计算机科学与工程系

Department of Computer Science and Engineering

CS307

MidTerm Exam Key, Fall 2017

Part 1 (Quiz): 15 Points
Part 2 (Database Design): 35 Points
Part 3 (SQL): 50 points

计算机科学与工程系

Department of Computer Science and Engineering

Part 1 - Quiz Questions

Please divide your answers on the exam paper by blocks of five, like this (w, x, y and z represent possible answers):

1-5:x,y,z,w,x

6-10: y,z,w,x,y

and so forth. If a question can have several correct answers (it will be said in the question), put them between parentheses.

1. With relational databases "DDL" refers to:

a. "Distributed Database Locking", a mechanism that ensures that two users cannot modify the same value at the same time.
b. "Derived Database Language", the ability to return columns derived through functions from the values stored in the database.
c. "Data Definition Language", the subset of SQL that deals with creating tables (among other things).
d. "Data Dictionary List", the list of the tables (and columns) in the database.
e. It's a made-up acronym with no meaning.

2. If you want to ensure that a column C can only take as values Y or N, the best way to enforce it is which constraint:

a. CHECK (C IN ('Y', 'N'))
b. CHECK (C = 'Y' AND C = 'N')
c. FOREIGN KEY (C) REFERENCES YES_NO(CODE) where YES_NO is a 2-row table containing in CODE the acceptable values
d. A constraint isn't needed, it would be better to check the value in the program before insertion

3. You can define a constraint only if the column is mandatory:

a. True
b. False

4. In the world of databases, "relation" means:

a. Table
b. Column
c. Foreign key
d. Functional dependency

5. **We have in a database the two following tables: Languages(language_code, language_name) and Countries(country_code, country_name, language).What is your opinion?**
   a. Good design
   b. Misses an intermediate table for the numerous countries where several languages are official
   c. Table Languages isn't really needed, the language name can be directly entered in table Countries

6. **In an Entity/Relationship (E/R) diagram**
   a. "Entity" refers to tables, "Relationship" to arrows between them
   b. "Entity" refers to tables that have a life of their own (for instance Countries, Films, People), and "Relationship" to tables that link two entities (such as Credits)
   c. "Relationship" is another name for "Table", "Entities" are columns.
   d. "Entity" refers to tables, "Relationship" to rows (related values)

7. **Group functions ignore NULLs.**
   a. True
   b. False

8. **If in a select statement you return 3 + C, where C is an int column; if the value of C for a row is undefined (NULL), the result will be:**
   a. 3
   b. NULL
   c. SQL error

9. **What is a Primary Key?**
   a. Primary keys are unique names of a table.
   b. Primary keys are integer ids in a table row.
   c. Primary keys are unique identifiers for each row in a table.
   d. None of the above.

10. Select the code that shows the population density of China, Australia, Nigeria and France

a. SELECT name, area/population FROM world WHERE name LIKE ('China', 'Nigeria', 'France', 'Australia')
b. SELECT name, area/population FROM world WHERE name IN ('China', 'Nigeria', 'France', 'Australia')
c. SELECT name, population/area FROM world WHERE name IN ('China', 'Nigeria', 'France', 'Australia')
d. SELECT name, population FROM world WHERE name IN ('China', 'Nigeria', 'France', 'Australia')
e. SELECT name, population/area FROM world WHERE name LIKE ('China', 'Nigeria', 'France', 'Australia')

11. With SQL, how can you return the number of rows in the "Persons" table?

a. SELECT ROWS(*) FROM Persons
b. SELECT ROWS() FROM Persons
c. SELECT COUNT(*) FROM Persons
d. SELECT COUNT() FROM Persons
e. SELECT COLUMNS(*) FROM Persons
f. SELECT COLUMNS() FROM Persons

12. The OR operator displays a record if ANY conditions listed are true. The AND operator displays a record if ALL of the conditions listed are true

a. True
b. False

13. What is the difference between UNION and UNION ALL

a. There is no difference, ALL is an optional keyword like AS when you rename a column
b. With UNION ALL there is no WHERE condition in the queries
c. UNION checks for duplicates and removes them, UNION ALL doesn't
d. UNION can be replaced by an inner join, UNION ALL by an outer join

14. (2 points) We have an employee table, with a column called salary.

