



INFO20003 DATABASE SYSTEMS REVISION WORKSHOP

Starting at ~ 2.05 pm!



Sign up for CISSA!

cissa.org.au/signup

Structure of Today's Workshop

1. ER Modelling
2. SQL
3. Query Optimisation
4. Normalisation
5. Transactions
6. Distributed DBs
7. No SQL
8. Group Work / Breakout rooms
9. General Questions & Answers!

If you have questions as we work through problems, please feel free to ask them right away!

CISSA

- Largest Tech Club in UoM.
- Representative of **Computer Science / Data Science** students in both undergrad and postgrad.
- Joint events and collaborations with...
 - **Tech Companies** (Google, Microsoft, Amazon, Canva etc.)
 - **Consulting Firms** (Deloitte)
 - **Trading Companies** (Optiver, Jane Street, IMC etc.)
 - **Researchers** in UoM
- **Social** events for you to make more friends in the tech community!



Join Our Club (it's FREE)!

- Access our [Discussion Space](#) on Facebook / Discord to connect with over 4000 people in the UoM tech community.
- Ask questions re. [Subjects](#), [Careers](#), [Interviews](#) etc.
- Events:
 - [Industry Connect & Tech talks](#)
 - [Mock Interviews + Office Tours](#)
 - [Hackathons - Codebrew + Catalyst](#)
 - [Start-Up & Entrepreneurship Competitions](#)
 - [Subject Revision + Interview Workshops](#)

Sign-Up!



cissa.org.au/signup

ER Modelling

ER modelling

Key concepts:

1. **Constraints:** Key constraints, participation constraints
2. **Weak Entities** (can be identified uniquely only by considering the primary key of another entity)
3. **Special attributes:** Multi-valued, composite
4. **Relationships:** Binary(most common), unary, ternary relationships
5. **Notation:** Chen's notation, Crow's foot notation.
6. **Translating design:** from conceptual design to logical design to physical design

ER modelling

Tips:

1. Do not include the actual business or company whose business processes you are modeling.
2. Avoid using vague words like “has” to label your relationships
3. Do not overuse weak entities (when an entity has a unique identifier, avoid modelling as a weak entity)
4. Remember to annotate underline for primary key (and dashed underline for partial key of weak entities)
 - a. Remember use surrogate key if you cannot find one from the case study
5. Proper use of relationship attributes is helpful
6. Ternary v.s. 2 binary relationships: Use ternary only when a relationship must involve all three entities.

ER modelling

Tips:

7. My steps to go through a question about ER modelling in the exam

- First reading: Read quickly and roughly, just to understand the business scenario
- Second reading: Read sentence by sentence. Decide entities and their attributes (focus on nouns), figure out relationships (normally in a sentence that involves two or more entities). Pay attention to some keywords when determining the relationship constraints (e.g. at least one, at most one, may, etc...)
 - Entity: Could be real world objects, events or concepts! Anything that we are interested in
- Based on the draft paper and understanding on the question, finish the model
- Double check special attributes

ER modelling

Tips:

8. List your assumptions if you made any, but only use assumptions when there are ambiguities instead for simplifying the case study question
 - Exam questions will be clear and straightforward, so only make assumptions when you have to!
9. Do not draw a messy final model
 - Please make sure your model is clear and readable.
 - Don't draw a circle like a rectangle! Don't draw a rectangle like diamond!
 - Make sure we can distinguish between bold lines and regular lines!
10. Check out your assignment 1 feedback

Question - Swim school (adapted from Practice data modelling task)

Consider the case of a privately run swim school in Melbourne. For now, data relating to staff, customers, lessons and others are stored in a mix of Excel spreadsheets on the front desk PC, and paper files in filing cabinets. The manager decided to employ you to set up an integrated data management solution for the swim schools. The following information is given for you:

The business employs a range of staff on a casual basis, meaning that each staff member is paid for each shift they perform. The business keeps track of the name, address and one or more phone numbers of all staff. Staff members are required to note down the start and end times of every shift and write a brief summary of the work they completed (for example, “cleaned filters on Pool A and fixed roof leak”). The manager would like the new database to also store the hourly pay rate for each shift to make it easier for them to pay staffs. Staff may take leaves for defined periods of time and the manager would like these to be recorded.

This swim school offers different swimming course plans for their customers. The manager plans to store customer information in an external Customer Relationship Management (CRM) system, so all that is to be stored in your database is their name and unique CRM ID. Customers sign up for swimming course plans, each consisting of a certain number of swim lessons with a particular teacher for each lesson. The manager asked you to store all related information about each course plan including its unique ID, location, total cost and the start date for each customer. The system should be able to capture the duration and type for each lesson as well. Each lesson has a unique identifier as well.

Question - Swim school (adapted from Practice data modelling task)

My step 1: First reading, understanding business scenario

Consider the case of a privately run swim school in Melbourne. For now, data relating to staff, customers, lessons and others are stored in a mix of Excel spreadsheets on the front desk PC, and paper files in filing cabinets. The manager decided to employ you to set up an integrated data management solution for the swim schools. The following information is given for you:

The business employs a range of staff on a casual basis, meaning that each staff member is paid for each shift they perform. The business keeps track of the name, address and one or more phone numbers of all staff. Staff members are required to note down the start and end times of every shift and write a brief summary of the work they completed (for example, “cleaned filters on Pool A and fixed roof leak”). The manager would like the new database to also store the hourly pay rate for each shift to make it easier for them to pay staffs. Staff may take leaves for defined periods of time and the manager would like these to be recorded.

