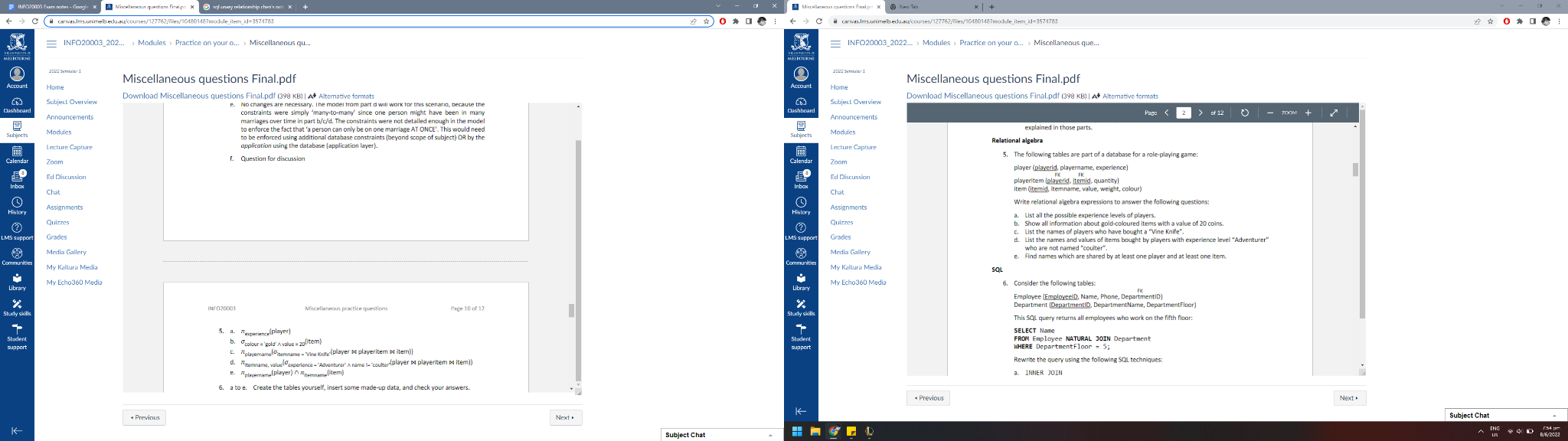
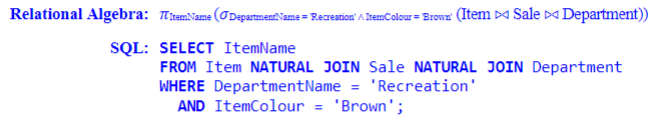


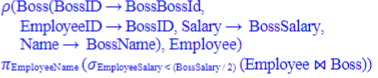
^ = and

v = or

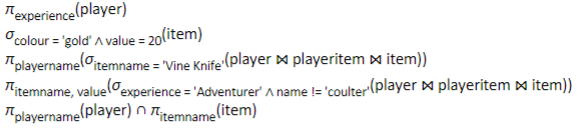
n = intersect

set difference: A-B is different that B-A. A-B means display rows that are in A but not in B. (Not symmetrical)



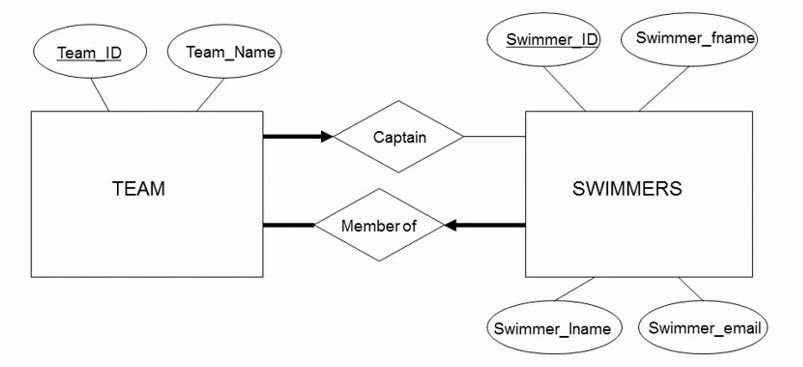


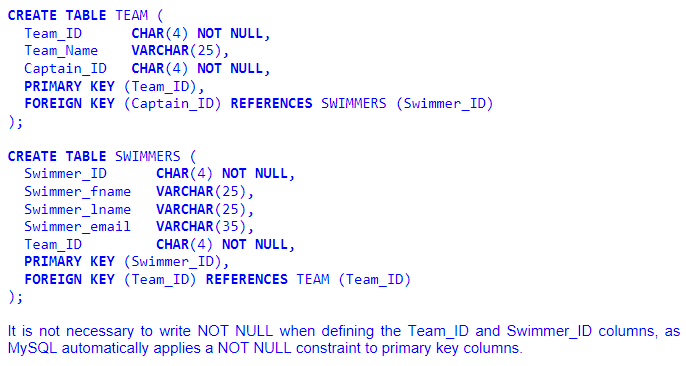
rename employee to boss



## SQL commands

**Create table**

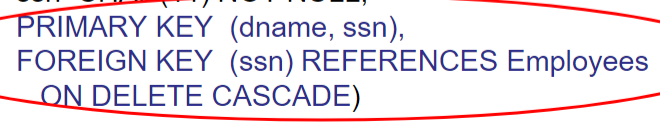




If it is a 0-1 or 1-1 relation, it has to be unique.

Primary key by default is unique.

If a 1-1 relation exists, we can put the referencing on that table. Otherwise, we have to create a new table.



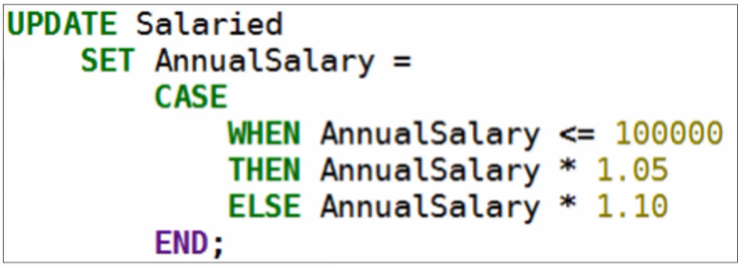
ON DELETE CASCADE needed for weak entities. It means remove the matching records from the child table when we delete the rows from the parent table.

Column2 ENUM ('value\_1','value\_2','value\_3'),

INSERT INTO Swimmers

(Swimmer\_id, swimmer\_fname, swimmer\_lname, swimmer\_email)

VALUES (1, “James”, “Hawkin”, NULL)



TRUNCATE

* same as delete but cannot roll back

DROP

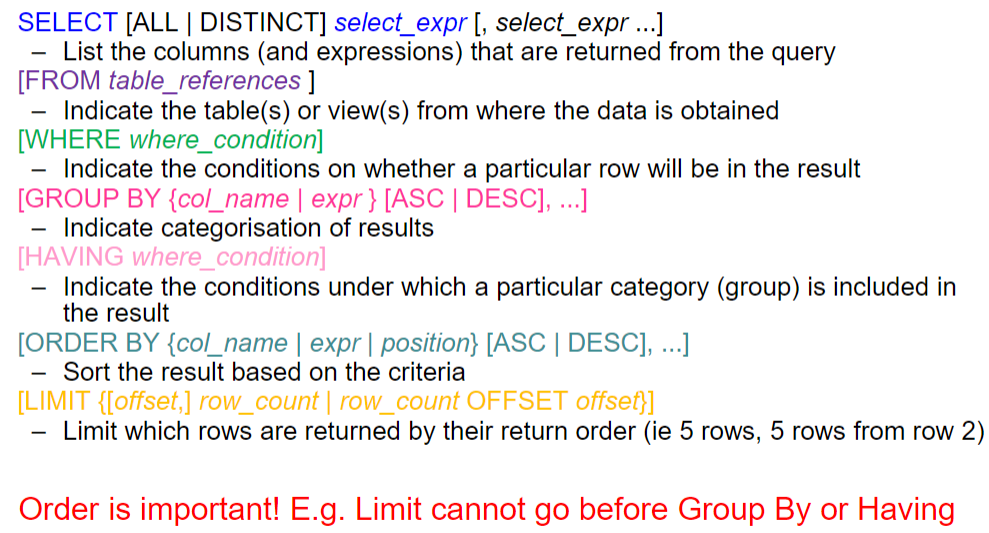
* Kills a relation - removes the data, removes the relation and no roll back

