Revision Lecture

Database Systems

Michael Pound

Revision Lectures

- Exam Overview
  - Structure
  - Exam Techniques
- Transactions and Schedules
- Two-Phased Locking Protocol
- Timestamping Protocol
- E/R Diagrams from Problem Specifications
- Normalisation

The Exam Structure

- The exam will contain ***FIVE*** questions. You must answer ***ANY THREE*** of these
- Question 1 will be a general question. It will contain a number of smaller questions that can be about anything on the course
- The remaining questions will focus on one or two topics.
- If you answer more than three questions, we will mark the first three only!
- If you answer more than three questions cross out the ones that you ***DON'T*** want us to mark

In the Exam

- Apart from a couple of exceptions, any topic from any lecture might be on the exam, including:
  - Relational Algebra
  - E/R Diagrams
  - SQL Data Definition
  - SQL SELECT (queries)
  - NULLs
  - Normalization
  - Transactions and Schedules
  - Locking and timestamps
  - Database Security, including Privileges

Not In The Exam

- The following are definitely ***not*** in the exam:
  - PHP code
  - SQL Injection Attacks
  - Modern Databases e.g. BigTable, OODBMSs etc.

Past Exam Papers

- Some papers (not answers unfortunately) are available on the University Portal
- To obtain these:
  - Login to my.nottingham.ac.uk with your IS (not CS) username and password.
  - Go to the "Library" tab, and find the link to "Exam Papers"
  - Search for DBS
- I will be doing some questions in these lectures – ask if there's a specific question you'd like covered

Exam Techniques

- Everyone approaches exams differently. Here are some things you might like to consider though:
  - Pay attention to how many marks a question is worth. Don't write a page for a 4 mark question
  - Have a scan through the exam questions before you start answering them
  - You are answering three questions in 2 hours, which means roughly 40 minutes per question

Exam Techniques

- Be concise, don't write more than you have to.
- For example:

Explain the lost update problem with respect to database concurrency.
(2 Marks)

Example concise answer: The lost update problem occurs when two concurrent transactions update the same resource in a database. The actions of the first transaction are lost when the value is overwritten by the second transaction.

Example non-concise answer: The lost update problem is related to database concurrency. Sometimes two transactions may need to read and write from the same resource. They may also need to operate concurrently. When this occurs, the write action of the first transaction might be lost when a second transaction overwrites this value. The lost update problem can be prevented using a locking or timestamping protocol…

SQL Exam Techniques

- Answering SQL questions is a little different
- Try to avoid worrying about where marks are allocated and instead aim to write a correct query
- Try to organise your query neatly so the marker can see what you've done
  - Don't forget brackets and semi-colons!
- There are often many ways to answer each SQL SELECT question, so focus on correctness
- Consider concision. E.g.
  - SELECT DISTINCT artName FROM Artist, CD, Track;
  - Is not a concise answer (You only need select from Artist)

Marking Scheme

- The exam has a very specific marking scheme. This means:
  - You will not get any marks for points unrelated to the question
  - You will miss marks if you leave something out, even if the rest of your answer is good.

Transactions and Concurrency Revision

Transaction

- Transactions are the 'logical unit of work' in a database
  - ACID properties
  - Also the unit of recovery
- It would be helpful to run many transactions at the same time
  - Many users
  - Possibly very long transactions
- Challenges with running transactions concurrently
  - Lost update
  - Uncommitted update
  - Inconsistent analysis
- The ACID properties are violated

ACID

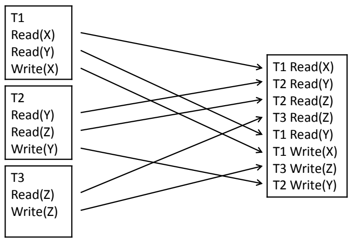
- Atomicity
  - Transactions are Atomic
  - Conceptually they do not have component parts
  - Transactions are either executed fully, or not at all
- Consistency
  - Transactions take the data base from one consistent state to another
  - Consistency isn't guaranteed mid-way through a transaction
- Isolation
  - All transactions execute independently of one another
  - The effects of an incomplete transaction are invisible to other transactions
- Durability
  - Once a transaction has completed, it is made permanent
  - Must be durable even after a system crash

