107502520 黃允誠 (答題一共四頁，第 4.題跨三頁，請教授及助教留意不要漏掉最後一頁)

1.

a)

可以直接先簡化成{A→B, A→C, BC→E, BC→F, E→B, C→D}

之後拆開：

ABC 構成的表是{A→B, A→C}

BCEF 構成的表是{BC→E, BC→F, E→B}

BD 構成的表竟沒有任何 dependency！

所以很明顯地：

b)

一定會出問題，It's not.

A	B	C	D	E	F
A1	Bsame	C1	D1	E1	F1
A2	Bsame	C2	D2	E2	F2

Decompose:

R1

A	B	C
A1	Bsame	C1
A2	Bsame	C2

R2

B	C	E	F
Bsame	C1	E1	F1
Bsame	C2	E2	F2

R3

B	D
Bsame	D1
Bsame	D2

Join:

A	B	C	D	E	F
A1	Bsame	C1	D1	E1	F1
A1	Bsame	C1	D2	E1	F1
A2	Bsame	C2	D1	E2	F2
A2	Bsame	C2	D2	E2	F2

演算法檢驗結果：(本來期末壓力大想偷懶等學期結束再看最後兩周的……後來還是乖乖先補課了)

	A	B	C	D	E	F
R1	O	O	O	X	O	O
R2	X	O	O	X	O	O
R3	X	O	X	O	X	X

2.

註：schema 只標出跟題目有關的部分 schema 及部分資料內容(若重要)。

(1)

WORK_ON

Employ_ID	Project_ID	Meet_place_ID	Hours

Employ_ID 跟 Project_ID 組成 Key，決定 Hours，但 Meet_place_ID 部分相依於 Project_ID，當今天一個 Project 要改變例行會議地點時，假設該 Project 有 100 個員工參與，就要找出 100 個 tuples 更新，很麻煩且可能出錯，為 update anomaly。

(2)

STUDENT

Student_ID	Student_name	Student_dept_ID	Student_college_ID

Student_ID 作為 KEY 決定其他 attributes，但同時系所也決定學院，因此 Student_ID→Student_dept_ID、Student_dept_ID→Student_college_ID 構成遞移相依。「系所屬於學院」的概念沒有被獨立出來，當學

校創立了「資電學院」之後，決定要將「資訊工程學系」由「工學院」移至「資電學院」，結果與前一題同理需對大量資訊工程學系的學生更新 Student_college_ID，為 update anomaly。
(3)
EMPLOY_RELATION

Employ_ID	Relative_ID	Coworker_ID
107502520	107502520mom	110522152
107502520	107502520dad	110522152
107502520	107502520mom	110525014
107502520	107502520dad	110525014
107502520	107502520mom	110522077
107502520	107502520dad	110522077
107502520	107502520mom	110522008
107502520	107502520dad	110522008

Employ_ID↠Relative_ID；Employ_ID↠Coworker_ID。今天我們的專案結束，這四位員工不再是我的 coworker 了。於是公司將這些資料移除，結果在有其他專案的 coworker 之前，我成沒父母的孤兒了，為 delete anomaly。
3.
a)
Requirement analysis: Manager, Expert, Clerks, Local IT
Manager 有他這個專案的(high level)需求；Clerks 有他操作方便的需求；Local IT 有他維護方便的需求，這一步要弄清楚所有人的需求，所以全部的人都要到。
Model design: Manager, Expert, Clerks, Local IT
Manager 要來提供需求端應用領域的領域知識(例如本次期末專案我認為學校的館舍和教室要進行完整的正規化，但教授提醒這些資料通常不會有高頻率變動，這就是應用領域知識影響 Model design 的選擇)；Expert 跟 Local IT 是兩公司負責處理資料庫較底層部分的人，本來就要到；Clerks 是因為從我們這學期的期末專案不難發現如果 Model 要有好的正規化就會比較複雜，若完全不懂 Schema，之後要下 query 會有點障礙，因此這一步 Clerks 也要參與。
Data preparation: Manager, Expert, Clerks
Manager 要管理這個專案，雖然他不懂資工，但至少資料庫裡放了甚麼資料他要知道；Clerks 在這步驟負責跟 Expert 溝通並進行資料處理的細節。Local IT 只負責資料庫的維護，理論上如果資料庫架構設計得好，維護跟資料的內容沒有太大關係，只跟前一步驟比較有關；如果這一步驟要放的資料真的太髒或處理得不好會影響資料庫的運作，由 B 公司的 Expert 負責監督跟溝通就好，因為資料庫的設計跟整合主要還是開發端主導。
Testing and confirmation: Manager, Expert, Clerks, Local IT
最後測試環節一樣有「跟需求端確認」的意思，跟「Requirement analysis」那一步驟同理，所有人都要到。
b)
其實每個步驟都很重要，頂多從其中挑一個比較沒那麼嚴重的：Data preparation
如前面所述，資料庫的長期維護和運作要好主要是看架構，內容到底放甚麼資料除非太髒不然影響沒有那麼大。其他三個步驟都絕對不能輕忽：頭尾是確保資料庫符合需求；中間是確保資料庫的架構契合需求端的應用領域以及嚴謹且表達力盡量完整，缺一不可。
4.
題目要盡量加速 referential constraints enforcing，特別是 CASCADE 的時候：
首先我們把每個 table 視為一個 vertex，然後如果 table A reference table B，就從 B 的 vertex 畫箭頭到 A 的 vertex。為甚麼反著畫？因為這代表的是 CASCADE 的方向。基本上我們只有對 referenced table 進行更動時能對 referencing table CASCADE，對 referencing table 進行更動時若違背 referential constraints 只能拒絕更動。
上述步驟結束後我們會得到一個有向圖。一個重要的問題：圖上有沒有環？有環就意味著有 referential cycle，實際上就形式主義而言，看 referential constraints 的定義並沒有限制不能有 referential cycle，但你很難想出一個實際的例子 model 出來的 schema 有這個情況。我上網查詢資料，普遍認可的說法是

