# ~~EXAMEN juni 2018~~ Proefexamen 2019

# Gegevensbanken

~~22 juni 2018~~ ~~9.00 u.~~

Het examen bestaat uit drie vragen die schriftelijk uitgewerkt worden.

***Instructies***

a. De vragen moeten worden opgelost in de volgorde waarin ze genummerd zijn. Elke vraag moet op een nieuw blad worden begonnen (of in het examenformulier ingevuld) – op ***elk blad moet je je naam, jaar, studierichting en het nummer van de vraag schrijven.*** Schrijf duidelijk en overzichtelijk.
b. De antwoorden moeten worden afgegeven ten laatste 3:00 u. na aanvang van het examen.
c. Het gebruik van een blad met de SQL-syntaxis is TOEGELATEN en is als bijlage toegevoegd. Het gebruik van cursusnota's, rekenmachines, print-outs van de slides, het boek en andere hulpmiddelen is niet toegelaten.

Veel succes!

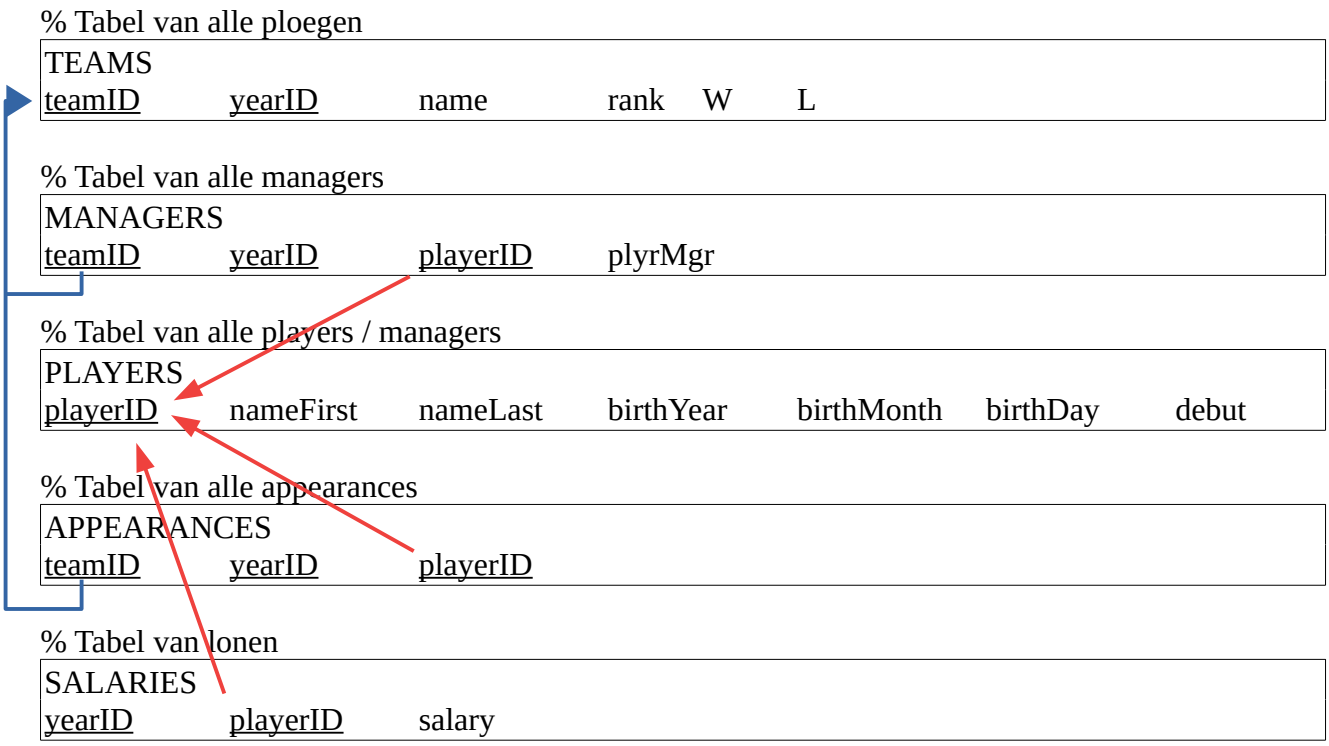
---

## 1. Meerkeuzevragen (3 punten)

*Kruis voor elke zin aan of hij waar of fout is.*
Punten voor de deze vraag = max (aantal correcte kruisjes – aantal verkeerde kruisjes, 0) / 2