We want to retrieve a single value, the 10th highest salary (we assume that on the database where we run them all queries run correctly and that there is no syntax problem).

Which query or queries will always return a correct result?

a.
```
select salary from employees e
where 9 = (select count(distinct salary)
           from employees e2
           where e2.salary > e.salary)
```

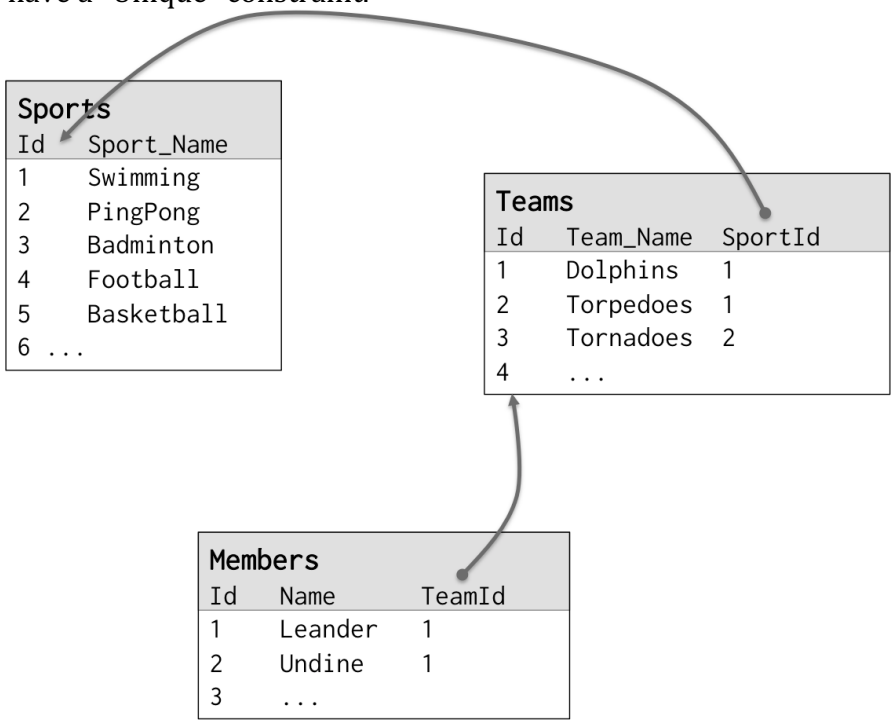
b.
```
select salary from employees e
where exists (select null
              from employees e2
              where e2.salary > e.salary
              group by e2.salary
              having count(*) = 9)
```
wrong grouping, counting groups of people with the same salary

c.
```
select salary
from (select distinct salary from employees
      order by salary desc
      limit 10) x
order by salary
limit 1
```

d.
```
select salary
from (select salary,
      rank() over (order by salary desc) as rnk
      from employees)
where rnk = 10
```
It would work with dense_rank(), not with rank() because with rank() we may not have any salary ranked 10th if there are ties.

Part 2 - Database Design

A small database was designed to record membership of sport teams, with foreign keys indicated by arrows; in all cases column Id is the primary key and the name columns have a "Unique" constraint.



Sports

Id	Sport_Name
1	Swimming
2	PingPong
3	Badminton
4	Football
5	Basketball
6	...

Teams

Id	Team_Name	SportId
1	Dolphins	1
2	Torpedoes	1
3	Tornadoes	2
4	...	

Members

Id	Name	TeamId
1	Leander	1
2	Undine	1
3	...	

For both questions, if you think that a new table or a new column is necessary, please specify:
- for tables: what is their primary key, what are the foreign keys and which columns are mandatory
-for columns: if they are mandatory or not and if they are involved in a foreign key constraint.

