**The answers to Lab 1**

1. Create a table book that contains information about books (title, format, and number of pages, authors, publisher, year, edition, ISBN-10 and 13). Find one book on the Web, for instance on amazon.com, to see some examples of the values of these attributes. Search the Web for available SQL domains (types).

CREATE TABLE book (

title VARCHAR(128) NOT NULL,

format CHAR(9),

pages INT,

authors VARCHAR(128),

publisher VARCHAR(32),

year DATE,

edition INT,

ISBN10 CHAR(10),

ISBN13 CHAR(14)

);

1. Delete the table book.

DROP TABLE book;

1. Create a table book that contains information about books (title, format (paperback or hardcover), number of pages, authors, publisher, year, edition, ISBN-10 and ISBN-13). Choose a primary key. Forbid NULL values for the title and ISBN-10 and 13, make sure the format is either “paperback” or “hardcover”.

CREATE TABLE book (

title VARCHAR(128) NOT NULL,

format CHAR(9) CHECK(format = ‘paperback’ OR format=’hardcover’ OR format IS NULL),

pages INT,

authors VARCHAR(128),

publisher VARCHAR(32),

year DATE,

edition INT,

ISBN10 CHAR(10) NOT NULL UNIQUE,

ISBN13 CHAR(14) PRIMARY KEY

);

1. Insert one book called “Introduction to Database Systems”. Go to the Web to find actual details.

INSERT INTO book VALUES ('Introduction to Database Systems', 'paperback', 168, 'Stephane Bressan and Barbara Catania', 'MacGraw-Hill', '2005-01-01', 1, '0071246509', '978-0071246507');

1. Insert half a dozen books with title containing “Introduction to Database Systems” or authored by C.J. Date. Go to the Web, for instance amazon.com, to find the details of such books.

INSERT INTO book VALUES ('An Introduction to Database Systems', 'paperback', 1024, 'C.J. Date', 'Addison-Wesley', ‘2003-08-01', 1, '0321197844','978-0321197849');

INSERT INTO book VALUES ('Introduction to Database Systems (Management Information Systems)', 'paperback', 29, 'R. Dixon, G. Rawlings', 'CIMA Publishing', '1998-12-01', 1, '0948036230', '978-0948036231');

INSERT INTO book VALUES ('Introduction to Database Systems', 'hardcover', 650, 'Bipin C. Desai', 'West Group', '1990-08-01', 1, ‘0314667717’, '978-0314667717');

INSERT INTO book VALUES ('SQL and Relational Theory: How to Write Accurate SQL Code', 'paperback', 432, 'C.J. Date', 'O Reilly Media', '2009-01-23’, 1, '0596523068', '978-0596523060');

1. Find all the information about all books.

SELECT \* FROM book;

1. Modify all books authored by C.J. Date to mention the author’s first name (find the author’s first name from the Web.)

UPDATE book SET authors=‘Christopher J. Date’ WHERE authors= ‘C. J. Date’;

1. Delete all books authored by C.J. Date.

DELETE FROM book

WHERE authors=’Christopher J. Date’;

DELETE FROM book

WHERE authors LIKE ‘C%Date’;

1. Find the title, format, number of pages, authors, publisher, year, edition, ISBN-10 and ISBN-13 of the books.

SELECT title, format, pages, authors, publisher, year, edition, ISBN10, ISBN13 FROM book;

1. Find the titles of the books.

SELECT title FROM book;

1. Find the authors of the books called “Introduction to Database Systems”.

SELECT authors FROM book WHERE title=’Introduction to Database Systems’;

1. Add a language attribute to all books. Set the default language to English.

ALTER TABLE book

ADD language VARCHAR(32) DEFAULT ‘English’;

1. Delete the table book.

DROP TABLE book;

**The answers to Lab 2**

1. Find the emails of students.

SELECT s.email

FROM student s;

1. Find the different emails of students.

SELECT s.email

FROM student s;