| | waar | fout |
|---|---|---|
| Twee-fasen-vergrendeling vermijdt altijd deadlock. | | |
| Je kan een rooster van transacties testen op conflict-serialiseerbaarheid a.d.h.v. een graaf. | | |
| Een van de regels voor vergrendeling bij lees-/schrijfvergrendeling is: een transactie T moet ofwel een leesgrendel ofwel een schrijfgrendel hebben op X om read_item(X) te mogen uitvoeren. | | |
| Een binaire zoekboom die geordend is heeft een zoektijd die precies $\log_2(n)$ is. | | |
| Beschouw de volgende twee verzamelingen functionele afhankelijkheden:<br><br>G = {D -> BZ, A -> D, F -> Y, C -> F, E -> F} en H = {A -> Z, C -> Y, E -> FZ}.<br><br>De volgende twee beweringen zijn waar:<br><br>H overdekt de verzameling G niet. G overdekt de verzameling H. | | |
| Zwakke entiteiten hebben altijd partiële participatie in hun identificerende relatie. | | |

Beschouw het volgende relationeel schema dat een gegevensbank met gegevens over het Baseball. (alle pijlen zijn verwijssleutels; de kleur dient enkel om in geval van twee kruisende lijnen het onderscheid te maken)




2.A.1) vul de **SQL** query in om de naam van de ploegen te krijgen die na 1980 minstens 1 manager hebben gehad die speler-manager was. De tabel mag geen dubbels bevatten.

```
                        t.name
from Teams as t
where t.yearID > 1980
      ____________ (
        select * from Managers as mgr
        where mgr.plyrMgr = 'Y'
            and
                                    
    );
```

2.A.2) geef in **relationele algebra** een lijst van de clubnaam en de voor- en achternaam van alle managers die ooit voor de club gewerkt hebben als niet-playermanager.

( PLAYERS ( TEAMS

( MANAGERS )))

2.A.3) Geef in **relationele calculus** een lijst van teamnaam, rang, jaar, aantal wins en losses van de teams waarvan dat jaar alle spelers die bij dat team speelden, meer verdienden bij dat team dan 20000**.**

**{ t.name, t.yearID, t.rank, t.W, t.L**
**| (t)**
**and not [ ( a)( s) (a) and (s)**
**and a.teamID = t.teamID**
**and a.yearID = t.yearID**
**and =**
**and =**
**and s.salary**
**]**
**}**

Aandacht: In het echte examen heeft 2.A *5* onderdelen, dus nog 2 vragen meer. De bovenstaande 3 geven een idee hoe de 5 eruitzien, en in elk geval gaat het om queries in de drie vraagtalen die we behandeld hebben.

