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 \( A \rightarrow B \) which is "bad" with respect to XNF, decompose R into \( \pi_A(B) \) and \( \pi_C(A) \) 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)

- \((\text{Cinema, Film, Day, Time})\) 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).
  - \((\text{Cinema, Film, Day, Time})\) and \((\text{Cinema, Film, Day, Time, Certificate})\) 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 \( A \rightarrow B \) 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: \( A \rightarrow B \) is non-trivial if B is not a subset of A.

Example (coursework 5)

- \(\{\text{Film}\} \rightarrow \{\text{Certificate}\}\) is a non-trivial functional dependency
- \(\{\text{Cinema, Film}\} \rightarrow \{\text{Certificate}\}\) is also a non-trivial functional dependency
- So is \(\{\text{Film}\} \cup X \rightarrow \{\text{Certificate}\}\) for any \(X \subseteq \{\text{Cinema, Day, Time}\}\)
- \(\{\text{Film, Certificate}\} \rightarrow \{\text{Certificate}\}\) is a trivial functional dependency

“Bad” functional dependencies

- For 2NF: \(A \rightarrow B\) 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: \(A \rightarrow B, B \rightarrow C\) where C is a not a key attribute (alternative definition of 3NF: bad fd \(A \rightarrow B\) is where A is not a superkey and B is non-key attribute)
- For BCNF: non-trivial \(A \rightarrow B\) where A is not a superkey.
- Example of 3NF but not BCNF: \(R(A,B,C)\), candidate keys \((A,B)\) and \((A,C)\), fd \(B \rightarrow C\).
Example (coursework 5)

- Relation Listing is not in 2NF because it has a functional dependency \( \text{(Film)} \rightarrow \{\text{Certificate}\} \) 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), \( \pi \) (projection), \( \sigma \) (selection)
- Other operations are definable using the ones above: \( \cap \) (intersection), \( \bowtie \) (natural join – can be defined using \( \times \), \( \sigma \) and \( \pi \))

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Email (domain: char(3))</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>bbb</td>
<td>222222</td>
</tr>
<tr>
<td>Chris</td>
<td>ccc</td>
<td>333333</td>
</tr>
</tbody>
</table>

Example: not union-compatible

(different domains for the second column)

<table>
<thead>
<tr>
<th>Name</th>
<th>Email (domain: char(3))</th>
<th>DOB (domain: int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>bbb</td>
<td>1971</td>
</tr>
<tr>
<td>Chris</td>
<td>ccc</td>
<td>1972</td>
</tr>
</tbody>
</table>

Example: union-compatible

<table>
<thead>
<tr>
<th>Name</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>1971</td>
</tr>
<tr>
<td>Sam</td>
<td>1985</td>
</tr>
<tr>
<td>Chris</td>
<td>1972</td>
</tr>
<tr>
<td>Steve</td>
<td>1986</td>
</tr>
</tbody>
</table>
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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Price</td>
<td>Name</td>
<td>Price</td>
</tr>
<tr>
<td>Milk</td>
<td>0.80</td>
<td>Cream</td>
<td>5.00</td>
</tr>
<tr>
<td>Bread</td>
<td>0.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eggs</td>
<td>1.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soap</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Price</td>
<td>Name</td>
<td>Price</td>
</tr>
<tr>
<td>Milk</td>
<td>0.80</td>
<td>Soap</td>
<td>1.00</td>
</tr>
<tr>
<td>Bread</td>
<td>0.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eggs</td>
<td>1.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Soap</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Extended Cartesian) product of relations

A relation containing all tuples of the form:

$$<\text{tuple from } R, \text{ tuple from } S> :$$

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

(this assumes R has n columns and S has m columns)
**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:** $\pi_{\text{Name,Telephone}}(R)$

$R$:

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Telephone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td><a href="mailto:bbb@cs.nott.ac.uk">bbb@cs.nott.ac.uk</a></td>
<td>0115222222</td>
</tr>
<tr>
<td>Chris</td>
<td><a href="mailto:ccc@cs.nott.ac.uk">ccc@cs.nott.ac.uk</a></td>
<td>0115333333</td>
</tr>
</tbody>
</table>

**Selection**

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

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

**Example: selection**

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

$R$:

<table>
<thead>
<tr>
<th>Title</th>
<th>Director</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insomnia</td>
<td>Nolan</td>
<td>2002</td>
</tr>
<tr>
<td>Magnolia</td>
<td>Anderson</td>
<td>1999</td>
</tr>
<tr>
<td>Insomnia</td>
<td>Skjoldbjaerg</td>
<td>1997</td>
</tr>
<tr>
<td>Memento</td>
<td>Nolan</td>
<td>2000</td>
</tr>
<tr>
<td>Gattaca</td>
<td>Niccol</td>
<td>1997</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Tel</td>
</tr>
<tr>
<td>Anne</td>
<td>111111</td>
</tr>
<tr>
<td>Bob</td>
<td>222222</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Tel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chris</td>
<td>333333</td>
</tr>
<tr>
<td>Dan</td>
<td>111111</td>
</tr>
</tbody>
</table>

