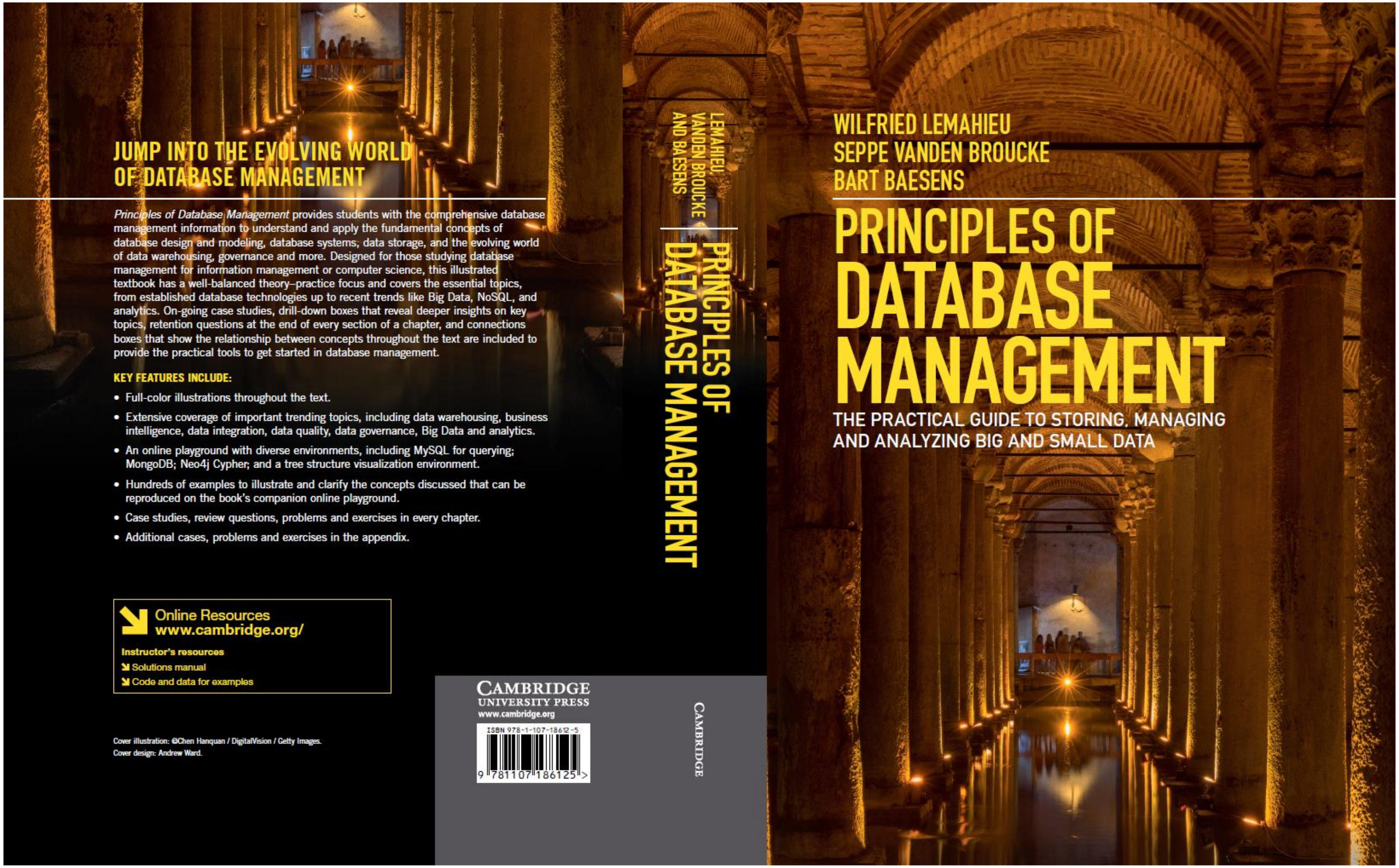
Extended Relational Databases



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources

- Solutions manual
- Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU,
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF
DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com

Introduction

- Success of the relational model
- Limitations of the Relational Model
- Active RDBMS Extensions
- Object-Relational RDBMS extensions
- Recursive SQL queries

Success of the Relational Model

- Relational model: tuples and relations
- The relational model requires all relations to be normalized
- The relational model is also referred to as a value based model (↔ identity based OO model)
- SQL is an easy to learn, descriptive and non-navigational data manipulation language (DML)