Schedules

- A *schedule* is a sequence of the operations in a set of concurrent transactions that preserves the order of operations in each of the individual transactions
- A *serial* schedule is a schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions (each must commit before the next can begin)

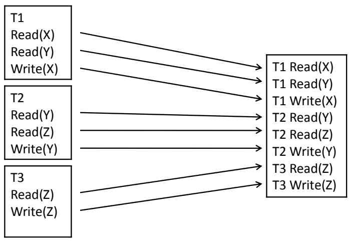
Example Schedule

- Three transactions:
- Example schedule



Example Schedule

- Three transactions:
- Example serial schedule

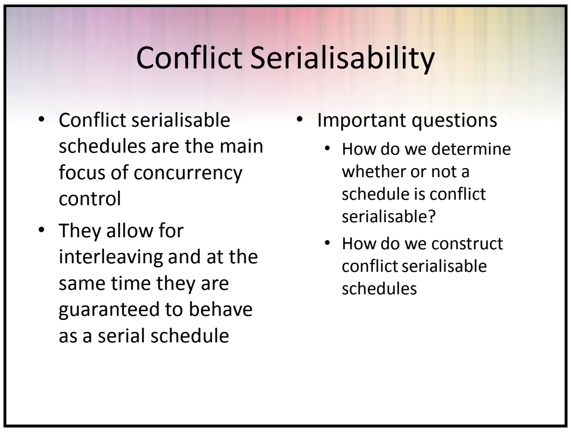


Serialisability

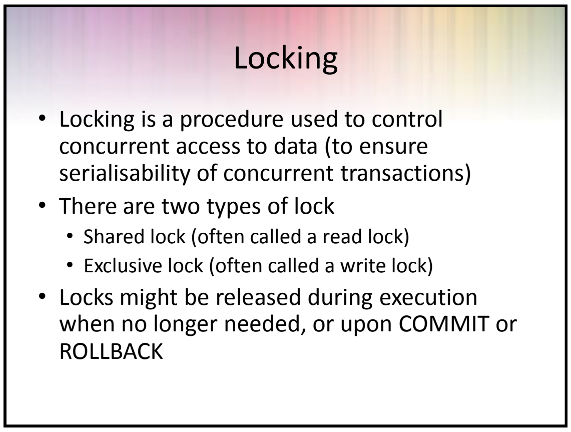
- Two schedules are equivalent if they always have the same effect
- A schedule is *serialisable* if it is equivalent to some serial schedule
- For example:
  - If two transactions only read from some data items, the order in which they do this is not important
  - If T1 reads and then updates X, and T2 reads then updates Y, then again this can occur in any order

Conflict Serialisability

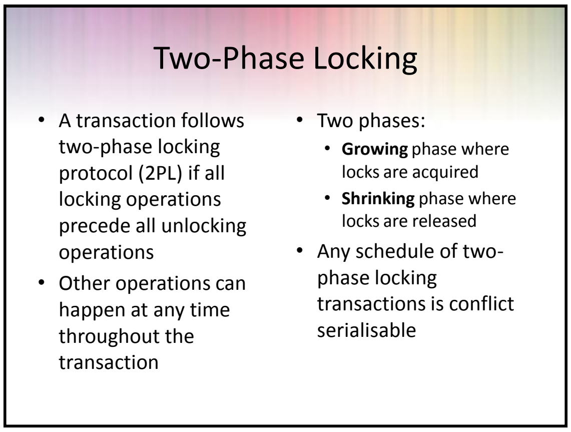
- Two transactions have a confict:
  - NO If they refer to different resources
  - NO If they only read
  - YES If at least one is a write and they use the same resource
- A schedule is conflict serialisable if the transactions in the schedule have a conflict, but the schedule is still serialisable


Conflict Serialisability

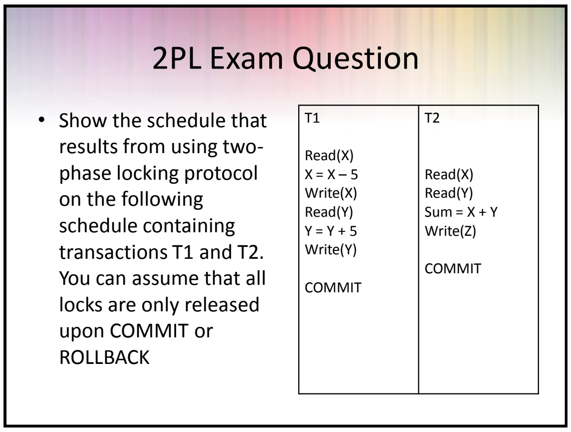
- Conflict serialisable schedules are the main focus of concurrency control
- They allow for interleaving and at the same time they are guaranteed to behave as a serial schedule
- Important questions
  - How do we determine whether or not a schedule is conflict serialisable?
  - How do we construct conflict serialisable schedules


