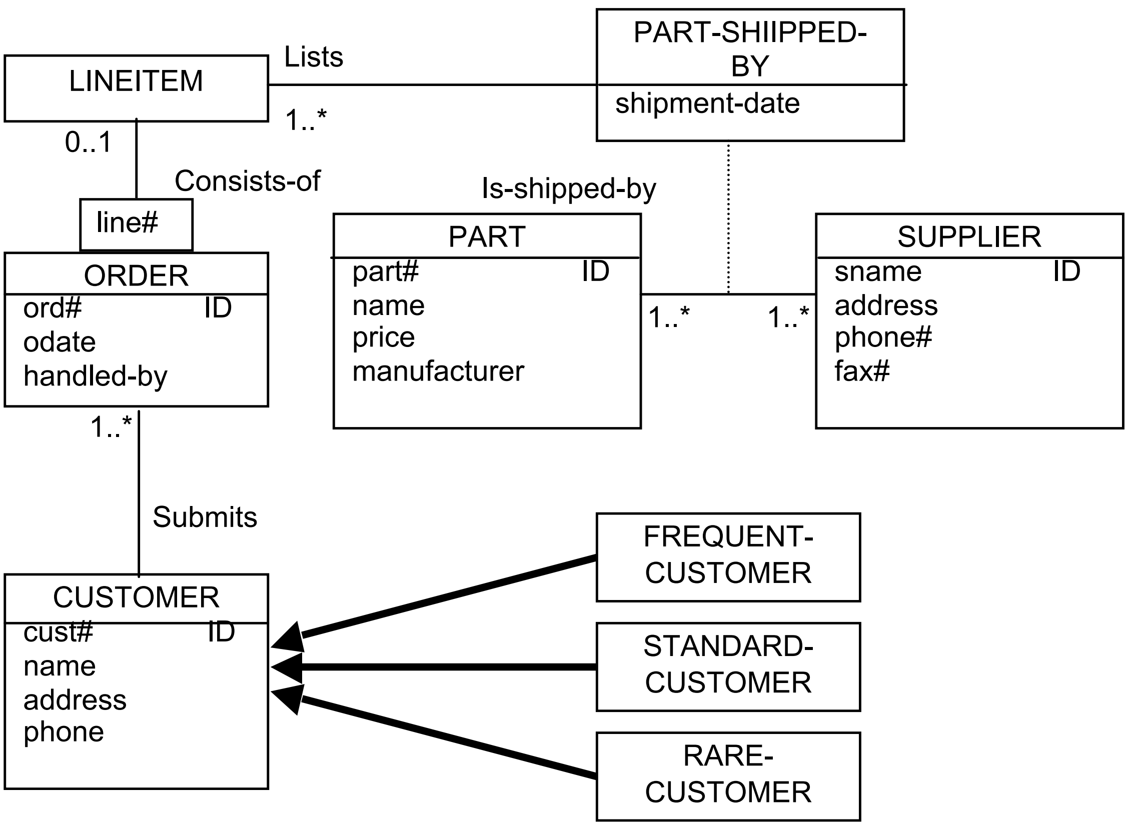
CSCI317 Database Performance Tuning
Sample problems to be solved as a preparation before a class test

---

Denormalizations

Task 1
Consider a conceptual schema of a simple ordering system given below.



Assume, that the following queries are frequently submitted for execution by the database applications. The objective of this task is to minimize their execution time by the transformations of the conceptual schema.

(1) *For each manufacturer find the total number of customers who purchased the products manufactured by the manufacturer.*
(2) *Find the names and numbers of all customers who purchased at least one part supplied by a supplier with a given name.*
(3) *For each frequent customer find the total value of all parts ordered by the frequent customer.*

Perform the simplifications of the conceptual schema given in Appendix A. Individually consider each one of the queries listed above and propose the transformations of the schema to minimize execution time of the queries given above. Integrate the results into a single schema.

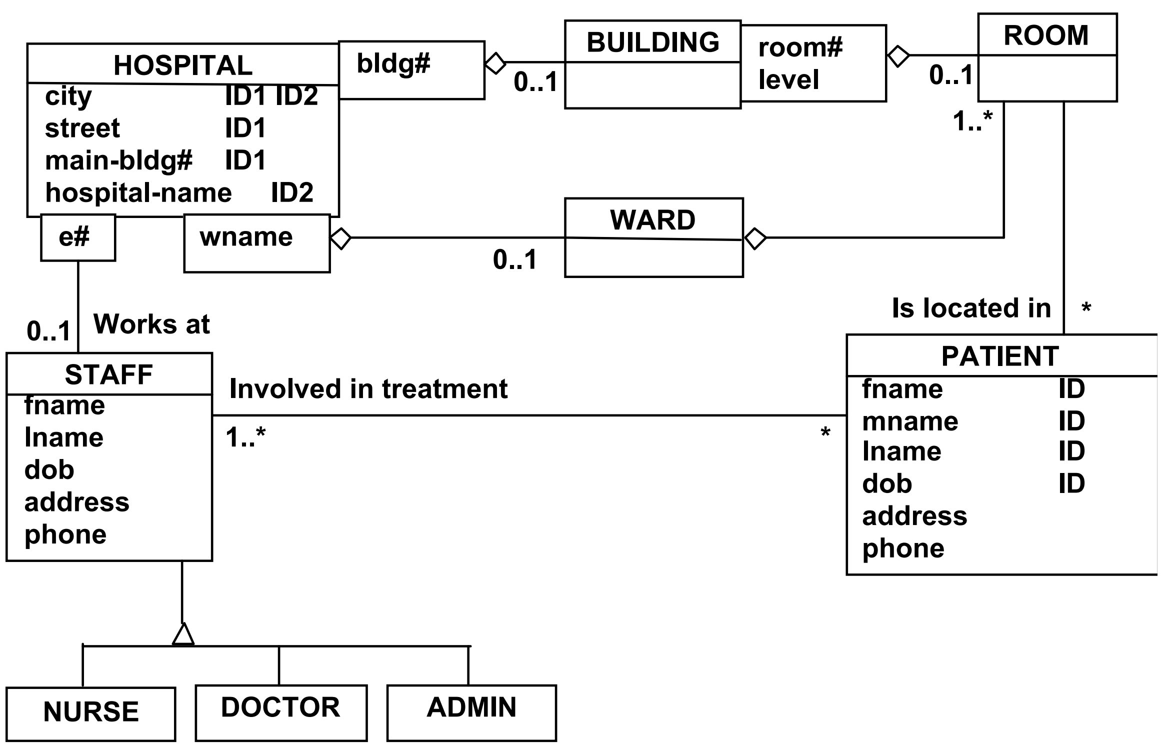
Task 2

Consider a conceptual schema of a hospital system given below. Assume, that we would like to denormalize the conceptual schema in order to improve the performance of the following class of queries:

> *Find a location (city, hospital-name, bldg#, room#) of a patient being treated by the doctors with a given last name (lname).*

For example, a query like `"find city, hospital name, building number, room number of a patient treated by a doctor whose last name is Smith"` belongs to a class of queries described above.