1. (10 points) How do you need to modify the schema if you want to allow a sport enthusiast to participate in several teams?
   - ◊ You need an intermediate table say team_membership that links a member to a team, with at least (member_id, team_id).
   - ◊ The primary key is the combination of both columns (which implies that both are mandatory)
   - ◊ member_id is a FK to Members(id)
   - ◊ team_id is a FK to Teams(id)

2. (25 points) The schema, possibly modified, may be appropriate to describe the current state of teams. But if we want the database to be used in a university context, we may expect membership to change every term - members leave the team when they graduate (or when they want to switch to another team), freshmen may arrive. We can assume that membership remains constant during a term. How would you modify it to keep historical information and be able to tell which were the members of a team say in Spring 2016?
    - ◊ The table to be modified is the one added in question 1. The term information can be added either as a single column ("Spring 2016") or as two columns (Term + Year), which would be better
    - ◊ There might be a FK to a table storing terms, which might be useful if the term information is stored as a single column. Other FKs are member_id and team_id as before.
    - ◊ What is important is that the primary key can NO LONGER be (member_id, team_id) because one student might be a team member one term, leave the team, and come back the next year. So the term information (one or two columns) must also be part of the PK, and therefore mandatory.

Part 3 - SQL Queries

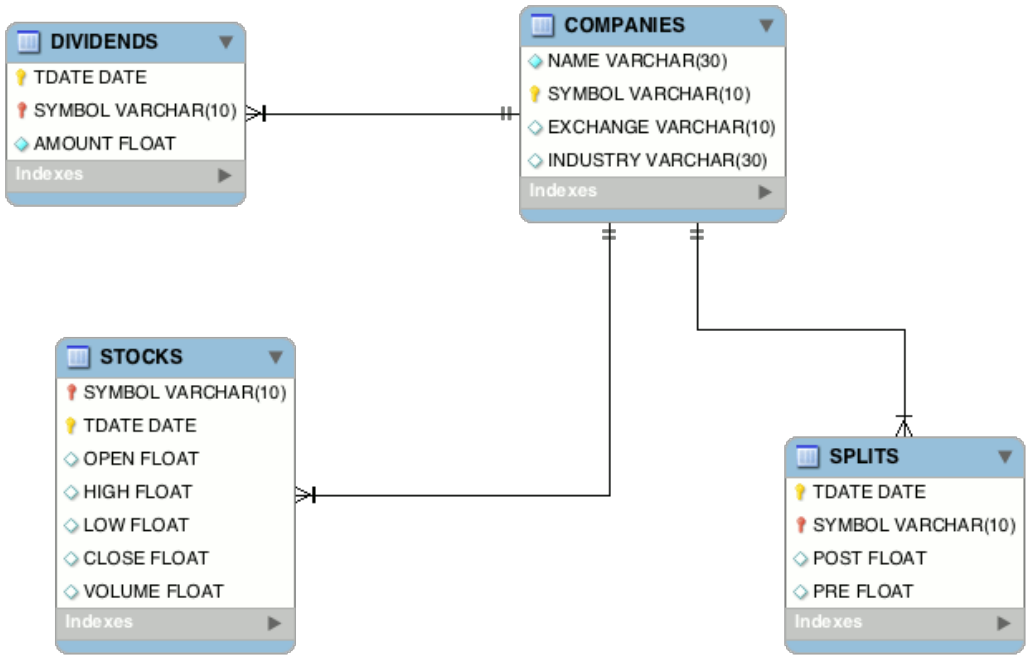
You will be asked in this part, not to write SQL queries, but to analyze a number of queries. Three questions are asked, and each time five queries are proposed as a possible answer. You must for each query say:

a. Does it provide the answer to the question?
b. Are there cases when it could not provide the answer to the question (for instance, an inner join might provide the right answer if there is no null in a nullable column, but fail to give the correct answer if one day a null is entered)
c. If the query provides the correct answer, is there something that is unnecessary in the query or misplaced?

Finally, you are asked for each one of the three questions to say which query looks best to you - either the most efficient, or the query that is closest to what you think is the solution if no queries provide the answer.

The questions use a financial database that contains stock market information for a small number of companies. Table descriptions follow.

Diagram



Tables

Companies

Identified by their symbol. They are quoted on one of two exchanges (NYSE, New-York Stock Exchange, or NASDAQ, National Association of Securities Dealers Automated Quotations). They may be classified by industry. Only name and symbol are mandatory columns, others are nullable.

symbol is the primary key, there is a unique constraint on **name**

Stocks

Stock values: trading date, company symbol, equity value when the market opened and when it closed, highest value and lowest value during the trading day, volume of equities traded. Only the primary key columns are mandatory, other columns are nullable.