Limitations of the Relational Model

- Complex objects are difficult to handle
- Due to the normalization, the relational model has a flat structure
  - expensive joins needed to de-fragment the data before it can be successfully used
- Specialization, categorization and aggregation cannot be directly supported

Limitations of the Relational Model

- Only two type constructors: tuple constructor and set constructor
- Tuple constructor can only be used on atomic values
- Set constructor can only be used on tuples
- Both constructors are not orthogonal
- Not possible to model behavior or store functions
- Poor support for audio, video, text

Active RDBMS Extensions

- Traditional RDBMSs are **passive**
- Modern day RDBMSs are **active**
- Triggers and stored procedures

Triggers

- A **trigger** is a piece of SQL code consisting of declarative and/or procedural instructions and stored in the catalog of the RDBMS
- Triggers can also reference attribute types in other tables

Triggers

EMPLOYEE(<u>SSN</u>, ENAME, SALARY, BONUS, JOBCODE, *DNR*)

DEPARTMENT(<u>DNR</u>, DNAME, TOTAL-SALARY, *MGNR*)

```
CREATE TRIGGER SALARYTOTAL
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.DNR IS NOT NULL)
UPDATE DEPARTMENT
SET TOTAL-SALARY = TOTAL-SALARY + NEW.SALARY
WHERE DNR = NEW.DNR
```

After Trigger!

Triggers

WAGE(<u>JOBCODE</u>, BASE_SALARY, BASE_BONUS)

```
CREATE TRIGGER WAGEDEFAULT
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AS NEWROW
FOR EACH ROW
SET (SALARY, BONUS) =
(SELECT BASE_SALARY, BASE_BONUS
FROM WAGE
WHERE JOBCODE = NEWROW.JOBCODE)
```

Before Trigger!

Triggers

- Advantages
  - Automatic monitoring and verification in case of specific events or situations
  - Modelling extra semantics and/or integrity rules without changing the user front-end or application code
  - Assign default values to attribute types for new tuples
  - Synchronic updates in case of data replication;
  - Automatic auditing and logging which may be hard to accomplish in any other application layer
  - Automatic exporting of data

Triggers

- Disadvantages
  - Hidden functionality, which may be hard to follow-up and manage
  - Cascade effects leading up to an infinite loop of a trigger triggering another trigger etc.
  - Uncertain outcomes if multiple triggers for the same database object and event are defined
  - Deadlock situations
  - Debugging complexities since they don’t reside in an application environment
  - Maintainability and performance problems

Stored Procedures

- A **stored procedure** is a piece of SQL code consisting of declarative and/or procedural instructions and stored in the catalog of the RDBMS
- Needs to be invoked explicitly

Stored Procedures

```sql
CREATE PROCEDURE REMOVE-EMPLOYEES
(DNR-VAR IN CHAR(4), JOBCODE-VAR IN CHAR(6)) AS
BEGIN
DELETE FROM EMPLOYEE
WHERE DNR = DNR-VAR AND JOBCODE = JOBCODE-VAR;
END
```

```java
import java.sql.CallableStatement;
…
CallableStatement cStmt = conn.prepareCall("{call REMOVE-EMPLOYEES(?, ?)}");
cStmt.setString(1, "D112");
cStmt.setString(2, "JOB124");
```

Stored Procedures

- Advantages
  - Similar to OODBMSs, they store behavior in the database
  - They can reduce network traffic
  - They can be implemented in an application-independent way
  - They improve data and functional independence
  - They can be used as a container for several SQL instructions that logically belong together
  - They are easier to debug in comparison to triggers

Object-Relational RDBMS extensions

- OODBMSs are perceived as very complex to work with
  - No good standard DML (e.g. SQL)
  - Lack of a transparent 3-layer database architecture
- Object-Relational DBMSs (ORDBMSs) keep the relation as the fundamental building block and SQL as the core DDL/DML, but with the following OO extensions:
  - User-Defined Types (UDTs)
  - User-Defined Functions (UDFs)
  - Inheritance
  - Behavior
  - Polymorphism
  - Collection types
  - Large objects (LOBs)

User-Defined Types (UDTs)