Transform a conceptual schema given below **A** such that any query that belongs to a class of queries described above can be computed faster than before a transformation. A transformation must consist of two steps. First transform the conceptual schema into simplified form. Next, perform implementation of generalizations (if any) and migrations of attributes to denormalize a schema obtained in the previous step.



Task 3

The conceptual schema given below represents a database domain where submit orders that consist of many lines.

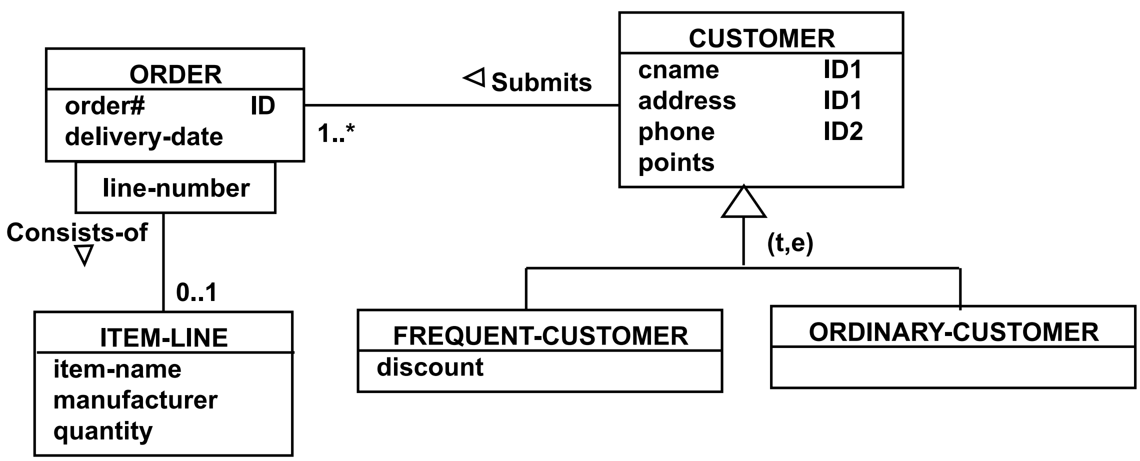
(1) Perform simplification of the conceptual schema above and re-draw the simplified conceptual schema.

(2) We would like to improve the performance of the following class of applications:

Find the names and manufacturers (attributes `item-name`, `manufacturer` *in a class* `ITEM-LINE`*) of all items delivered to the frequent customers in a given year (attribute* `delivery-date` *in a class* `ORDER`*).*

A sample application that belongs to a class described above is a as follows:

Find the names and manufacturers of all items delivered to the frequent customers `2010`.



Find the denormalizations of the simplified conceptual schema that improves the performance of the class of applications described above. When performing the denormalizations apply the following transformations of the simplified conceptual schema: migration of attributes, partitioning of classes of objects, and elimination of generalization. Re-draw the simplified conceptual schema after the denormalizations.

Quantitative analysis of indexing

Task 4
Consider a relational table
`TRANSACTION(t#,address,price,seller,buyer,agent,contract)`
that contains information about the real estate transactions. An attribute `t#` is a primary key.

Assume that:

(i) a relational table `TRANSACTION` occupies 10^3 data blocks,
(ii) a relational table `TRANSACTION` contains $5*10^3$ rows,
(iii) an attribute `address` has `4500` distinct values,
(iv) an attribute `price` has `2000` distinct values,
(v) an attribute `agent` has `100` distinct values,
(vi) a primary key is automatically indexed,
(vii) the attributes `price` and `address` are indexed,
(viii) all indexes are implemented as B*-trees with a fanout equal to `10`,
(ix) a leaf level of an index on attribute `price` consists of `50` data blocks,
(x) a leaf level of an index on attribute `address` consists of `100` data blocks.

Find the total number of read block operations needed to compute the following queries:

(1)
```
SELECT DISTINCT price
FROM TRANSACTION;
```

(2)
```
SELECT *
FROM TRANSACTION
WHERE agent = 'James';
```

(3)
```
SELECT *
FROM TRANSACTION
WHERE t# = 777 AND seller = 'Kate';
```

(4)
```
SELECT *
FROM TRANSACTION
WHERE address = 'Sydney, Station St. 5';
```

(5)
```
SELECT *
FROM TRANSACTION
WHERE agent = 'James' AND price = 100000;
```

Task 5
Consider a relational table:

```
TRANSACTION( t#,   product,   amount,   salesman,   customer,
               amount, delivery-date, comments)
```

that contains information about the industrial transactions. An attribute `t#` is a primary key.

Assume that:

(i) a relational table `TRANSACTION` occupies 10^6 data blocks,
(ii) a relational table `TRANSACTION` contains $2*10^6$ rows,
(iii) an attribute `product` has `1000` distinct values,
(iv) an attribute `amount` has `2000` distinct values,
(v) an attribute `salesman` has `100` distinct values,
(vi) an attribute `customer` has 10^6 distinct values,
(vii) an attribute `amount` has 10^4 distinct values,
(viii) an attribute `delivery-date` has $3*10^3$ distinct values
(ix) an attribute `comments` has 10^3 distinct values
(x) a primary key is automatically indexed,
(xi) the attributes `product`, `amount`, `delivery-date` are indexed,
(xii) all indexes are implemented as B*-trees with a fanout equal to `10`,
(xiii) a leaf level of an index on attribute `product` consists of `100` data blocks,
(xiv) a leaf level of an index on attribute `amount` consists of `200` data blocks.
(xv) a leaf level of an index on attribute `delivery-date` consists of `500` data blocks.
(xvi) a leaf level of an index on attribute `t#` consists of `100` data blocks.

For each one of `SELECT` statements listed above describe the best query processing plan and find the total number of read block operations needed to compute each statement.

Note, that each processing plan must be precisely described and it must be the best plan for a given indexing schema. A solution without the best plan scores no marks.

(1)
```
SELECT DISTINCT product
FROM TRANSACTION;
```

(2)
```
SELECT *
FROM TRANSACTION
ORDER BY PRODUCT;
```

(3)
```
SELECT *
FROM TRANSACTION
WHERE salesman = 'James' and customer = 'George'
```

(4)
```
SELECT *
FROM TRANSACTION
```

```
WHERE t# = 777;
```

(5)

```
SELECT *
FROM TRANSACTION
WHERE amount > 0;
```

(6)

```
SELECT *
FROM TRANSACTION
WHERE product = 'bolt' AND amount = 100;
```

(7)

```
SELECT delivery-date, count(*)
FROM TRANSACTION
GROUP BY delivery-date;
```

(8)

```
SELECT count(customer)
FROM TRANSACTION;
```

(9)

```
SELECT customer
FROM TRANSACTION
WHERE TO_CHAR(delivery-date,'YYYY') = '2009';
```

(10)

```
SELECT COUNT(*)
FROM TRANSACTION
WHERE COMMENT IS NOT NULL;
```

Task 6
Consider a relational table:

`PRODUCT(name, manufacturer, price, description, quality, mdate)`

where a pair of attributes (`name, manufacturer`) is a primary key.

