University of Basel

Dr. Ilir Fetai
Prof. Dr. Heiko Schuldt
David Lengweiler, M. Sc.
Martin Vahlensieck, M. Sc.

Database Systems — Autumn 2025

Exercise 4 — Hand-in: 23.11.2025 (11:59 pm)

Solving the Exercises: The exercises must be solved in groups of three people. Use the notations introduced in the lecture. Please choose the same partners for exercises and project. The DMI plagiarism guidelines apply for this lecture.

Submission Information: Please upload your solutions BEFORE the deadline to ADAM using the team hand-in feature. Solutions that are handed in too late cannot be considered. The use of the exercise template, located in "Exercise Material", is mandatory.

Task 1: PL/SQL Stored Procedures (15 points)

As the database administrator for a large bank, you have been assigned the task of designing and setting up a new database from scratch. After careful planning, you have defined the following table definitions:

```
CREATE TABLE Customer (
    customer_id     INT NOT NULL PRIMARY KEY,
    first_name      VARCHAR(50) NOT NULL,
    last_name       VARCHAR(50) NOT NULL,
    email           VARCHAR(100) UNIQUE NOT NULL,
    balance         DECIMAL(10, 2) DEFAULT 0
);

CREATE TABLE BankTransaction (
    transaction_id      INT PRIMARY KEY,
    customer_id         INT NOT NULL,
    transaction_date    DATE NOT NULL,
    amount              DECIMAL(10, 2) NOT NULL,
    description         VARCHAR(255),
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);
```

a) Each customer's balance should always match the sum of their transactions. Write a function with PL/SQL which checks this for a specified customer and returns whether the balance is consistent with the total of all associated transactions. You can assume that the database is in a consistent state and the costumer exists.

(5 points)

b) For regulatory compliance, the balance consistency check from the previous task must be performed regularly for each customer. Initially, this was planned as part of the banking client application, but you identified an opportunity to simplify the process by using a stored procedure in the database.

Write a stored procedure with PL/SQL that executes the balance check function for all customers and reports whether all account balances are consistent with their respective transactions. It is sufficent to report if the database is consistent, the specific reason can be ignored.

(4 points)

c) When a customer receives money, updates are required in both the *Customer* and *Transaction* tables. If not handled carefully, this can lead to data inconsistencies.

Write a PL/SQL stored procedure to ensure both tables are updated correctly and consistently when a customer receives a payment. Make sure to check that the customer exists in the database, and add handling when this is not the case.

(6 points)

Task 2: Data Storage — Movie database (15 points)

Given is a movie database with the following schema definition:

```
CREATE TABLE Movie (
  MovieID       NUMBER PRIMARY KEY NOT NULL,
  MovieTitle    VARCHAR(255) UNIQUE
);

CREATE TABLE Scene (
  MovieID       NUMBER REFERENCES Movie,
  SceneID       NUMBER,
  Title         VARCHAR(255),
  PRIMARY KEY   (MovieID, SceneID)
);

CREATE TABLE Person (
  PID           NUMBER PRIMARY KEY NOT NULL,
  Name          VARCHAR(40) NOT NULL,
  Nationality   CHAR(5)
);

CREATE TABLE Actor (
  PID           NUMBER NOT NULL REFERENCES Person,
  MovieID       NUMBER NOT NULL,
  SceneID       NUMBER NOT NULL,
  Occupation    VARCHAR(20) NOT NULL,
  PRIMARY KEY   (PID, MovieID, SceneID),
  FOREIGN KEY   (MovieID, SceneID) REFERENCES Scene
);
```

We assume the following cardinalities and average tuple lengths for the given relations:

Movie	1000 tuples at	100 Bytes
Scene	10 000 tuples at	50 Bytes
Person	6000 tuples at	40 Bytes
Actor	30 000 tuples at	40 Bytes

You can assume a uniform distribution of key values. The following *indirect* and *unclustered* indices are generated as B^+-trees:

```
CREATE UNIQUE INDEX MovieIDIdx ON Movie (MovieID);
CREATE INDEX MovieTitleIdx ON Movie (MovieTitle);
CREATE INDEX ActorIdx ON Actor (PID);
```

Assume that the relation occupies its own table space with page size 4 KiB (4096 B) and the page header constitutes 96 B. Data pages are filled up to 90 %, index pages up to 70 %. The data type `NUMBER` occupies 10 B. Further, the Movie title is unique for the relation `Movie` and has an average length of 90 B. Slot array pointers, tuple identifiers (TID) and the pointers in the B^+-tree have each a length of 6 B.

For the following questions hand in both your calculations and the final results.

a) Calculate the storage space used to store the primary data, i.e., calculate the number

of tuples per page and the number of pages that are necessary to store the data in the relations of the given schemas.

Relation	Tuples per page	Number of pages
`Movie`		
`Scene`		
`Person`		
`Actor`		

```
\begin{tabular}{|l||c|c|}
  \hline
  Relation & Tuples per page & Number of pages \\
  \hline\hline
  \texttt{Movie}          &  &  \\ \hline
  \texttt{Scene}          &  &  \\ \hline
  \texttt{Person}         &  &  \\ \hline
  \texttt{Actor} &  &  \\ \hline
\end{tabular}
```

Listing 1: LaTeXcode for the table above

(4 points)

b) Calculate the storage space used to store the indices, i.e., calculate the number of entries per leaf node and per inner node, as well as – using these results – the number of leaf nodes and the number of inner nodes.

Index	Entries per leaf nodes	Entries per inner nodes	Number of leaf nodes	Number of inner nodes	Index Size [KiB]
`MovieIDIdx`					
`MovieTitleIdx`					
`ActorIdx`					

```
\begin{tabular}{|l||c|c||c|c||c|}
  \hline
  & \multicolumn{2}{c||}{Entries per}
  & \multicolumn{2}{c||}{Number of}
  & \multicolumn{1}{c|}{Index Size} \\
  Index & leaf nodes & inner nodes & leaf nodes & inner nodes & [KiB] \\
  \hline\hline
  \texttt{MovieIDIdx}      &  &  &  &  &  \\ \hline
  \texttt{MovieTitleIdx}  &  &  &  &  &  \\ \hline
  \texttt{ActorIdx} &  &  &  &  &  \\ \hline
\end{tabular}
```

Listing 2: LaTeXcode for the table above

(7 points)

c) Calculate the number of page accesses (index and data page accesses) in the following three queries by using the corresponding index structures. You can assume that all tuples queried for exist in the relation.

```
SELECT * FROM Movie WHERE MovieID = 4711;
SELECT * FROM Movie WHERE MovieTitle = 'Opelgang';
SELECT * FROM Actor WHERE PID = 1199;
```

Relation	Attribute	Page accesses
Movie	MovieID	
Movie	MovieTitle	
Actor	PID	

```
\begin{tabular}{|l|l||c|}
  \hline
  Relation              & Attribute & Page accesses \\
  \hline\hline
  \texttt{Movie}         & \texttt{MovieID}    &  \\ \hline
  \texttt{Movie}         & \texttt{MovieTitle} &  \\ \hline
  \texttt{Actor} & \texttt{PID}       &  \\ \hline
\end{tabular}
```

Listing 3: LaTeXcode for the table above

(4 points)