Example exam question

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

<table>
<thead>
<tr>
<th>R.Name</th>
<th>R.Tel</th>
<th>S.Name</th>
<th>S.Tel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anne</td>
<td>111111</td>
<td>Chris</td>
<td>333333</td>
</tr>
<tr>
<td>Anne</td>
<td>111111</td>
<td>Dan</td>
<td>111111</td>
</tr>
<tr>
<td>Bob</td>
<td>222222</td>
<td>Chris</td>
<td>333333</td>
</tr>
<tr>
<td>Bob</td>
<td>222222</td>
<td>Dan</td>
<td>111111</td>
</tr>
</tbody>
</table>

Example exam question

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

<table>
<thead>
<tr>
<th>R.Name</th>
<th>S.Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anne</td>
<td>Dan</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

This update is lost

Only this update succeeds
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

<table>
<thead>
<tr>
<th>Interleaved Schedule</th>
<th>Serial Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 Read(X)</td>
<td>T2 Read(X)</td>
</tr>
<tr>
<td>T2 Read(X)</td>
<td>T2 Read(Y)</td>
</tr>
<tr>
<td>T2 Read(Y)</td>
<td>T2 Read(Z)</td>
</tr>
<tr>
<td>T1 Read(Z)</td>
<td>T1 Read(X)</td>
</tr>
<tr>
<td>T1 Read(Y)</td>
<td>T1 Read(Z)</td>
</tr>
<tr>
<td>T2 Read(Z)</td>
<td>T1 Read(Y)</td>
</tr>
</tbody>
</table>

This schedule is serialisable:
Conflict Serialisable Schedule

Interleaved Schedule
- T1 Read(X)
- T1 Write(X)
- T2 Read(Y)
- T2 Write(Y)

Serial Schedule
- T1 Read(X)
- T1 Write(Y)
- 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

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>write-lock(Y)</td>
<td>unlock(X)</td>
</tr>
<tr>
<td>read-lock(Y)</td>
<td>write-lock(Y)</td>
</tr>
<tr>
<td>Read(Y)</td>
<td>Read(Y)</td>
</tr>
<tr>
<td>Y = Y + X</td>
<td>Y = Y + X</td>
</tr>
<tr>
<td>Write(Y)</td>
<td>Write(Y)</td>
</tr>
<tr>
<td>unlock(Y)</td>
<td>unlock(Y)</td>
</tr>
</tbody>
</table>

Serialisability Theorem

Any schedule of two-phased transactions is conflict serialisable

Lost Update can’t happen with 2PL

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>cannot acquire write-lock(X): T2 has read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

Uncommitted Update cannot happen with 2PL

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>write-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Cannot acquire write-lock(X): T2 has read-lock(X)</td>
<td>Cannot acquire write-lock(X): T1 has read-lock(X)</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

Inconsistent analysis cannot happen with 2PL

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>write-lock(X)</td>
<td>read-lock(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Cannot acquire write-lock(X): T2 has read-lock(Y)</td>
<td>Cannot acquire write-lock(X): T1 has read-lock(Y)</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

read-lock(Y) | read-lock(Y) |
| X = Y + 5 | X = Y + 5 |
| Cannot acquire write-lock(Y)| Cannot acquire write-lock(Y)|
| Write(Y) | Write(Y) |
| COMMIT | COMMIT |
Exam revision

**Timestamping**

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

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

Exim revision

**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)

Exim revision

**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

Exim revision

**Timestamp Example**

Exim revision
Timestamp Example

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

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

Y = Y + X  Z = Y - X

TS 1 2

Exam revision

Timestamp Example

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

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

Y = Y + X  Z = Y - X

TS 1 2

Exam revision
Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   2  2
W   2  2

T1  T2
TS   3  2

Exam revision

Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   3  2
W   2  2

T1  T2
TS   3  2

Exam revision

Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   3  2
W   2  2

T1  T2
TS   3  2

Exam revision

Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   3  3
W   2  2

T1  T2
TS   3  3

Exam revision

Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   3  3
W   3  2

T1  T2
TS   3  3

Exam revision

Timestamp Example

T1
Read(X) Read(Y)
Y = Y + X 
Write(Y)

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

X  Y  Z
R   3  3
W   3  2

T1  T2
TS   3  3

Exam revision
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?