## B. Theorie (4 punten)

### 2.B.1 Chase algoritme

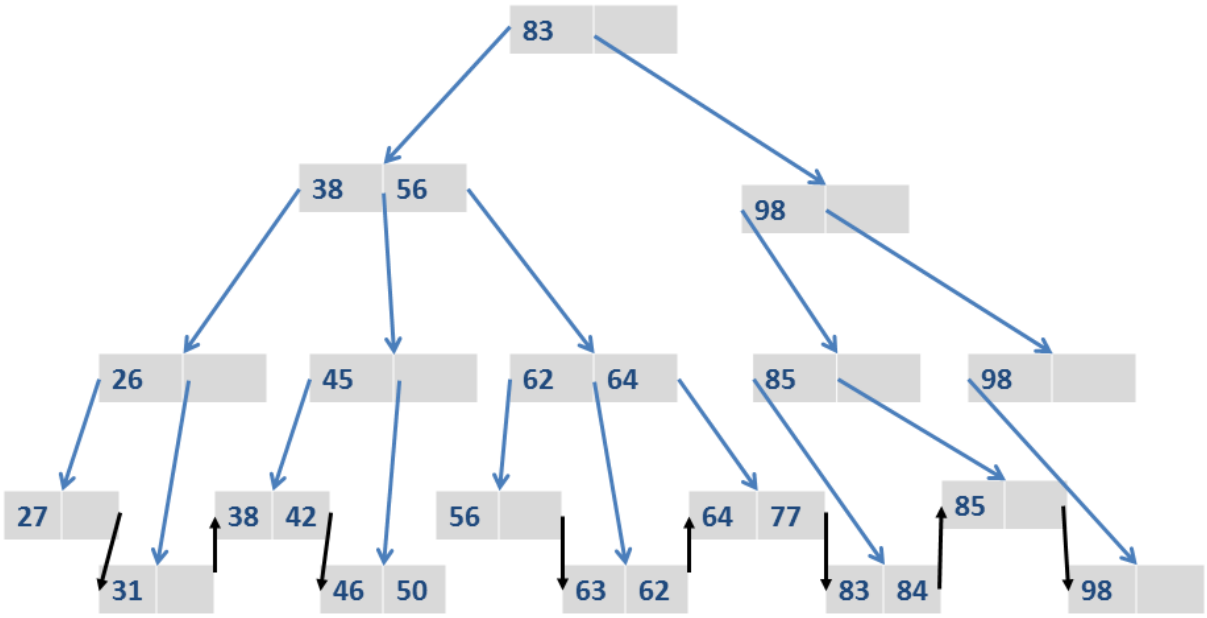
Waarvoor dient het chase algoritme?
Leg uit (in woorden, geen pseudocode!) hoe dit werkt.

### 2.B.2 SQL

Wat betekent het voor relaties in SQL om “union compatible” te zijn? Voor welke operaties is dit belangrijk?

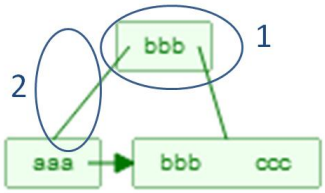
## 3. Fysiek model

*3.A (4 punten)* Deze datastructuur is (bijna) een B+-boom:



1. De meeste knopen en wijzers in deze datastructuur zijn correct. Maar er zitten 6 fouten in de datastructuur (dingen die in een B+ boom *niet* mogen voorkomen, en die ervoor zorgen dat het geen geldige B+ boom is). Verbeter ze **op de volgende manier**:

   EERST: Markeer de fouten met nummers **in de tekening op de vorige pagina**, volgens dit voorbeeld.

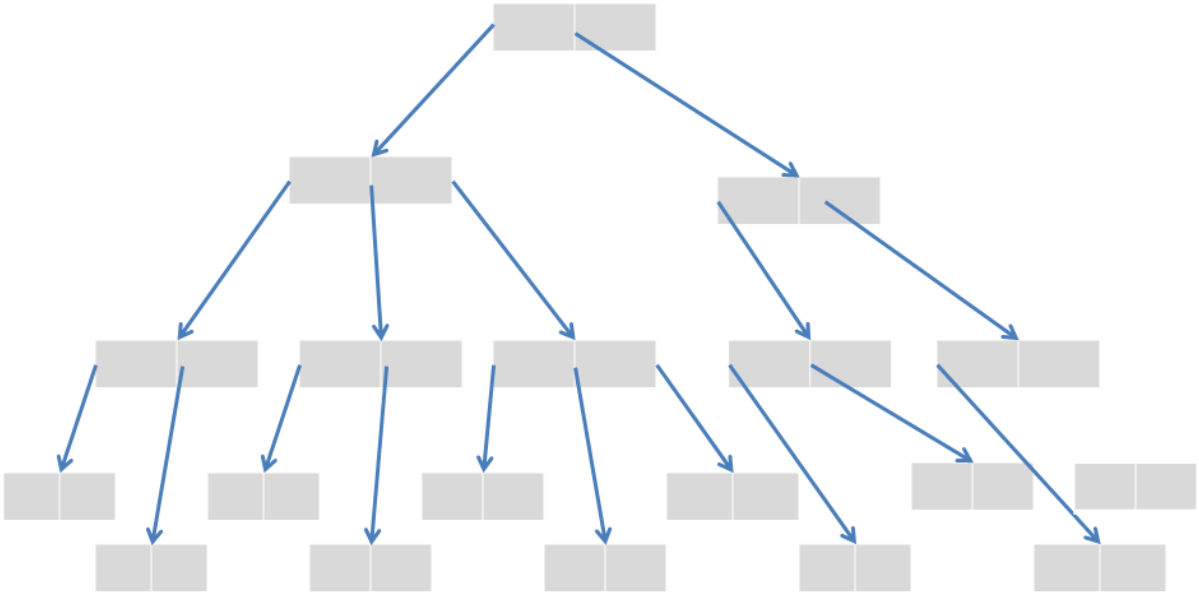



   DAN: Leg kort uit wat de fouten zijn.

