

Høgskolen in Telemark

**EXAM**

**865 DATABASES II**

**9.05.2005**

| | |
|---|---|
| Time: | 3 hours |
| Language: | English |
| Pages: | 8 including front page and attachments |
| Utilities: | None |
| Remarks: | Se next page. |
| Attachments: | *Simplified syntax, SQL* and *Simplified syntax, PL/SQL* |

**The results will be published via internett using "Arena høgskole". All students have received a letter containing their password for Arena, at the start of each semester. In addition you will find the results on lists at the outside of the exam office. The lists will only show your candidate number, so please write it down before starting the exam.**



Avdeling for allmenne fag

## Remarks

The exam consists of 4 separate tasks. These should be answered by **all** the candidates. Candidates that have extended exam (4 hours) should also answer a multiple choice test. This is handed out separately.

The weight of each task is shown in the table below. Note: These weights are used only for deciding the mark for the final exam. For students taking the 3 hours final exam, the final marks are calculated by weighing the final exam, the mandatory task and the mid-term. In this case the final exam counts for 60% of the final result. For students with 4 hours extended exam, the final exam counts 100%.

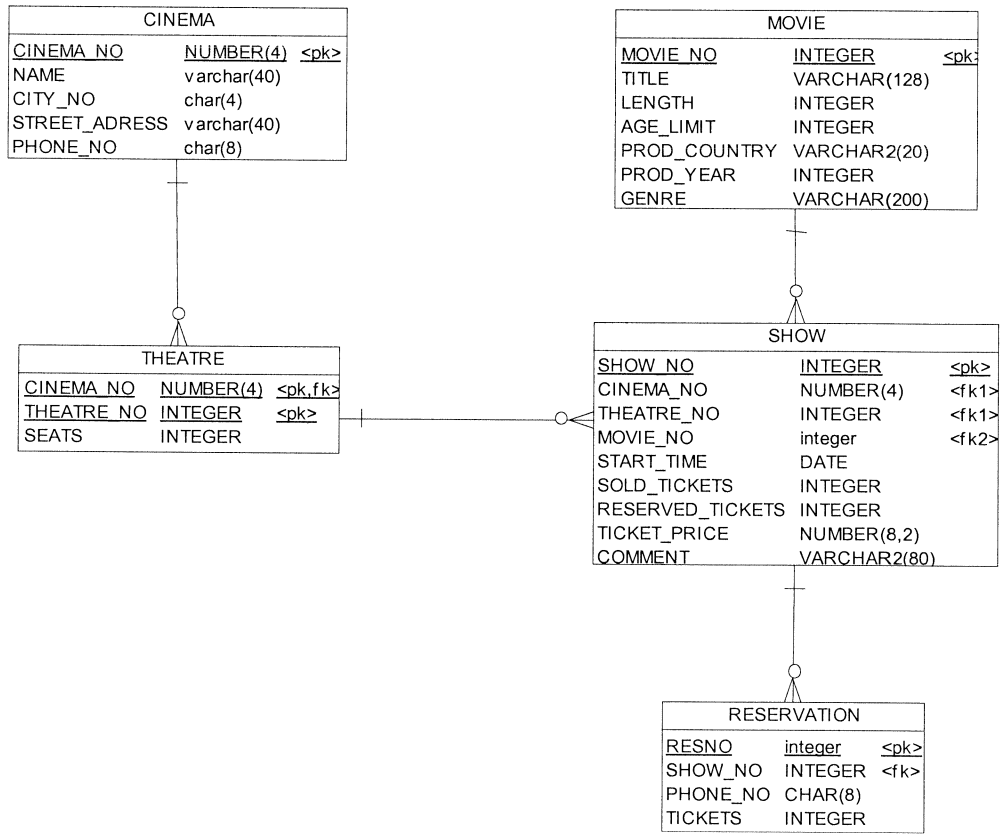
| Task | Weight 3 hours | Weight 4 hours |
|---|---|---|
| 1 | 40 % | 30 % |
| 2 | 28 % | 20 % |
| 3 | 18 % | 15 % |
| 4 | 14 % | 10 % |
| Multiple choice | Should not be answered | 25 % |
| Sum | 100% | 100% |

**Read the text thoroughly, and answer as many tasks as you can. Don't think to complicated!**

Good luck!