Locking

- Locking is a procedure used to control concurrent access to data (to ensure serialisability of concurrent transactions)
- There are two types of lock
  - Shared lock (often called a read lock)
  - Exclusive lock (often called a write lock)
- Locks might be released during execution when no longer needed, or upon COMMIT or ROLLBACK

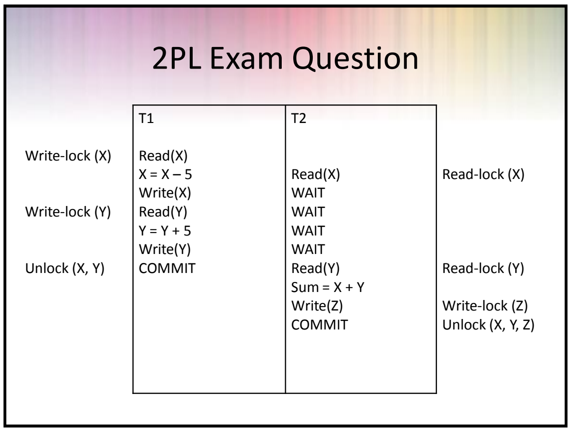

Two-Phase Locking

- A transaction follows two-phase locking protocol (2PL) if all locking operations precede all unlocking operations
- Other operations can happen at any time throughout the transaction
- Two phases:
  - **Growing** phase where locks are acquired
  - **Shrinking** phase where locks are released
- Any schedule of two-phase locking transactions is conflict serialisable

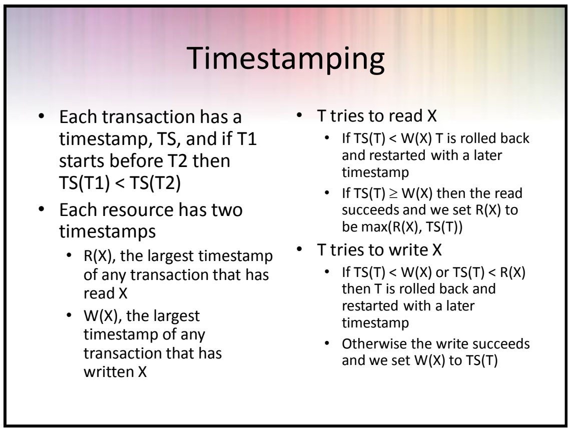

2PL Exam Question

- Show the schedule that results from using two-phase locking protocol on the following schedule containing transactions T1 and T2. You can assume that all locks are only released upon COMMIT or ROLLBACK

T1	T2
Read(X)	
X = X – 5	Read(X)
Write(X)	Read(Y)
Read(Y)	Sum = X + Y
Y = Y + 5	Write(Z)
Write(Y)	
	COMMIT
COMMIT	


2PL Exam Question

	T1	T2	
Write-lock (X)	Read(X)		
	X = X – 5	Read(X)	Read-lock (X)
	Write(X)	WAIT	
Write-lock (Y)	Read(Y)	WAIT	
	Y = Y + 5	WAIT	
	Write(Y)	WAIT	
Unlock (X, Y)	COMMIT	Read(Y)	Read-lock (Y)
		Sum = X + Y	
		Write(Z)	Write-lock (Z)
		COMMIT	Unlock (X, Y, Z)


Timestamping

- Each transaction has a timestamp, TS, and if T1 starts before T2 then TS(T1) < TS(T2)
- Each resource has two timestamps
  - R(X), the largest timestamp of any transaction that has read X
  - W(X), the largest timestamp of any transaction that has written X
- T tries to read X
  - If TS(T) < W(X) T is rolled back and restarted with a later timestamp
  - If TS(T) ≥ W(X) then the read succeeds and we set R(X) to be max(R(X), TS(T))
- T tries to write X
  - If TS(T) < W(X) or TS(T) < R(X) then T is rolled back and restarted with a later timestamp
  - Otherwise the write succeeds and we set W(X) to TS(T)

