Omtenta i DATABASER

Svar:

Obs! Lärarversion, med lösningar

DAG: 5 April, 2002 **TID:** kl. 8.45 – 12.45 **PLATS:** V-huset

Ansvarig:	Patrik Jansson
Förfrågningar:	Patrik Jansson, ankn. 5415
Resultat:	anslås den 24 April, 2002
Poängantal:	sammanlagt maximalt 60 poäng.
Betygsgränser:	CTH: 3:a 24 p., 4:a 36 p., 5:a 48 p. GU: Godkänd 28 p., Väl godkänd 48 p. Doktorander: Godkänd 28 p.
Hjälpmedel:	Ett A4-blad (båda sidor får användas) med valfritt innehåll får medföras och skall i så fall lämnas in med svaren på tentan. (Skriv då namn och personnummer även på detta blad.)

Observera:

- Skriv tydligt och disponera pappret på ett lämpligt sätt.
- Börja varje uppgift på nytt blad. Skriv endast på en sida av pappret.
- Alla svar skall **motiveras väl** och ej vara onödigt komplicerade!
- Ange på tentan om du går på GU eller Chalmers och vilken linje/program du går.

Lycka till!

Uppgift 1. **Transaktioner:** Följande fyra krav ställs på ett databassystems transaktionshantering:
12p

- `A`. "Atomicity" — "Allt eller inget": Varje transaktion ska genomföras antingen helt eller inte alls.
- `C`. "Consistency" — "Se upp för konsistensvillkoren": En slutförd transaktion som började i ett konsistent tillstånd slutade också i ett konsistent tillstånd.
- `I`. "Isolation" — "Ingen Insyn": Inga (eventuellt inkonsistenta) mellantillstånd av en transaktion ska vara synliga för någon annan.
- `D`. "Durability" — "Det som är gjort är gjort" Om en transaktion har avslutats och bekräftats med `COMMIT`, så gäller dess ändringar.

Transaktionshanteraren har som indata ett antal transaktioner, varje trasaktion representeras av en lista av operationer (exempel: transaktion T1 = [Begin, Read A, Write B, Commit]) och har som uppgift att utföra dessa transaktioner enligt någon lämplig strategi.

Ge för vardera av nedanstående fyra fall exempel på en strategi som uppfyller tre av ACID-reglerna men som bryter mot den fjärde . Om någon eller några av kombinationerna är omöjliga — förklara varför.

a) Uppfyller `CID`, men ej `A`

b) Uppfyller `AID`, men ej `C`

c) Uppfyller `ACD`, men ej `I`

d) Uppfyller `ACI`, men ej `D`

Svar: För att uppfylla alla ACID-kraven behövs exempelvis konservativ 2-faslåsning, test av konsistenskrav senast vid commit och fördröjd ändring + loggfil + återställning vid krasch. Nedan beskrivs vad som krävs i a) till d) relativt detta.

CID Utför några av operationerna men låt bli att göra commit.

AID Ignorera konsistenskontrollerna vid commit.

ACD Använd inte låsning.

ACI Utför inga operationer på databasen!

Bättre SENT än aldrig

Tågbolaget SENT (Svensk Effektiv Naturvänlig Tågtrafik) har anlitat dig (nykläckt konsult) för att modellera deras verksamhet i en databas. Deras tidigare konsult (nu utbränd efter åtta års hårt förarbete på detta projekt) har lämnat efter sig en uppsättning attribut och funktionella beroenden som beskriver det de vill modellera:

Kortform	**Attribut**	**Förklaring**
AP	AntalPlatser	Antal platser i en viss vagn
Avg	Avgångstid	Tiden då tåget avgår från avreseorten
Avr	Avreseort	Den ort tåget startar ifrån
Bi	Biljettnummer	
Bo	Bokningsnummer	
F	Fönster	Är platsen vid ett fönster?
K	Klass	Första klass eller andra klass
M	MedBord	Finns det bord vid platsen?
Pl	Platsnummer	Platsnummer inom en vagn
Pr	Pris	Biljettens pris
S	Slutstation	Tågets slutstation
T	Tågnummer	Tågets nummer
V	Vagnnummer	Vagnens ordningnummer i ett tåg
VI	VagnId	Vagnens identifikationsnummer

