Øvingsoppgaver INF1300 ß

Anta at vi har et databaseskjema med følgende tabeller:

Hotell:	hotellnr, hotellnavn, by
Rom:	romnr, hotellnr, type, pris
Bestilling:	hotellnr, romnr, fraDato, tilDato, gjestnr
Gjest:	gjestnr, gjestNavn, gjestAdresse

Alle tabellenes primærnøkler er understreket.

Oppg 1
Identifiser de naturlige fremmednøklene i skjemaet.

Oppg 2
Definer tabellene med CREATE TABLE-setninger. Logg deg deretter inn i postgreSQL og opprett tabellene i databasen din.

Oppg 3
Legg inn data i databasen med INSERT-setninger. Lek deg gjerne med å slette forekomster med DELETE, oppdatere forekomster med UPDATE, osv. Legg inn i hvert fall ett hotell i Oslo, Oslo Plaza, og bestilling for gjest Ole Olsen for å ha data å jobbe med i oppg. 4 og 5.

Tips når du jobber med SQL mot databaser: bruk din yndlings-teksteditor og lagre alle SQL-kommandoer du bruker til fil, da kan du klippe og lime og slipper å skrive alt på nytt neste gang du skal jobbe med det (eller sletter skjemaene/dataene dine ☺). Du kan også kjøre sql fra fil direkte i psql.

Oppg 4
a) Finn alle detaljer for alle hoteller
b) Finn alle detaljer for alle hoteller i Oslo
c) Finn alle byer det er registrert hoteller i
d) Finn gjestnr til alle gjester som en eller annen gang har hatt en bestilling.
e) Finn navn og adresse til alle gjester som bor i Oslo, alfabetisk sortert etter navn
f) Finn alle dobbelt- og familierom med en pris på under 800,- pr. natt, sortert med lavest pris først.

Oppg 5
a) Finn pris og type for alle rom på Oslo Plaza
b) Finn alle bestillinger for en gjest med navn Ole Olsen. Listen skal inneholde gjestnr, hotellnavn, by, romnr, fraDato og tilDato, sortert etter gjestnr og deretter fraDato, med de nyeste overnattingene først.
c) Finn alle rom det i øyeblikket bor noen på ved Oslo Plaza
d) Finn alle gjester som i øyeblikket bor på Oslo Plaza

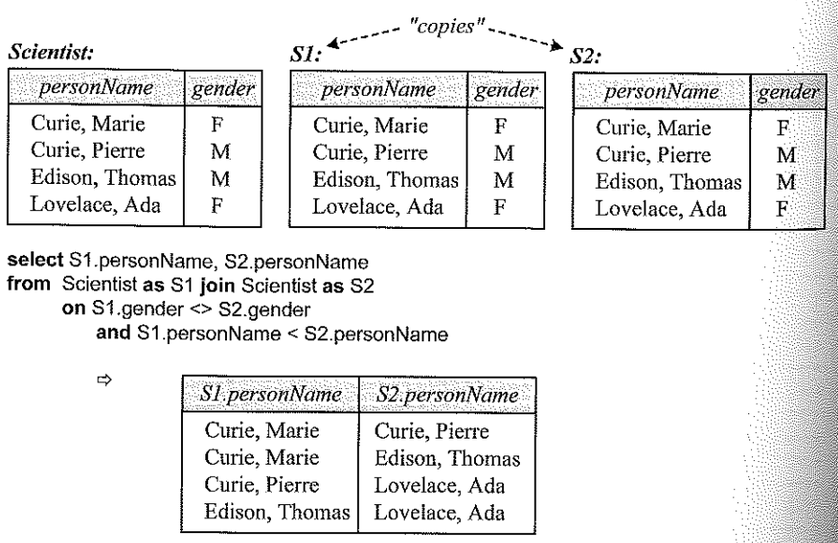
Oppg 6 Oppgaver fra læreboka: Exercise 12.6 s. 581, oppgave 1 ÏÇ ç.

row from the table and is sometimes called a "correlation variable", "range variable", or "table label". For clarity, **as** may be used to introduce the alias.