Timestamping Exam Question

2008-2009 Paper
5. (g)

Trace the timestamping protocol for the following two transactions T1 and T2, assuming that the statements are executed in a strictly alternating way (First statement of T1 followed by the first statement of T2 followed by the second statement of T1, and so on) and there are no other transactions. At each step, indicate what the time stamps of T1 and T2 are, and what the read and write timestamps of resources X, Y are. Assume that before T1 and T2 are executed the timestamps of resources are 0. Trace until both transactions can commit.

(5 Marks)

Transaction 1	Transaction 2
Write(X)	Read(Y)
Write(Y)	Write(X)

Timestamp Example

→ T1	→ T2
Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	0	0	0

	T1	T2
TS	1	2

Start

Timestamp Example

→ T1	→ T2
Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	0	0	0

	T1	T2
TS	1	2

Start

T1	→ T2
→ Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	1	0	0

	T1	T2
TS	1	2

Timestamp Example

→ T1	→ T2
Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	0	0	0

	T1	T2
TS	1	2

Start

T1	→ T2
→ Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	1	0	0

	T1	T2
TS	1	2

T1	T2
→ Write(X)	→ Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	2	0
W	1	0	0

	T1	T2
TS	1	2

Timestamp Example

→ T1	→ T2
Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	0	0	0

	T1	T2
TS	1	2

Start

T1	→ T2
→ Write(X)	Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	0	0
W	1	0	0

	T1	T2
TS	1	2

T1	T2
→ Write(X)	→ Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	2	0
W	1	0	0

	T1	T2
TS	1	2

→ T1	T2
Write(X)	→ Read(Y)
Write(Y)	Write(X)

	X	Y	Z
R	0	2	0
W	1	0	0

	T1	T2
TS	3	2

T1 Restarted

Timestamp Example

Answer Continued...

→ T1	T2
Write(X)	Read(Y)
Write(Y)	→ Write(X)

	X	Y	Z
R	0	2	0
W	2	0	0

	T1	T2
TS	3	2

T2 Completed

Timestamp Example

Answer Continued...

```
→T1          T2
 Write(X)   Read(Y)
 Write(Y)→Write(X)
```

	X	Y	Z
R	0	2	0
W	2	0	0

	T1	T2
TS	3	2

T2 Completed

```
 T1          T2
→Write(X)   Read(Y)
 Write(Y)→Write(X)
```

	X	Y	Z
R	0	2	0
W	3	0	0

	T1	T2
TS	3	2

Timestamp Example

Answer Continued...

```
→T1          T2
 Write(X)   Read(Y)
 Write(Y)→Write(X)
```

	X	Y	Z
R	0	2	0
W	2	0	0

	T1	T2
TS	3	2

T2 Completed

```
 T1          T2
→Write(X)   Read(Y)
 Write(Y)→Write(X)
```

	X	Y	Z
R	0	2	0
W	3	0	0

	T1	T2
TS	3	2

```
 T1          T2
 Write(X)   Read(Y)
→Write(Y)→Write(X)
```

	X	Y	Z
R	0	2	0
W	3	3	0

	T1	T2
TS	3	2

T1 Completed

E/R Diagrams

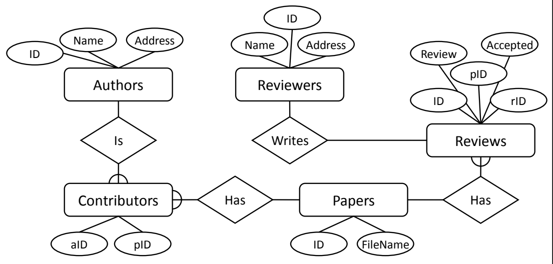
- You might be asked to draw an E/R diagram based on a problem specification. If you are, you should identify and draw:
  - Entities
    - Often nouns, things with component parts that will become tables in your database
  - Attributes
    - All the component parts of the entities you've identified
  - Relationships
    - Remember to include a name and cardinality ratio, and remove M:M relationships with an intermediate entity

E/R Diagram Exam Question

2009-2010 Paper
Question 2

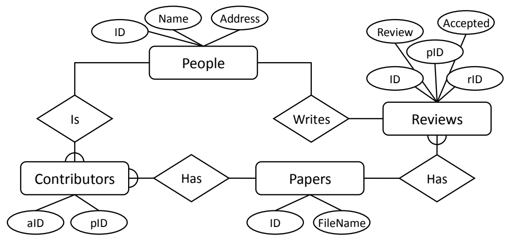
E/R Diagram Exam Question

The completed diagram with M:M relationships removed



E/R Diagram Exam Question

A better answer would probably be to have a single entity for authors and reviewers – their attributes are the same



E/R Diagram Exam Question

(c) Write an SQL query which returns the list of IDs of accepted papers, without repetitions.