DISTINCT is not needed

1. Print the names of students in descending alphabetical order.

SELECT s.name

FROM student s

ORDER BY name DESC;

1. Are there students with the same name?

SELECT \* FROM student s1, student s2 WHERE s1.name=s2.name AND s1.email < s2.email;

1. Find the different names of students. Is the result sorted? Look at the execution plan.

SELECT DISTINCT s.name

FROM student s;

1. Find the names of students who owned a copy of the book ‘978-0262033848’.

SELECT s.name

FROM student s, copy c

WHERE c.owner=s.email AND c.book=‘978-0262033848’;

Do not use table book as it is guaranteed by referential integrity that there is such a book if it appears in table copy.

1. Find the names of students who owned a copy of a book with more than 100 pages whose title contains the word ‘Computer’.

SELECT s.name

FROM student s, copy c, book b

WHERE c.owner=s.email AND c.book=b.ISBN13 AND b.title LIKE ‘%computer%’ AND b.pages > 100;

1. Find the number of A4 pages needed to photocopy the two books with ISBN-13 ‘978-0262033848’ and ‘978-0321295354’ (2 pages of a book can be copied on one A4 page).

SELECT (b1.pages + b2.pages)/2FROM book b1, book b2WHERE b1.ISBN13 = '978-0262033848' AND b2.ISBN13='978-0321295354';

1. Find the different names of students who owned a copy of a book other than of ‘978-0262033848’.

SELECT DISTINCT s.name

FROM student s, copy c

WHERE c.owner=s.email AND c.book <> ‘978-0262033848’;

1. Find the names of students who borrowed a copy of the book ‘978-0262033848’.

SELECT s.name

FROM student s, loan l

WHERE l.borrower=s.email AND l.book=‘978-0262033848’;

1. Find the names of students who owned or borrowed a copy of the book ‘978-0262033848’. Use UNION.

SELECT s.name

FROM student s, copy c

WHERE c.owner=s.email AND c.book=‘978-0262033848’

UNION

SELECT s.name

FROM student s, loan l

WHERE l.borrower=s.email AND l.book=‘978-0262033848’;

1. Find the names of students who owned or borrowed a copy of the book ‘978-0262033848’. USE OR.

SELECT s.name

FROM student s, copy c, loan l

WHERE (c.owner=s.email AND c.book=‘978-0262033848’)

OR (l.borrower=s.email AND l.book=‘978-0262033848’);

1. Delete all the data in table loan.

DELETE FROM loan;

1. Try again the last two queries.

We see that the query with OR returns no results. This is wrong. That means that the query was not correct. It happens because the Cartesian product in the FROM clause is empty. When is it correct to use OR? OR should be used among conditions on the exactly same tables.

**The answer to Lab 3**

1. Find the total number of copies

SELECT COUNT(\*)

FROM copy c;

1. Find, for each book, the corresponding number of copies. [Print the primary key of the book and the number of copies.]

SELECT c.book, COUNT(\*)

FROM copy c

GROUP BY c.book;

1. Find the available books with the largest number of copies.

SELECT c.book

FROM copy c

WHERE c.available='TRUE'

GROUP BY c.book

HAVING COUNT(\*) >= ALL (SELECT COUNT(\*) FROM copy c WHERE c.available='TRUE' GROUP BY c.book);

1. Find the names of the students who have borrowed some book by ‘Charles Dickens.

SELECT s.name

FROM student s, loan l, book b

WHERE l.borrower=s.email AND l.book=b.ISBN13 AND b.authors='Charles Dickens';

1. Find the number of different books by ‘Charles Dickens.

SELECT COUNT(\*)

FROM book b

WHERE b.authors='Charles Dickens';

1. Find the names of the different students who have borrowed all the books by ‘Charles Dickens’. Use aggregate functions.