REPLACE

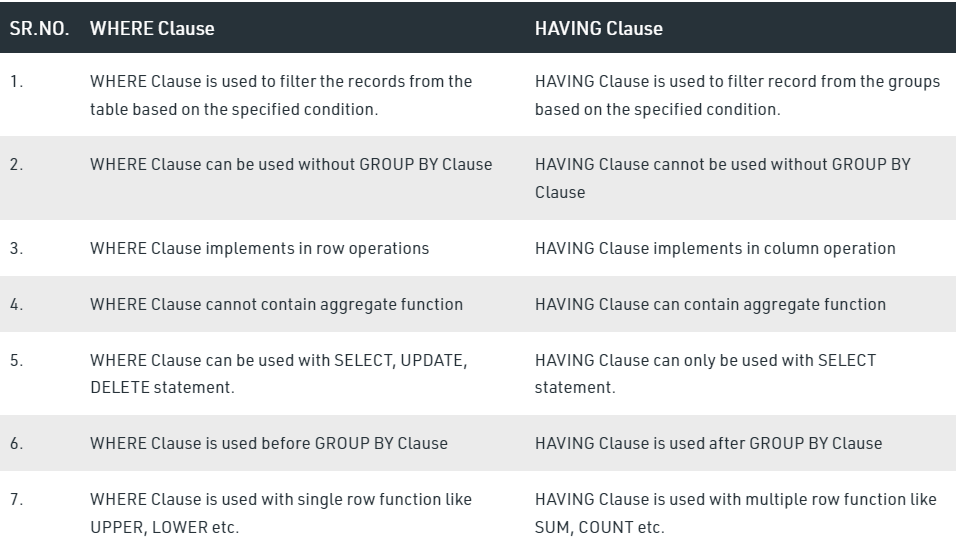
DELETE

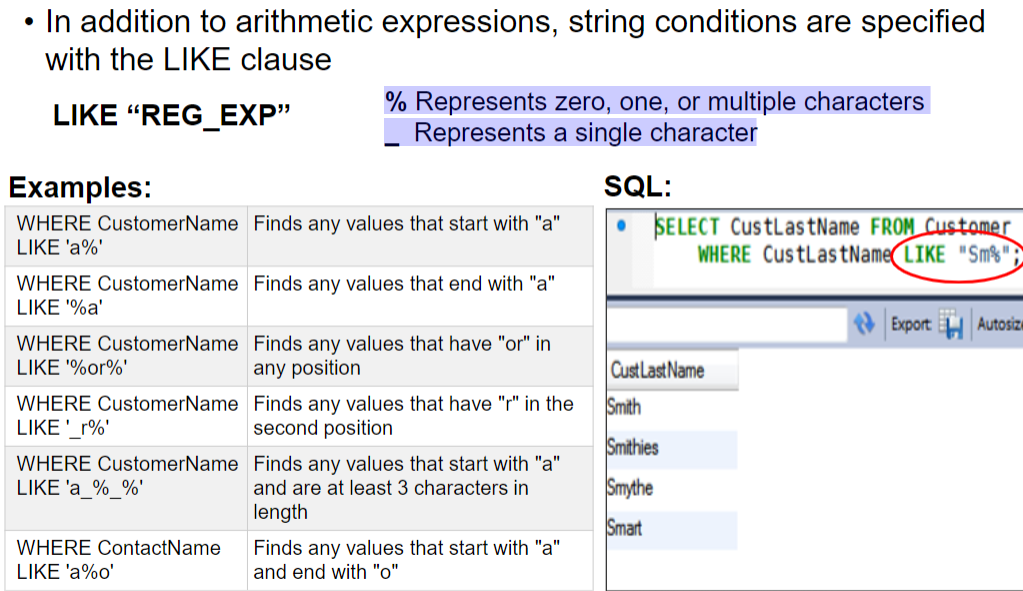
ALTER

RENAME



Difference between having and where





**Aggregate functions**

COUNT()

AVG()

SUM()

MAX()

MIN()

**Clause**

ORDER BY

GROUP BY

HAVING

LIMIT

OFFSET

SELECT \* FROM T1,T2 (cross product)

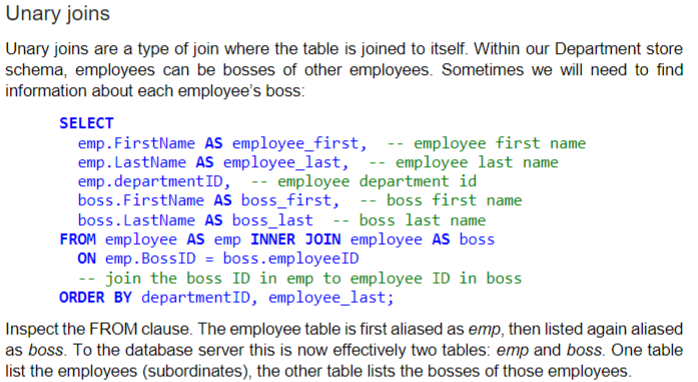
INNER JOIN

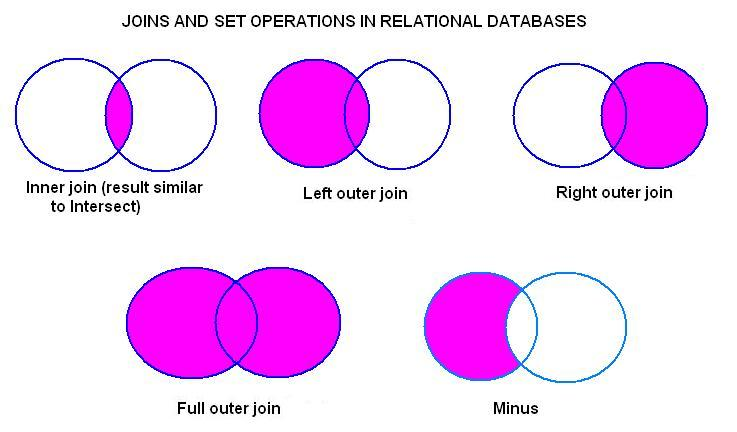
NATURAL JOIN (same column name)

LEFT/RIGHT/FULL OUTER JOIN (can combine to NATURAL LEFT JOIN)

UNION / UNION ALL - Use ALL if you want duplicate rows to be shown

UNARY JOIN





FROM Customers LEFT NATURAL JOIN Orders (Means that customers with no orders will be displayed)

**Subquery comparison**

IN/ NOT IN

ANY - must satisfy at least one of the inner conditions (any of)

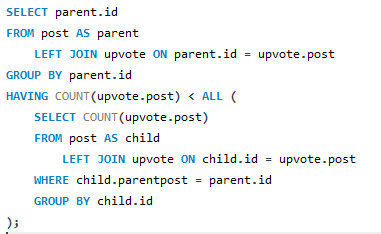
ALL - must satisfy all inner conditions

EXISTS - the inner query returns at least one record

DISTINCT - unique entries

IS NOT NULL/ IS NULL

If the subquery also uses an attribute that is also used in the outer query, you can rename the attributes to specify.