- There are many ways to answer this question
  - Two possibilities are subqueries to find lists of either accepted or rejected papers, and use ALL, IN, NOT IN etc.

E/R Diagram Exam Question

```
SELECT pID FROM Papers
WHERE 'Yes' = ALL
 (SELECT Accept FROM Review
  WHERE Review.pID = Papers.pID);

SELECT pID FROM Papers
WHERE pID NOT IN
 (SELECT DISTINCT Review.PID
  FROM Review
  WHERE Accept = 'No');
```

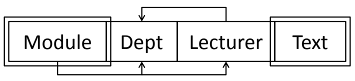
Normalisation Revision

Functional Dependencies

- Normalisation is the process of removing redundancy in a database
- Redundancy is often caused by functional dependencies
- If some set of attributes A functionally determine some set B (A $\rightarrow$ B), then for every combination of values in A, we will always have the same combination of values in B.

FD Diagrams

- FDs can be represented simply using the attribute names:



- {Module , Text} is a candidate key, so we put a double box around them
- {Lecturer} $\rightarrow$ {Dept}, so we have an arrow from Lecturer to Dept
- {Module} $\rightarrow$ {Dept} and {Module} $\rightarrow$ {Lecturer} , so we have
  {Module} $\rightarrow$ {Dept, Lecturer}

Note: Trivial FDs and FDs dependent on an entire candidate key are not included

1NF

- 1NF – Removes all non-atomic values
- E.g.

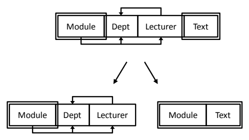
CustomerID	OrderDate	OrderNumber
321	11/03/11, 12/03/11	1101225,1101229
135	14/03/11	1101331
947	14/03/11, 15/03/11	1101303, 1101541

$\downarrow$

CustomerID	OrderDate	OrderNumber
321	11/03/11	1101225
321	12/03/11	1101229
135	14/03/11	1101331
947	14/03/11	1101303
947	15/03/11	1101541

2NF

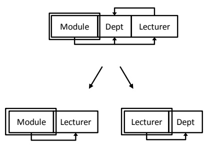
- 2NF – Removes partial FDs e.g.



- For some FD A → B where
  - A is the set of attributes on the left of the FD
  - B is the set of attributes on the right of the FD
  - C is all other attributes
- Create two new relations
  - A ∪ B and A ∪ C

3NF

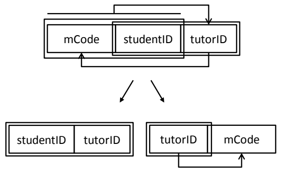
- 3NF – Removes transitive FDs e.g.



- For some FD A → B → C where
  - A is the set of attributes on the left of the FD
  - B is the intermediate attributes
  - C is the right hand side
  - D is all other attributes
- Create two new relations
  - B ∪ C and A ∪ B ∪ D

BCNF

- BCNF – Removes partial FDs where the dependant attributes can also be keys e.g.



- For some FD that violates BCNF A → B where
  - A is the set of attributes on the left of the FD
  - B is the right hand side
  - C is the right hand side
- Create two new relations
  - A ∪ B and A ∪ C

Normalisation Exam Question

2009-2010 Paper
4.
(b)
List all non-trivial functional dependencies in the relation Person (ID, FirstName, LastName, Nationality, EU) where ID is unique, and EU has a value yes or no depending on whether the person's nationality is in a country belonging to the EU.