Figure 12.46 provides a simple example. The base table Scientist stores the name and gender of various scientists, and the query lists pairs of scientists of opposite gender. Here the aliases S1 and S2 are declared in the **from** clause. To understand the self-join, it helps to think of S1 and S2 as copies of the original base table, as shown. The conditional join performs a <>-join on gender (to ensure opposite gender) and a <-join on PersonName (to ensure that each pair appears only once, rather than in both orders).

Some versions of SQL also provide a "create synonym" command for declaring a permanent alias definition. However, this is not part of the standard. In simplified form, the **from** clause syntax of SQL queries may be summarized in BNF as follows (for alternatives in braces, exactly one is required):

```
from table [ [as ] alias ]
   [ , | cross join table [ [ as ] alias ]
     | natural [ inner | [ outer] { left | right | full } ] join table [ [as ] alias ]
          | [ inner | [ outer ] { left | right | full } ] join table [ [ as ] alias ]
                    { on condition | using ( col-list ) }
          | union join table [ [ as ] alias ]
          [ ,... ] ]
```




```
select S1.personName, S2.personName
from  Scientist as S1 join Scientist as S2
      on S1.gender <> S2.gender
         and S1.personName < S2.personName
```

⇨

S1.personName	*S2.personName*
Curie, Marie	Curie, Pierre
Curie, Marie	Edison, Thomas
Curie, Pierre	Lovelace, Ada
Edison, Thomas	Lovelace, Ada

Figure 12.46 A self-join is used to list pairs of scientists of opposite gender.

In formulating an SQL query, the guidelines discussed in relational algebra usually apply. First state the query clearly in English. Then try to solve the query yourself, watching how you do this. Then formalize your steps in SQL. This usually entails the following moves. What tables hold the required information? Name these in the **from** clause. What columns do you want, and in what order? Name these (qualified if needed) in the **select** list. If the **select** list doesn't include a key, and you wish to avoid duplicate rows, use the **distinct** option. If you need n tables, specify the *n - 1* join conditions. What rows do you want? Specify the search condition in a **where** clause. What order do you want for the rows? Use an **order by** clause for this.

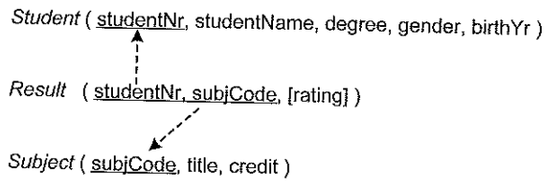
The new join syntax introduced in SQL-92 is convenient, but adds little in the way of functionality. The most useful notations are those for conditional and natural joins (both inner and outer). If you are using a version of SQL that does not support the new syntax, you should take extra care to specify the join conditions in detail and to qualify column names when required.

Any SQL statement may include *comments*. These might be used for explanation, documentation, or to simply comment out some code for debugging (comments are ignored at execution time). In SQL-92, a comment begins with two contiguous hyphens "--" and terminates at the end of the line. In addition to these single-line comments, SQL:1999 introduced multiline comments, starting with "/*" and ending with "*/". SQL Server supports both these comments styles. Here's an example:

```
-- This query retrieves details about the employees who drive a car
select *        -- select all the columns
from   Employee natural join Drives
/* If the natural join syntax is not supported,
      then omit "natural", and specify the join condition in an on-clause */
```

Exercise 12.6

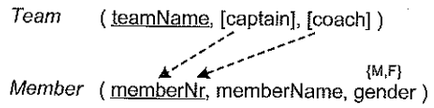
1. This question refers to the student database discussed in Exercise 12.1. Its relational schema is shown. Formulate the following queries in SQL.



   (a) List studentNr, name, degree, and birthYr of the students in ascending order of degree. For students enrolled in the same degree, show older students first.
   (b) For each student named 'Smith J', list the studentNr and the codes of subjects (being) studied.
   (c) List the studentNr, name, and gender of all students studying CS113.
   (d) List the titles of the subjects studied by the student with studentNr 863.
   (e) List the studentNr, name, and degree of those male students who obtain a rating of 5 in a subject titled "Logic". Display these in alphabetic order of name.

