Exam revision

Database Systems Lecture 18

Natasha Alechina

In This Lecture

- Exam format
- Main topics
- How to revise

Exam format

- Answer three questions out of five
- Each question is worth 25 points
- I will only mark three questions in the order you answer them
- (So cross out any answers you don't want marked)
- Final mark for the module is your coursework mark (at most 25) plus your exam mark (at most 75).

Main topics

- What is a database, what is a DBMS, data manipulation language, data definition language, data control language
- Relational model
  - Relations, attributes, domains
  - candidate keys, primary keys, foreign keys, entity integrity, referential integrity)
- Relational algebra
- SQL (the kind of questions you did in cw2, cw3, cw4)
- Normalisation (1NF, 2NF, 3NF, BCNF)
- Security, privileges (how to grant and revoke them)
- Transactions, recovery
- Concurrency

How to revise

- Do all the exercises, then check the model solutions
- Remember SQL syntax – you will have to write SQL queries in the exam
- Look at the previous exam papers (for G51DBS06-07, G51DBS07-08, G51DBS08-09 and G52DBS)
- Exam for last year and answers are now on the web
- If you get stuck with some previous exam paper questions, send me an email – I will either answer by email or, if I get a lot of similar questions, arrange a tutorial

Particular topics

- Normalisation
- Relational Algebra
- Concurrency control: 2PL and Timestamping

Normalisation

- General idea: given a relation R
  - Find all candidate keys in R
  - Find all non-trivial functional dependencies in R
  - Decomposing to XNF: for every functional dependency A → B which is "bad" with respect to XNF, decompose R into π_{AB} (R) and π_{AC} (R) where C is the rest of R's attributes.

Candidate keys

- A set of attributes A (can be a singleton set) is a *candidate key* for relation R if it has properties of
  - Uniqueness: no two different tuples in R can have the same values for attributes in A
  - Minimality: no subset of A has the uniqueness property
- A set of attributes is a *superkey* if it includes a candidate key
- We call an single attribute a *key attribute* if it is part of a candidate key.

Example (coursework 5)

- (Cinema,Film,Day,Time) is a candidate key:
  - It uniquely identifies each tuple: there are no two tuples which agree on these four attributes and have different values for other attributes (in this case, there is only one other attribute Certificate).
  - It is minimal (removing one of the attributes with make the resulting set not unique).
- (Cinema, Film, Day, Time) and (Cinema, Film, Day, Time, Certificate) are superkeys.
- Cinema is a key attribute. Certificate is a non-key attribute.

Functional dependencies

- A, B are sets of attributes of relation R.
- A *functional dependency* A → B holds for R if for every two tuples in R, if they have the same values for A, then they have the same values for B.
- *Non-trivial* functional dependency: A → B is non-trivial if B is not a subset of A.

Example (coursework 5)

- {Film} → {Certificate} is a non-trivial functional dependency
- {Cinema, Film} → {Certificate} is also a non-trivial functional dependency
- So is {Film} ∪ X → {Certificate} for any X ⊆ {Cinema,Day,Time}
- {Film, Certificate} → {Certificate} is a trivial functional dependency

"Bad" functional dependencies

- For 2NF: A → B where A is a part of a candidate key and B is a non-key attribute (so, a table is in 2NF if it has no such dependencies)
- For 3NF: A → B, B → C where C is a not a key attribute

(alternative definition of 3NF: bad fd A → B is where A is not a superkey and B is non-key attribute)

- For BCNF: non-trivial A → B where A is not a superkey.
- Example of 3NF but not BCNF: R(A,B,C), candidate keys (A,B) and (A,C), fd B → C.

Example (coursework 5)

- Relation Listing is not in 2NF because it has a functional dependency {Film} → {Certificate} where Film is part of a candidate key and Certificate is a non-key attribute.