(5 Marks)

(c)
Is the table in part (b) in BCNF? Explain your answer. If the table is not in BCNF, decompose it to BCNF

(10 Marks)

Normalisation Exam Question

(b)

- You might like to begin by drawing out an FD diagram, although this has not specifically been asked for
- These are the attributes identified in the question:

ID	FirstName	LastName	Nationality	EU

Normalisation Exam Question

(b)

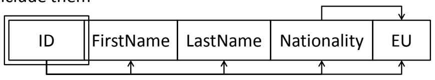
- ID is unique, so is a candidate key. It's unlikely any other combination of attributes are also unique and minimal

ID	FirstName	LastName	Nationality	EU

Normalisation Exam Question

(b)

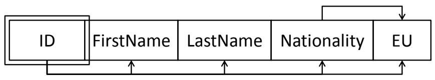
- Non-trivial FDs are those A → B where B is not a subset of A. This question asks for ***all*** non-trivial FDs, do not omit those that are dependent on an entire candidate key
- This question is a bit confusing with regard to dependencies on an entire candidate key. They're basically implied, but aren't strictly speaking a trivial FD. We should probably include them



Normalisation Exam Question

(b)

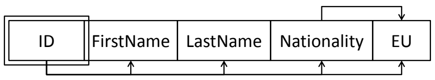
- Non-trivial FDs:
  - {ID} → {FirstName, LastName, Nationality, EU}
  - {ID} → ... All combinations
  - {ID} → {Nationality}
  - {Nationality} → {EU}



Normalisation Exam Question

(c)

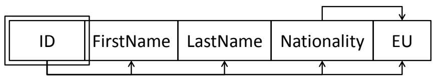
- We need to determine if the relation below is in BCNF. Remember that each normal form has the previous as a prerequisite.



Normalisation Exam Question

(c)

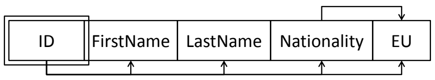
- We will assume the relation is in 1NF, since it seems to have no non-atomic values
- Is the relation in 2NF?



Normalisation Exam Question

(c)

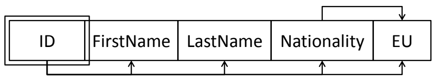
- A relation is in 2NF if there are no non-key attributes partially dependent on a candidate key
- In this case, {ID} is the only member of a candidate key, so there are no partial dependencies. This means the relation is in 2NF



Normalisation Exam Question

(c)

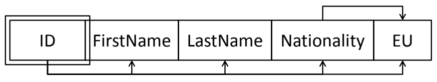
- Is the relation in 3NF?



Normalisation Exam Question

(c)

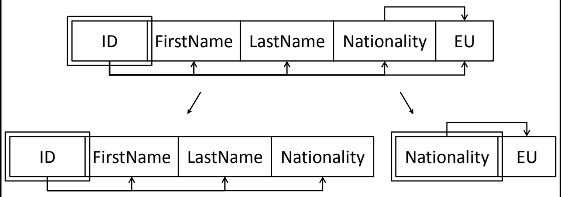
- A relation is in 3NF if there are no non-key attributes transitively dependent on a primary key
- In this case, {ID} → {Nationality} → {EU} breaches 3NF



Normalisation Exam Question

(c)

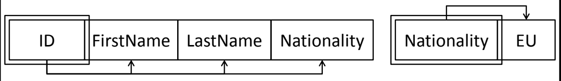
- To normalise, we split into B ∪ C and A ∪ B ∪ D



Normalisation Exam Question

(c)

- A relation is in BCNF if there are no non-trivial functional dependencies where a set of attributes B is dependent on a non-superkey. This is the same as 3NF, but B can be a key attribute
- These relations are already in BCNF



SQL SELECT Revision

Answering SELECT Questions

- Focus mainly on correctness, then clarity, then conciseness
- Split a difficult question into smaller parts, which could be sub queries
- Avoid excessive use of joins, although note that there's nothing wrong with using joins in general
- Give the question a go, you may pick up some marks for good syntax etc.

SELECT Exam Question

2008-2009 Paper
3.

This question refers to the following tables: *Children*, *Playgroups*, *Activities*. The *Children* table contains data about children (names, ages and addresses of parents) – we assume for simplicity that names are unique. *Playgroups* says which child is in which playgroup and *Activities* says what children in the playgroup did on a certain date (for example, went to a zoo).