「這樣的 schema 並沒有錯誤，但代表著 modeling 很爛。(poor design)」，換句話說如果做到這一步發現圖上有環，我們應該做的是重新設計或調整 schema 把環弄掉。
得到有向無環圖(DAG)後，我們就對這個圖做拓撲排序，便會從「只被 reference；不 reference 別人」的 table 到「只 reference 別人；不被 reference」的 table，中間的部分不會有排在前面的 table reference 排在後面的 table 的情況，換句話說不會有排在後面的 table CASCADE 到排在前面的 table 的情況。這個排序其實也就是第十一組期末專題報告寫 schema 時將表格從上到下排版的順序，也是大部分情況下畫完 reference 箭頭(只是跟現在畫的方向相反)看起來會最清楚的排序。
有了這個排序後我們挑出那些「只被 reference；不 reference 別人」的 table，這些 tables 就是我們決定物理上 site distribution 的基準。例如校務行政系統，這些 tables 可能有：學院、校區(假設學院運作可能跨校區)、外部廠商(清潔公司等)……
我們先討論簡單暴力的做法：這些 tables 中每個 row 都架一個 site。一個學院一個 site；一個校區一個 site；一個外部廠商我們學校架一個 site 紀錄它的相關資訊(或架在他們那邊上面放個 APP 給他們用)。
接著對於這些 table，將所有其下 reference 它的 tables 進行 complete horizontal fragment (sharding)，然後將對應的資料分配到對應的 site。例如：學生 reference 班級；班級 reference 系所；系所 reference 學院；我們對學院建 site。先看系所資料，每個 row 屬於哪個學院就分配到哪個學院的 site；再看班級資料，每個 row 屬於哪個系所，那個系所屬於哪個學院，就把這個 row 分配到那個學院的 site；以此類推。
這樣一來，在進行 referential constraints enforcing 時，乍看之下 CASCADE 下來都保證在同一個 site 內，非常快速。不過實際上還有個問題要解決：一個 table reference 多個 tables。反過來不是問題，一個 table 被多個 tables reference 的話，一樣按照前面的原則往下將所有下面的 table 全都做 sharding 然後分配到對應的 site 就好，沒有差別。重點是一個 table reference 多個 tables，絕大部分情況這通常都是「relationship relation」，例如：WORK_ON 同時 reference 員工和計畫；員工 reference 部門；計畫 reference 外部需求方；我們對每個部門有一個 site；對每個外部需求方跟他們的 site 連動，在其上放一部份我們的資料庫。這時 WORK_ON 的資料到底是要根據員工 ID 這個 FK 分配到部門的 sites，還是根據計畫 ID 這個 FK 分配到外部需求方的 sites？
這邊可以進行 replication，用兩份 WORK_ON，分別根據前述的原則分配到部門對應的 sites 跟外部需求方的 sites。但題目說跨 sites 的連線慢 50 倍！沒關係，其實我們只需要傳輸極少量的資訊，這個資訊量是常數的複雜度，不管你要一次 CASCADE 多少 row 都是常數：例如某個外部需求方的某個計畫結束或是終止合作，我們在那個外部需求方的 site 將該計畫 delete，然後要 CASCADE 到 WORK_ON。在那個 site local 端可以很快的 CASCADE，把所有員工 WORK_ON 那個計劃的資料都在那個 site 上的 WORK_ON 中 delete。接著我們只要傳送：「WORK_ON」(table 的識別 ID)、「計畫 ID」(CASCADE 目標 FK 欄位)、某個計畫 ID (CASCADE 目標 FK 值)、delete (CASCADE 動作)，然後屬於部門這邊的 sites 收到通知，查看其下 WORK_ON table 的內容，各自進行 local 端的 CASCADE 即可。這邊 network 的 prototype 有點影響，因為一個計劃有很多員工參與，這些員工可能屬於多個不同部門，如果只能點對點傳送，那麼剛才那個某外部需求方的 site 就要對所有部門對應的 sites 一個一個傳送這四筆資料，而且從它那邊看它很難知道哪些部門的 sites 其實不甘他們的事；但如果 network 支援「廣播式」的傳輸，也就是類似一個 pool 的概念，我發一個訊息出去，然後大家自己看你要不要做甚麼事情，這樣題目說每個 site 計算跟通信能力都相近，我發出去，其他所有 sites 同時接收，需要 CASCADE 的就同時開始，會比較快。
那麼我們還能有更細緻的作法：對 WORK_ON 做 complete vertical fragment，在它 reference 出去的兩端(或多端)當中，將各個 FK 分配到那一端對應的 sites 上，而至於不是 PK 也不是 FK 的欄位，就考量 query 等等存取資料的情況中，哪邊比較常使用到就分配到哪邊。但壞消息是，這個優化只對 update 的 CASCADE 有好處，例如某員工的員工 ID 改了，你直接在那個部門的 site 上 CASCADE 到 WORK_ON 就好，就不用管外部需求方這邊；但如果是 delete 的就沒用，還是要廣播過來。
綜上所述，這個做法保證在單一的「reference 鏈」上，所有 table 雖然都分散，但相互有關聯(也就是 JOIN 時要放在一起)的 row 都在同一 site 上，CASCADE 迅速。而對於一個 table reference 到多個 tables 的「CASCADE 路徑匯合」情況，也有相應的機制盡量加速。