The primary key is **(tdate, symbol)**

Dividends

Dividends paid by a company (per share). All columns are mandatory.

The primary key is (**tdate, symbol**)

Splits

When the price of one share becomes too high, many companies prefer splitting shares into less expensive cheaper new shares to improve liquidity (investors can move their money around more easily); the stock-market capitalization remains the same, you simply get more, less expensive, shares. Column **pre** is the number of old shares that are exchanged against the number of new shares in **post**. Only the primary key columns are mandatory.

The primary key is (**tdate, symbol**)

Question 1

What is the company in the database with the biggest value variation (in percentage) in one day ? The percentage of value variation can be computed as 100 * (high – low) / open.

Query1.1	`select name,s.tdate,s.open,s.close,s.high,s.low,s.vary`
	`from companies c`
	`  inner join`
	`  (select symbol,tdate,open,close,high,low,`
	`  max(100*(high-low)/open) vary`
	`  from stocks) s`
	`  on s.symbol = c.symbol`

	a. Doesn't answer the question, returns biggest variation for every company (at least tries to, wrong GROUP BY)

Query1.2	`select c.name, s.tdate, s.open, s.close, s.high, s.low, s.variation`
	`from companies c`
	`  join (select symbol, tdate, open, close,`
	`  high, low,`
	`  round(100*(high - low)/open, 2) as variation`
	`  from stocks`
	`  order by variation desc`
	`  limit 1) s`
	`on c.symbol = s.symbol`

	a. More info than needed b. Doesn't handle ties: if two companies have exactly the same variation, returns only one. c. Should compute max variation overall, and compare variation to this.

Query1.3	`select name`
	`from (select symbol, (100*(high-low)/open) ValueVariation`
	`  from stocks`
	`  order by ValueVariation desc`
	`  limit 1) ed`
	`  left join companies c`
	`  on ed.symbol = c.symbol`

	a. Returns the right info b. Doesn't handle ties: if two companies have exactly the same variation, returns only one. c. Should compute max variation overall, and compare variation to this.

Query1.4	`select c.name, s.tdate, s.open, s.close, s.high, s.low,`
	`  round((100 * (s.high - s.low) / s.open), 2)`
	`  as variation`
	`from companies c`

	join stocks s on c.symbol = s.symbol where variation = (select round(max(100 * (high - low) / open), 2) from stocks)
	a. Returns too much information b. Always returns the proper result c. Using the column alias in the where clause doesn't necessarily work with every DBMS but students cannot no, so no apart from not returning columns that weren't asked for.
Query1.5	select c.name, x.tdate, x.open, x.close, x.high, x.low, round(x.maxvar,2) from (select * from (select max(100*((s.high-s.low)/s.open)) maxvar from stocks s) m inner join stocks s where (100*((s.high-s.low)/s.open)) = maxvar) x inner join companies c on c.symbol = x.symbol
	a. Returns too much information b. Always returns the proper result c. Join syntax with the max a bit questionable. Should use "cross join"

Best query: 1.4. (1.5 would be an acceptable answer)

Question 2

List chronologically the dates when ALL the companies in the 'Finance' sector lost value (*all* being of course *all in the* database - note that there are currently 15 Financial companies in the database).

Query2.1	select date from (select date, count(*) num_lost_money from (select s.tdate as date from companies c join (select * from stocks where (open - close) > 0) s on s.symbol = c.symbol where c.sector = 'Finance') f group by date) a where a.num_lost_money = (select count(*) from companies where sector = 'Finance') order by date asc
	a. Yes, returns what was wanted b. Can have a problem if new Finance companies are

	introduced of if some finance companies (for instance Lehman Brothers ...) disappear. Only works if the number of Finance companies is constant. c. OK

Query2.2	`select tdate datesfinancelostmoney`
	`from (select name, sector, symbol`
	`from companies`
	`where sector is 'Finance') ed`
	`left join stocks s`
	`on ed.symbol = s.symbol`
	`where close < open`
	`order by tdate`

	a. Wrong result. Returns dates when at least one finance company lost money. b. Wrong in most cases. c. OK

Query2.3	`select tdate from stocks`
	`where tdate`
	`not in (select tdate from stocks`
	`where symbol in (select symbol from companies`
	`where sector = 'Finance')`
	`and close >= open)`
	`group by tdate`
	`order by tdate ASC`

	a. Correct result b. Always works, even when the set of finance companies changes. c. Nothing obviously wrong.