## Task 1

In this task you should write PL/SQL code for an Oracle database that is used for reservation and sale of cinema tickets. The database contains information about *cinemas* and *movies*, information about *shows* and *theatres,* and about the *movies* shown. Finally the database contains a table with *ticket reservations*. The database is modelled in an ER-diagram:



The ER-diagram is implemented as 5 tables:

- CINEMA(cinema_no, name, city_no, street_adress, phone_no)
- THEATRE(cinema_no*, theatre_no, seats)
- MOVIE(cinema_no, title, length, age_limit, prod_country, prod_year, genre )
- SHOW(show_no, cinema_no*, theatre_no*, cinema_no*, start_time, sold_tickets, reserved_tickets, ticketprice, comment)
- RESERVATION(resno, show_no*, phone_no, tickets)

Primary keys are underlined (and marked with <pk> in the ER-model). Foreign keys are marked with an asterix (*) (and with <fk> in the ER-model). The Oracle data type DATE stores both a date and a time.

The Oracle database also contains a sequence that is defined like this:

```
CREATE SEQUENCE sq_resno INCREMENT BY 1 START WITH 1;
```

**Subtasks**

a) Rule: *The total number of sold and reserved tickets for a show can not be greater than the number of seats in the theatre where the film is shown.* Write a trigger *tr_theatre* that makes sure that this rule is not violated if the number of seats in a theatre is changed. Explain why the rule will not be violated when new theatres are registered.

b) Write a stored function *sf_reserve_seats* that should be used to reserve seats for a show. The function should have data type INTEGER, and it should have three in-parameters: *show_no*, *phone_no*, and the number of *seats* you wish to reserve.

Before the reservation is registered in the database, the function should first check that there are enough available seats for the show. If the reservation can be done, a new reservation should be inserted and automaticly given a reservation number. The number should be taken from the Oracle sequence *sq_resno*. The return value for the function should be the reservation number for the new reservation. If there are not enough available seats for the show, the function should not insert any reservation. Instead the function should return the value -1. This value should also be returned if any other database error occurs during execution of the function.

c) At the cinema the ticked keeper can sell tickets to shows even if the customer have not reserved any seats. Create a procedure *sp_sell_tickets* that can be used for this. The procedure should increase the number of sold tickets for the show with the number of tickets being sold (in one sale). You must decide all parameters for the procedure. The procedure should have sound error-handling. It must also make sure that you don't sell more tickets than the current number of available seats for the show. Seats are available when not already sold or reserved. Assume that reservations are deleted and registered as sold tickets in *show,* when reserved tickets are collected.

e) Suggest what indexes should be created in this database. Give a short explanation for the your answer. Write SQL code to create **one of** the indexes you suggest. The SQL code should store the index physically in a table space named CINEMA.

## Task 2

Note: The questions in **task 2 should be answered very shortly**, i.e. with a few single words, or 1-2 sentences.

a) Write a SQL statement to create an Oracle user with username *peter* and password *pan.*
b) Give an example of a *system privilege* and an *object privilege* in Oracle.
c) Write a SQL statement to give the user *peter* privilege to read data from the table *cinema* in the database used in task 1.
d) You are running the SQL statement **SELECT * FROM theatre WHERE cinema_no = 1**. Explain how the Oracle access method TABLE ACCESS FULL finds the correct data when this statement is executed.
e) The concept ETL is used in relation to data warehouses. Explain in brief words what this three letter abbreviation means, and what tasks are done in an ETL process.
f) Explain the meaning of the concept *Drill-down* in relation to data warehouses.
g) Give the name of two object-oriented mechanisms in SQL-99 (SQL-3)
h) Explain the difference between the command SHUTDOWN NORMAL and SHUTDOWN ABORT in Oracle.