A database administrator created B*-Tree index on an attribute `price`. B*-tree index on a primary key has been automatically created by a database system.

Assume that:

- (i) a relational table `PRODUCT` occupies 10^4 data blocks,
- (ii) a relational table `PRODUCT` contains 10^5 rows,
- (iii) a height of an index on the primary key is equal to `4`,
- (iv) a height of an index on an attribute `price` is equal to `2`,
- (v) the total number of distinct values in a column price is equal to 10^3,
- (vi) a leaf level of an index on the primary key consists of `500` data blocks,
- (vii) a leaf level of an index on attribute `price` consists of `100` data blocks.

Find the total number of read block operations needed to compute the following queries **(show all calculations):**

(1)
```
SELECT *
FROM PRODUCT
WHERE manufacturer = 'IBM' AND name = 'computer';
```

(2)
```
SELECT name, manufacturer
FROM PRODUCT
WHERE price = 500 OR quality = 'A';
```

(3)
```
SELECT *
FROM PRODUCT
WHERE price = 300;
```

(4)
```
SELECT COUNT(DISTINCT manufacturer)
FROM PRODUCT;
```

(5)
```
SELECT name, COUNT(*)
FROM PRODUCT
GROUP BY name;
```

Indexing relational tables

Task 7

The following `SELECT` statements suppose to retrieve information from TPC R benchmark database:

(1)

```
SELECT P_NAME, P_TYPE, P_SIZE
FROM PART
ORDER BY P_BRAND;
```

(2)

```
SELECT COUNT( P_NAME )
FROM PART;
```

(3)

```
SELECT P_BRAND, P_TYPE, COUNT(*)
FROM PART
GROUP BY P_BRAND, P_TYPE;
```

(4)

```
SELECT COUNT( DISTINCT P_NAME )
FROM PART;
```

(5)

```
SELECT P_RETAILPRICE, P_COMMENT
FROM PART
WHERE P_NAME = 'BOLT' AND P_SIZE = 10;
```

Find the smallest collection of indexes that speeds up the processing of all of the queries listed above. Write SQL script that creates the indexes and lists the execution plans for each one of `SELECT` statements given above after the indexes have been created.

Task 8
Indexing

Implement the queries listed below as SELECT statements and for each one of the queries propose the indexing schema (one or more indexes) that speeds up query processing. Consider all queries as independent such that each indexing schema for one query is independent from an indexing schema for another query.

(1)
Find name (P_NAME) and retail price (P_RETAILPRICE) of all parts in a brand (P_BRAND) Brand#51 and supplied by a supplier from CANADA (N_NAME).

(2)
Find the total number of orders issued by a customer (C_NAME) Customer#000000374 and an order includes at least one part with quantity (L_QUANTITY) greater than 40.

(3)
Find the total number of lines included in each order, list order status (O_ORDERSTATUS), order date (O_ORDERDATE), and order total price (O_TOTALPRICE).

(4)
Find the names of customers (C_NAME) from EUROPE (R_NAME) who did not include into their orders the parts supplied by a supplier Supplier#000000400 (S_NAME).

(5)
Find the quantities of items (L_QUANTITY) that got discount equal to 0.1. Do not display the same quantities more than one time and display the quantities ordered in an ascending way.

Task 9

Find the smallest collection of indexes that speed up the processing of ALL `SELECT` statements listed below. Note, that it is not allowed to use the "hints" in order to force the system to use an index for query processing.

```
SELECT *
FROM LINEITEM
WHERE L_DISCOUNT = 0.1  AND
L_TAX = 0.05 AND
L_QUANTITY = 37
ORDER BY L_QUANTITY;

SELECT *
FROM LINEITEM
WHERE (L_TAX > 20 OR L_QUANTITY = 40) AND
       L_DISCOUNT = 0.1;

SELECT DISTINCT(L_TAX)
FROM LINEITEM
ORDER BY L_TAX ASC;

SELECT L_QUANTITY, L_DISCOUNT, COUNT(*)
FROM LINEITEM
GROUP BY L_QUANTITY, L_DISCOUNT;

SELECT COUNT(*)
FROM LINEITEM
WHERE L_DISCOUNT = 0.1 AND
       L_QUANTITY = 10;
```

Task 10

Implement the queries listed below as `SELECT` statements and for each one of the queries propose the indexing schema (one or more indexes) that speeds up query processing in the best possible way. Consider all queries as independent such that each indexing schema for one query is independent from an indexing schema for another query.

(1)

```
SELECT *
FROM ORDERS
WHERE O_ORDERDATE = '12-DEC-2004' AND
       O_TOTALPRICE = 777;
```

(2)

```
SELECT *
FROM SUPPLIER
WHERE S_NAME = 'JONES' AND
S_PHONE = 1234567;
```

(3)

```
SELECT *
FROM SUPPLIER
WHERE S_PHONE = 9999999;
```

(4)

```
SELECT *
FROM SUPPLIER
WHERE S_NAME = 'JONES' OR
       S_PHONE = 1234567;
```

(5)

```
SELECT *
FROM SUPLIER
WHERE S_NAME = 'JONES';
```

(6)

```
SELECT COUNT(O_ORDERDATE)
FROM ORDERS;
```

(7)

```
SELECT O_TOTALPRICE, COUNT(*)
FROM ORDERS
```

Clustering relational tables

Task 11

Consider the following SELECT statements that join the relational tables included in the TPC R benchmark database owned by a user CSCI315.

(1)

```
SELECT CUSTOMER.C_NAME,
        CUSTOMER.C_ADDRESS,
        ORDERS.O_ORDERDATE
FROM    CUSTOMER JOIN ORDERS
ON      CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY;
```

(2)

```
SELECT CUSTOMER.C_NAME,
       CUSTOMER.C_ADDRESS,
        NATION.N_NAME
FROM   CUSTOMER JOIN NATION
ON     CUSTOMER.C_NATIONKEY = NATION.N_NATIONKEY
WHERE  NATION.N_NAME = 'SINGAPORE';
```

(3)

```
SELECT SUPPLIER.S_NAME,
        SUPPLIER.S_PHONE,
        PARTSUPP.PS_AVAILQTY
FROM    SUPPLIER JOIN PARTSUPP
ON      SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY;
```

(4)

```
SELECT SUPPLIER.S_NAME,
        SUPPLIER.S_PHONE,
        NATION.N_NAME
FROM    SUPPLIER JOIN NATION
ON      SUPPLIER.S_NATIONKEY = NATION.N_NATIONKEY;
```

Assume that the statements listed above are frequently executed by the database applications and we would like to improve performance of the applications through clustering of the relational tables. Implement SQL script `task8.sql` that performs the following actions:

(1) Creates the cluster that speeds up the processing of the queries given above.
(2) Creates the copies of relational included into the clusters and loads and load data into the relational table included in the clusters.
(3) Lists an execution plan for the queries given above after the cluster have been created. Note, that you have to change the names of relational tables used in the queries.
(4) Drops the relational tables included in the clusters.
(5) Drops the clusters.