Query2.4	`select a.tdate FinanceDecreaseDates`
	`from (select s.tdate, count(*) numDecreased from stocks s`
	`inner join companies c`
	`on s.symbol = c.symbol`
	`where c.sector = 'Finance'and (s.close < s.open)`
	`group by s.tdate`
	`order by s.tdate) a`
	`where a.numDecreased =`
	`(select count(*)`
	`from companies`
	`where sector = 'Finance')`

	a. Yes, returns what was wanted but may be an order problem.

	b. Only works if the number of Finance companies is constant. c. Misplaced ORDER BY for the date (should be on the outside query)

Query2.5	`select tdate as dates_where_all_finance_lost`
	`from (select distinct s.tdate,`
	`count(tdate) as all_finance,`
	`(close - open < 0) as lost_val_day`
	`from companies c`
	`join stocks s`
	`on s.symbol = c.symbol`
	`where sector = 'Finance'`
	`and lost_val_day=1`
	`group by s.tdate`
	`having all_finance = 15)`

	a. Not ordered explicitly, otherwise returns what was wanted b. Only works if the number of Finance companies is constant. c. DISTINCT redundant with GROUP BY

Best query: 2.3

Question 3

For companies which have a sector, we can globally define a daily open price and a daily close price per sector by summing up these values for all the companies in that sector.

Using this definition, list for all sectors on the same row the number of positive days (when the global close price was higher than the global open price) and the number of negative days (when the global close price was lower than the global open price).

Query3.1	`select sector,`
	`case s.close-s.open`
	`when s.close-s.open< 0 then 'negative'`
	`else 'positive'`
	`end as income,`
	`count(*)`
	`from companies c`
	`join stocks s`
	`on s.symbol= c.symbol`

	where sector not null group by sector,income
	a. No, doesn't display everything on one row and no aggregation per sector (most days will be both positive and negative)
Query3.2	select "positive", count(*) as days from (select c.sector, sum(s.open) as daily_open, sum(s.close) as daily_close from companies c join stocks s on s.symbol = c.symbol group by c.sector having daily_close > daily_open) union all select "negative", count(*) as days from (select c.sector, sum(s.open) as daily_open, sum(s.close) as daily_close from companies c join stocks s on s.symbol = c.symbol group by c.sector having daily_open > daily_close)
	a. No, not on one row, doesn't give the sector name
Query3.3	select p.sector, p.positive_days, n.negative_days from (select sector, count(*) as positive_days from (select c.sector, s.tdate, sum(s.open) as open, sum(s.close) as close from companies c join stocks s on c.symbol = s.symbol where c.sector is not null group by s.tdate, c.sector) where open > close group by sector) p join (select sector, count(*) as negative_days from (select c.sector, s.tdate, sum(s.open) as open, sum(s.close) as close from companies c join stocks s on c.symbol = s.symbol where c.sector is not null group by s.tdate, c.sector) where open < close group by sector) n on p.sector = n.sector

	group by p.sector
	a. Yes b. May be a problem if for one sector all days were positive or all days were negative (unlikely, but ...) c. Two identical GROUP BY that could be processed in a single pass over the tables.
Query3.4	select z.sector, count() as positive from (select x.tdate, x.sector, sum(x.net_gain) as net_gain from (select s.tdate, c.name, c.sector, s.close - s.open as net_gain from companies c inner join stocks s on s.symbol = c.symbol) x group by x.tdate, x.sector) z where z.net_gain > 0 group by z.sector
	a. No, only counts positive days.
Query3.5	select f.sector, count(f.sector),count(x.sector) from (select c.sector, s.tdate from stocks s join (select symbol, sector from companies where sector is not NULL) c on c.symbol = s.symbol where s.open > s.close group by s.tdate, c.sector) f join (select c.sector, s.tdate from stocks s join (select symbol, sector from companies where sector is not NULL) c on c.symbol = s.symbol where s.open < s.close group by s.tdate, c.sector) x on x.sector = f.sector group by f.sector
	a. No, counts days when result was positive/negative for at least one company in the sector

Best query: 3.3