最後再回到最一開始說的「簡單暴力」的部分：實際上不一定對最上層 table 要分配 sites 時都要一筆資料(row)一個 site，例如學院，你可以讓工學院跟資電學院共用一個物理的 sites，之後的步驟也都一樣照前述的原則，並不會出甚麼問題。這取決於你建 sites 的方便，你是要建很多 sites，還是比較少的 sites 但每個 site 比較強，這還牽扯到地理上、物理設備上等等各種其他領域的考量，已經不侷限在資料庫了。

這個方案還有一個好處，就是絕大部分情況下你在 model 的時候，會屬於那種「不 reference 任何 table」的 table，通常也就對應到你會想建 site 的「概念」，例如學院、部門，一筆資料一個 site 本來就也合理。反例如「學生」這個概念，在資料庫大尺度的考量下你自然不會想針對學生建 site。如果說要有就會比較接近課本說的「機動銷售員」那種概念(A special case of partial replication occurring heavily in applications)，另外 fetch 一小部分資料出去，很小型的 site：想像未來某新概念的學校架構中學生不再 reference 到班級系所，一個學生在手機(site)上退選一門課，從學生修課紀錄刪除這筆資料，學生修課紀錄同時也 reference 到課程；課程 reference 到老師；老師 reference 到系所，而一個課程可能由多位老師聯合授課，於是這個學生的 site 廣播要求所有系所的 sites 確認它們是否要對它們底下的學生修課紀錄 CASCADE：學生修課紀錄、學生 ID、107502520、delete，這樣也是合理的。從這個例子可以看出，實際應用上會根據我提出的方法被關聯到 site 的 table 通常其對應的概念也就是終端的 entity。當然也有可能有些個數龐大的概念沒有 reference 任何人，又不好像這樣關聯到 site，我目前想不到例子，但可能有，所以才說要看情況分配，不是一定這些「最上層 table」都全部一筆資料一個 site。

所以題目提到多少 sites，這部分其實不是重點，看你實際應用，要多少 sites 以上的方式都能套用。

最後稍微說一下 overheads 的部分：referential constraints enforcing 的部分前面都說了，盡量保持在同個 site 內。那麼這個架構絕大部分是 no replication，只在像 WORK_ON 這樣的 table 上有 replication，所以你的 update (update、insert、delete)在這些 table 上會比較慢，在其他大部分的 table 上都很快。但 availability 就會比較差，特別是例如你這個部門要看其他部門的資料，甚至是想看所有部門的資料的時候，JOIN 起來那大量的網路連線非常難受。

但課本有提到，replication 少就是 availability 差然後 update 快；replication 多(極端是每個 site 都複製一整份資料庫)就是 availability 很好但 update 會很慘，這是不可兼得的。而此題題目要求盡量加速 referential constraints enforcing，特別是 CASCADE 的情況下，那麼這些動作其實是有點接近 update 的，所以我選擇偏向前者。

還有就是因為還是有 replication，所以還是有 Concurrency Control 的 overheads。