Make sure that you pick for the clustering right relational tables.

Task 12

Consider a relational database created by the execution of the following `CREATE TABLE` statements.

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE SKILL(
sname         VARCHAR(30)      NOT NULL, /* Skill name              */
       CONSTRAINT SKILL_pkey PRIMARY KEY ( sname ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE SREQUIRED(
sname          VARCHAR(30)    NOT NULL, /* Skill name       */
requires       VARCHAR(30)    NOT NULL, /* Skill required   */
slevel         NUMBER(2)      NOT NULL, /* Level required   */
       CONSTRAINT SREQUIRED_pkey PRIMARY KEY ( sname, requires ),
       CONSTRAINT SREQUIRED_fkey1 FOREIGN KEY ( sname)
              REFERENCES SKILL( sname ),
       CONSTRAINT SREQUIRED_fkey2 FOREIGN KEY ( requires )
              REFERENCES SKILL( sname ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE APPLICANT(
anumber        NUMBER(6)      NOT NULL, /* Applicant number        */
fname          VARCHAR(20)    NOT NULL, /* First name              */
lname          VARCHAR(30)    NOT NULL, /* Last name               */
dob            DATE           NOT NULL, /* Date of birth           */
city           VARCHAR(30)    NOT NULL, /* City                    */
state          VARCHAR(20)    NOT NULL, /* State                   */
phone          NUMBER(10)     NOT NULL, /* Phone number            */
fax            NUMBER(10)             , /* Fax number              */
email          VARCHAR(50)            , /* E-mail address          */
       CONSTRAINT APPLICANT_pkey PRIMARY KEY ( anumber ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE EMPLOYER(
ename          VARCHAR(100)   NOT NULL, /* Employer name           */
city           VARCHAR(30)    NOT NULL, /* City                    */
state          VARCHAR(20)    NOT NULL, /* State                   */
phone          NUMBER(10)     NOT NULL, /* Phone number            */
fax            NUMBER(10)             , /* Fax number              */
email          VARCHAR(50)            , /* E-mail address          */
web            VARCHAR(50)            , /* Web site address        */
        CONSTRAINT EMPLOYER_pkey PRIMARY KEY ( ename ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE EMPLBY(
anumber        NUMBER(6)      NOT NULL, /* Applicant number        */
ename          VARCHAR(100)   NOT NULL, /* Employer name           */
fromdate       DATE           NOT NULL, /* Employed from           */
todate         DATE                   , /* Employed to             */
       CONSTRAINT EMPLBY_pkey PRIMARY KEY ( anumber, ename, fromdate ),
       CONSTRAINT EMPLBY_fkey1 FOREIGN KEY ( anumber )
              REFERENCES APPLICANT( anumber ),
       CONSTRAINT EMPLBY_fkey2 FOREIGN KEY ( ename )
              REFERENCES EMPLOYER( ename ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE POSITION(
pnumber        NUMBER(8)      NOT NULL, /* Position number         */
title          VARCHAR(30)    NOT NULL, /* Position title          */
salary         NUMBER(9,2)    NOT NULL, /* Salary                  */
extras         VARCHAR(50)             , /* Extras                 */
bonus          NUMBER(9,2)            , /* End of year bonus       */
specification  VARCHAR(2000)  NOT NULL, /* Specification           */
ename          VARCHAR(100)   NOT NULL, /* Employer name           */
       CONSTRAINT POSITION_pkey PRIMARY KEY ( pnumber ),
       CONSTRAINT POSITION_fkey FOREIGN KEY ( ename)
              REFERENCES EMPLOYER( ename ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE SPOSSESSED(
anumber        NUMBER(6)      NOT NULL, /* Applicant number        */
sname          VARCHAR(30)    NOT NULL, /* Skill name              */
slevel         NUMBER(2)      NOT NULL, /* Skill level             */
       CONSTRAINT SPOSSESSED_pkey PRIMARY KEY ( anumber, sname ),
       CONSTRAINT SPOSSESSED_fkey1 FOREIGN KEY ( anumber )
```

```
                              REFERENCES APPLICANT ( anumber )
                              ON DELETE CASCADE,
       CONSTRAINT SPOSSESSED_fkey2 FOREIGN KEY ( sname )
                              REFERENCES SKILL ( sname ),
       CONSTRAINT SPOSSESSED_check1 CHECK ( slevel IN
                                      ( 1,2,3,4,5,6,7,8,9,10 ) ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE SNEEDED(
pnumber        NUMBER(8)      NOT NULL, /* Position number         */
sname          VARCHAR(30)    NOT NULL, /* Skill name              */
slevel         NUMBER(2)      NOT NULL, /* Skill level             */
       CONSTRAINT SNEEDED_pkey PRIMARY KEY ( pnumber, sname ),
       CONSTRAINT SNEEDED_fkey1 FOREIGN KEY ( pnumber )
                              REFERENCES POSITION ( pnumber )
                              ON DELETE CASCADE,
       CONSTRAINT SNEEDED_fkey2 FOREIGN KEY ( sname )
                              REFERENCES SKILL ( sname ),
       CONSTRAINT SNEEDED_check1 CHECK ( slevel IN
                                      ( 1,2,3,4,5,6,7,8,9,10 ) ) );
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
CREATE TABLE APPLIES(
anumber        NUMBER(6)      NOT NULL, /* Applicant number        */
pnumber        NUMBER(8)      NOT NULL, /* Position number         */
appdate        DATE           NOT NULL, /* Application date        */
       CONSTRAINT APPLIES_pkey PRIMARY KEY ( anumber, pnumber ),
       CONSTRAINT APPLIES_fkey1 FOREIGN KEY ( anumber )
                              REFERENCES APPLICANT ( anumber )
                              ON DELETE CASCADE,
       CONSTRAINT APPLIES_fkey2 FOREIGN KEY ( pnumber )
                              REFERENCES POSITION ( pnumber )
                              ON DELETE CASCADE);
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
```

The database contains information about applicants for the positions advertised by employers, skills, skills possessed by applicants, skills needed for positions and skills required by other skills.

After loading data into the database the relational tables have the following sizes:

SKILL	50 data blocks
SREQUIRED	200 data blocks
APPLICANT	500data blocks
EMPLOYER	300 data blocks
EMPLBY	5000 data blocks
POSITION	500 data blocks
SPOSSESSED	300 data blocks
SNEEDED	600 data blocks
APPLIES	1000 data blocks

We would like to use clustering to improve performance of the following types of queries:

(i) Find full information about the applicants who applied for a position offered by a given employer.
(ii) Find full information about the applicants who posses a give skill.
(iii) Find full information about the skills possessed by a given applicant.
(iv) Find full information about the positions applied by a given applicant.
(v) Find full information about employers who advertise more than a given number positions.

Express the queries above as `SELECT` statements.

Assume, that queries *(i)* and *(ii)* are processed 5 times per day. Assume that queries *(iii)* and *(iv)* are processed 10 times per day. Assume that query *(v)* is processed 10 times per day.

Assume that if the relational tables `r` and `s` consist of b_r and b_s blocks then their sequential scan requires b_r and b_s read block operations and their join, i.e. `r JOIN s` requires $3 * (b_r + b_s)$ read block operations.

Use a method of finding suboptimal clustering explained to you during the lecture classes in a presentation 18 Clustering relational tables to find suboptimal clustering of the sample database that improves the performance of the queries listed above.

Denormalization of relational tables

Task 13
Consider the following query template:

> Find an order key, (O_ORDERKEY), order status (O_ORDERSTATUS), order value (O_TOTALPRICE) and region name (R_NAME) of all customers who submitted an order in a given year (O_ORDERDATE).

The following is a sample query is consistent with the template above:

> Find an order key, (O_ORDERKEY), order status (O_ORDERSTATUS), order value (O_TOTALPRICE) and region name (R_NAME) of all customers who submitted an order in 1998 (O_ORDERDATE).

Write a sample query consistent with the template above as SELECT statement operating on the original relational table ORDERS and other tables required by the query.

Next, write SQL statements that denormalize a relational table ORDERS such that a query listed above can be implemented as more efficient SELECT statement.

Finally, write a sample query consistent with the template above as SELECT statement operating on a relational table ORDERS denormalized such that the new query would provide an answer faster than a query implemented in the previous step.

Performance driven storage management

Task 14

`CREATE TABLE` statements listed below can be used to create a TPC W sample database (`www.tpc.org`).