Revision of relational algebra

- Operations: $\cup$ (union), $-$ (difference), $\times$ (product), π (projection), σ (selection)
- Other operations are definable using the ones above: $\cap$ (intersection), $\bowtie$ (natural join – can be defined using $\times$, σ and π)

Revision of relational algebra

- Union-compatible relations: same number of attributes/columns, with the same domains
- For *named* relations (as SQL tables, where columns have names), also the names of the attributes/columns should be the same

Example: not union-compatible

Name	email	telephone
Bob	bbb	222222
Chris	ccc	333333

Name	DOB
Sam	1985
Steve	1986

Example: not union-compatible

(different domains for the second column)

Name	Email (domain: char(3))
Bob	bbb
Chris	ccc

Name	DOB (domain: int)
Sam	1985
Steve	1986

Example: union-compatible

Name	DOB
Bob	1971
Chris	1972

Name	DOB
Sam	1985
Steve	1986

Union of two relations

- Let R and S be two union-compatible relations. Then their union $R \cup S$ is a relation which contains tuples from both relations:

$R \cup S = \{x: x \in R \text{ or } x \in S\}$.

Example: shopping lists

R

Name	Price
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S

Name	Price
Cream	5.00

$R \cup S$

Name	Price
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00
Cream	5.00

Difference of two relations

Let R and S be two union-compatible relations. Then their *difference* $R - S$ is a relation which contains tuples which are in R but not in S:

$R - S = \{x: x \in R \text{ and } x \notin S\}$.

Example

R

Name	Price
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S

Name	Price
Soap	1.00

$R - S$

Name	Price
Milk	0.80
Bread	0.60
Eggs	1.20

(Extended Cartesian) product of relations

A relation containing all tuples of the form:

<tuple from R, tuple from S>:

$R \times S = \{<c_1,\ldots,c_n, c_{n+1},\ldots,c_{n+m}>:$

$<c_1, \ldots,c_n> \in R, \; <c_{n+1},\ldots,c_{n+m}> \in S\}$

(this assumes R has n columns and S has m columns)

Example

R

Name	Price
Milk	0.80
Bread	0.60
Eggs	1.20
Soap	1.00

S

Name	Calories
Milk	200
Bread	300

R×S

Name	Price	Name	Cal
Milk	0.80	Milk	200
Bread	0.60	Milk	200
Eggs	1.20	Milk	200
Soap	1.00	Milk	200
Cheese	1.34	Bread	300
Milk	0.80	Bread	300
Bread	0.60	Bread	300
Eggs	1.20	Bread	300
Soap	1.00	Bread	300

Projection

- Let R be a relation with n columns, and X is a set of column names. Then *projection of R on X* is a new relation $\pi_X(R)$ which only has columns from X.

Example: $\pi_{Name,Telephone}(R)$

R:

Name	Email	Telephone
Bob	bbb@cs.nott.ac.uk	0115222222
Chris	ccc@cs.nott.ac.uk	0115333333

Example: $\pi_{Name,Telephone}(R)$

$\pi_{Name,Telephone}(R)$:

Name	Telephone
Bob	0115222222
Chris	0115333333

Selection

- Let R be a relation and α is a property of tuples from R.
- *Selection from R subject to condition* α is defined as follows:

$\sigma_{\alpha}(R) = \{<a_1,\ldots,a_n> \in R: \alpha(a_1,\ldots,a_n)\}$

Example: selection

- $\sigma_{\text{Year} < 2002 \text{ and Director} = \text{Nolan}}(R)$

R

Title	Director	Year
Insomnia	Nolan	2002
Magnolia	Anderson	1999
Insomnia	Skjoldbjaerg	1997
Memento	Nolan	2000
Gattaca	Niccol	1997

Example: selection

- $\sigma_{\text{Year} < 2002 \text{ and Director} = \text{Nolan}}(R)$

Title	Director	Year
Memento	Nolan	2000

Example (small) exam question

- What is the result of $\pi_{R.Name,S.Name}(\sigma_{R.Tel = S.Tel}(R \times S))$, where R and S are:

R

Name	Tel
Anne	111111
Bob	222222

S

Name	Tel
Chris	333333
Dan	111111

Example exam question

$R \times S$

R.Name	R.Tel	S.Name	S.Tel
Anne	111111	Chris	333333
Anne	111111	Dan	111111
Bob	222222	Chris	333333
Bob	222222	Dan	111111

Example exam question

$\sigma_{R.Tel = S.Tel}(R \times S)$

R.Name	R.Tel	S.Name	S.Tel
Anne	111111	Dan	111111

Example exam question

$\pi_{R.Name,S.Name}(\sigma_{R.Tel = S.Tel}(R \times S))$

R.Name	S.Name
Anne	Dan

Revision of concurrency

- Transactions running concurrently may interfere with each other, causing various problems (lost updates etc.)
- Concurrency control: the process of managing simultaneous operations on the database without having them interfere with each other.

Lost Update

	T1	T2	
	`Read(X)`		
	`X = X - 5`		
		`Read(X)`	
		`X = X + 5`	
This update Is lost	`Write(X)`		
		`Write(X)`	Only this update succeeds
	`COMMIT`		
		`COMMIT`	

Uncommitted Update ("dirty read")

T1	T2
Read(X)	
X = X - 5	
Write(X)	
	Read(X) ← This reads the value of X which it should not have seen
	X = X + 5
	Write(X)
ROLLBACK	
	COMMIT

Inconsistent analysis

T1	T2
Read(X)	
X = X - 5	
Write(X)	
	Read(X)
	Read(Y)
	Sum = X+Y
Read(Y)	
Y = Y + 5	
Write(Y)	

Summing up data while it is being updated

Schedules

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

Serial schedules

- Serial schedules are guaranteed to avoid interference and keep the database consistent
- However databases need concurrent access which means interleaving operations from different transactions

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 some data items, then the order is which they do it is not important
  - If T1 reads and updates X and T2 reads and updates a different data item Y, then again they can be scheduled in any order.

Serial and Serialisable

Interleaved Schedule	Serial Schedule
T1 Read(X)	T2 Read(X)
T2 Read(X)	T2 Read(Y)
T2 Read(Y)	T2 Read(Z)
T1 Read(Z)	
T1 Read(Y)	T1 Read(X)
T2 Read(Z)	T1 Read(Z)
	T1 Read(Y)

This schedule is serialisable: → Serial Schedule

Conflict Serialisable Schedule

Interleaved Schedule

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

Serial Schedule

T1 Read(X)
T1 Write(X)
T1 Read(Y)
T1 Write(Y)

T2 Read(X)
T2 Write(X)
T2 Read(Y)
T2 Write(Y)

This schedule is serialisable, even though T1 and T2 read and write the same resources X and Y: they have a conflict

Conflict Serialisability

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

Concurrency control

- Our aim is to constrain concurrent transactions so that all resulting schedules are conflict serialisable
- Two approaches:
  - Locks
  - Time stamps

Locking

- Locking is a procedure used to control concurrent access to data (to ensure serialisability of concurrent transactions)
- In order to use a ‘resource’ (table, row, etc) a transaction must first acquire a *lock* on that resource
- This may deny access to other transactions to prevent incorrect results

Two types of locks

- Two types of lock
  - Shared lock (S-lock or read-lock)
  - Exclusive lock (X-lock or write-lock)
- Read lock allows several transactions simultaneously to read a resource (but no transactions can change it at the same time)
- Write lock allows one transaction exclusive access to write to a resource. No other transaction can read this resource at the same time.
- The lock manager in the DBMS assigns locks and records them in the data dictionary

Locking

- Before reading from a resource a transaction must acquire a read-lock
- Before writing to a resource a transaction must acquire a write-lock
- Locks are released on commit/rollback
- A transaction may not acquire a lock on any resource that is write-locked by another transaction
- A transaction may not acquire a write-lock on a resource that is locked by another transaction
- If the requested lock is not available, transaction waits

Two-Phase Locking

- A transaction follows the *two-phase locking protocol* (2PL) if all locking operations precede the first unlock operation in the transaction
- Two phases
  - Growing phase where locks are acquired on resources
  - Shrinking phase where locks are released