SELECT s.name

FROM student s, loan l, book b

WHERE l.borrower=s.email AND l.book=b.ISBN13 AND b.authors='Charles Dickens'

GROUP BY s.name, s.email

HAVING COUNT(DISTINCT b.ISBN13) = (SELECT COUNT(\*)

FROM book b

WHERE b.authors='Charles Dickens');

1. Find the names of students who owned a copy of a book with more than 100 pages whose title contains the word ‘Computer’. Use nested queries. This is not the preferred answer.

SELECT s.name

FROM student s, copy c

WHERE c.owner=s.email AND c.book IN (

SELECT b.ISBN13 FROM book b WHERE b.title LIKE '%computer%' AND b.pages > 100);

SELECT s.name

FROM student s

WHERE s.email IN (

SELECT c.owner FROM copy c WHERE c.book IN (

SELECT b.ISBN13 FROM book b WHERE b.title LIKE '%computer%' AND b.pages > 100));SELECT s.name

FROM student s, copy c, book b

WHERE s.email = c.owner AND c.book = b.ISBN13 AND b.title LIKE '%computer%' AND b.pages <100;

1. Find the different names of students who never owned a copy of a book other than of the book ‘978-0262033848’.

SELECT DISTINCT s.name

FROM student s

WHERE s.email NOT IN (SELECT c.owner FROM copy c WHERE c.book <> '978- 0470128725');

1. Find the names of the different students who have borrowed all the books by ‘Charles Dickens’. Use NOT EXISTS. (You may also try with NOT IN and EXCEPT.)

SELECT s.name

FROM student s

WHERE NOT EXISTS

(SELECT \*

FROM book b

WHERE b.authors=‘Charles Dickens’ AND NOT EXISTS

(SELECT \*

FROM loan l

WHERE l.book=b.ISBN13 AND l.borrower=s.email));

1. Find the names of the different students who have borrowed all the books by ‘Amelie Nothomb’.

There is no such book so … every student has borrowed all her books! Or none?

SELECT s.name

FROM student s, loan l, book b

WHERE l.borrower=s.email AND l.book=b.ISBN13 AND b.authors='Amelie Nothomb'

GROUP BY s.name, s.email

HAVING COUNT(DISTINCT b.ISBN13) = (SELECT COUNT(\*)

FROM book b

WHERE b.authors='Amelie Nothomb');

SELECT s.name

FROM student s

WHERE NOT EXISTS

(SELECT \*

FROM book b

WHERE b.authors='Amelie Nothomb' AND NOT EXISTS

(SELECT \*

FROM loan l

WHERE l.book=b.ISBN13 AND l.borrower=s.email));

1. Create and query views for the copies and loans for which the owner is a Computer Science student.

create view copy\_cs as

select c.owner as owner, c.book as book, c.copy as copy, c.available as available

from copy c, student s

where c.owner=s.email and s.faculty='School of Computing';

create view loan\_cs as

select l.borrower as borrower, l.owner as owner, l.book as book, l.copy as copy, l.borrowed as borrowed, l.returned as returned

from loan l, student s

where l.owner=s.email and s.faculty='School of Computing';

1. We could not cascade the update and deletion of book borrowers to the loan table. Create a trigger that propagates the update of a student’s email to the loan.

**The answer to Problem I**

CREATE TABLE members (

name VARCHAR(20) NOT NULL,

card\_number NUMERIC PRIMARY KEY,

address VARCHAR(50) NOT NULL

)

CREATE TABLE wines (

name VARCHAR(20),

appellation VARCHAR(20),

vintage NUMERIC,

alcohol\_degree NUMERIC NOT NULL,

vineyard VARCHAR(100) NOT NULL,

certifier VARCHAR(50) NOT NULL,

country VARCHAR(20) NOT NULL,

PRIMARY KEY (name, vintage, appellation)

)

CREATE TABLE bottles (

wine\_name VARCHAR(20),

vintage NUMERIC,

appellation VARCHAR(20),

number NUMERIC,

in\_cellar BOOLEAN NOT NULL,

PRIMARY KEY (number, wine\_name, vintage, appellation),

FOREIGN KEY (wine\_name, vintage, appellation) REFERENCES wines(name, vintage, appellation))