```
/* ------------------------------------------------------------------- */
/*                                                                     */
/* -------- The relational database schema for TPC-W benchmark -------- */
/*                                                                     */
/* ------------------------------------------------------------------- */


CREATE TABLE COUNTRY(
CO_ID          NUMERIC(4)     NOT NULL, /* Unique country ID              */
CO_NAME        VARCHAR(50)    NOT NULL, /* Name of country         */
CO_EXCHANGE    NUMERIC(12,6)  NOT NULL, /* Exchange rate to US$    */
CO_CURRENCY    VARCHAR(18)    NOT NULL, /* Name of currency        */
       CONSTRAINT COUNTRY_PKEY PRIMARY KEY (CO_ID),
       CONSTRAINT COUNTRY_CHECK1 CHECK(CO_ID > 0) );


CREATE TABLE ADDRESS(
ADDR_ID        NUMERIC(10)    NOT NULL, /* Unique address ID              */
ADDR_STREET1   VARCHAR(40)    NOT NULL, /* Street address, line 1  */
ADDR_STREET2   VARCHAR(40)    NOT NULL, /* Street address, line 2  */
ADDR_CITY      VARCHAR(30)    NOT NULL, /* Name of city            */
ADDR_STATE     VARCHAR(20)    NOT NULL, /* Name of state           */
ADDR_ZIP       VARCHAR(10)    NOT NULL, /* Zip code or postal code */
ADDR_CO_ID     NUMERIC(4)     NOT NULL, /* Unique ID of country    */
       CONSTRAINT ADDRESS_PKEY PRIMARY KEY (ADDR_ID),
       CONSTRAINT ADDRESS_FKEY FOREIGN KEY (ADDR_CO_ID)
              REFERENCES COUNTRY(CO_ID),
       CONSTRAINT ADDRESS_CHECK1 CHECK(ADDR_ID > 0) );

CREATE TABLE AUTHOR(
A_ID           NUMERIC(10)    NOT NULL, /* Unique author ID        */
A_FNAME        VARCHAR(20)    NOT NULL, /* First name of author    */
A_MNAME        VARCHAR(20)    NOT NULL, /* Last name of author     */
A_LNAME        VARCHAR(20)    NOT NULL, /* Middle name of author   */
A_DOB          DATE           NOT NULL, /* Date of birth of author */
A_BIO          VARCHAR(500)   NOT NULL, /* About the author        */
       CONSTRAINT AUTHOR_PKEY PRIMARY KEY (A_ID),
       CONSTRAINT AUTHOR_CHECK1 CHECK(A_ID > 0) );

CREATE TABLE ITEM(
I_ID           NUMERIC(10)    NOT NULL, /* Unique ID of item              */
I_TITLE        VARCHAR(60)    NOT NULL, /* Title of item           */
I_A_ID         NUMERIC(10)    NOT NULL, /* Author ID of item              */
I_PUB_DATE     DATE           NOT NULL, /* Date of release of an item */
I_PUBLISHER    VARCHAR(60)    NOT NULL, /* Publisher of item              */
I_SUBJECT      VARCHAR(60)    NOT NULL, /* Subject of book         */
I_DESC         VARCHAR(500)   NOT NULL, /* Description of item     */
I_RELATED1     NUMERIC(10)    NOT NULL, /* Unique item ID (I_ID) of related item */
I_RELATED2      NUMERIC(10)     NOT NULL, /* Unique item ID (I_ID) of related item */
I_RELATED3      NUMERIC(10)     NOT NULL, /* Unique item ID (I_ID) of related item */
I_RELATED4      NUMERIC(10)     NOT NULL, /* Unique item ID (I_ID) of related item */
I_RELATED5      NUMERIC(10)     NOT NULL, /* Unique item ID (I_ID) of related item */
I_THUMBNAIL    VARCHAR(10),            /* Pointer to thumbnail image of item */
I_IMAGE        VARCHAR(10),            /* Pointer to  image of item       */
I_SRP          NUMERIC(15,2)  NOT NULL, /* Suggested retail price  */
I_COST         NUMERIC(15,2)  NOT NULL, /* Cost of item            */
I_AVAIL        DATE           NOT NULL, /* When item is available  */
I_STOCK        NUMERIC(4)     NOT NULL, /* Quatity in stock        */
I_ISBN         CHAR(13)       NOT NULL, /* Product ISBN            */
I_PAGE         NUMERIC(4)     NOT NULL, /* Number of pages of  book */
I_BACKING      VARCHAR(15)    NOT NULL, /* Type of book:paper,hardback */
I_DIMENSIONS   VARCHAR(25)    NOT NULL, /* Size of book in inches  */
```

```
        CONSTRAINT ITEM_PKEY PRIMARY KEY (I_ID),
        CONSTRAINT ITEM_FKEY FOREIGN KEY (I_A_ID)
               REFERENCES AUTHOR(A_ID),
        CONSTRAINT ITEM_CHECK1 CHECK(I_ID > 0),
        CONSTRAINT ITEM_CHECK2 CHECK(I_A_ID > 0) );