**Gelieve je antwoorden HIER in te vullen:**

| Fout nummer | Beschrijving van het probleem: |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

2. Teken de gecorrigeerde boom met alle waardes en wijzers (inclusief degenen die al correct waren). Maak MINIMALE veranderingen tegenover de originele boom.



3. Voeg de waarde 40 aan de boom in 2. toe. Teken de nieuwe boom HIER.

*3.B. (4 punten)* ***Op de 3 volgende pagina's*** *zie je*

- de output van EXPLAIN voor een query op de database van het werkje,
- info over de betrokken tabellen.

**Vul alle antwoorden HIER in.**

1. Hoeveel records (totaal) verwacht MySQL te moeten verwerken ?

   _____ records

2. Is het mogelijk dat het uiteindelijk verwerkte aantal datarecords daarvan afwijkt? Geef EEN voorbeeld en leg kort uit.

   Van tabel _____ moeten eventueel meer/minder [onderstreep wat past] records verwerkt worden, omdat

3. Teken de structuur van de indexen gebruikt in rijen R1 en R2:

| Naam van de index gebruikt in R1 | Geïndexeerde waardes (attribuut/en) | Verwijzen naar |
|---|---|---|
| | | |

| Naam van de index gebruikt in R2 | Geïndexeerde waardes (attribuut/en) | Verwijzen naar |
|---|---|---|
| | | |

4. Leg kort uit wat en hoe geselecteerd en gejoind wordt. Maak gebruik van de EXPLAIN-velden type, key, ref, Extra.

| | |
|---|---|
| R1 | |
| R2 | |
| R3 | |

**QUERY**

```
EXPLAIN SELECT hof.playerID, hof.yearID, hof.votedBy,
p.birthYear, m.yearID, m.inseason
FROM HallOfFame hof, Master p, Managers m
WHERE hof.playerID = p.playerID
AND p.playerID = m.playerID
AND m.yearID <= p.birthYear+30
```

| RIJ | id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| R1 | 1 | SIMPLE | m | Index | PRIMARY,yearID_i | yearID_i | 6 | *NULL* | 3436 | Using index | |
| R2 | 1 | SIMPLE | hof | Ref | PRIMARY | PRIMARY | 767 | lahman2016.m.playerID | 1 | Using index | |
| R3 | 1 | SIMPLE | p | eq_ref | PRIMARY | PRIMARY | 767 | lahman2016.m.playerID | 1 | Using where | |

**TABELLEN**

# HallOfFame

| Column | Type | Null | Default |
|---|---|---|---|
| playerID *(Primary)* | varchar(255) | No | |
| yearid *(Primary)* | int(11) | No | |
| votedBy *(Primary)* | varchar(255) | No | |
| ballots | int(11) | Yes | *NULL* |
| needed | int(11) | Yes | *NULL* |
| votes | int(11) | Yes | *NULL* |
| inducted | varchar(255) | Yes | *NULL* |
| category | varchar(255) | Yes | *NULL* |
| needed_note | varchar(255) | Yes | *NULL* |

## Indexes

| Keyname | Type | Unique | Packed | Column | Cardinality | Collation | Null |
|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | Yes | No | playerID | 4156 | A | No |
| | | | | yearid | 4156 | A | No |
| | | | | votedBy | 4156 | A | No |
| yearid | BTREE | No | No | yearid | 159 | A | No |