## Task 3

a) Structured data from a relational database (for example from a table or a query result), can be stored as a XML document. Two alternative methods can be used to represent column values in XML. Shortly describe these two methods and what pros and cons they have.
b) The table below shows the data from one row in the table *cinema*. Write a XML document that represents the data in this row. Use one of the methods for representation from step a).

| Cinema_no | Name | City_no | Street_adress | phone_no |
|---|---|---|---|---|
| 1 | Colosseum | 0301 | Fr. Nansens vei 6 | 82030000 |

c) Explain why it can be useful to store XML documents in a database, and how this can be done.

## Task 4

a) Explain shortly the difference between *instance error* and *media error* in Oracle. Explain what events can cause these errors, and how the errors can result in lost data.

b) Explain how recovery of a database can be done after a *media error*. You should mention how data files, log files and archive log files are used in the recovery process.

# Appendiks F – Forenklet syntaks, SQL

## Appendiks F – Foreklet syntaks, SQL

```
ALTER TABLE [skjema.]tabellnavn {
ADD ({kolonnenavn datatype [DEFAULT uttrykk] [kolonnefø-
  ring] | tabellføring} [,...])
|MODIFY ({kolonnenavn [datatype] [DEFAULT uttrykk] [kolonnefø-
  ring]} [,...])
|DROP {PRIMARY [CASCADE] | UNIQUE (kolonnenavn [,...])
  [CASCADE] | CONSTRAINT føring}
|DROP COLUMN kolonnenavn
|DISABLE {CONSTRAINT føring/PRIMARY KEY/UNIQUE (kolonne
  [,kolonne],...)} | [CASCADE]}
|ENABLE{VALIDATE|NOVALIDATE}{CONSTRAINT føring/PRIMARY
  KEY/UNIQUE (kolonne [,kolonne],...)}

ALTER USER brukernavn
IDENTIFIED {BY passord | EXTERNALLY}

COMMIT

CREATE [UNIQUE] INDEX [skjema.]indeksnavn ON [skjema.]tabell-
  navn (kolonnenavn [ASC|DESC ] [,...])

CREATE TABLE [skjema.]tabellnavn {({kolonnenavn datatype
  [DEFAULT uttrykk] [kolonneføring]
|tabellføring} [,...])] |AS subspørring};

CREATE USER brukernavn
IDENTIFIED {BY passord | EXTERNALLY}

CREATE [OR REPLACE] [FORCE|NO FORCE] VIEW [skjema.]viewnavn
  [(kolonnealias ...])]
AS subspørring [WITH CHECK OPTION] [CONSTRAINT føringnavn] |WITH
  READ ONLY]

DELETE [FROM] {[skjema.]tabellnavn|[skjema.]view-
  navn|(subspørring)} [alias] [WHERE betingelse]

DROP {INDEX [skjema.]indeksnavn|SEQUENCE [skjema.]sekvens-
  navn|TABLE [skjema.]tabellnavn [CASCADE CONSTRAINTS] |VIEW
  [skjema.]viewnavn}

GRANT {objektprivilegium [( kolonnenavn[,...])] |ALL [PRIVI-
  LEGES] [(kolonnenavn[,...])]} [,...]

ON [skjema.]{tabellnavn|viewnavn}TO