CREATE TABLE CUSTOMER(
C_ID           NUMERIC(10)    NOT NULL, /* Unique ID of customer    */
C_UNAME        VARCHAR(20)    UNIQUE NOT NULL, /* Unique user name  */
C_PASSWD       VARCHAR(20)    NOT NULL, /* User password            */
C_FNAME        VARCHAR(15)    NOT NULL, /* First name of customer   */
C_LNAME        VARCHAR(15)    NOT NULL, /* Last name of customer    */
C_ADDR_ID      NUMERIC(10)    NOT NULL, /* Address ID of customer   */
C_PHONE        VARCHAR(16)    NOT NULL, /* Phone number of customer */
C_EMAIL        VARCHAR(50)    NOT NULL, /* Email for purchase confirmation */
C_SINCE        DATE           NOT NULL, /* Date of registration     */
C_LAST_VISIT   DATE           NOT NULL, /* Date of last visit              */
C_LOGIN        TIMESTAMP      NOT NULL, /* Start of current customer session */
C_EXPIRATION   TIMESTAMP      NOT NULL, /* Current customer session expiry */
C_DISCOUNT     NUMERIC(3,2)   NOT NULL, /* Percentage discount      */
C_BALANCE      NUMERIC(15,2)  NOT NULL, /* Balance of customer      */
C_YTD_PMT      NUMERIC(15,2)  NOT NULL, /* Year-to-date payments    */
C_BIRTHDATE    DATE           NOT NULL, /* Birth date                      */
C_DATA         VARCHAR(500)   NOT NULL, /* Miscellaneous information       */
        CONSTRAINT CUSTOMER_PKEY PRIMARY KEY (C_ID),
        CONSTRAINT CUSTOMER_FKEY FOREIGN KEY (C_ADDR_ID)
               REFERENCES ADDRESS(ADDR_ID),
        CONSTRAINT CUSTOMER_CHECK1 CHECK(C_ID > 0),
        CONSTRAINT CUSTOMER_CHECK2 CHECK(C_ADDR_ID > 0) );


CREATE TABLE ORDERS(
O_ID           NUMERIC(10)    NOT NULL, /* Unique ID of order              */
O_C_ID         NUMERIC(10)    NOT NULL, /* Customer ID              */
O_DATE         TIMESTAMP      NOT NULL, /* Order date an time              */
O_SUB_TOTAL    NUMERIC(15,2)  NOT NULL, /* Subtotal of all order-line items */
O_TAX          NUMERIC(15,2)  NOT NULL, /* tax over subtotal               */
O_TOTAL        NUMERIC(15,2)  NOT NULL, /* Total for this order     */
O_SHIP_DATE    TIMESTAMP      NOT NULL, /* Method of delivery              */
O_SHIP_TYPE    VARCHAR(10)    NOT NULL, /* Order ship date          */
O_BILL_ADDR_ID  NUMERIC(10)   NOT NULL, /* Address ID to bull              */
O_SHIP_ADDR_ID        NUMERIC(10)    NOT NULL, /* Address ID to ship order */
O_STATUS       VARCHAR(15)    NOT NULL, /* Order status             */
        CONSTRAINT ORDERS_PKEY PRIMARY KEY (O_ID),
        CONSTRAINT ORDERS_FKEY1 FOREIGN KEY (O_C_ID)
               REFERENCES CUSTOMER(C_ID),
        CONSTRAINT ORDERS_FKEY2 FOREIGN KEY (O_SHIP_ADDR_ID)
               REFERENCES ADDRESS(ADDR_ID),
         CONSTRAINT ORDERS_FKEY3 FOREIGN KEY (O_BILL_ADDR_ID)
        REFERENCES ADDRESS(ADDR_ID),
        CONSTRAINT ORDER_CHECK1 CHECK(O_ID > 0),
        CONSTRAINT ORDER_CHECK2 CHECK(O_C_ID > 0),
        CONSTRAINT ORDER_CHECK3 CHECK(O_BILL_ADDR_ID > 0),
        CONSTRAINT ORDER_CHECK4 CHECK(O_SHIP_ADDR_ID > 0) );



CREATE TABLE CC_XACTS(
CX_O_ID         NUMERIC(10)   NOT NULL, /* Unique order ID          */
CX_TYPE        VARCHAR(10)    NOT NULL, /* Credit cardc type               */
CX_NUM         NUMERIC(16)    NOT NULL, /* Credit card number              */
CX_NAME        VARCHAR(31)    NOT NULL, /* Name of credit card      */
CX_EXPIRY      TIMESTAMP      NOT NULL, /* Expiration date          */
CX_AUTH_ID     CHAR(15)       NOT NULL, /* Authorization for transaction amount */
CX_XACT_AMT    NUMERIC(15,2)  NOT NULL, /* Amount for this transaction */
CX_XACT_DATE   TIMESTAMP      NOT NULL, /* Date and time of authorization */
CX_CO_ID       NUMERIC(4)     NOT NULL, /* Country when transaction originated */
        CONSTRAINT CC_XACTS_PKEY PRIMARY KEY (CX_O_ID),
        CONSTRAINT CC_XACTS_FKEY1 FOREIGN KEY (CX_CO_ID)
               REFERENCES COUNTRY(CO_ID),
```

```
        CONSTRAINT CC_XACTS_FKEY2 FOREIGN KEY (CX_O_ID)
                REFERENCES ORDERS(O_ID),
        CONSTRAINT CC_XACTS_CHECK1 CHECK(CX_O_ID > 0),
        CONSTRAINT CC_XACTS_CHECK2 CHECK(CX_CO_ID > 0) );