CREATE TABLE taste (

Member NUMERIC(20),

wine\_name VARCHAR(20),

vintage NUMERIC,

appellation VARCHAR(20),

rating VARCHAR(9) NOT NULL,

bottle\_no NUMERIC,

date DATE NOT NULL,

PRIMARY KEY (member, bottle\_no, wine\_name, vintage, appellation),

FOREIGN KEY member REFERENCES members(card\_number),

FOREIGN KEY (bottle\_no, wine\_name, vintage, appellation) REFERENCES bottles(number, name, vintage, appellation))

member

bottle

wine

taste

contain

card\_number

address

name

date

rating

number

name

appellation

vintage

Other…

**The answer to Problem II**

**Tuple Relational Calculus**

1. Find the names of pizzas that come in a 10 inch size

{T | ∃T1

(T1 ∈ pizza ∧ T1.size = 10 ∧ T1.name = T.name)}

1. Find the names of pizzas that come in a 10 inch or a 12 inch size

{T | ∃T1

(T1 ∈ pizza ∧ (T1.size = 10 ∨ T1.size = 12) ∧ T1.name = T.name)}

1. Find the names of pizzas that come in both a 10 inch and a 12 inch size

{T | ∃T1 ∃T2

(T1 ∈ pizza ∧ T2 ∈ pizza ∧ T1.name = T2.name ∧ T1.size = 10 ∧ T2.size = 12 ∧ T1.name = T.name)}

1. Find the pairs of different codes of pizzas with the same name and the same size (is there any?)

{T | ∃T1 ∃T2

(T1 ∈ pizza ∧ T2 ∈ pizza ∧ T1.code <> T2.code ∧ T1.name = T2.name ∧ T1.size = T2.size

∧ T.code1 = T1.code ∧ T.code2 = T2.code)}

Yes there some possibly {code} is the key, not {name, size}

1. Find the names and phone numbers of the stores in "College Park" or "Greenbelt" that sell a 10 inch pizza named "pepperoni" for less than $8

{T | ∃T1 ∃T2 ∃T3

(T1 ∈ pizza ∧ T2 ∈ store ∧ T3 ∈ sells ∧ T1.code= T3.code ∧ T2.name = T3.store\_name ∧ (T2.area = ‘ College Park’ ∨ T2.area = ‘Greenbelt ‘) ∧ T1.name = ‘pepperoni’ ∧ T1.size = 10 ∧ T3.price < 8 ∧ T2.name = T.name ∧ T2.phone = T.phone)}

1. Find the codes of the most expensive pizzas – assume the scheme of the database is reduced to a relation pizza(code, price) to simplify –

{T | ∃T1 ∀T2

(T1 ∈ pizza ∧ (T2 ∈ pizza ⇒ T1.price ≥ T2.price) ∧ T1.code = T.code)}

1. Find the names of the stores that sell all the pizzas

{T | ∃T1 ∀T2 ∃T3

(T1 ∈ store ∧ (T2 ∈ pizza⇒ (T3 ∈ sells ∧ T2.code= T3.code ∧ T1.name = T3.store\_name ))

∧ T1.name = T.name)}

**Domain Relational Calculus**

1. Find the names of pizzas that come in a 10 inch size

{<N> | ∃C ∃S (pizza(C, N, S)∧ S = 10)}

1. Find the names of pizzas that come in a 10 inch or a 12 inch size

{<N> | ∃C ∃S (pizza(C, N, S) ∧ (S = 10 ∨ S = 12))}

1. Find the names of pizzas that come in both a 10 inch and a 12 inch size

{<N1> | ∃C1 ∃S1 ∃C2 ∃N2 ∃S2

(pizza(C1, N1, S1) ∧ pizza(C2, N2, S2) ∧ N1 = N2 ∧ S1 = 10 ∧ S2 = 12)}

1. Find the pairs of different codes of pizzas with the same name and the same size (is there any?)

{<C1, C2> | ∃N1 ∃S1 ∃N2 ∃S2

(pizza(C1, N1, S1) ∧ pizza(N2, C2, S2) ∧ C1 <> C2 ∧ N1 = N2 ∧ S1 = S2 )}

1. Find the names and phone numbers of the stores in "College Park" or "Greenbelt" that sell a 10 inch pizza named "pepperoni" for less than $8