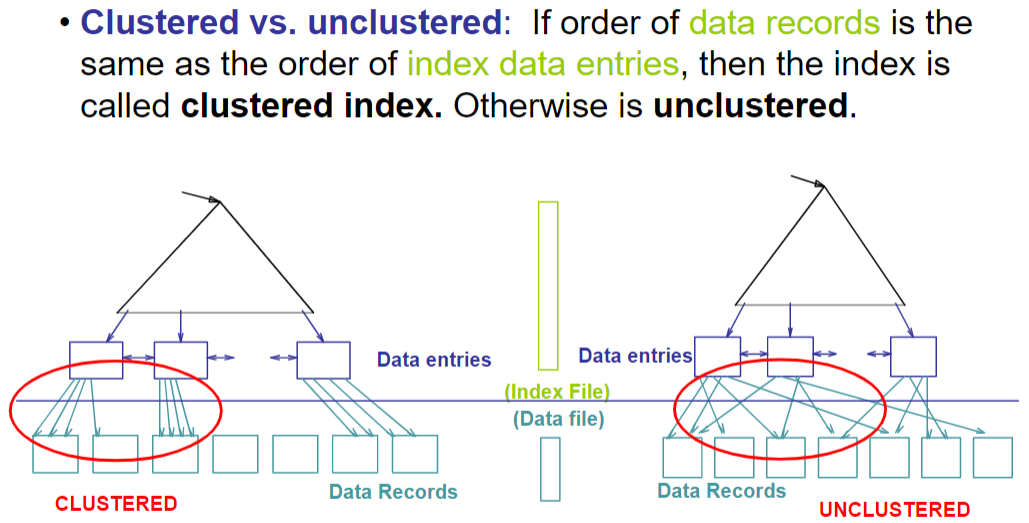
## Storage indexing

File organisation types

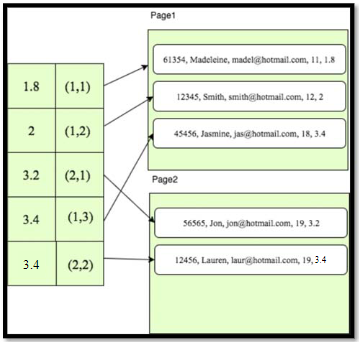
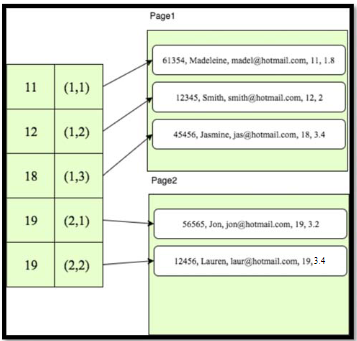
* File is a single table

| **Heap** | **Sorted** | **Index** |
| --- | --- | --- |
| Fastest insert  Suitable to retrieve all records | Best for retrieval (of a range of records) in some order | Good for selection on search key fields  eg: WHERE A = 5 AND B > 9 |
| No order | Bad for maintenance |  |

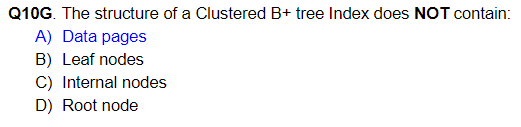
There are two types of index, clustered (in the same order as data entries) and unclustered.



**Clustered (index on age): Unclustered (index on gpa):**



(a,b) where a is page and b is row number



Cost of retrieving data records through index varies greatly based on whether index is clustered (cheaper for clustered)

Clustered indexes are more expensive to maintain (require file reorganization), but are really efficient for range search

(Approximated) cost of retrieving records found in range scan:

1. Clustered: cost ≈ # pages in data file with matching records

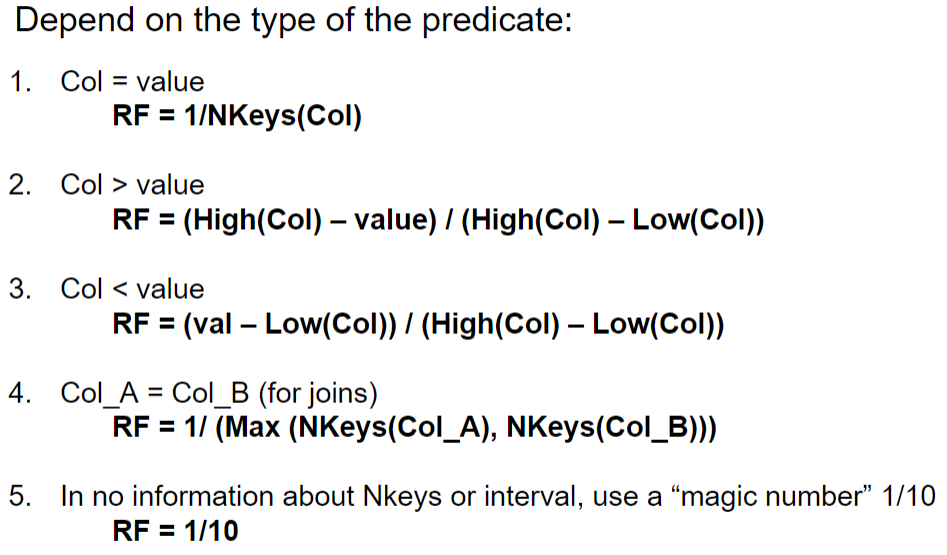
2. Unclustered: cost ≈ # of matching index data entries (data records)

Can be indexed as a hash map, binary tree. **trees are better for range. Hash better for equality**. **Range queries cannot be applied to a hash index**.

trees also give the ability to do an index only scan where all the required info is at index level. That way, we do not have to enter data level (expensive) at all.

## Query calculations

**Calculating reduction factor**

Basically probability. Easier not to refer to this table.

RF(E.id=D.id) = 1/Number of distinct keys (smaller number)

Finding expected number of tuples = multiple of tuples in each table \* all reduction factor

**Result size**

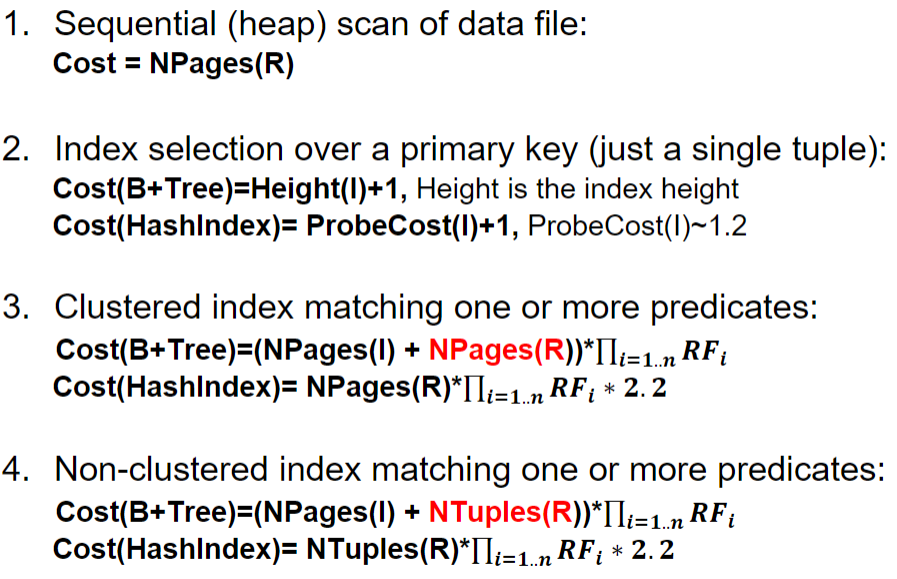
Single table selection

* NTuples \* reduction factor

Joins

* Size of tuples = NTuple (Inner) \* NTuple (Outer) \* 1/maximum number of distinct keys in either tables (usually the smaller value. If there is a projection, it is smaller number when comparing the original tuple size before projection vs new join)