This swim school offers different swimming course plans for their customers. The manager plans to store customer information in an external Customer Relationship Management (CRM) system, so all that is to be stored in your database is their name and unique CRM ID. Customers sign up for swimming course plans, each consisting of a certain number of swim lessons with a particular teacher for each lesson. The manager asked you to store all related information about each course plan including its unique ID, location, total cost and the start date for each customer. The system should be able to capture the duration and type for each lesson as well. Each lesson has a unique identifier as well.

Question - Swim school (adapted from Practice data modelling task)
My step 2: Analyse sentence by sentence, focus on keywords (maybe circle some if allowed)

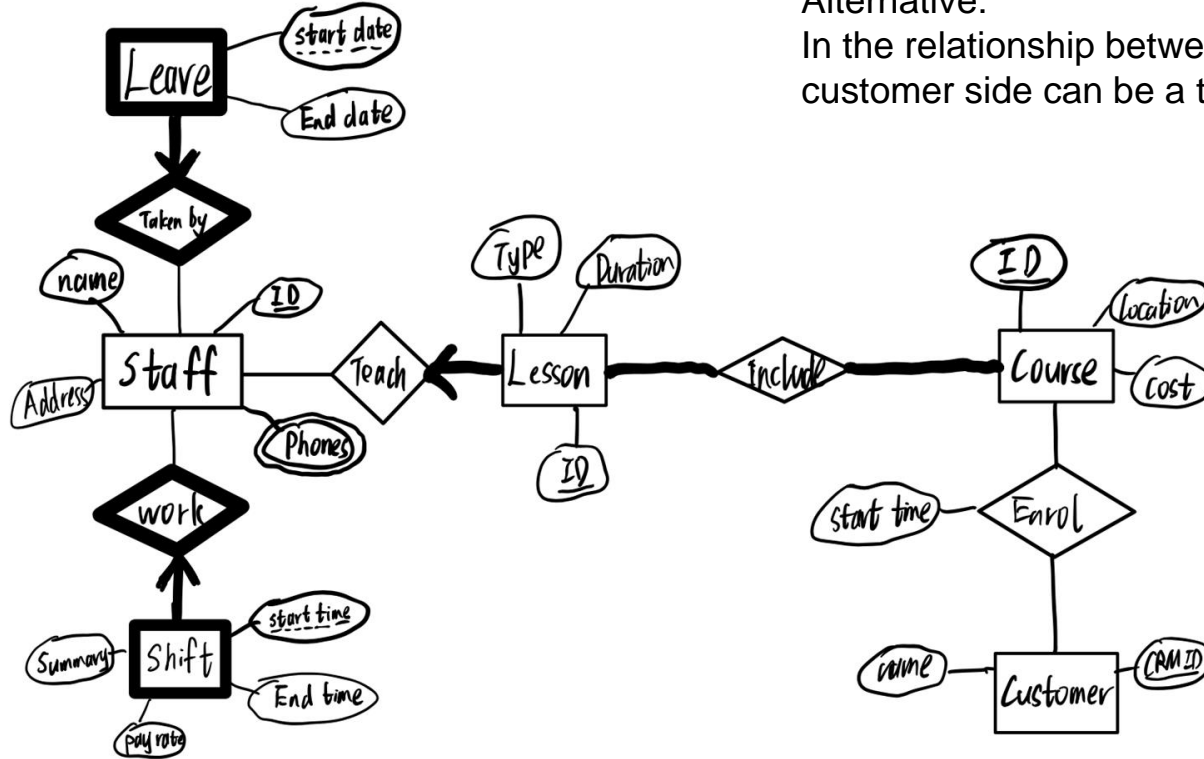
Consider the case of a privately run swim school in Melbourne. For now, data relating to staff, customers, lessons and others are stored in a mix of Excel spreadsheets on the front desk PC, and paper files in filing cabinets. The manager decided to employ you to set up an integrated data management solution for the swim schools. The following information is given for you:

The business employs a range of staff on a casual basis, meaning that each staff member is paid for each shift they perform. The business keeps track of the name, address and one or more phone numbers of all staff. Staff members are required to note down the start and end times of every shift and write a brief summary of the work they completed (for example, “cleaned filters on Pool A and fixed roof leak”). The manager would like the new database to also store the hourly pay rate for each shift to make it easier for them to pay staffs. Staff may take leaves for defined periods of time and the manager would like these to be recorded.

This swim school offers different swimming course plans for their customers. The manager plans to store customer information in an external Customer Relationship Management (CRM) system, so all that is to be stored in your database is their name and unique CRM ID. Customers sign up for swimming course plans, each consisting of a certain number of swim lessons with a particular teacher for each lesson. The manager asked you to store all related information about each course plan including its unique ID, location, total cost and the start date for each customer. The system should be able to capture the duration and type for each lesson as well. Each lesson has a unique identifier as well.

Question - Swim school (adapted from Practice data modelling task)

My step 3: Based on draft and understanding, finish the model



Alternative:

In the relationship between customer and course, the customer side can be a total participation

SQL

SQL Questions

- It's very likely that there will be SQL questions on the exam related to the A2 case study, this has been the case for most past exams
- **Revise the A2 case study!**
 - Redoing it is a great idea!
- Can try making up your own questions as well

SQL

- In the past, we commonly have exam questions that require a left join so make sure you know how that works
- Most important thing is to try these questions yourself, without looking at any answers first
- In the exam, you can open up MySQL and run your queries and check them!

SQL Summary - Keywords

SELECT colA, ...

- DISTINCT - This removes any duplicates in the result
- May use aggregation functions here e.g. COUNT(), SUM(), AVG(), MIN(), MAX()

FROM TableA ...

- Can join with other tables here

WHERE ... AND ... OR

- Ensures the rows returned satisfy these conditions

GROUP BY

- Groups rows together
- e.g. GROUP BY forum.id would group rows so that all rows with the same forum.id get combined in some way to one row. Aggregation functions can be used here.