Playgroups	
PlaygroupID	Name

Children		
Name	Age	Address

Activities		
PlaygroupID	ADate	Description

SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(a) Find a list of names of all children. (1 mark)

```
SELECT Name from Children;
```

SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(b) Find a list of names of all children aged 4. (2 marks)

```
SELECT Name from Children
 WHERE Age = 4;
```

SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(c) Return a list of names and addresses for all children in the playgroup with ID equal to 1 (3 marks)

- This question requires either a join, I'll use an INNER JOIN
- Or a subquery to first find the names of the children. Either approach would be worth the same amount of marks
- This is made easier because we've been told names are unique

SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(c) Return a list of names and addresses for all children in the playgroup with ID equal to 1 (3 marks)

```
SELECT Name, Address
FROM Children INNER JOIN Playgroups USING (Name)
WHERE PlaygroupID = 1;

SELECT Name, Address FROM Children
WHERE Name IN (SELECT Name FROM Playgroups
               WHERE PlaygroupID = 1);
```

SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(d) Find a list of names and ages of children in a playgroup which went to the zoo on the 21st of February 2009 (check for Activities.Description value zoo). (5 marks)

- This question is much easier using a subquery, do the question in two parts
- First you find the PlaygroupID in the Activities table
- Then we obtain the children's names and ages using a join or another subquery as with the last question.

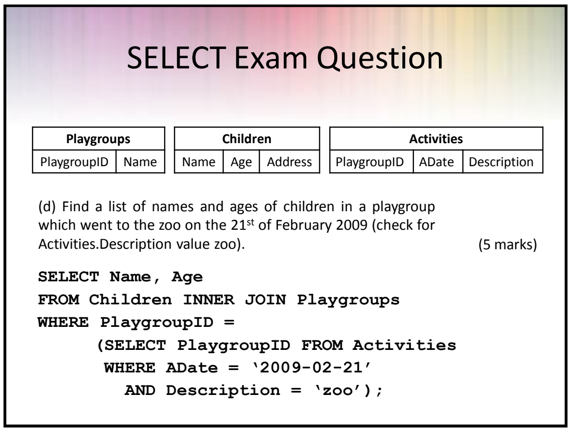
SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(d) Find a list of names and ages of children in a playgroup which went to the zoo on the 21st of February 2009 (check for Activities.Description value zoo). (5 marks)

- Subquery to find the correct PlaygroupID:

```
SELECT PlaygroupID FROM Activities
WHERE ADate = '2009-02-21'
  AND Description = 'zoo';
```

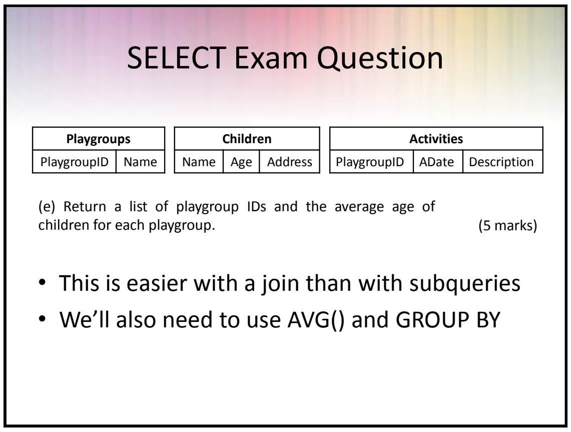


SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(d) Find a list of names and ages of children in a playgroup which went to the zoo on the 21st of February 2009 (check for Activities.Description value zoo). (5 marks)

```
SELECT Name, Age
FROM Children INNER JOIN Playgroups
WHERE PlaygroupID =
      (SELECT PlaygroupID FROM Activities
       WHERE ADate = ‘2009-02-21’
         AND Description = ‘zoo’);
```

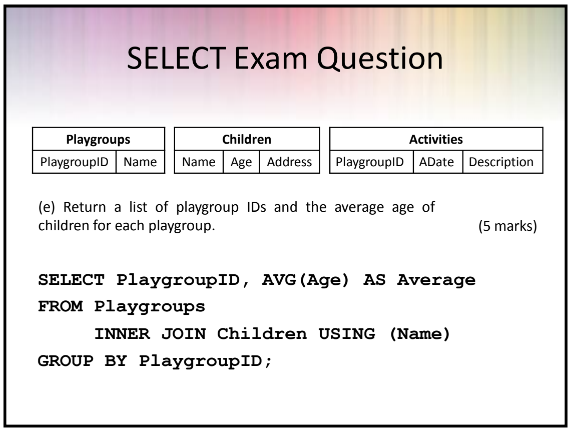


SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(e) Return a list of playgroup IDs and the average age of children for each playgroup. (5 marks)