Dessa funktionella beroenden skall modelleras:

1. Tågnummer → Avgångstid, Avreseort, Slutstation
2. Tågnummer, Vagnnummer → VagnId
3. VagnId → AntalPlatser
4. Biljettnummer → Bokningsnummer, Tågnummer, Vagnnummer, Platsnummer, Pris
5. VagnId, Platsnummer → Klass, Fönster, MedBord
6. VagnId, Avgångstid → Tågnummer, Vagnnummer

Svar: Här är samma beroenden med kortformer:

1. T → Avg, Avr, S
2. T, V → VI
3. VI → AP
4. Bi → Bo, T, V, Pl, Pr
5. VI, Pl → K, F, M
6. VI, Avg → T, V

Uppgift 2. 12p **DB-design:** Gör en förlustfri uppdelning av den universella tabellen *sent(AP, Avg, Avr, Bi, Bo, F, K, M, Pl, Pr, S, T, V, VI)* i lämpliga delar så att delarna är på BCNF, och så att alla beroenden utom nummer 6 bevaras. Ange för varje deltabell: tabellhuvud, nyckel, främmande nycklar från tabellen, samt vilka funktionella beroenden som den uppfyller. Din uppdragsgivare,

SENT, bryr sig inte om ifall uppdelningen gjorts med hjälp av normaliseringsteori, sunt förnuft eller (E)ER-diagram + översättning, bara slutresultatet är komplett, korrekt och väl motiverat.

Svar:

Beroende 1: *avgång(T, Avg, Avr, S)*

Beroende 2: *tågset(T, V, VI)*
$tågset.VI \rightarrow vagn.VI$
$tågset.T \rightarrow avgång.T$

Beroende 3: *vagn(VI, AP)*

Beroende 4: *biljett(Bi, Bo, T, V, Pl, Pr)*
$biljett.T \rightarrow avgång.T$
$biljett.\{T, V\} \rightarrow tågset.\{T, V\}$

Beroende 5: *plats(VI, Pl, K, F, M)*
$plats.VI \rightarrow vagn.VI$

Uppgift 3. 12p **Relationsalgebra:** Använd helst kortformerna av attributnamnen i svaren.

a) Skriv ett uttryck som testar om beroende nummer 6 (VI, Avg → T, V) är uppfyllt.

b) Definiera vyn *sent* som återskapar den universella tabellen från delarna.

c) Skriv en sökfråga som ger Biljettnummer (kortform: Bi) för alla dubbelbokningar av sittplatser.

Svar med SQL-uttryck istället för relationsalgebra ger maximalt halva poängtalet.

Svar:

a) $|\pi_{VI,Avg}(sent)| = |\pi_{VI,Avg,T,V}(sent)|$

b) $sent = biljett \bowtie avgång \bowtie vagn \bowtie tågset \bowtie plats$

c)

$$\begin{aligned} kopia &= biljett \\ svar &= \pi_{biljett.Bi}(\sigma_{biljett.Bi \neq kopia.Bi}(biljett \bowtie_{T,V,Pl} kopia)) \end{aligned}$$

Ski Jump

You are about to help implement a database system for a ski jumping competition. Each jumper in the competition has a unique name associated with him and the country he is coming from. The ski jumping competition takes two rounds. During each round the distance of the jump is measured (with accuracy up to 0.5 meters) and five judges give their points for the style (you can assume it's between 0 and 20 with one decimal point accuracy). Based on the distance and judges' points the total points for a single round are calculated. After the second round the total points for the first and the second round are added to give the final result. The following tables will be needed for the competition:

- `jumper(Name, Country)`
  `Name` — jumper name
  `Country` — a three letter country code

- `judge(JudgeNum, Round, Name, Points)`
  `JudgeNum` — judge id number (1 to 5)
  `Round` — round number (1 or 2)
  `Name` — jumper name
  `Points` — points given by a judge

- `distance(Round, Name, Distance)`
  `Round`, `Name` — same as before
  `Distance` — jump distance in meters

- `results(Round, Name, Points)`
  `Round`, `Name` — same as before
  `Points` — accumulated points for the jumper: for round number one the total points for that round, for round number two the sum of total points for round number one and round number two (you can assume that the total result is between 0 and 1000 with one decimal point accuracy)

Uppgift 4. (SQL design)
12p

a) What are the primary keys in the Ski Jump tables? What are the foreign keys? Which are the basic integrity constraints for this design?

b) Write SQL statements to create the tables for Ski Jump. Choose field types carefully.

c) Before a value is inserted into the `results` table for a given round both the jump distance and points from all five judges should be in the database. How would you specify the integrity constraint in SQL to enforce this requirement?