HAVING

- Similar to the WHERE clause, but here we are applying a condition on each group formed in the GROUP BY

ORDER BY ... ASC / DESC

- Sorts the results by certain attribute(s)

LIMIT ...

- Limit so that not all rows are returned e.g. LIMIT 1 returns the first row

SQL Summary - Types of Joins

- NATURAL JOIN
 - Attributes being joined on must have the same name
- INNER JOIN
 - Specify the condition being joined on
 - E.g. TableA INNER JOIN TableB ON TableA.id = TableB.id
- Outer Joins
 - Includes records which do not have a match with a record in the other table
 - Gets merged with NULLs
 - LEFT OUTER JOIN - Includes all records from the **left** table, even if there's no matches
 - RIGHT OUTER JOIN - Includes all records from the **right** table, even if there's no matches
 - FULL OUTER JOIN - Includes all the matches as well all the unmatched records from **both** tables

SQL Summary - Subqueries & Set operations

- Using subqueries (nested queries) are very useful!

Sub-query operators

- **IN / NOT IN**
 - can check if an attribute is in / not in a subquery e.g. WHERE userID IN (SELECT user FROM ...)
- **ANY**
 - True if **any** value returned meets the condition (OR) e.g. WHERE userID = ANY(1, 8, 11)
- **ALL**
 - True if **all** values returned meet the condition (AND) e.g. WHERE userID = ALL(1, 8, 11)
- **EXISTS**
 - True if the subquery returns one or more rows e.g. WHERE EXISTS(SELECT user FROM ...)

Set operations (which are supported on MySQL)

- **UNION** - Returns all records from both inputs
- **UNION ALL** - Use ALL if you want duplicate rows to be shown

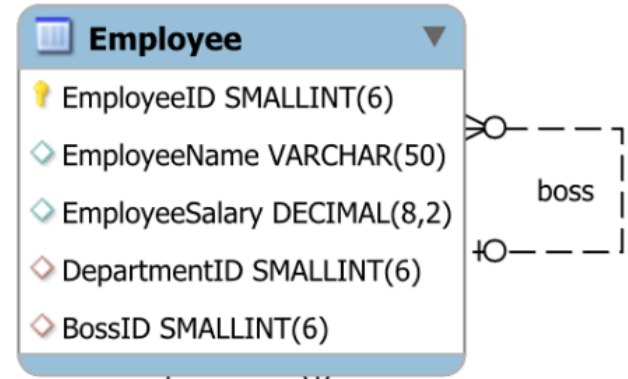
SQL Summary - Unary Joins, Aliases

From Tutorial 5:

Find the employees whose salary is less than half that of their managers.

This requires a unary join. Use the 'AS' keyword as an alias for the table

```
SELECT Emp.EmployeeName
FROM Employee AS Emp
  INNER JOIN Employee AS Boss
    ON Emp.BossID = Boss.EmployeeID
WHERE Emp.EmployeeSalary < (Boss.EmployeeSalary / 2);
```



SQL question

This is in the **same context as the case study for A2** so revise that before you attempt this question:

From the users which have friends, find the number of friends each user is currently friends with.

Order your results from highest to lowest number of friends.

Return your results in the form (userId, numFriends)

Query Optimisation

Query optimization

Tips:

1. Understand the formula in the cheatsheet. Memorizing != understanding
2. Understanding = able to answer questions such as:
 - What is 2 in $2 * N_{Passes} * N_{Pages}$ in the Sort Merge Join formula?
 - 2 -> One for reading and one for writing
 - What is B in the Block-oriented NLJ?
 - B -> Buffer size, how many pages can fit in the memory
 - What is 3 in the Hash Join formula?
 - 3 -> One for reading (to bring the pages into memory for hashing), one for writing(after hashing), one for reading again to join
 - And so on

You should be able to answer all those kind of questions!!! They may not be directly examined but they are very important when you analyse a given pipeline!!!

Query optimization

Tips:

3. Be careful for usability of indexes!

- Hash index: No range query
- Cluster index: Compare search key fields prefix and query predicates.

4. Reduction factor decides cost and result size:

$$\mathbf{ResultSize} = \prod_{j=1..k} \mathbf{NTuples}(R_j) \prod_{i=1..n} \mathbf{RF}_i$$

- Cost depends on the search key fields of the index and the query predicates, only consider the matched attributes.

Question - Tutorial 8 take home question

Consider the following schema:

Student (studentid, name, dob, degreename)

StudentSubject (studentid, subjectid, grade)

Subject (subjectid, name, level, coordinatorname, budget)

The number of tuples in Student is 20,000 and each page can hold 20 records. The StudentSubject relation has 50,000 tuples and each page contains 50 records. Subject has 1,000 tuples and each page can contain 10 records. One page can fit 100 resulting tuples of Student JOIN StudentSubject. 100 tuples resulting from the join of StudentSubject and Subject also fit onto a page. Assume that Subject.subjectid and Student.studentid are candidate keys. Sorting can be done in 2 passes. There are 3 available indexes: an unclustered hash index on Student(degreeame), an unclustered B+ tree index on Subject(level), and a clustered B+ index on StudentSubject(studentid). All indexes have 50 pages. There are 10 distinct values for Subject.level, ranging from 1-10. There are known to be 40 distinct values for Student.degreeame. Consider the following query:

```
SELECT Stu.studentid, Sub.subjectid
```

```
FROM Student AS Stu, Subject AS Sub, StudentSubject AS SS
```

```
WHERE Stu.studentid=SS.studentid AND SS.subjectid=Sub.subjectid AND Stu.degreeame='CompSci' AND Sub.level=10
```

For this query, estimate the cost of the following plans. If selections are not marked on the tree, assume that they are done on the fly (in memory) after having completed all joins.