- This is easier with a join than with subqueries
- We’ll also need to use AVG() and GROUP BY

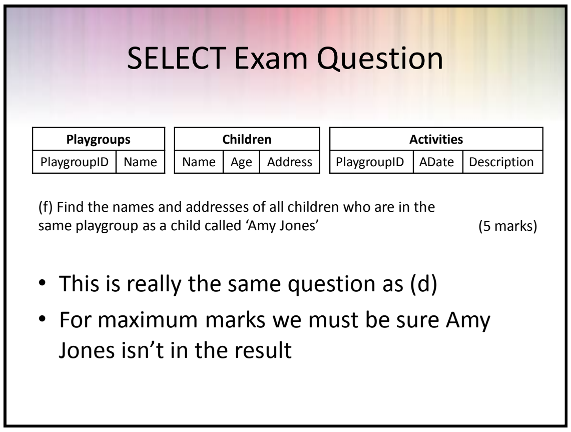


SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(e) Return a list of playgroup IDs and the average age of children for each playgroup. (5 marks)

```
SELECT PlaygroupID, AVG(Age) AS Average
FROM Playgroups
     INNER JOIN Children USING (Name)
GROUP BY PlaygroupID;
```

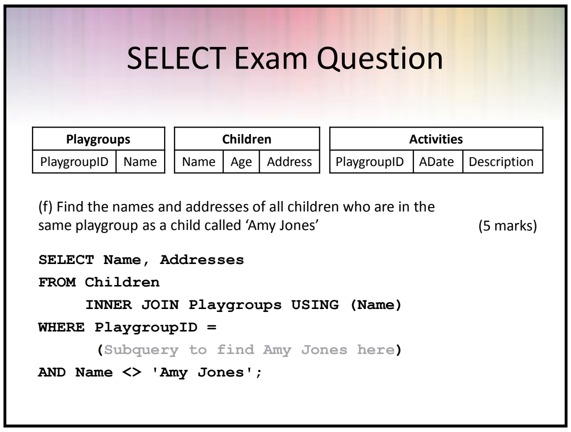


SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(f) Find the names and addresses of all children who are in the same playgroup as a child called ‘Amy Jones’ (5 marks)

- This is really the same question as (d)
- For maximum marks we must be sure Amy Jones isn’t in the result

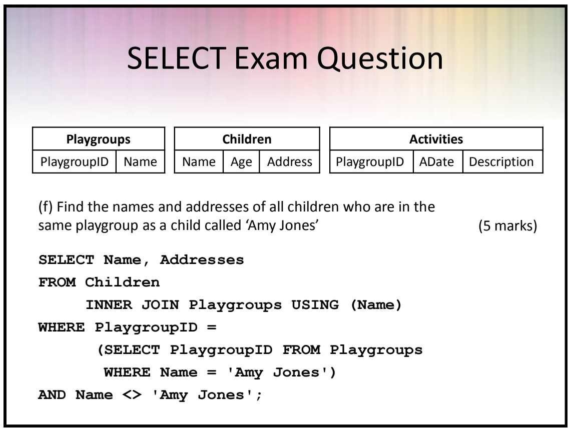


SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(f) Find the names and addresses of all children who are in the same playgroup as a child called ‘Amy Jones’ (5 marks)

```
SELECT Name, Addresses
FROM Children
     INNER JOIN Playgroups USING (Name)
WHERE PlaygroupID =
      (Subquery to find Amy Jones here)
AND Name <> 'Amy Jones';
```



SELECT Exam Question

Playgroups		Children			Activities		
PlaygroupID	Name	Name	Age	Address	PlaygroupID	ADate	Description

(f) Find the names and addresses of all children who are in the same playgroup as a child called ‘Amy Jones’ (5 marks)

```
SELECT Name, Addresses
FROM Children
     INNER JOIN Playgroups USING (Name)
WHERE PlaygroupID =
      (SELECT PlaygroupID FROM Playgroups
       WHERE Name = 'Amy Jones')
AND Name <> 'Amy Jones';
```