(f) List the code and credit points of all subjects for which at least one male student enrolled in a BSc scores a rating of 7. Display the subjects with higher credit points first, with subjects of equal credit points listed in alphabetic order of code. Ensure that no duplicate rows occur in the result.
(g) List the code and title of all the subjects, together with the ratings obtained for the subject (if any).

2. The relational schema shown is a fragment of an application dealing with club members and teams. Formulate the following queries in SQL.



(a) List the teams as well as the member numbers and names of their captains.
(b) Who (member number) is captain and coach of the same team?
(c) Who (member number) captains some team and coaches some team.

To record who plays in what team, the following table scheme is used:

PlaysIn (memberNr, teamName)

(d) What intertable constraints apply between this table and the other two tables?

Formulate the following queries in SQL.
(e) List details of all the members as well as the teams (if any) in which they play.
(f) List details of all the members as well as the teams (if any) that they coach.
(g) Who (memberNr) plays in the judo team but is not the captain of that team?
(h) Who (memberNr and name) plays in a team with a female coach?

12.7 SQL: in, between, like, and is null Operators

This section examines four special operators (other than the comparators =, <, etc.) that SQL provides for use within search conditions: **in**; ...**between**...**and**...; **like** and **is null**. These are sometimes called functions, and the conditions they are used to express are called "predicates" in the SQL standard.

A function is something that takes zero or more values as arguments and returns a single value as its result. You are probably familiar with functions from mathematics or programming, such as cos(x) or sqrt(x). As these examples illustrate, syntactically a function is usually represented as a function identifier preceding its arguments, which are typically included in parentheses.

When the action performed by a function is represented without bracketing all the arguments, we usually describe the notation as involving operators and operands rather than functions and arguments. Operators may be represented in infix, prefix, postfix, and mixfix notation according to whether the operator appears between, before, after, or mixed among the operands. For instance, the sum of 2 and 3 might be set out as:


```
sum (2, 3)        -- function
2 + 3             -- infix operator
+ 2 3             -- prefix operator
2 3 +             -- postfix operator
sum of 2 and 3    -- mixfix operator
```

The four operators we are about to discuss are used to express search conditions. The first three return the value True, False, or Unknown, while the **is null** operator returns True or False. Our initial treatment focuses on the SQL-89 version of these operators. Extensions for SQL-92 onwards are mentioned later. Most of our examples are based on the Person table, reproduced here as Table 12.9.

Suppose we want the names and birth years of the people born in 1950, 1967, or 1974. One way of requesting this information is shown in the following SQL query. As an exercise, check that this results in four rows.

```
select firstname, birthyr from Person
where birthyr = 1950 or birthyr = 1967 or birthyr = 1974
```

Imagine how tedious this way of phrasing the request would be if there were a dozen or more years involved. Partly to make life easier in such situations, SQL includes an **in** operator to handle *bag membership* and hence *set membership*. Using this infix operator, the aforementioned request may be formulated more briefly as:

```
select firstname, birthyr from Person
where birthyr in (1950,1967,1974)
```

Here the search condition is that the birthyr value is a member of the bag containing the values 1950, 1967, and 1974. The order in which these values are written does not matter, nor would it matter if any are duplicated. In general, if x is some expression (e.g., a column name) and a, b, etc. are data values (e.g., numeric or string constants), then the SQL condition shown here on the left is equivalent to the mathematical expression shown on the right:

x **in** (a, b ...) means $x \in [a, b ...]$

Table 12.9 A relational table storing personal details.

Person:

firstname	*gender*	*starsign*	*birthyr*
Bob	M	Gemini	1967
Eve	F	Aquarius	1967
Fred	M	Gemini	1970
Norma	F	Aries	1950
Selena	F	Taurus	1974
Terry	M	Aquarius	1946