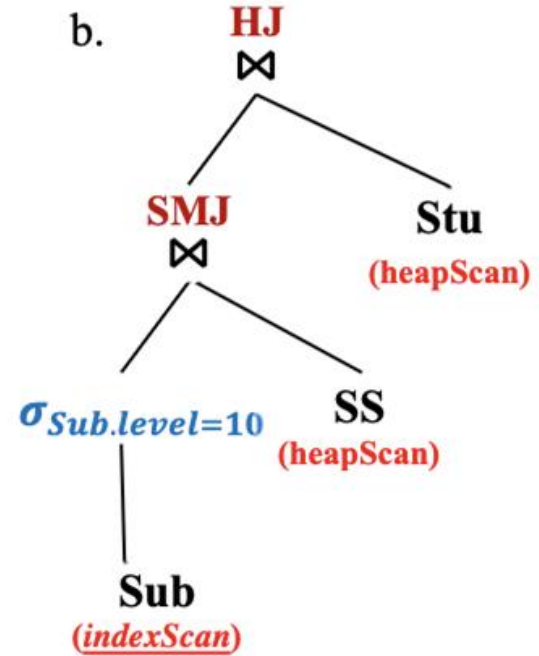
Extract info from text

	ntuples	tup/pg	pages	Index #vals pages
Student	20,000	20	1,000	UC hash degree name 40 50
SS	50,000	50	1,000	C B+ student 50
Subject	1,000	10	100	UC B+ level 10 50
Stu X SS	???	100	???	
SS X Sub	???	100	???	

Index: 50 pages
Sorting 2 passes

Steps

- 1) Calculate selection cost + size
- 2) Calculate 1st join cost + size
- 3) Calculate 2nd join cost (don't need size)

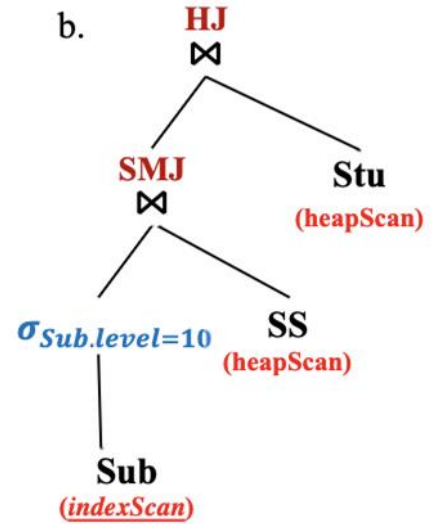


Steps

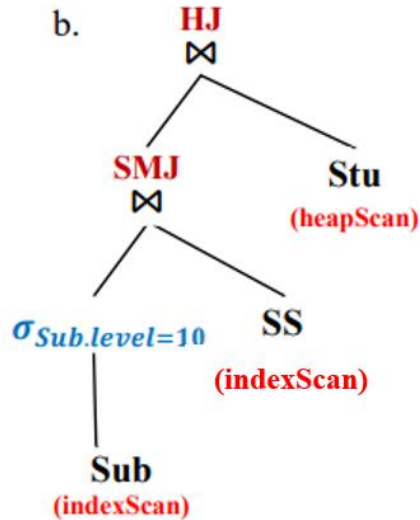
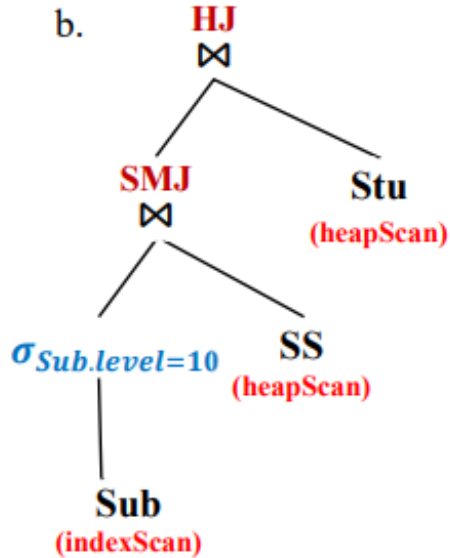
- 1) Calculate selection cost + size
- 2) **Calculate 1st join cost + size**
- 3) Calculate 2nd join cost (don't need size)

For step 2, what do we subtract in the formula?

$$\begin{aligned} \text{CostSMJ} &= \text{NPages}(\text{Sub selection}) + \text{NPages}(\text{SS}) \\ &+ 2 * \text{NPages}(\text{Sub selection}) * \text{num_passes}(\text{Sub selection}) \\ &+ 2 * \text{NPages}(\text{SS}) * \text{num_passes}(\text{SS}) \end{aligned}$$



What's the difference?