{<SN, P> | ∃C ∃N ∃S ∃A ∃Pr

(pizza(C,N, S) ∧ store(SN, A, P) ∧ sells(SN, C, Pr) ∧ (A = “College Park” ∨ A = “Greenbelt”) ∧ N = “pepperoni” ∧ S = 10 ∧ Pr<8)}

1. Find the codes of the most expensive pizzas – assume the scheme of the database is reduced to a relation pizza(code, price) to simplify –

{<C1> | ∃N1 ∃S1 ∀C2 ∀N2 ∀S2

(pizza(C1, N1, S1) ∧ (pizza(C2, N2, S2) ⇒ P1 ≥ P2))}

1. Find the names of the stores that sell all the pizzas

{<SN> | ∃A ∃P ∀C ∀N ∀S ∃Pr

(store(SN, A, P) ∧ (pizza(C,N, S) ⇒ sells(SN, C, Pr) ))}

**Relational algebra**

*P* → Pizza(*code, pname, size*)

*S* → Store(*sname, area, phone*)

*L* → Sells(*sname, code, price*)

1. Find the names of pizzas that come in a 10 inch size.

πpname(σsize=10(P))

1. Find the names of pizzas that come in a 10 inch or a 12 inch size.

πpname(σsize=10∨size=12(P))

1. Find the names of pizzas that come in both a 10 inch and a 12 inch size.

πpname(σsize=10(P)) ∩ πpname(σsize=12(P))

1. Find the pairs of diﬀerent codes of pizzas with the same name and the same size (is there any?).

πP1.code,P2.code(σP1.code ≠ P2.code ∧ P1.name = P2.name ∧ P1.size = P2.size (ρ(P1,P) × ρ(P2,P)))

1. Find the names and phone numbers of the stores in “College Park” or “Greenbelt” that sell a 10 inch pizza named “pepperoni” for less than $8.

πS.sname,phone( σsize=10 ∧ pname=‘pepperoni’∧price<8∧(area=‘CollegePark’ ∨ area=‘Greenbelt’)(P ⋈n S ⋈n L))

please note that the attribute names are changed, as given at the beginning of this answer sheet. Otherwise, we cannot use nature join here.

1. Find the codes of the most expensive pizzas assume the scheme of the database is reduced to a relation P → Pizza(code, price) to simplify.

πP1.code,P2.code(σP1.price≥P2.price(ρ(P1,P) × ρ(P2,P)))/πcode(ρ(P1,P))

The intuition is: (i) Find all pairs (code1, code2) of pizza codes where the price of code1 is more or equal to the price of code2. (ii) For a speciﬁc c ∈ code1, if c is paired with all possible codes, it means that its price is more or equal to all prices. Therefore c is the most expensive pizza. Note that many such pizzas may exist (if all have the same high price).

Recall that, for two tables A(x, y), B(y), division is deﬁned as:

A/B = πx(A) − πx((πx(A) × B) − A)

1. Find the names of the stores that sell all the pizzas.

πsname,code(L)/πcode(P)= πsname(L) − πsname((πsname(L) × πcode(P )) − πsname,code(L))

Observe. This is division. In this case:

A ≡ πsname,code(L)

B ≡ πcode(P )