  {bruker|rolle|PUBLIC} [,...] [WITH GRANT OPTION]

INSERT INTO {[skjema.]tabellnavn|[skjema.]view-
  navn|(subspørring1)} [(kolonnenavn [,...])]
{VALUES (uttrykk [,...])|subspørring2}

REVOKE {objektprivilegium|ALL [PRIVILEGES]} [,...] ON
  [skjema.]{tabellnavn|viewnavn}
FROM {bruker |rolle|PUBLIC} [,...] [CASCADE CONSTRAINTS]

ROLLBACK

SELECT [ALL|DISTINCT]{*|{[skjema.]{tabellnavn|view-
  navn}.*/uttrykk [[AS] kolonnealias]}[,...]}
FROM {[skjema.]{tabellnavn|view-
  navn}[alias]|(subspørring) [alias]}[,...]
[WHERE betingelse]
[GROUP BY uttrykk [,...]
[HAVING betingelse]]
[{UNION|UNION ALL|MINUS|INTERSECT} selectsetning]
[ORDER BY {kolonnenavn|kolonneposisjon|kolonnealias} [ASC|DESC]
  [,...]]

UPDATE {[skjema.]{tabellnavn|viewnavn}|(subspørring1)} [alias]
SET {kolonnenavn={(subspørring2)|uttrykk}|(kolonne-
  navn[,...])=(subspørring3)}[,...]
[WHERE betingelse]

betingelse::=uttrykk {=|>|<|<>|<=|>=}
  {uttrykk|(subspørring)}|uttrykksliste{=|<>}(subspørring)
|uttrykk {=|>|<|<>|<=|>=}{ANY|ALL|SOME}{uttrykks-
  liste|(subspørring)}
|uttrykksliste {=|<>}{ANY|ALL|SOME}({subspørring|uttrykksliste
  [,...]})
|uttrykk [NOT] IN {uttrykksliste|(subspørring)}
|uttrykksliste [NOT] IN ({subspørring|uttrykksliste [,...]})
|uttrykk [NOT] BETWEEN uttrykk AND uttrykk] |uttrykk IS [NOT]
  NULL|EXISTS (subspørring)
|streng1 [ NOT ] LIKE streng2 | (betingelse) | NOT betingelse |
  betingelse {AND|OR} betingelse
|tabell1.kolonne = tabell2.kolonne (+) | tabell1.kolonne (+) =
  tabell2.kolonne

datatyper::= CHAR|CHAR(n)|VARCHAR2(n)|NUMBER(p,s)|DATE|LONG

funksjoner ::=
  ADD_MONTHS(date,months_to_add)|AVG([DISTINCT|ALL] uttrykk)|
  COUNT({*|[DISTINCT|ALL] uttrykk})|INITCAP
  (streng)|LENGTH(streng)|LOWER(streng)| MAX([DISTINCT|ALL]
  uttrykk)| MIN([DISTINCT|ALL] uttrykk|NVL(uttrykk1,uttrykk2)|
  SUM([DISTINCT|ALL] uttrykk)|SYSDATE| UPPER(streng)|USER

kolonneføring ::= [CONSTRAINT føringsnavn]{[NOT]
  NULL|UNIQUE|PRIMARY KEY|
```

```
REFERENCES [skjema.]tabellnavn[(kolonnenavn)][ON DELETE
  CASCADE]|CHECK (betingelse)}[DISABLE]

objektprivilegier ::= ALTER|DELETE|EXECUTE|INDEX|INSERT|REFE-
  RENCES|SELECT|UPDATE

tabellføring ::= [CONSTRAINT føringnavn]{{UNIQUE|PRIMARY
  KEY}(kolonnenavn [,...])
|FOREIGN KEY(kolonnenavn[,...]) REFERENCES [skjema.]tabellnavn
  [(kolonnenavn [,...])]
[ON DELETE CASCADE]|CHECK (betingelse)} [DISABLE]}

uttrykk ::= NULL|[[skjema.]{tabellnavn.|viewnavn.} kolonne-
  navn]|' tekst'|tall|(uttrykk [,...])
|funksjon [([DISTINCT|ALL] uttrykk [,...])|(uttrykk)|{+|-}
  uttrykk|uttrykk {*|/|+|-| ||}uttrykk
```