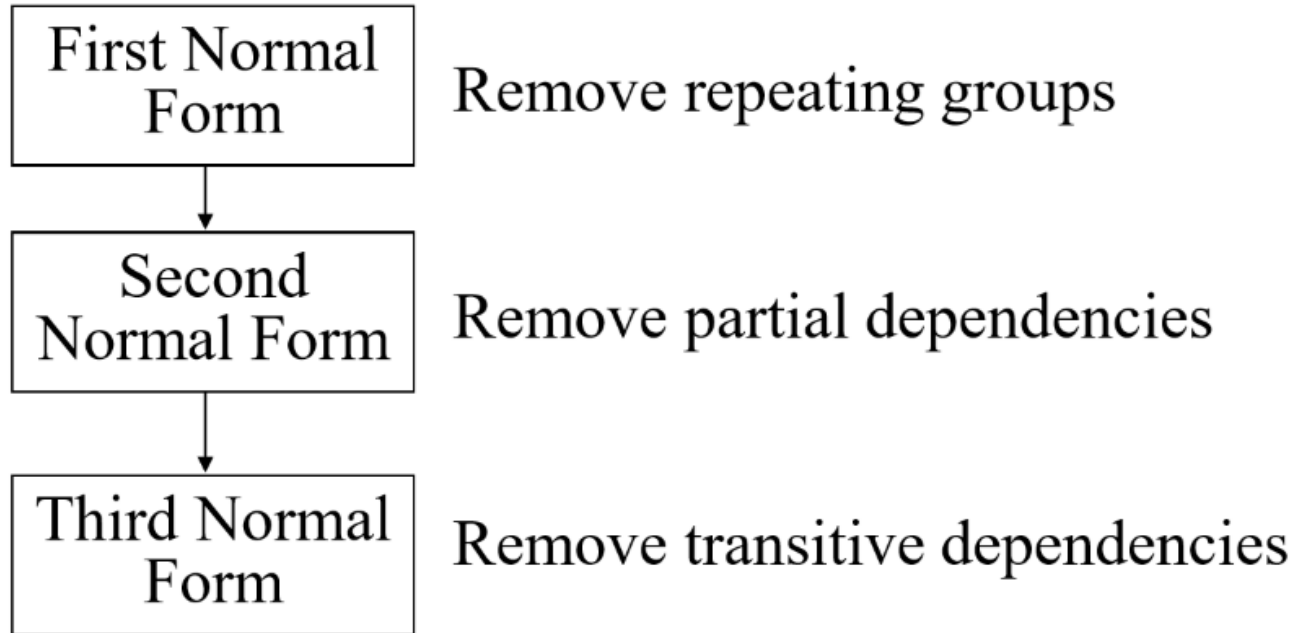
	Index #vals pages
Student	UC hash degree name 40 50
SS	C B+ stuld 50
Subject	UC B+ level 10 50

Normalisation

Why do we normalise relations?

- We want to reduce data duplication / redundancy (storing the same piece of data in many places)
- Allow users to insert, delete and update rows without anomalies.
 - **Insertion** Anomaly
 - Cannot insert certain attributes without other attributes
 - **Deletion** Anomaly
 - Accidental loss of attributes due to the deletion of other attributes
 - **Update** Anomaly
 - Happens when one or more instances of duplicated data are updated but not all
- Normalisation is the process of taking larger, denormalised tables and breaking them into smaller tables

How to normalise:



How to normalise:

To achieve 1NF

- Remove cells which have multiple values inside of them
- Remove repeating groups of attributes
 - Order-Item(Order#, Customer#, ((Item#, Desc, Qty))
 - These brackets denote there is a repeating group

To achieve 2NF

- Remove partial dependencies
- This is where a non-key attribute depends on a part of the PK (but not the whole PK)

To achieve 3NF

- Remove transitive dependencies
- This is where a non-key attributes depends on another non-key attribute

Normalise this table into 3NF

staffNo	name	position	salary	branchNo	branchAddress	telNos
S1500	Tom Daniels	Manager	50000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 03-555-3618
S0003	Sally Adams	Assistant	30000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 03-555-3618
S0010	Mary Martinez	Manager	50000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 06-555-6756
S3250	Robert Chin	Supervisor	32000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 06-555-6756
S2250	Sally Stern	Manager	50000	B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131, 06-555-3131
S0415	Art Peters	Manager	50000	B003	14 – 8th Avenue, New York, NY 10012	212-371-3000, 12-371-3000

Normalise this StaffBranch table into 3NF

The dependencies are:

- staffNo → name, position
- position → salary
- branchNo → branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

Transactions

DB Transactions + ACID

- We want to have certain guarantees regarding the state of the database before/while/after performing a series of SQL CRUD operations + Schema manipulations
 - PARTICULARLY important when multiple users are accessing the database at the same time, trying to read/write the same data all at the same time!
- Transactions are tools that allow us to consider related groups of commands as being tied together
- In relational databases, ACID is generally king. We want our transactions to be:
 - Atomic
 - Consistent
 - Isolated (there are many levels, we consider '*serializable*', the default/strongest/safest level)
 - Durable

(for interest only)

The SQL standard and PostgreSQL-implemented transaction isolation levels are described in **Table 13.1**.

Table 13.1. Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

Transactions

Question:

A DBMS which strictly implements ACID compliant transactions can never encounter the 'Inconsistent retrieval problem'.

Which ACID property is primarily responsible for preventing this problem? Explain how it prevents the inconsistent retrieval problem.

Transactions

Alice	Bob
SELECT SUM(Salary) FROM Employee;	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 33;
	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 44;
(finishes calculating sum)	COMMIT;

Transactions

Time	Trans- action	Action	Value	T1 SUM	Comment
1	T1	Read Salary for EmpID 11	10,000	10,000	
2	T1	Read Salary for EmpID 22	20,000	30,000	
3	T2	Read Salary for EmpID 33	30,000		
4	T2	Salary = Salary * 1.01			
5	T2	Write Salary for EmpID 33	30,300		
6	T1	Read Salary for EmpID 33	30,300	60,300	<i>after update</i>
7	T1	Read Salary for EmpID 44	40,000	100,300	<i>before update</i>
8	T2	Read Salary for EmpID 44	40,000		<div>we want either <i>before</i> \$210,000 or <i>after</i> \$210,700</div>
9	T2	Salary = Salary * 1.01			
10	T2	Write Salary for EmpID 44	40,400		
11	T2	COMMIT			
12	T1	Read Salary for EmpID 55	50,000	150,300	
13	T1	Read Salary for EmpID 66	60,000	210,300	

Transactions

A:

C:

I:

D:

Transactions

A: Atomic

C: Consistent

I: Isolated

D: Durable

Solution:

Isolation

Isolation guarantees that concurrent execution of transactions leaves the database in the same state as if the transactions were executed sequentially (one after another)

Transactions

Time



Alice's Transaction

Bob's Transaction



Correct
(no issue
with
Inconsistent
Retrieval
problem)

OR

Bob's Transaction

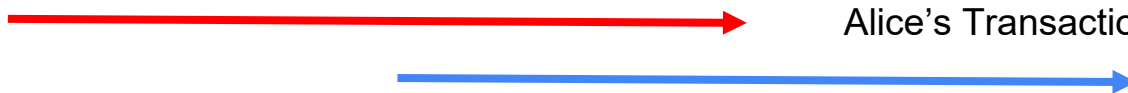
Alice's Transaction



Incorrect
(Inconsistent
Retrieval)

Bob's Transaction

Alice's Transaction



Distributed DBs

Distributed DBs PRO/CON

- Distributed DBs are needed when you get to a certain scale, or you need global availability
- PRO:
 - Scalability
 - Availability + Reliability
 - Better response time for localised data
- CON:
 - More complex (cost, security issues, etc)
 - CAP tradeoffs

Distributed DBs types

- Types of distributed DBs that we discussed:
 - Data replication (ie *full* replication)
 - Horizontal partitioning
 - Vertical partitioning
 - Combinations (most common)



Distributed DBs

Question:

As business grew over time for hiTech Computer Repairs Inc, the company has opened multiple new stores in Australia, and also recently spread to Europe. The management at hiTech Headquarters is interested in upgrading the database associated with the `PC repair services` booking system from centralised to a distributed structure. HiTech repair centers in Australia are unlikely to regularly need access to repair records for European customers, and vice versa.

Which of the following type of distributed database would be best suited for this case: **a) Vertically partitioned b) Horizontally Partitioned c) Fully Replicated DB**

Distributed DBs

Answer:

Horizontally Partitioned.

- Same type of data being accessed (ie from same table) for the 'service repairs', BUT:
- Rows that are for the 'Australian' stores are not likely to be used by the 'European' stores.
- Split the European rows into a European server, and keep Australian rows on the Australian server

No SQL

NoSQL conceptually

- Safe to think about NoSQL as being aggregate-esque
- Instead of decomposing everything into tables in a way that requires joins to get meaningful data, we group the related data together ahead of time so it's easier to get to

A code editor window with a light gray background and a title bar with three colored dots (red, yellow, green). It contains a JSON document representing a user profile.

```
{
  "_id": "5cf0029cafff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

NoSQL Pros/cons

- Often easier to work with data which is 'BIG' in some/many ways:
 - Velocity
 - Variety
 - Volume
- In general are designed with BASE in mind: scalable horizontally and favour availability
 - Better for certain use cases... not as good for flexible data analysis
- We spoke about a few types of NoSQL database:
 - Aggregate: Key/value, Document-store, Column-family
 - Graph

No SQL

Question:

FaceTube™ is a new online worldwide social video sharing platform that you and your friend have started.

Your friend asks if you think a No-SQL based database for storing the metadata + comments for video uploads is a good idea.

Discuss some pros/cons for using a No-SQL database for this scenario. Your answer should make reference to the properties of the data being stored and BASE.

No SQL

Answer:

Yes, NoSQL is likely a good choice here

Pros:

- High availability when partitioned (BA)
- *OK if not completely consistent (BA), as long as other users will eventually (e.g. in a few minutes) see new comments etc (S + E)*
- Can deal with large number of comments in rapid succession

Cons:

- Harder to perform data analysis
- Potentially takes up marginally more storage space

Time for Group Work!

Solutions

SQL Solution

From the users which have at least one friend, find the number of friends each user is currently friends with.
Order your results from highest to lowest number of friends.
Return your results in the form (userId, numFriends)

```
SELECT userID, SUM(numFriends) as totalFriends
FROM
(
    (SELECT user1 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested IS NOT NULL AND WhenRejected IS NULL
    AND WhenConfirmed IS NOT NULL AND WhenUnfriended IS NULL
    GROUP BY user1)

    UNION

    (SELECT user2 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested IS NOT NULL AND WhenRejected IS NULL
    AND WhenConfirmed IS NOT NULL AND WhenUnfriended IS NULL
    GROUP BY user2)

) AS friendList

GROUP BY userID
ORDER BY totalFriends DESC;
```

friendof	
📌	user1 INT(11)
📌	user2 INT(11)
🔹	WhenRequested TIMESTAMP
🔹	WhenRejected TIMESTAMP
🔹	WhenConfirmed TIMESTAMP
🔹	WhenUnfriended TIMESTAMP
Indexes ▶	

From the users which have at least one friend, find the number of friends each user is currently friends with. Order your results from highest to lowest number of friends.
Return your results in the form (userId, numFriends)

```
SELECT userID, SUM(numFriends) as totalFriends
FROM
(
    (SELECT user1 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested IS NOT NULL
    AND WhenRejected IS NULL AND
    WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user1)

    UNION

    (SELECT user2 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested
    IS NOT NULL
    AND WhenRejected IS NULL
    AND WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user2)

) AS friendList

GROUP BY userID
ORDER BY totalFriends DESC;
```

Regardless of whether the user appears in user1 or user2, we should be accounting for both.

For example, for two users who are friends in the form: (user1, user2)

- If both (3, 7) and (7, 10) are currently friends, then user 7 should be shown to have two friends.

This highlighted inner query finds how many friends each user1 has.

Use a similar query for user2.

From the users which have at least one friend, find the number of friends each user is currently friends with. Order your results from highest to lowest number of friends.
Return your results in the form (userId, numFriends)