Example

- T1 follows 2PL protocol
  - All of its locks are acquired before it releases any of them
- T2 does not
  - It releases its lock on X and then goes on to later acquire a lock on Y

T1	T2
read-lock(X)	read-lock(X)
Read(X)	Read(X)
write-lock(Y)	unlock(X)
unlock(X)	write-lock(Y)
Read(Y)	Read(Y)
Y = Y + X	Y = Y + X
Write(Y)	Write(Y)
unlock(Y)	unlock(Y)

Serialisability Theorem

Any schedule of two-phased transactions is conflict serialisable

Lost Update can't happen with 2PL

	T1	T2	
read-lock(X)	**Read(X)**		
	X = X - 5		
		Read(X)	read-lock(X)
		X = X + 5	
cannot acquire write-lock(X): T2 has read-lock(X)	**Write(X)**		
		Write(X)	cannot acquire write-lock(X): T1 has read-lock(X)
	COMMIT		
		COMMIT	

Uncommitted Update cannot happen with 2PL

	T1	T2	
read-lock(X)	**Read(X)**		
	X = X - 5		
write-lock(X)	**Write(X)**		
		Read(X)	Waits till T1 releases write-lock(X)
		X = X + 5	
		Write(X)	
Locks released	**ROLLBACK**		
		COMMIT	

Inconsistent analysis cannot happen with 2PL

	T1	T2	
read-lock(X)	**Read(X)**		
	X = X - 5		
write-lock(X)	**Write(X)**		
		Read(X)	Waits till T1 releases write-locks on X and Y
		Read(Y)	
		Sum = X+Y	
read-lock(Y)	**Read(Y)**		
	Y = Y + 5		
write-lock(Y)	**Write(Y)**		

Timestamping

- Transactions can be run concurrently using a variety of techniques
- We looked at using locks to prevent interference
- An alternative is timestamping
  - Requires less overhead in terms of tracking locks or detecting deadlock
  - Determines the order of transactions before they are executed

Timestamping

- Each transaction has a timestamp, TS, and if T1 starts before T2 then TS(T1) < TS(T2)
  - Can use the system clock or an incrementing counter to generate timestamps
- 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

Timestamp Protocol

- If 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 Example

- Given T1 and T2 we will assume
  - The transactions make alternate operations
  - Timestamps are allocated from a counter starting at 1
  - T1 goes first

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R			
W			

	T1	T2
TS		

Timestamp Example

T1	T2
→ Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	1		
W			

	T1	T2
TS	1	

Timestamp Example

T1	T2
→ Read(X)	→ Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2		
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	→ Read(X)
→ Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	1	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
→ Read(Y)	→ Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	→ Read(Y)
→ Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
→ Y = Y + X	→ Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	→ Z = Y - X
→ Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	1	2

Timestamp Example

T1	T2
→ T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	→ Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	2	2	
W			

	T1	T2
TS	3	2

Timestamp Example

T1	T2
→ T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	→ Write(Z)

	X	Y	Z
R	2	2	
W			2

	T1	T2
TS	3	2

Timestamp Example

T1	T2
T1	T2
→ Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	3	2	
W			2

	T1	T2
TS	3	2

Timestamp Example

T1	T2
T1	T2
Read(X)	Read(X)
→ Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W			2

	T1	T2
TS	3	2

Timestamp Example

T1	T2
T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
→ Y = Y + X	Z = Y - X
Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W			2

	T1	T2
TS	3	2

Timestamp Example

T1	T2
T1	T2
Read(X)	Read(X)
Read(Y)	Read(Y)
Y = Y + X	Z = Y - X
→ Write(Y)	Write(Z)

	X	Y	Z
R	3	3	
W		3	2

	T1	T2
TS	3	2

Timestamping

- The protocol means that transactions with higher times take precedence
  - Equivalent to running transactions in order of their final time values
  - Transactions don't wait - no deadlock
- Problems
  - Long transactions might keep getting restarted by new transactions - starvation
  - Rolls back old transactions, which may have done a lot of work

Any questions?