CREATE TABLE ORDER_LINE(
OL_ID           NUMERIC(10)     NOT NULL, /* Unique order-line item ID       */
OL_O_ID         NUMERIC(10)     NOT NULL, /* Order ID                    */
OL_I_ID         NUMERIC(10)     NOT NULL, /* Unique item ID              */
OL_QTY          NUMERIC(3)      NOT NULL, /* Quantity of item            */
OL_DISCOUNT     NUMERIC(3,2)    NOT NULL, /* Percentage discount off of stock retail price
*/
OL_COMMENTS     VARCHAR(100)    NOT NULL, /* Special instructions        */
        CONSTRAINT ORDER_LINE_PKEY PRIMARY KEY (OL_ID, OL_O_ID),
        CONSTRAINT ORDER_LINE_FKEY1 FOREIGN KEY (OL_I_ID)
                REFERENCES ITEM(I_ID),
        CONSTRAINT ORDER_LINE_FKEY2 FOREIGN KEY (OL_O_ID)
                REFERENCES ORDERS(O_ID),
        CONSTRAINT ORDER_LINE_CHECK1 CHECK(OL_ID > 0),
        CONSTRAINT ORDER_LINE_CHECK2 CHECK(OL_O_ID > 0),
        CONSTRAINT ORDER_LINE_CHECK3 CHECK(OL_I_ID > 0) );
```

The TPC W database contains information about the orders submitted by the customers, the items included in the orders, the authors of the items, the addresses and the countries the customers belong to and credit card numbers.

Assume, that to avoid the conflicts with the accesses to the relational tables of TPC W sample database we would like to distribute the relational tables and automatically created indexes on primary keys on three different hard drives. Do not worry if your computer does not have three hard drives. We shall simulate the drives through three different tablespaces `DRIVE_C`, `DRIVE_D`, and `DRIVE_E`. To find out, which relational tables and, which indexes should be located on each drive we shall consider the following database applications.

(1) *Find a complete information about the items whose total ordered quantity (attribute `OL_QTY`) is higher than a give value.*
(2) *Find the first name and the last name (attributes `C_FNAME`, `C_LNAME`) of the customers who ordered more than `10` items in a single order.*
(3) *Find the credit card numbers (attribute `CC_NUM`) of the customers who submitted more than `100` orders.*
(4) *Find the full addresses of the customers who ordered a given item (attribute `I_TITLE`).*
(5) *Find the dates of orders (attribute `O_DATE`) submitted by the customers living in a given country (attribute `CO_NAME`).*

Analyze the applications listed above and find which relational tables and which indexes will be used by each application and distribute the relational tables and indexes over the hard drives simulated by the tablespaces `DRIVE_C`, `DRIVE_D`, and `DRIVE_E` such, that the relational tables and indexes used in the same application are located on different hard drives. If it is impossible to distribute the relational tables and indexes used by the same application on the different hard drives then try to minimize the total number of conflicts.

Performance driven re-organization of relational tables
Task 15
Write SQL script that implements the following operations on a sample database:

First, the script connects as a user SYSTEM and increases the size of tablespace CSCI315 by the additional 200 Mbytes.

Next, the script connects as a user CSCI315 and finds the total number of rows in the relational tables LINEITEM, ORDERS, and CUSTOMER. The total number of rows for all tables must be listed in one (!) line.

Next, the script executes a statement ANALYZE TABLE for each one of the relational tables LINEITEM, ORDERS, and CUSTOMER. Then, the script lists an average amount free space counted in bytes in the data blocks used for the implementation of each one of the relational tables LINEITEM, ORDERS, and CUSTOMER.. An average amount of free space in data blocks can be found in a relational view USER_TABLES in a column AVG_SPACE.

Next, the script executes a script delcust.sql, which creates a relational table DELCUST and fills the table with the identifiers of customers (attribute D_CUSTKEY) to be deleted from the database. At the end of this step the script lists the total number of rows in a relational table DELCUST.

Next, the script deletes from a relational table CUSTOMER all customers such that identifier of each customer is included in a relational table DELCUST. Information about all other customers must be left in a relational table CUSTOMER.

Then, the script deletes information about all orders submitted by the customers delete from a relational table CUSTOMER. Together with orders all information about ordered items must be removed from a relational table LINEITEM. In fact, step (5) and step (6) can be performed in any order, which is convenient for you.

Next, the script executes a statement ANALYZE TABLE for each one of the relational tables LINEITEM, ORDERS, and CUSTOMER. Then, the script lists an average amount free space counted in bytes in the data blocks used for the implementation of each one of the relational tables LINEITEM, ORDERS, and CUSTOMER. An average amount of free space in data blocks can be found in a relational view USER_TABLES in a column AVG_SPACE.

Next, the script performs re-organization of the relational tables LINEITEM, ORDERS, and CUSTOMER to minimize an average amount of free space in data blocks. A way how you re-organize the relational tables is up to you. You can follow the methods presented in Homework 8. It is not allowed to change the size of a database block. The values of the other parameters of the relational tables are up to you.

Next, the script executes a statement `ANALYZE TABLE` for each one of the relational tables `LINEITEM`, `ORDERS`, and `CUSTOMER`. Then, the script lists an average amount free space counted in bytes in the data blocks used for the implementation of each one of the relational tables `LINEITEM`, `ORDERS`, and `CUSTOMER`. An average amount of free space in data blocks can be found in a relational view `USER_TABLES` in a column `AVG_SPACE`.

Finally, the script lists the total number of rows in the relational tables `LINEITEM`, `ORDERS`, and `CUSTOMER`. The total number of rows for all tables must be listed in one (!) line.

Rewriting `SELECT` statements

Task 16

For each one of `SELECT` statements listed below construct a new `SELECT` statement equivalent to the original one and more efficient than the original one.

(1)

```
SELECT L_TAX
FROM LINEITEM
WHERE L_QUANTITY > 100
UNION
SELECT L_TAX
FROM LINEITEM
WHERE L_QUANTITY < 10;
```

(2)

```
SELECT  S1.S_NAME
FROM SUPPLIER S1 JOIN SUPPLIER S2
     ON S1.S_NAME >= S2.S_NAME
GROUP BY S1.S_NAME
HAVING COUNT(*) = ( SELECT COUNT( DISTINCT S_NAME)
                    FROM SUPPLIER );
```

(3)

```
SELECT PS_PARTKEY, PS_AVAILQTY
FROM PARTSUPP
WHERE PS_PARTKEY IN (SELECT P_PARTKEY
                     FROM PART
                     WHERE P_PARTKEY = 100);
```

(4)

```
SELECT DISTINCT P_TYPE, P_RETAILPRICE
FROM PART
WHERE P_RETAILPRICE =
      (SELECT MAX(P.P_RETAILPRICE)
       FROM PART P
       WHERE P.P_TYPE = PART.P_TYPE)
ORDER BY P_TYPE ASC;
```

(5)

```
SELECT P_PARTKEY
FROM PART
WHERE P_RETAILPRICE > 900 AND
      P_PARTKEY IN ( SELECT P_PARTKEY
                     FROM PART
                 WHERE P_RETAILPRICE < 920 );
```

(6)

```
SELECT LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, LINEITEM.L_ORDERKEY
FROM LINEITEM
WHERE ( LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, LINEITEM.L_ORDERKEY )
IN
      ( SELECT LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY,
LINEITEM.L_ORDERKEY
        FROM LINEITEM JOIN ORDERS
             ON LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
        WHERE LINEITEM.L_PARTKEY IN (46557,20193,45690,45123)
        UNION
        SELECT LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY,
LINEITEM.L_ORDERKEY
        FROM LINEITEM
        WHERE LINEITEM.L_PARTKEY IN (46557,45690,45123) AND
              LINEITEM.L_SUPPKEY IN
              ( SELECT PS_SUPPKEY
                FROM PARTSUPP
                WHERE PARTSUPP.PS_PARTKEY IN (46557,45690,45123) ) )
ORDER BY LINEITEM.L_PARTKEY ASC, LINEITEM.L_SUPPKEY ASC,
LINEITEM.L_ORDERKEY ASC
```

(7)