**Indexing cost/ index scan**



5. if doing selection on candidate key using hash index, (from tut 8)

1.2 is for bucket check and 1 to fetch page from the disk. 2.2 per tuple

**Cost = hash lookup cost + 1 datapage access = 1.2+1= 2.2**

**Joins**

Tuple oriented nested loop join

* **Cost = Npages (outer) + Ntuples (outer) \* NPages (Inner)**

Page Oriented Nested loops join

* min buffer pages needed = 3
* **Cost = Npages (outer) + Npages (Outer)\*Npages (Inner)**

Block nested loops join

* min buffer pages needed = B (buffer blocks available)
* Cost: NPages (Outer) + NBlocks(Outer) \* NPages(Inner)
* NBlocks(Outer) = Npages (outer) / (B-2)
* **Cost = Npages(outer) + (Npages (outer) / (B-2)) \* Npages(inner)**

Sort-merge join

* Cost (SMJ) = Sort(Outer) + Sort(Inner) + NPages(Outer) + NPages(Inner)
* Sort(R) = External Sort Cost = 2\*NumPasses\*NPages(R)
* Check if it is sorted: clustered index scan on joining attribute.
* **Cost = Npages(r) + Npages(s) + 2\*Npages(r)\*NumPasses(r) + 2\*Npages(s)\*num\_passes(s)**
  + NumPasses = given in question

Hash join (Hash the keys into subgroups)

* **Cost = 2 \* NPages(Outer) + 2\* NPages(Inner) + NPages(Outer) + NPages(Inner)**

Remove the scanned portion if it is the second level of join.

If there is a clustered B+ index on the same attribute, then it is already sorted.

**Other**

What is the lowest possible I/O cost be for joining R and S using any join algorithm, and

how much buffer space would be needed to achieve this cost:

Lowest I/O = NPages(Outer) + NPages(Inner)

How much space is needed: Minimum number of pages in either tables + 1 (other table input page) + 1 (space for output)

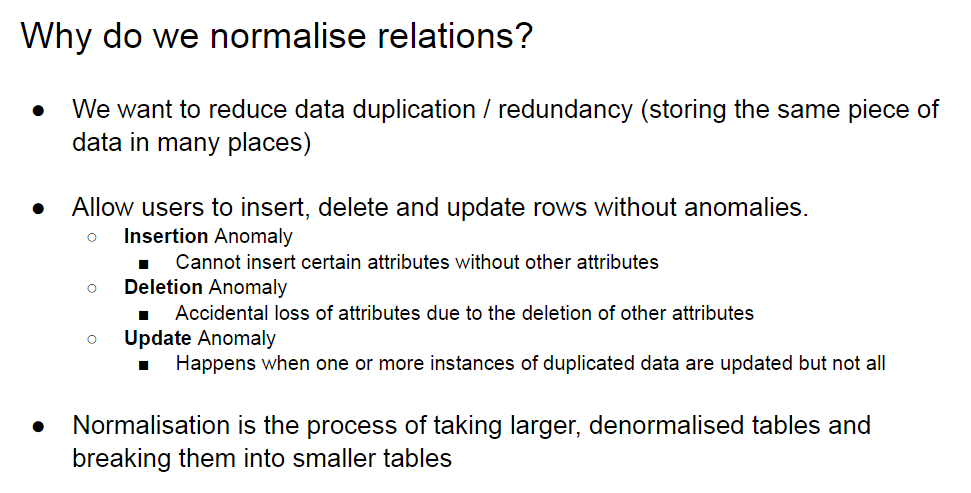
**Why we use I/O to calculate**

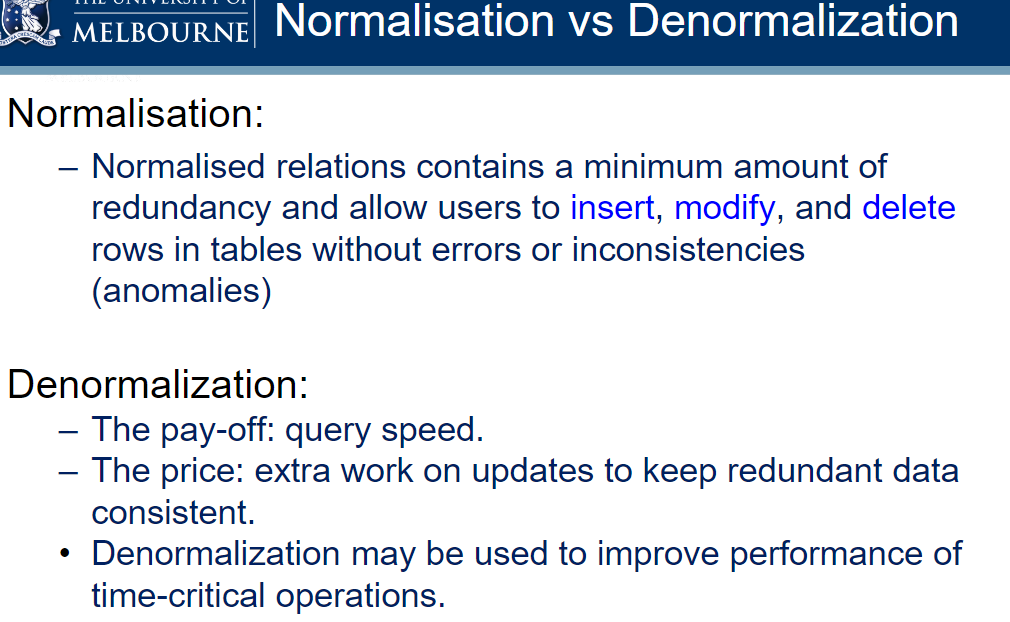
In comparing the cost of different algorithms, we count I/O (page accesses) and ignore

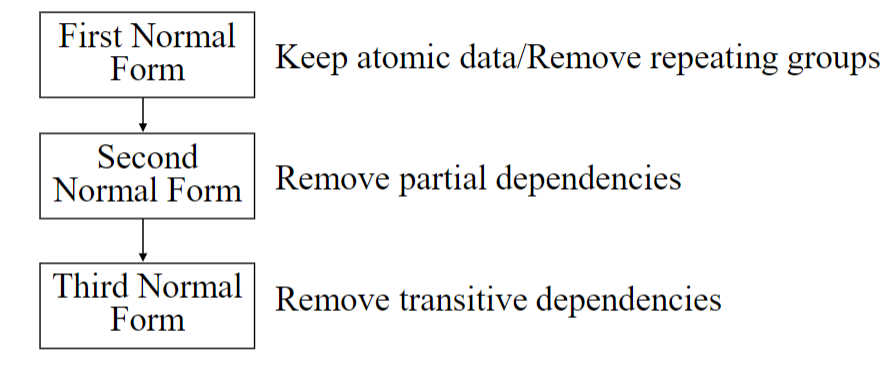
all other costs. What is the reason behind this approach?