- Standard SQL: CHAR, VARCHAR, INT, FLOAT, DOUBLE, DATE, TIME, BOOLEAN, etc.
- **User-Defined Types (UDT)** define customized data types with specific properties
- Five types:
  - Distinct data types: extend existing SQL data types
  - Opaque data types: define entirely new data types
  - Unnamed row types: use unnamed tuples as attribute values
  - Named row types: use named tuples as attribute values
  - Table data types: define tables as instances of table types

Distinct data types

- A **distinct data type** is a user-defined data type which specializes a standard, built-in SQL data type.

```
CREATE DISTINCT TYPE US-DOLLAR AS DECIMAL(8,2)
CREATE DISTINCT TYPE EURO AS DECIMAL(8,2)

CREATE TABLE ACCOUNT
(ACCOUNTNO SMALLINT PRIMARY KEY NOT NULL,
…
AMOUNT-IN-DOLLAR US-DOLLAR,
AMOUNT-IN-EURO EURO)
```

Distinct data types

- Once a distinct data type has been defined, the ORDBMS will automatically create two casting functions

```
SELECT *
FROM ACCOUNT
WHERE AMOUNT-IN-EURO > 1000
```

ERROR!

```
SELECT *
FROM ACCOUNT
WHERE AMOUNT-IN-EURO > EURO(1000)
```

Opaque data types

- An **opaque data type** is an entirely new, user-defined data type, not based upon any existing SQL data type.
- Examples: data types for image, audio, video, fingerprints, text, spatial data, RFID tags, QR codes, etc.

Opaque data types

```
CREATE OPAQUE TYPE IMAGE AS <…>
CREATE OPAQUE TYPE FINGERPRINT AS <…>

CREATE TABLE EMPLOYEE
 (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

Unnamed Row Types

- An **unnamed row type** includes a composite data type in a table by using the keyword ROW.
- It consists of a combination of data types such as built-in types, distinct types, opaque types, etc.

```
CREATE TABLE EMPLOYEE
 (SSN SMALLINT NOT NULL,
  NAME ROW(FNAME CHAR(25), LNAME CHAR(25)),
  ADDRESS ROW(
   STREET ADDRESS CHAR(20) NOT NULL,
   ZIP CODE CHAR(8),
   CITY CHAR(15) NOT NULL),
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

Named row types

- A **named row type** is a user-defined data type which groups a coherent set of data types into a new composite data type and assigns a meaningful name to it
- can be used in table definitions, queries, or anywhere else a standard SQL data type can be used
- Note: the usage of (un)named row types implies the end of the first normal form!

Named row types

```
CREATE ROW TYPE ADDRESS AS
(STREET ADDRESS CHAR(20) NOT NULL,
ZIP CODE CHAR(8),
CITY CHAR(15) NOT NULL)


CREATE TABLE EMPLOYEE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS ADDRESS,
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

Named row types

```
SELECT LNAME, EMPADDRESS
FROM EMPLOYEE
WHERE EMPADDRESS.CITY = 'LEUVEN'
```

```
SELECT E1.LNAME, E1.EMPADDRESS
FROM EMPLOYEE E1, EMPLOYEE E2
WHERE E1.EMPADDRESS.CITY =
E2.EMPADDRESS.CITY
AND E2.SSN = '123456789'
```

Table data types

- A **table data type** (or typed table) defines the type of a table.
  - Similar to a class in OO

```
CREATE TYPE EMPLOYEETYPE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS ADDRESS
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

Table data types

```
CREATE TABLE EMPLOYEE OF TYPE
EMPLOYEETYPE PRIMARY KEY (SSN)

CREATE TABLE EX-EMPLOYEE OF TYPE
EMPLOYEETYPE PRIMARY KEY (SSN)
```

Table data types

```
CREATE TYPE DEPARTMENTTYPE
  (DNR SMALLINT NOT NULL,
  DNAME CHAR(25) NOT NULL,
  DLOCATION ADDRESS
  MANAGER REF(EMPLOYEETYPE))
```

Note: reference can be replaced by the actual data it refers to by means of the DEREF (from dereferencing) function.