**SQL**

1. Find the names of pizzas that come in a 10 inch size

SELECT name FROM pizza WHERE size = 10

1. Find the names of pizzas that come in a 10 inch or a 12 inch size

SELECT name FROM pizza WHERE size = 10 OR size = 12

1. Find the names of pizzas that come in both a 10 inch and a 12 inch size

SELECT P1.name FROM pizza P1, pizza P2 WHERE P1.size = 10 AND P2.size = 12 AND P1.name=p2.name

1. Find the pairs of different codes of pizzas with the same name and the same size (is there any?)

SELECT T1.code, T2.code FROM pizza T1, pizza T2   
WHERE T1.code <> T2.code AND T1.name = T2.name AND T1.size = T2.size

1. Find the names and phone numbers of the stores in "College Park" or "Greenbelt" that sell a 10 inch pizza named "pepperoni" for less than $8

SELECT T2.name, T2.phone FROM pizza T1, store T2, sells T3   
WHERE T1.code= T3.code AND T2.name = T3.store\_name   
AND (T2.area = ‘College Park’ OR T3.area = ‘Greenbelt’)   
AND T1.name = ‘pepperoni’ ∧ T1.size = 10

1. Find the codes of the most expensive pizzas – assume the scheme of the database is reduced to a relation pizza(code, price) to simplify

SELECT p.code FROM pizza p

WHERE p.price >=ALL(SELECT p1.price FROM pizza p1)

1. Find the names of the stores that sell all the pizzas

SELECT st.name FROM store st

WHERE NOT EXISTS (SELECT \* FROM pizza p

WHERE NOT EXISTS (SELECT \* FROM sells s

WHERE st.name=s.sname and p.code=s.code))

**The answer to Problem III**

1. The rule is not correct. For instance, {id} -> {name}, but {name} is not a subset of {id}.
2. Pseudo-transitivity
3. Armstrong:

Assume that X -> Y (1), Z -> V (2), and Z (belongs) Y (3)

Since Z (belongs) Y (3) then Y -> Z (4), by reflexivity

Since X -> Y (1) and Y -> Z (4) then X -> Z (5), by transitivity

Since X -> Z (5) and Z -> V (2) then X -> V (QED), by transitivity

1. Transitivity can be deduced from pseudo transitivity alone; therefore the Armstrong axioms in which transitivity is replaced by pseudo-transitivity are still complete.
2. F={ {A}->{B},{C}->{D}, {B,D}->{E}, {D}->{A,D}, {A,C}->{E,B} }
3. Empty instance or an instance with only one tuple.
4. (1,1,1,1,1) and (1,2,2,2,2).
5. ……
6. AB->A
7. A->B
8. AC->BC
9. C+ (0) = {C}

C+ (1) = {C, D} by using {C}->{D}

C+ (2) = {C, D, A} by using {D}->{A,D}

C+ (3) = {C, D, A, B} by using {A}->{B}

C+ (4) = {C, D, A, B, E} by using {B,D}->{E}

C+ = {C, D, A, B, E}, we can stop, we have every attribute. {C} is a superkey

There is no proper subset which is a superkey (only one proper subset -> and it is not a superkey), therefore {C} is a candidate key.

It is the only one. {C} is a primary key.

h. 1. Simplify the right-hand side

F'={ {A}->{B},{C}->{D}, {B,D}->{E}, {D}->{A}, {D}->{D}, {A,C}->{E} , {A,C}->{B} }

2. Simplify the left-hand side

F"={ {A}->{B},{C}->{D}, {D}->{E}, {D}->{A}, {D} ->{D}, {C}->{E} }

{A,C}->{B} can be removed because {A}->{B} is there (and {A} -> {A,B})

{B,D}->{E}, can be replaced by {D}->{E}, (because {D}->{A} and {A}->{B})

{A,C}->{E} can be replaced by {C}->{E}, (because {C}->{D} and {D}->{E})

3. Eliminate redundant rules

Min(F)={ {A}->{B},{C}->{D}, {D}->{E}, {D}->{A} }

{D} ->{D}, can be removed because it is trivial

{C}->{E} can be removed because it can obtained from {C}->{D}, {D}->{E}.