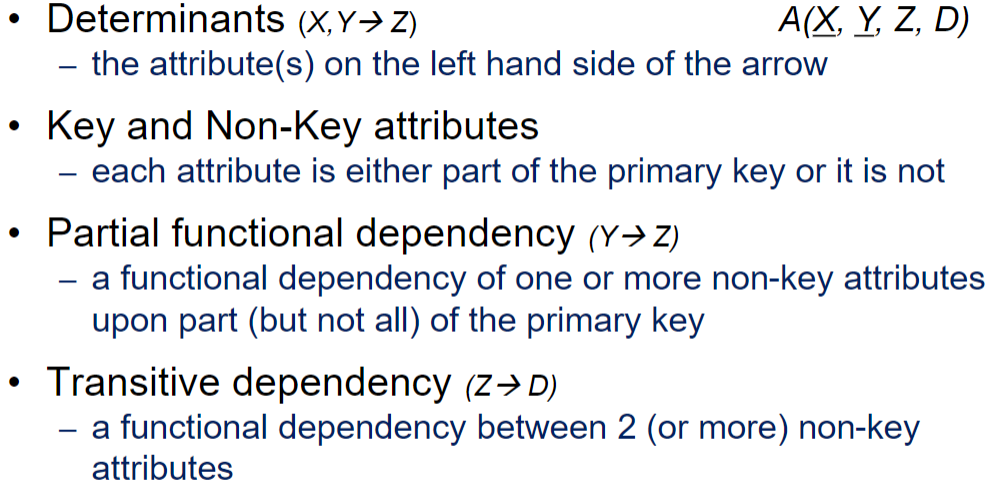
Because the I/O cost dominates the overall cost. One I/O corresponds to millions of CPU cycles.

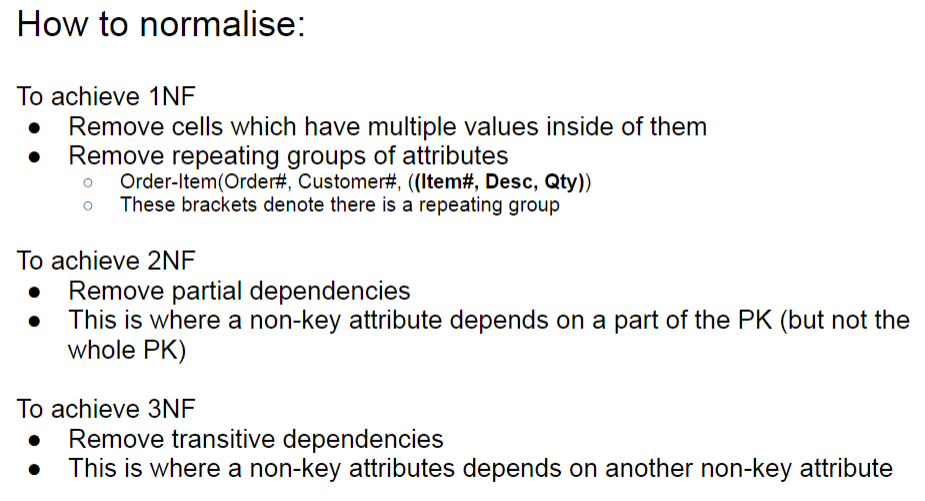
## Normalisation











First normal form

Remove Repeating Groups

* repeating groups of attributes cannot be represented in a flat, two dimensional table
* removing cells with multiple values (keep atomic data)
* Multiple values mean having two entries in one column. eg: location: sg, au
* After doing so, it will be in 1NF

Because it is multi dimensional, we have to flatten it. Items and customer is diff table

Order-Item (Order#, Customer#, (Item#, Desc, Qty))

becomes

Order-Item (Order#, Item#, Desc, Qty)

and

Order (Order#, Customer#)

Just remove only the repeated value and the candidate keys for it. Remove the multiple value from the main table.

Eg:

status is repeating

Year, TranNum → Type, Customer, BSB, AccNum, Status

report(tranNum, type, customer, bank, status, BSB, AccNum, year)

becomes

status (tranNum, Year, status)

report(tranNum, type, customer, bank, BSB, AccNum, year)

Second normal form

Remove Partial Dependencies

* a non-key attribute cannot be identified by part of a composite key
* After doing so, it will be in 2NF

Because Desc is solely dependent on one underlined (Item#), which is a key (underlined) we split

order, item -> qty, comment

item -> desc

**Order-Item (Order#, Item#, Desc, Qty, comment)**

becomes

**Item (Item#, Desc)**

and

**Order-Item (Order#, Item#, Qty, comment)**

Third normal form

Remove Transitive Dependencies

* a non-key attribute cannot be identified by another non-key attribute
* After doing so, it will be in 3NF

Department Id (non key attribute) determines department name.

Employee (Emp#, Ename, Dept#, Dname)

becomes

Employee (Emp#, Ename, Dept#)

and

Department(Dept#, Dname)

Remember to add new foriegn keys whenever appropriate.

## Transactions

**ACID**

• 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

• Consistency

* Constraints that hold before a transaction must also hold after it
* multiple users accessing the same data see the same value

• Isolation

* Changes made during execution of a transaction cannot be seen by other transactions until this one is completed

• Durability

* When a transaction is complete, the changes made to the database are permanent, even if the system fails

Transactions solve TWO problems:

1. users need the ability to define a unit of work

2. concurrent access to data by >1 user or program

**Problem 1: Transactions as a unit of work**

• Transaction needs to be treated as an indivisible unit of work

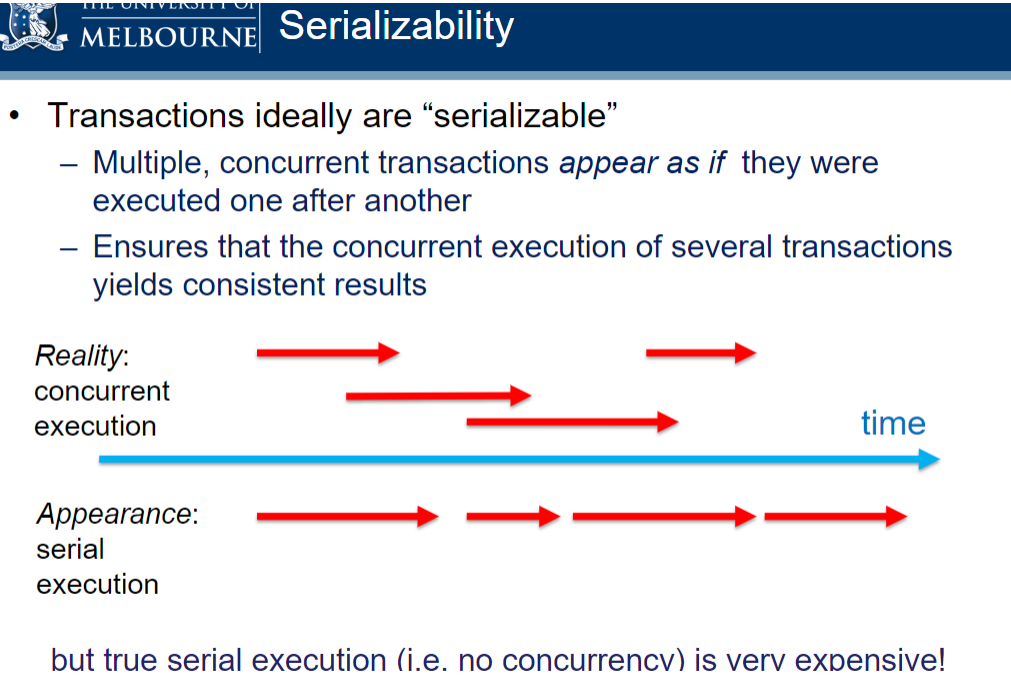
• “Indivisible” means that either the whole job gets done, or none gets done: if an error occurs, we don’t leave the database with the job half done, in an inconsistent state

**Problem 2: Concurrent access**

Multiple users at one time

What could go wrong?