User-Defined Functions (UDFs)

- Every RDBMS comes with a set of built-in functions, e.g., MIN(), MAX(), AVG(), etc.
- **User-Defined Functions (UDFs)** allow users to extend these by explicitly defining their own functions
- Every UDF consists of
  - name
  - input and output arguments
  - implementation

User-Defined Functions (UDFs)

- UDFs are stored in the ORDBMS and hidden from the applications
- UDFs can be overloaded
- Types
  - sourced functions
  - external functions

Sourced function

- UDF which is based on an existing, built-in function

```
CREATE DISTINCT TYPE MONETARY AS DECIMAL(8,2)


CREATE TABLE EMPLOYEE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS ADDRESS,
  SALARY MONETARY,
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

Sourced function

```
CREATE FUNCTION AVG(MONETARY)
RETURNS MONETARY
SOURCE AVG(DECIMAL(8,2))

SELECT DNR, AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNR
```

External functions

- **External functions** are written in an external host language
  - Python, C, Java, etc.
- Can return a single value (scalar) or table of values

Inheritance

- ORDBMS extends an RDBMS by providing explicit support for inheritance, both at
  - the level of a data type
  - the level of a typed table

Inheritance at data type level

- Child data type inherits all the properties of a parent data type and can then further specialize it by adding specific characteristics

```
CREATE ROW TYPE ADDRESS AS
  (STREET ADDRESS CHAR(20) NOT NULL,
  ZIP CODE CHAR(8),
  CITY CHAR(15) NOT NULL)
```

```
CREATE ROW TYPE INTERNATIONAL_ADDRESS AS
 (COUNTRY CHAR(25) NOT NULL) UNDER ADDRESS
```

Inheritance at data type level

```
CREATE TABLE EMPLOYEE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS INTERNATIONAL_ADDRESS,
  SALARY MONETARY,
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)
```

```
SELECT FNAME, LNAME, EMPADDRESS
FROM EMPLOYEE
WHERE EMPADDRESS.COUNTRY = 'Belgium'
AND EMPADDRESS.CITY LIKE 'Leu%'
```

Inheritance at Table Type Level

```
CREATE TYPE EMPLOYEETYPE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS INTERNATIONAL_ADDRESS
  …
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE)

CREATE TYPE ENGINEERTYPE AS
  (DEGREE CHAR(10) NOT NULL,
  LICENSE CHAR(20) NOT NULL) UNDER EMPLOYEETYPE

CREATE TYPE MANAGERTYPE AS
  (STARTDATE DATE,
   TITLE CHAR(20)) UNDER EMPLOYEETYPE
```

Inheritance at Table Type Level

```
CREATE TABLE EMPLOYEE OF TYPE EMPLOYEETYPE PRIMARY KEY (SSN)
CREATE TABLE ENGINEER OF TYPE ENGINEERTYPE UNDER EMPLOYEE
CREATE TABLE MANAGER OF TYPE MANAGERTYPE UNDER EMPLOYEE
```

```
SELECT SSN, FNAME, LNAME, STARTDATE, TITLE
FROM MANAGER
```

```
SELECT SSN, FNAME, LNAME
FROM EMPLOYEE
```

```
SELECT SSN, FNAME, LNAME
FROM ONLY EMPLOYEE
```

Behavior

- E.g, triggers, stored procedures or UDFs
- ORDBMS can include the signature or interface of a method in the definitions of data types and tables
    - Information hiding

Behavior

```
CREATE TYPE EMPLOYEETYPE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS INTERNATIONAL_ADDRESS,
  …
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE,
  FUNCTION AGE(EMPLOYEETYPE) RETURNS INTEGER)
```

```
CREATE TABLE EMPLOYEE OF TYPE EMPLOYEETYPE PRIMARY KEY (SSN)
```

```
SELECT SSN, FNAME, LNAME, PHOTOGRAPH
FROM EMPLOYEE
WHERE AGE = 60
```

Polymorphism

- Subtype inherits both the attribute types and functions of its supertype
- Subtype can also override functions to provide more specialized implementations
- Polymorphism: same function call can invoke different implementations

Polymorphism

```
CREATE FUNCTION TOTAL_SALARY(EMPLOYEE E)
RETURNING INT
AS SELECT E.SALARY

CREATE FUNCTION TOTAL_SALARY(MANAGER M)
RETURNING INT
AS SELECT M.SALARY + <monthly_bonus>

SELECT TOTAL_SALARY FROM EMPLOYEE
```

Collection Types

- Can be instantiated as a collection of instances of standard data types or UDTs
- Set: unordered collection, no duplicates
- Multiset or bag: unordered collection, duplicates allowed
- List: ordered collection, duplicates allowed
- Array: ordered and indexed collection, duplicates allowed
- Note: end of the first normal form!

Collection Types