Svar:

```
drop table RESULTS;
drop view HELP;
drop table JUDGE;
drop table DISTANCE;
drop table JUMPER;

create table JUMPER (
  NAME     varchar(30)  not null primary key,
  COUNTRY  char(3)      not null
);

create table DISTANCE (
  ROUND    integer      not null check (ROUND between 1 and 2),
  NAME     varchar(30)  not null references JUMPER,
  DISTANCE numeric(4,1) not null check (DISTANCE between 0.0 and 300.0),
  primary key (ROUND, NAME)
);

create table JUDGE (
  JUDGENUM integer      not null check (JUDGENUM between 1 and 5),
  ROUND    integer      not null check (ROUND between 1 and 2),
  NAME     varchar(30)  not null references JUMPER,
  POINTS   numeric(3,1) not null check (POINTS between 0.0 and 20.0),
```

```
  primary key (JUDGENUM, ROUND, NAME)
);

create table RESULTS (
  ROUND    integer      not null check (ROUND between 1 and 2),
  NAME     varchar(30)  not null references JUMPER,
  POINTS   numeric(5,1) not null check (POINTS between 0.0 and 1000.0),
  primary key (ROUND, NAME),
  foreign key (ROUND, NAME) references DISTANCE (ROUND, NAME)
);


-- part of Uppg. 4c
create view HELP as
  select ROUND, NAME from JUDGE group by (ROUND,NAME) having sum(JUDGENUM) = 15;
  intersect
  select ROUND, NAME from DISTANCE;

-- example data
insert into JUMPER values ('John Smith', 'USA');
insert into JUMPER values ('Bill Jones', 'CAN');
insert into JUMPER values ('Adam Malysz', 'POL');
insert into JUMPER values ('Wojciech Pochwala', 'POL');

commit;
```

Uppgift 5. (Programming against the Ski Jump database in Java)
12p

Eftersom JDBC inte ingår i Chalmerskursen är denna uppgift frivillig för Chalmerister. Chalmeristerna kan få poäng på denna uppgift på ett av två sätt: (Den beräkningsmetod som ger flest poäng används.)

- *Genom att lösa uppgften och få den poängbedömd (som på GU)*
- *Som 25% av poängsumman på de andra uppgifterna*

Assume the `jumpers` table is filled with proper entries. Also assume the following secret Java method for calculating single round points based on the distance and five judges results is given:

```
public static float calcPoints(
  float distance, float p1, float p2, float p3, float p4, float p5) {...}
```

Write the code for the following methods (assume they are `static` and may throw `SQLException`):

- `void jump(Connection conn, int round, String name, float distance);`
  This method is called after the distance of a single jump is measured.
- `void judge(Connection conn, int judgeNum, int round, String name, float points);`
  This method should be used by judges after each jump to report their results.
- `boolean calcResults(Connection conn, int round, String name);`
  This method is called to update `results` table. It should calculate the points for a given round using the given `calcPoints` method. If there was not enough data to calculate the result (you can use the answer to **4c** to determine that) the method should return `false`, otherwise the `results` table should be updated and `true` returned.
- `void printResults(Connection conn);`
  This method should print all the jumpers positions (sorted) with country and final point results, like the following:

  ```
  1. John Smith, USA, 825 points
  2. Bill Jones, CAN, 740 points
  ...
  ```

Java JDBC hints

- Executing SQL `select` statement and getting the results:

  ```
  Statement stmt = conn.createStatement();
  ResultSet rset = stmt.executeQuery ("select ... from ...");
  while(rset.next()) {
    // access the column values by rset.getString(colnum) or
    // rset.getInt(colnum) or rset.getFloat(colnum)
  }
  ```

- Executing SQL `insert` statement:

  ```
  Statement stmt = conn.createStatement ();
  stmt.executeUpdate ("insert into ... values ...");
  ```