* The lost update problem: two users updating data updates of initial value instead of updated value from the other user.
* Uncommitted data problem: Uncommitted data occurs when two transactions execute concurrently and the first is rolled back after the second has already accessed the uncommitted data
* Inconsistent retrieval: Occurs when one transaction calculates some aggregate functions over a set of data, while other transactions are updating the data. Isolation is responsible



Methods to achieve concurrency:

**Locking:**

Guarantees exclusive use of a data item to a current transaction

Only one user is granted access at any given point of time.

Can be:

* database level lock (sqlite)
* table level lock
* page level lock
* row level lock (mysql)
* field level lock: Allows 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

Eliminates lost update problem

Types of locks:

* Binary (locked or unlocked/ 1 or 0) Considered too restrictive to yield optimal concurrency, as it locks even for two READs (when no update is being done)
* Shared lock:

– Access is reserved for the transaction that locked the object

– Must be used when transaction intends to WRITE

– Granted if and only if no other locks are held on the data item

– In MySQL: “select ... for update”

* Exclusive lock

– Other transactions are also granted Read access

– Issued when a transaction wants to READ data, and no Exclusive lock is held on that data item

• Multiple transactions can each have a shared lock on the same data item if they are all just reading it

– In MySQL: “select ... for share”

**Time stamping:**

– Assigns a global unique timestamp to each transaction

– Each data item accessed by the transaction gets the timestamp

– Thus for every data item, the DBMS knows which transaction

performed the last read or write on it

– When a transaction wants to read or write, the DBMS compares

its timestamp with the timestamps already attached to the item

and decides whether to allow access

**Optimistic concurrency control:**  
– Based on the assumption that the majority of database

operations do not conflict

– Transaction is executed without restrictions or checking

– Then when it is ready to commit, the DBMS checks whether any

of the data it read has been altered – if so, rollback

## Types of failure

• Statement failure

– Syntactically incorrect

• User Process failure

– The process doing the work fails (errors, dies)

• Network failure

– Network failure between the user and the database

• User error

– User accidentally drops the rows, table, database

• Memory failure

– Memory fails, becomes corrupt

• Media Failure

– Disk failure, corruption, deletion

## Types of backup

• Physical vs Logical

• Online vs Offline

• Full vs Incremental

• Onsite vs Offsite

**Logical backups can only be done online**

Physical:

– raw copies of files and directories

– suitable for large databases that need fast recovery

– database is preferably offline (“cold” backup) when backup occurs

• MySQL Enterprise automatically handles file locking,

so database is not wholly off line

– backup = exact copies of the database directories and files

– backup should include logs

– backup is only portable to machines with a similar configuration

– to restore

• shut down DBMS

• copy backup over current structure on disk

• restart DBMS

Logical:

– Logical backup can only be done online

– backup completed through SQL queries

– slower than physical

• SQL Selects rather than OS copy

– output is larger than physical

– doesn’t include log or config files

– machine independent

– server is available during the backup

– in MySQL can use the backup using

• Mysqldump

• SELECT ... INTO OUTFILE

– to restore

• Use mysqlimport, or LOAD DATA INFILE within the mysql client

Online (or HOT) backup

– backups occur when the database is “live”

– clients don’t realise a backup is in progress

– need to have appropriate locking to ensure integrity of data

Offline (or COLD) backup

– backups occur when the database is stopped

– to maximize availability to users take backup from replication server not live server

– simpler to perform

– cold backup is preferable, but not available in all situations

e.g. applications without downtime

Full

– a full backup is where the complete database is backed up

• may be Physical or Logical, Online or Offline

– it includes everything you need to get the database operational in the event of a failure

Incremental

– only the changes since the last backup are backed up

– for most databases this means only backup log files

– to restore:

• stop the database, copy backed up log files to disk

• start the database and tell it to redo the log files

Offsite

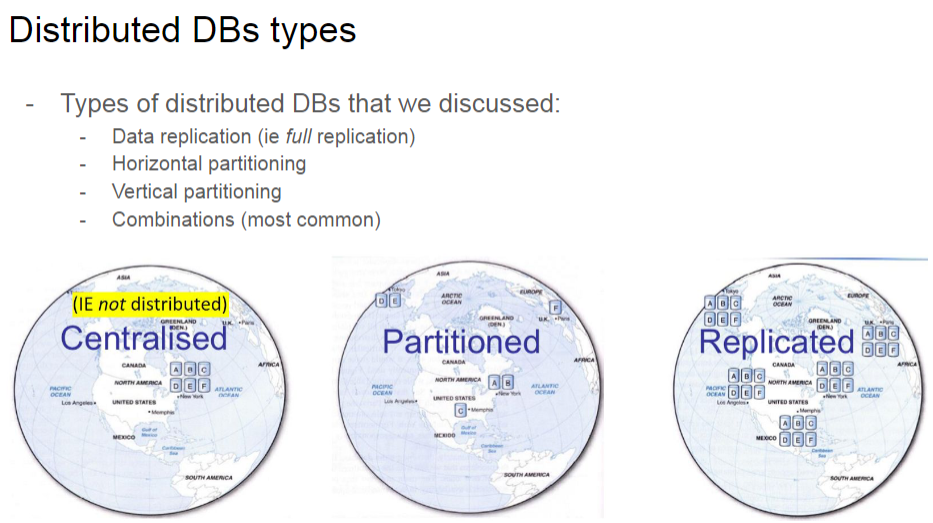
• Enables disaster recovery (because backup is not physically near the disaster site)

• Example solutions:

• backup tapes transported to underground vault

• remote mirror database maintained via replication

• backup to Cloud (see figure below)



Distributed Database (replicated is a form of distributed database)

• a single logical database physically spread across multiple computers in multiple locations that are connected by a data communications link

• appears to users as though it is one database

Decentralised Database

• a collection of independent databases which are not networked together as one logical database

• appears to users as though many databases

**Decentralised Advantages:**

• Good fit for geographically distributed organisations / users

• Utilise the internet

• Data located near site with greatest demand

• Faster data access (to local data)

• Faster data processing

• Workload split amongst physical servers

• Allows modular growth

• add new servers as load increases (**horizontal scalability**)

• Increased reliability and availability

• less danger of a single-point of failure (SPOF), IF data is replicated

• Supports database recovery

• When data is replicated across multiple sites

**Decentralised Disadvantages:**

• Complexity of management and control

• Database or/and application must stitch together data across sites

• Who and where is the current version of the record (row & column)?

• Who is waiting to update that information and where are they?

• How does the logic display this to the web & application server?

• Data integrity

• If two users in two locations update the record at the exact same

time who decides which statement should “win”?