```
SELECT LINEITEM.L_ORDERKEY, LINEITEM.L_LINENUMBER
FROM LINEITEM JOIN PART
     ON LINEITEM.L_PARTKEY = PART.P_PARTKEY
WHERE PART.P_PARTKEY IN (46557,20193,19110,45690,45123)
  MINUS
      (SELECT LINEITEM.L_ORDERKEY, LINEITEM.L_LINENUMBER
       FROM LINEITEM JOIN PART
            ON LINEITEM.L_PARTKEY = PART.P_PARTKEY
       WHERE PART.P_PARTKEY IN (46557,20193,19110,45690,45123)
         MINUS
       SELECT LINEITEM.L_ORDERKEY, LINEITEM.L_LINENUMBER
       FROM LINEITEM JOIN SUPPLIER
            ON LINEITEM.L_SUPPKEY = SUPPLIER.S_SUPPKEY
       WHERE SUPPLIER.S_SUPPKEY IN (4567,2323,1987,2194,1111) );
```

Other interesting problems

Task 17

Implement SQL script that creates an index IDXT7(P_NAME, P_BRAND, P_SIZE) over a relational table PART. Then, find SELECT statements that will use the index in the following ways:

(i) Execution of the first SELECT statement must traverse the index vertically and it must not access a relational table PART.

(ii) Execution of the second SELECT statement must traverse the index vertically and later on horizontally and it must not access a relational table PART.

(iii) Execution of the third SELECT statement must traverse the leaf level of the index horizontally and it must not access a relational table PART.

(iv) Execution of the fourth SELECT statement must traverse the index vertically and it must access a relational table PART.

(v) Execution of the fifth SELECT statement must traverse the index vertically and later on horizontally and it must access a relational table PART.

(vi) Execution if the sixth SELECT statement must traverse the leaf level of the index horizontally and it must access a relational table PART.

End of sample problems

Task z
Indexing
Consider TPC-R benchmark database owned by a user CSCI315 (see an Experiment 9.1 for more details related to TPC-R database).

Implement the queries listed below as SELECT statements and for each one of the queries propose the indexing schema (one or more indexes) that speeds up query processing. Consider all queries as independent such that each indexing schema for one query is independent from an indexing schema for another query. A good idea is to drop all indexes implemented for one query before testing an indexing schema for another query.

Use SQL script testsql.sql provided in Experiment 9.2 to find the execution plans and execution statistics for each implemented query. Use testsql.sql twice, first time before indexing and second time after indexing. Repeat the testing for each query.

(1)
Find name (P_NAME) and retail price (P_RETAILPRICE) of all parts in a brand (P_BRAND) Brand#51 and supplied by a supplier from CANADA (N_NAME).

(2)
Find the total number of orders issued by a customer (C_NAME) Customer#000000374 and an order includes at least one part with quantity (L_QUANTITY) greater than 40.

(3)
Find the total number of lines included in each order, list order status (O_ORDERSTATUS), order date (O_ORDERDATE), and order total price (O_TOTALPRICE).

(4)
Find the names of customers (C_NAME) from EUROPE (R_NAME) who did not include into their orders the parts supplied by a supplier Supplier#000000400 (S_NAME).

(5)
Find the quantities of items (L_QUANTITY) that got discount equal to 0.1. Do not display the same quantities more than one time and display the quantities ordered in an ascending way.

Storage management
Task y
Download a file `tpcw.pdf` and SQL script `dbcreate5.sql` and `dbdrop.sql`.

A script `dbcreate5.sql` contains SQL statements that can be used to create a TPC W sample database (`www.tpc.org`). A conceptual schema of TPC W database is included in a file `tpcw.pdf`.

The TPC W database contains information about the orders submitted by the customers, the items included in the orders, the authors of the items, the addresses and the countries the customers belong to and credit card numbers. Analyze a conceptual schema of the sample database and the referential integrity constraints to find out how information is structured in the database.

Assume, that to avoid the conflicts with the accesses to the relational tables of TPC W sample database we would like to distribute the relational tables and automatically created indexes on primary keys on three different hard drives. Do not worry if your computer does not have three hard drives. We shall simulate the drives through three different tablespaces `DRIVE_C`, `DRIVE_D`, and `DRIVE_E`. To find out, which relational tables and, which indexes should be located on each drive we shall consider the following database applications.

(6) *Find a complete information about the items whose total ordered quantity (attribute `OL_QTY`) is higher than a give value.*
(7) *Find the first name and the last name (attributes `C_FNAME`, `C_LNAME`) of the customers who ordered more than `10` items in a single order.*
(8) *Find the credit card numbers (attribute `CC_NUM`) of the customers who submitted more than `100` orders.*
(9) *Find the full addresses of the customers who ordered a given item (attribute `I_TITLE`).*
(10) *Find the dates of orders (attribute `O_DATE`) submitted by the customers living in a given country (attribute `CO_NAME`).*

Analyze the applications listed above and find which relational tables and which indexes will be used by each application and distribute the relational tables and indexes over the hard drives simulated by the tablespaces `DRIVE_C`, `DRIVE_D`, and `DRIVE_E` such, that the relational tables and indexes used in the same application are located on different hard drives. If it is impossible to distribute the relational tables and indexes used by the same application on the different hard drives then try to minimize the total number of conflicts.

In the next step of the implementation task modify SQL script `dbcreate5.sql` such that its execution creates the tablespaces `DRIVE_C`, `DRIVE_D`, and `DRIVE_E` with the parameters required below, and it creates the relational tables of TPC W sample database in the tablespaces. The requirements imposed on the tablespaces are the following.

(1) A tablespace DRIVE_C be a locally managed tablespace and it must have a uniform allocation of extents with the size of each extent equal to 64 Kbytes. The size of the tablespace must be 10 Mbytes. It must have automatic management of free space. It must not be possible to automatically extend the tablespaces created and it must consist of only one file.
(2) A tablespace DRIVE_D must be a locally managed tablespace and it must have a uniform allocation of extents with the size of each extent equal to 128 Kbytes. It must have automatic management of free space. The size of the tablespace must be 20 Mbytes. It must not be possible to automatically extend the tablespaces created and it must consist of two files each 10 Mbytes large.
(3) A tablespace DRIVE_E must be a locally managed tablespace with automatic allocation of extents. The size of tablespace must be 5 Mbytes.

The TPC W sample database must be owned by a user with the roles RESOURCE and CONNECT granted and revoked UNLIMITED TABLESPACE privilege. The user must have access to all disk space available in the tablespaces created in the previous step.

Finally, an updated script dbcreate5.sql must create the relational tables of TPC W sample database in the tablespaces DRIVE_C, DRIVE_D, and DRIVE_E in a way that minimizes the total number of conflicts when accessing the tablespace by the applications listed above. When ready, execute the updated script dbcreate5.sql and produce a report from the execution.