# Managers

| Column | Type | Null | Default |
|---|---|---|---|
| playerID *(Primary)* | varchar(255) | No | |
| yearID *(Primary)* | int(11) | No | |
| teamID | varchar(255) | Yes | *NULL* |
| lgID | varchar(255) | Yes | *NULL* |
| inseason *(Primary)* | int(11) | No | |
| G | int(11) | Yes | *NULL* |
| W | int(11) | Yes | *NULL* |
| L | int(11) | Yes | *NULL* |
| rank | int(11) | Yes | *NULL* |
| plyrMgr | varchar(255) | Yes | *NULL* |

## Indexes

| Keyname | Type | Unique | Packed | Column | Cardinality | Collation | Null |
|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | Yes | No | playerID | 1718 | A | No |
| | | | | yearID | 3436 | A | No |
| | | | | inseason | 3436 | A | No |
| plyrMgr_i | BTREE | No | No | plyrMgr | 4 | A | Yes |
| yearID_i | BTREE | No | No | yearID | 312 | A | No |
| lgID | BTREE | No | No | lgID | 14 | A | Yes |

# Master

| Column | Type | Null | Default |
|---|---|---|---|
| playerID *(Primary)* | varchar(255) | No | |
| birthYear | int(11) | Yes | *NULL* |
| birthMonth | int(11) | Yes | *NULL* |
| birthDay | int(11) | Yes | *NULL* |
| birthCountry | varchar(255) | Yes | *NULL* |
| birthState | varchar(255) | Yes | *NULL* |
| birthCity | varchar(255) | Yes | *NULL* |
| deathYear | varchar(255) | Yes | *NULL* |
| deathMonth | varchar(255) | Yes | *NULL* |
| deathDay | varchar(255) | Yes | *NULL* |
| deathCountry | varchar(255) | Yes | *NULL* |
| deathState | varchar(255) | Yes | *NULL* |
| deathCity | varchar(255) | Yes | *NULL* |
| nameFirst | varchar(255) | Yes | *NULL* |
| nameLast | varchar(255) | Yes | *NULL* |
| nameGiven | varchar(255) | Yes | *NULL* |
| weight | int(11) | Yes | *NULL* |
| height | int(11) | Yes | *NULL* |
| bats | varchar(255) | Yes | *NULL* |
| throws | varchar(255) | Yes | *NULL* |
| debut | varchar(255) | Yes | *NULL* |
| finalGame | varchar(255) | Yes | *NULL* |
| retroID | varchar(255) | Yes | *NULL* |
| bbrefID | varchar(255) | Yes | *NULL* |

## Indexes

| Keyname | Type | Unique | Packed | Column | Cardinality | Collation | Null |
|---|---|---|---|---|---|---|---|
| PRIMARY | BTREE | Yes | No | playerID | 19067 | A | No |
| debut_i | BTREE | No | No | debut | 19067 | A | Yes |

**Table 8.1** Summary of SQL Syntax

```
CREATE TABLE <table name> (<column name> <column type> [<attribute constraint>]
                          {, <column name> <column type> [<attribute constraint>] }
                          [<table constraint> {,<table constraint>}])

DROP TABLE <table name>

ALTER TABLE <table name> ADD <column name> <column type>

SELECT [DISTINCT] <attribute list>
FROM (<table name> { <alias>} | <joined table>) {, (<table name> { <alias>} | <joined table>) }
[WHERE <condition>]
[GROUP BY <grouping attributes> [HAVING <group selection condition> ] ]
[ORDER BY <column name> [<order>] {, <column name> [<order>] } ]

<attribute list>::= (* | ( <column name> | <function>(([DISTINCT]<column name> | *)))
                        {,( <column name> | <function>(([DISTINCT] <column name> | *)) } ) )
<grouping attributes>::= <column name> { , <column name>}
<order>::= (ASC | DESC)

INSERT INTO <table name> [( <column name>{, <column name>} ) ]
(VALUES ( <constant value> , { <constant value>} ){,(<constant value>{,<constant value>})}
| <select statement>)

DELETE FROM <table name>
[WHERE <selection condition>]

UPDATE <table name>
SET <column name>=<value expression> { , <column name>=<value expression> }
[WHERE <selection condition>]

CREATE [UNIQUE] INDEX <index name>*
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[CLUSTER]

DROP INDEX <index name>

CREATE VIEW <view name> [ ( <column name> { , <column name> } ) ]
AS <select statement>

DROP VIEW <view name>
```

*These last two commands are not part of standard SQL2.