```
SELECT userID, SUM(numFriends) as totalFriends
FROM
(
    (SELECT user1 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested IS NOT NULL
    AND WhenRejected IS NULL AND
    WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user1)

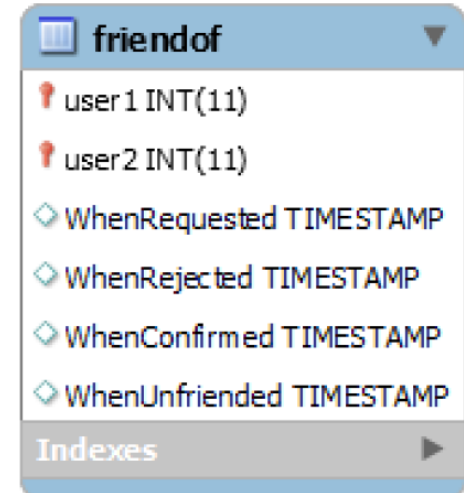
    UNION

    (SELECT user2 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested
    IS NOT NULL
    AND WhenRejected IS NULL
    AND WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user2)

) AS friendList

GROUP BY userID
ORDER BY totalFriends DESC;
```

This highlighted part just checks that this is a current, confirmed friend.



friendof	
user1	INT(11)
user2	INT(11)
WhenRequested	TIMESTAMP
WhenRejected	TIMESTAMP
WhenConfirmed	TIMESTAMP
WhenUnfriended	TIMESTAMP

Indexes

From the users which have at least one friend, find the number of friends each user is currently friends with. Order your results from highest to lowest number of friends.
Return your results in the form (userId, numFriends)

```
SELECT userID, SUM(numFriends) as totalFriends
FROM
(
    (SELECT user1 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested IS NOT NULL
    AND WhenRejected IS NULL AND
    WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user1)

    UNION

    (SELECT user2 AS userID, COUNT(*) as numFriends
    FROM friendof
    WHERE WhenRequested
    IS NOT NULL
    AND WhenRejected IS NULL
    AND WhenConfirmed IS NOT NULL
    AND WhenUnfriended IS NULL
    GROUP BY user2)