# Appendiks G – Forenklet syntaks, PL/SQL

## Appendiks G – Forenklet syntaks, PL/SQL

```
basic scalar datatypes:
VARCHAR2 (maximum_length)
| NUMBER[(precision,scale)]
| DATE | BOOLEAN | BINARY_INTEGER

datatypes:
scalar datatype[ datatype_option] | variable_name%TYPE [
  datatype_option] | table.column_name%TYPE [ datatype_option]
  | table%ROWTYPE [ datatype_option]

datatype_option:
{NOT NULL:=expression|:=expression}

variable declaration:
variable_name  datatype;

record:
TYPE record_name
IS RECORD (field_name   datatype ;
[field_name   datatype; ...] );

pl/sql-block:
BEGIN
   statement; [statement; ...]
[EXCEPTION
   statement; [statement; ...]]
END;

valg og løkker:
IF condition THEN
   statement;[statement;...]
[ELSIF condition THEN
   statement; [statement; ...] ...]
[ELSE
    statement; [statement; ...]
END IF;

LOOP
   statement; [statement; ...]
EXIT [WHEN condition ];
   [statement; [statement; ...]]
END LOOP;

FOR counter IN [REVERSE]
   lower_bound .. upper_bound LOOP
  statement; [statement; ...]
END LOOP;

WHILE condition  LOOP
   statement; [statement; ...]
END LOOP;

Definisjon av databaseobjekter:

CREATE [OR REPLACE] PROCEDURE prosedyrenavn
[(argument [IN|OUT|IN OUT] datatype
[,argument [IN|OUT|IN OUT] datatype ...])]
IS|AS
pl/sql-block

CREATE [OR REPLACE] FUNCTION function_name [(argument [IN]
  datatype
[,argument [IN] datatype ...])]
RETURN datatype
IS|AS
pl/sql-block

CREATE [OR REPLACE] PACKAGE package_name
  IS|AS
<public type and item specifications>
<subprogram specifications>
END package_name;

CREATE [OR REPLACE] PACKAGE BODY package_name
IS|AS
   <private type and item specifications>
   <subprogram bodies>
END package_name;

select i PL/SQL:

SELECT select_list
  INTO {variable_name [,variable_name ...]
       |record_name}
  FROM table
 WHERE condition;

trapping exceptions:

EXCEPTION
   {WHEN exception [OR exception ...] THEN
      statement; [statement; ...]
   [WHEN exception [OR exception ...] THEN
      statement; [statement;...] ...]
   [WHEN OTHERS THEN
      statement;[statement;...]]
   |WHEN OTHERS THEN statement;
      [statement; ...] }
```

user-defined exception (declaration):
*exception_name* EXCEPTION;

user-defined exception (raising):
RAISE *exception_name;*

associate non-predefined Oracle-server error and user_defined exception:
*exception_name* EXCEPTION;

PRAGMA EXCEPTION_INIT(*exception_name*,
*error_number*);

Innebygd prosedyre som kan kommunisere en serverfeil til klienten (får en PL/SQL-blokk til å feile):
PROCEDURE RAISE_APPLICATION_ERROR (
feilnummer IN NUMBER,
feilmelding IN VARCHAR2 );

der feilnummer er et valgfritt tall fra -20 999 til -20 000.

some predefined Oracle-server errors:
NO_DATA_FOUND|TOO_MANY_ROWS|INVALID_CURSOR|
ZERO_DIVIDE|DUP_VAL_ON_INDEX

markør:
CURSOR *markoer_name*
[(*parameter_name datatype*
[,*parameter_name datatype* ...)]]
IS *select_statement;*

OPEN *markoer_name;*

FETCH *markoer_name*
INTO {*variable_name* [,*variable_name* ...]
|*record_name*};

CLOSE *markoer_name;*

cursor attributes:
%ROWCOUNT|%FOUND|%NOTFOUND|%ISOPEN

event:
INSERT|UPDATE [OF *column*]|DELETE

timing:
BEFORE|AFTER

statement trigger:
CREATE [OR REPLACE] TRIGGER *trigger_name*
*timing event1* [OR *event2* OR *event3*]
ON *table_name*
[DECLARE *deklarasjoner* ]
*pl/sql-block;*

row trigger:
CREATE [OR REPLACE] TRIGGER *trigger_name*
*timing event1* [OR *event2* OR *event3*]
ON *table_name*
[REFERENCING OLD AS *old* /NEW AS *new*]
FOR EACH ROW [ WHEN *condition*]
[DECLARE *deklarasjoner* ]
pl/sql-block