• Solution: Transaction Manager or Master-slave design

• Security

• Many server sites -> higher chance of breach

• Multiple access sites require protection including network and

storage infrastructure from both cyber & physical attacks

• Lack of standards

• Increased training & maintenance costs

• Increased storage requirements

• Replication model

**Replicated advantages:**

• High reliability due to redundant copies of data

• Fast access to data at the location where it is most accessed

• May avoid complicated distributed integrity routines

• Replicated data is refreshed at scheduled intervals

• Decoupled nodes don’t affect data availability

• Transactions proceed even if some nodes are down

• Reduced network traffic at prime time

• If updates can be delayed

• This is currently popular as a way of achieving high availability for global systems

• Most SQL & NoSQL databases offer replication

**Replicated disadvantages:**

• Need more storage space

• Each server stores a copy of the row

• Data Integrity:

• High tolerance for out-of-date data may be required

• Updates may cause performance problems for busy nodes

• Retrieve incorrect data if updates have not arrived

• Takes time for update operations

• High tolerance for out-of-date data may be required

• Updates may cause performance problems for busy nodes

• Network communication capabilities

• Updates can place heavy demand on telecommunications/networks

**Horizontal partitioning**

• Table rows distributed across nodes (sides)

• Advantages

• data stored close to where it is used

• efficiency

• local access optimization

• better performance

• only relevant data is stored locally

• security

• unions across partitions

• ease of query

• Disadvantages

• accessing data across partitions

• inconsistent access speed

• no data replication

• backup vulnerability (SPOF)

**Vertical partitioning**

• Table columns distributed across nodes (sides)

Advantages and disadvantages are the same as for horizontal partitioning, except:

• combining data across partitions is more difficult because it requires joins (instead of unions)

## Dimensional modelling/ Data warehousing

**Data warehouse**

* A big centralised database holding all the information from different departments in a company.
* Supports numerical aggregations(average/total number of sales) and dimensions (Sales by product by store by quarter)
* The data is integrated from multiple sources internal and external to the organisation by converting it into a common format and validating before storing it to ensure credibility of the warehouse.
* Unlike an ER model, where data is organised around conceptual entities, the data in a data warehouse is organised around business processes such as sales, finance, or marketing.
* Meant for data analysis

• Subject oriented

– Data warehouses are organised around particular subjects (sales, customers, products)

• Validated, Integrated data

– Data from different systems converted to a common format: allows comparison and consolidation of data from different sources

– Data from various sources validated before storing it in a data warehouse

• Time variant

– Historical data

– Trend analysis crucial for decision support: requires historical data

– Data consists of a series of “snapshots” which are time stamped

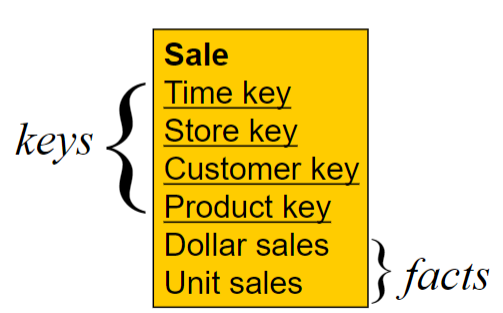
• Non-volatile

– Users have read access only – all updating done automatically by ETL process and periodically by a DBA

**Fact table**

• A fact table contains the actual business measures (additive, aggregates), called facts

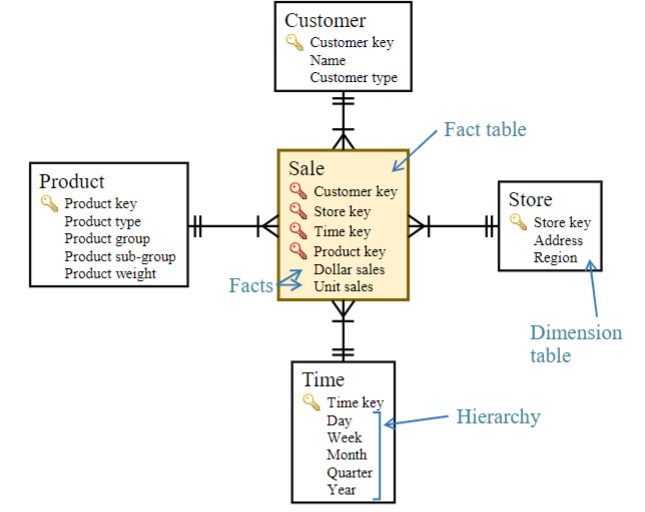
• The fact table also contains foreign keys pointing to dimensions

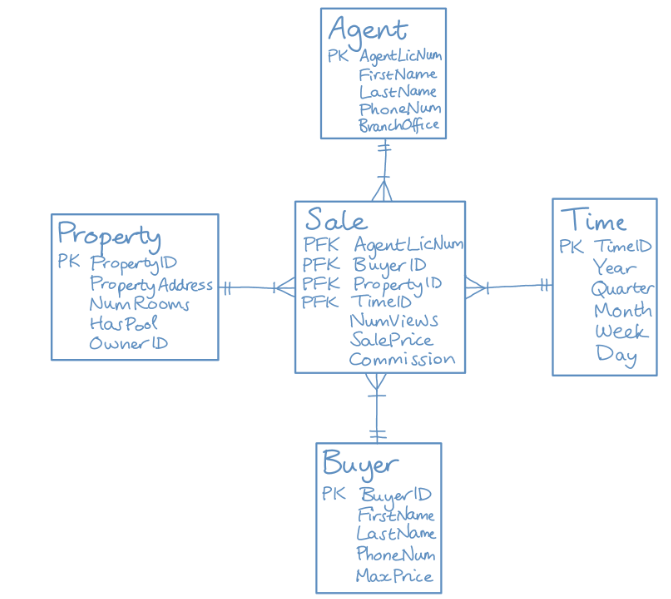


**Dimensional table**

Kimball’s four-step dimensional design process

1. Select and explain the business process.
2. Declare the grain and justify your choice.
3. Identify and explain the dimensions.
4. Identify and explain the facts.

Embedded Hierarchies in Dimensional Tables



star schemas are denormalized. Notice houw product sub-group and product group are transitive functional dependencies.

Why is star schema preferred to relational database for decision making?

Star schemas are organised around facts (business measures) and dimensions that help with managerial decision making. What’s more, they are denormalized, making it faster to aggregate and query data

## No SQL

**What is big data**

Data that exist in very large volumes and many different varieties (data types) and that need to be processed at a very high velocity (speed)

Pros of relational databases

– simple, can capture (nearly) any business use case

– can integrate multiple applications via shared data store

– standard interface language SQL

– ad-hoc queries, across and within "data aggregates"