) AS friendList

GROUP BY userID
ORDER BY totalFriends DESC;
```

Then, we take a **union** to find out the total friends users have.

This allows us to account for a specific user appearing in either the user1 and user 2 columns.

After the union, the rows from the two tables are just combined into one table.

- The same userID might appear in two rows so we need to **GROUP BY userID**
- Also, need to **SUM** up the number of friends to find totalFriends for each user.

Then we order by totalFriends in a descending order as the question specifies

Normalisation Solution

Normalisation Solution:

staffNo	name	position	salary	branchNo	branchAddress	telNos
S1500	Tom Daniels	Manager	50000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 03-555-3618
S0003	Sally Adams	Assistant	30000	B001	8 Jefferson Way, Portland, OR 97201	503-555-3618, 03-555-3618
S0010	Mary Martinez	Manager	50000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 06-555-6756
S3250	Robert Chin	Supervisor	32000	B002	City Center Plaza, Seattle, WA 98122	206-555-6756, 06-555-6756
S2250	Sally Stern	Manager	50000	B004	16 – 14th Avenue, Seattle, WA 98128	206-555-3131, 06-555-3131
S0415	Art Peters	Manager	50000	B003	14 – 8th Avenue, New York, NY 10012	212-371-3000, 12-371-3000

Normalise this StaffBranch table into 3NF

The dependencies are:

- staffNo → name, position
- position → salary
- branchNo → branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

Remember: We must check if **ALL** **previous** normal forms are satisfied before trying to put it into 3NF

Which NF is not satisfied?

What do we do to fix this?

Achieving 1NF

Original Table is **StaffBranch**(staffNo PK, name, position, salary, branchNo PK, branchAddress, telNos)

Need to make telNo **part of the PK** to be able to store multiple telNos for each branch!

So we get:

StaffBranch(staffNo PK, name, position, salary, branchNo ?, branchAddress)
?(branchNo ?, telNos ?)

After making the table, think about what might be a good name for this table?

Which attribute should we make the PK and/or which one the FK?

Achieving 1NF

Original Table is **StaffBranch**(staffNo PK, name, position, salary, branchNo PK, branchAddress, telNos)

Need to make telNo **part of the PK** to be able to store multiple telNos for each branch!

So we get:

StaffBranch(staffNo PK, name, position, salary, branchNo PK, branchAddress)
?(branchNo PFK, telNo PK)

After making the table, think about what might be a good name for this table?

StaffBranch(staffNo PK, name, position, salary, branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

Now, it is in 1NF

Is this in 3NF yet?

StaffBranch(staffNo PK, name, position, salary, branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

The dependencies are:

- staffNo \rightarrow name, position
- position \rightarrow salary
- branchNo \rightarrow branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

Achieving 2NF

staffNo \rightarrow name, position

position \rightarrow salary

branchNo \rightarrow branchAddress, telNos

StaffBranch(staffNo PK, name, position, salary, branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

There are attributes that depend only on a PART of the PK (partial dependencies)

- staffNo \rightarrow name, position, salary
 - (by Armstrong's Transitivity Axiom: $A \rightarrow B$ and $B \rightarrow C \Rightarrow A \rightarrow C$)
- branchNo \rightarrow branchAddress

How do we remove these partial dependencies to put it into 2NF?

Achieving 2NF

staffNo → name, position

position → salary

branchNo → branchAddress, telNos

StaffBranch(staffNo PK, name, position, salary, branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

There are attributes that depend only on a PART of the PK (partial dependencies)

- staffNo → name, position, salary (by Armstrong's Transitivity Axiom)
- branchNo → branchAddress

How do we remove these partial dependencies to put it into 2NF?

Need to decompose staff details and branch details into their own relations!

StaffBranch(staffNo PFK, branchNo PFK)

?(staffNo PK, name, position, salary)

?(branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

Remember: We still need to keep the original table StaffBranch so that we still have information which links staff and branch (we still want to know which staff work at which branch)
What should we name relations?

Are we in 3NF yet?

StaffBranch(staffNo **PFK**, branchNo **PFK**)

Staff(staffNo **PK**, name, position, salary)

Branch (branchNo **PK**, branchAddress)

BranchTelephone(branchNo **PFK**, telNo **PK**)

Now all the relations are in 2NF

Are all the relations in 3NF? Why or why not?

The dependencies are:

- staffNo \rightarrow name, position
- position \rightarrow salary
- branchNo \rightarrow branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

Are we in 3NF yet?

StaffBranch(staffNo **PFK**, branchNo **PFK**)

Staff(staffNo **PK**, name, position, salary)

Branch (branchNo **PK**, branchAddress)

BranchTelephone(branchNo **PFK**, telNo **PK**)

Now all the relations are in 2NF

Are all the relations in 3NF? Why or why not?

Nope! We've got a **transitive dependency** in the **Staff** table.

A non-key attribute determines another non-key attribute (position \rightarrow salary)

Need to separate this out into its own relation

Staff(staffNo **PK**, name, position ?)

? (position ?, salary)

The dependencies are:

- staffNo \rightarrow name, position
- position \rightarrow salary
- branchNo \rightarrow branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

What should we name the table? Where do we put the FK and PKs?

Achieving 3NF

StaffBranch(staffNo PFK, branchNo PFK)

Staff(staffNo **PK**, name, position **FK**)

Position(position **PK**, salary)

Branch (branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

Are we done yet?

Are **all** the relations in 3NF?

The dependencies are:

- staffNo \rightarrow name, position
- position \rightarrow salary
- branchNo \rightarrow branchAddress, telNos

A candidate key is (StaffNo, BranchNo)

We've now achieved 3NF!

StaffBranch(staffNo PFK, branchNo PFK)

Staff(staffNo PK, name, position FK)

Position(position PK, salary)

Branch (branchNo PK, branchAddress)

BranchTelephone(branchNo PFK, telNo PK)

Transactions solutions

Transactions

Question:

A DBMS which strictly implements ACID compliant transactions can never encounter the 'Inconsistent retrieval problem'.

Which ACID property is primarily responsible for preventing this problem? Explain how it prevents the inconsistent retrieval problem.

Transactions

Alice	Bob
SELECT SUM(Salary) FROM Employee;	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 33;
	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 44;
(finishes calculating sum)	COMMIT;

Transactions

Time	Trans- action	Action	Value	T1 SUM	Comment
1	T1	Read Salary for EmpID 11	10,000	10,000	
2	T1	Read Salary for EmpID 22	20,000	30,000	
3	T2	Read Salary for EmpID 33	30,000		
4	T2	Salary = Salary * 1.01			
5	T2	Write Salary for EmpID 33	30,300		
6	T1	Read Salary for EmpID 33	30,300	60,300	<i>after update</i>
7	T1	Read Salary for EmpID 44	40,000	100,300	<i>before update</i>
8	T2	Read Salary for EmpID 44	40,000		<div>we want either <i>before</i> \$210,000 or <i>after</i> \$210,700</div>
9	T2	Salary = Salary * 1.01			
10	T2	Write Salary for EmpID 44	40,400		
11	T2	COMMIT			
12	T1	Read Salary for EmpID 55	50,000	150,300	
13	T1	Read Salary for EmpID 66	60,000	210,300	

Transactions

A:

C:

I:

D:

Transactions

A: Atomic

C: Consistent

I: Isolated

D: Durable

Solution:

Isolation

Isolation guarantees that concurrent execution of transactions leaves the database in the same state as if the transactions were executed sequentially (one after another)

Transactions

Time



Alice's Transaction

Bob's Transaction



Correct
(no issue
with
Inconsistent
Retrieval
problem)

OR

Bob's Transaction

Alice's Transaction



Incorrect
(Inconsistent
Retrieval)

Bob's Transaction

Alice's Transaction



Distributed Solutions

Distributed DBs

Question:

As business grew over time for hiTech Computer Repairs Inc, the company has opened multiple new stores in Australia, and also recently spread to Europe. The management at hiTech Headquarters is interested in upgrading the database associated with the `PC repair services` booking system from centralised to a distributed structure. HiTech repair centers in Australia are unlikely to regularly need access to repair records for European customers, and vice versa.

Which of the following type of distributed database would be best suited for this case: **a) Vertically partitioned b) Horizontally Partitioned c) Fully Replicated DB**

Distributed DBs

Answer:

Horizontally Partitioned.

- Same type of data being accessed (ie from same table) for the 'service repairs', BUT:
- Rows that are for the 'Australian' stores are not likely to be used by the 'European' stores.
- Split the European rows into a European server, and keep Australian rows on the Australian server

NoSQL solutions

No SQL

Question:

FaceTube™ is a new online worldwide social video sharing platform that you and your friend have started.

Your friend asks if you think a No-SQL based database for storing the metadata + comments for video uploads is a good idea.

Discuss some pros/cons for using a No-SQL database for this scenario. Your answer should make reference to the properties of the data being stored and BASE.

No SQL

Answer:

Yes, NoSQL is likely a good choice here

Pros:

- High availability when partitioned (BA)
- *OK if not completely consistent (BA), as long as other users will eventually (e.g. in a few minutes) see new comments etc (S + E)*
- Can deal with large number of comments in rapid succession

Cons:

- Harder to perform data analysis
- Potentially takes up marginally more storage space

Questions & Answers