– fast, reliable, concurrent, consistent

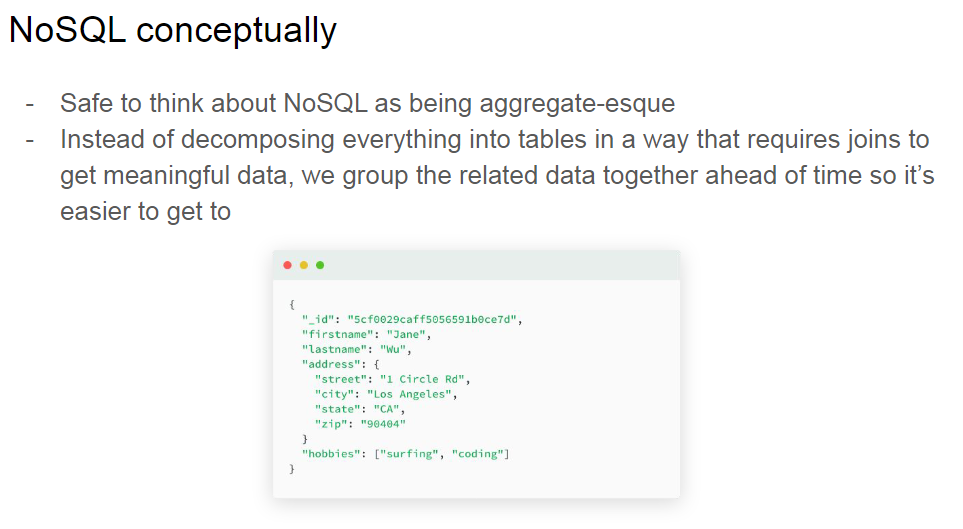
• Cons of relational databases

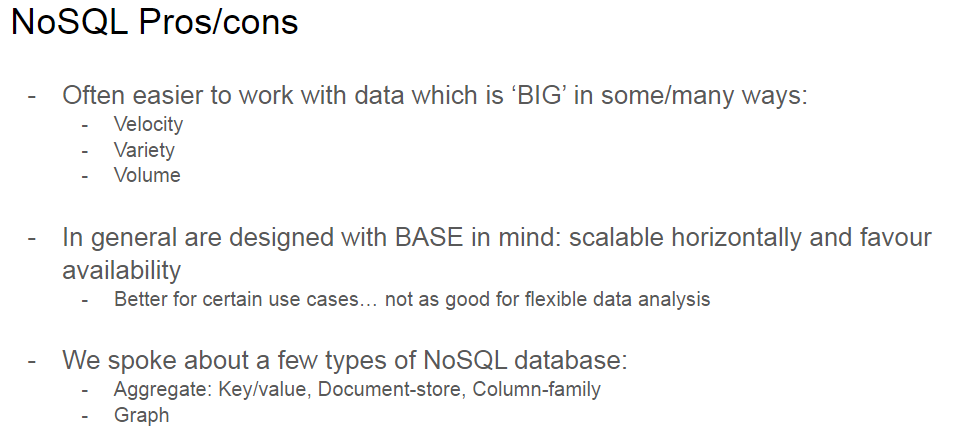
– Object Relational (OR) impedance mismatch

– not good with big data

– not good with clustered/replicated servers

NoSQL aims to solve the cons

**What is no sql**



NoSQL typically has schemas on read. Transactional database has schema on write.

Pros:

**Flexible modelling**

* flexible data models, less structured data

**Scalability**

* capacity can be removed and added easily using horizontal scaling

**Performance**

* high scalability means efficient read, writes, storage.

**High availability**

* Stored in partitions across multiple database instances without any shared resource. If any nodes fail, the database can continue on a different node.

Cons:

- Harder to perform data analysis - we likely have no way of knowing ahead of time which

data are relevant and which aren’t, need to traverse most of it

- Potentially takes up marginally more storage space

- Not completely consistent

Aggregate oriented

Key-value, document store and column-family are “aggregate-oriented- store business object in its entirety” databases (in Fowler’s terminology)

Pros:

– entire aggregate of data is stored together (no need for

transactions)

– efficient storage on clusters / distributed databases

Cons:

– hard to analyse across subfields of aggregates

– e.g. sum over products instead of orders

Key value stores

A simple pair of a key and an associated collection of values. Key is usually a string. The database has no knowledge of the structure or meaning of the values

Most flexible but least structured.

Good for pictures/ pdf/ non structured

Document

Like a key-value store, but “document” goes further than “value”. The document is structured, so specific elements can be manipulated separately.

stored in JSON, XML

Column-family

Data is grouped in “column groups/families” for efficiency reasons

Each row in a column-family table may have a different set of columns associated with it.

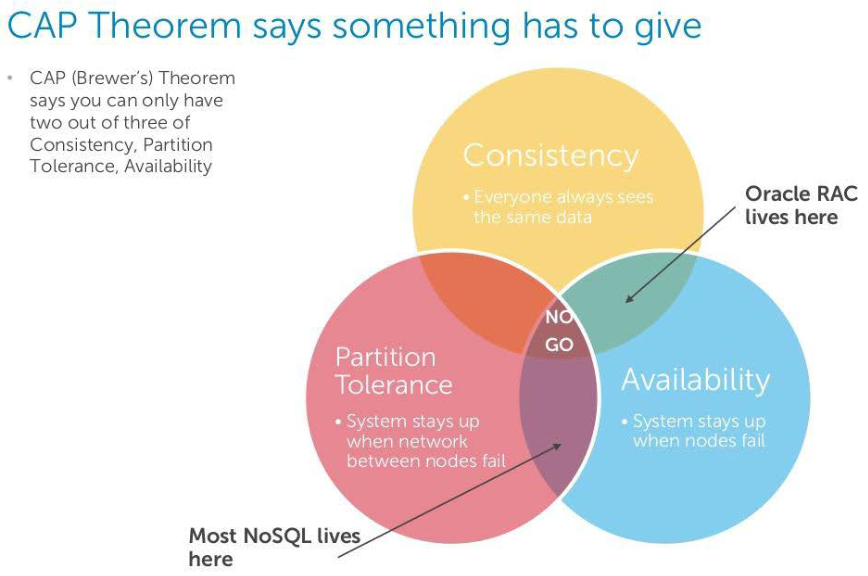
Graph

Maintain information regarding the relationships between data items. Nodes with properties.

Good for node and edge relations eg - book (node) and references (edge)

**CAP theorem / BASE**

If you have a distributed database, when a partition occurs, you must then choose consistency OR availability.



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.

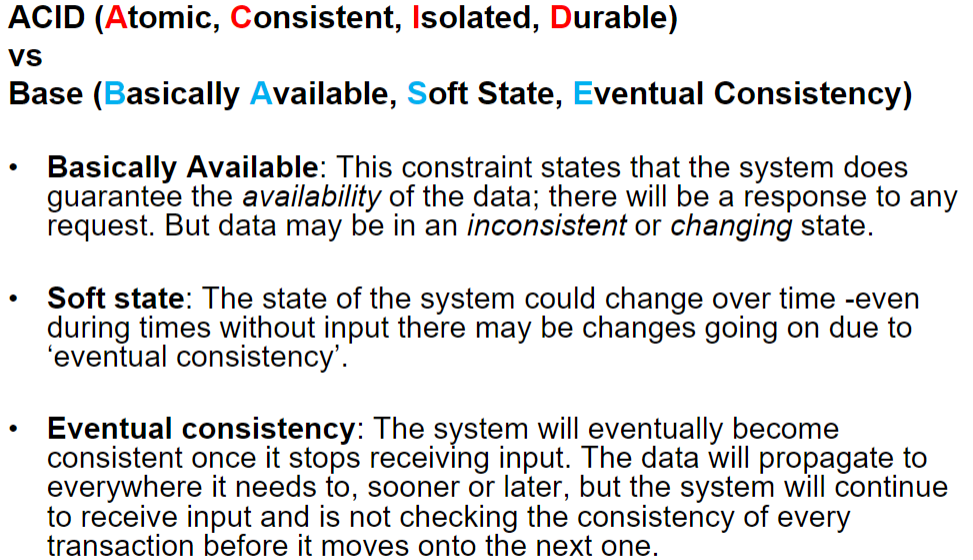
• CP: The database stops all the operations until the latest copy of data is available on

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.



Transactional database: ACID

NoSQL: BASE