```
CREATE TYPE EMPLOYEETYPE
  (SSN SMALLINT NOT NULL,
  FNAME CHAR(25) NOT NULL,
  LNAME CHAR(25) NOT NULL,
  EMPADDRESS INTERNATIONAL_ADDRESS,
  …
  EMPFINGERPRINT FINGERPRINT,
  PHOTOGRAPH IMAGE,
  TELEPHONE SET (CHAR(12)),
 FUNCTION AGE(EMPLOYEETYPE) RETURNS INTEGER)

CREATE TABLE EMPLOYEE OF TYPE EMPLOYEETYPE (PRIMARY KEY SSN)

SELECT SSN, FNAME, LNAME
FROM EMPLOYEE
WHERE '2123375000' IN (TELEPHONE)
```

Collection Types

```
SELECT T.TELEPHONE
FROM THE (SELECT TELEPHONE FROM EMPLOYEE) AS T
ORDER BY T.TELEPHONE
```

Collection Types

```
CREATE TYPE DEPARTMENTTYPE AS
  (DNR CHAR(3) NOT NULL,
  DNAME CHAR(25) NOT NULL,
  MANAGER REF(EMPLOYEETYPE),
  PERSONNEL SET (REF(EMPLOYEETYPE))
```

```
CREATE TABLE DEPARTMENT OF TYPE DEPARTMENTTYPE (PRIMARY KEY DNR)
```

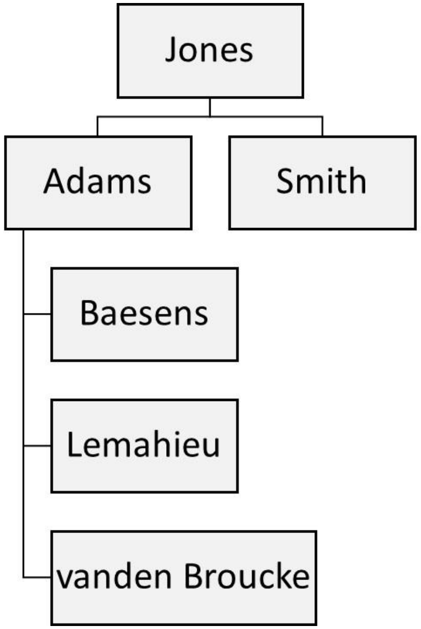
```
SELECT PERSONNEL
FROM DEPARTMENT
WHERE DNR = '123‘
```

```
SELECT DEREF(PERSONNEL).FNAME, DEREF(PERSONNEL).LNAME
FROM DEPARTMENT
WHERE DNR = '123'
```

Large objects

- Multimedia database applications
- LOB data will be stored in a separate table and tablespace
- Types of LOB data:
  - **BLOB (Binary Large Object):** a variable-length binary string whose interpretation is left to an external application
  - **CLOB (Character Large Object):** variable-length character strings made up of single-byte characters
  - **DBCLOB (Double Byte Character Large Object):** variable-length character strings made up of double-byte characters.

Recursive SQL Queries



Jones

Adams

Smith

Baesens

Lemahieu

vanden Broucke

- Employee(SSN, Name, Salary, *MNGR*)

Recursive SQL Queries

SSN	Name	Salary	MNGR
1	Jones	10.000	NULL
2	Baesens	2.000	3
3	Adams	5.000	1
4	Smith	6.000	1
5	vanden	3.000	3

Recursive SQL Queries

```
WITH SUBORDINATES(SSN, NAME, SALARY, MNGR, LEVEL)
AS
((SELECT SSN, NAME, SALARY, MNGR, 1
FROM EMPLOYEE
WHERE MNGR=NULL)
UNION ALL
(SELECT E.SSN, E.NAME, E.SALARY, E.MNGR, S.LEVEL+1
 FROM SUBORDINATES AS S, EMPLOYEE AS E
 WHERE S.SSN=E.MNGR)


SELECT * FROM SUBORDINATES
ORDER BY LEVEL
```

Recursive SQL Queries

SSN	NAME	SALARY	MNGR	LEVEL
1	Jones	10.000	NULL	1

SSN	NAME	SALARY	MNGR	LEVEL
3	Adams	5.000	1	2
4	Smith	6.000	1	2

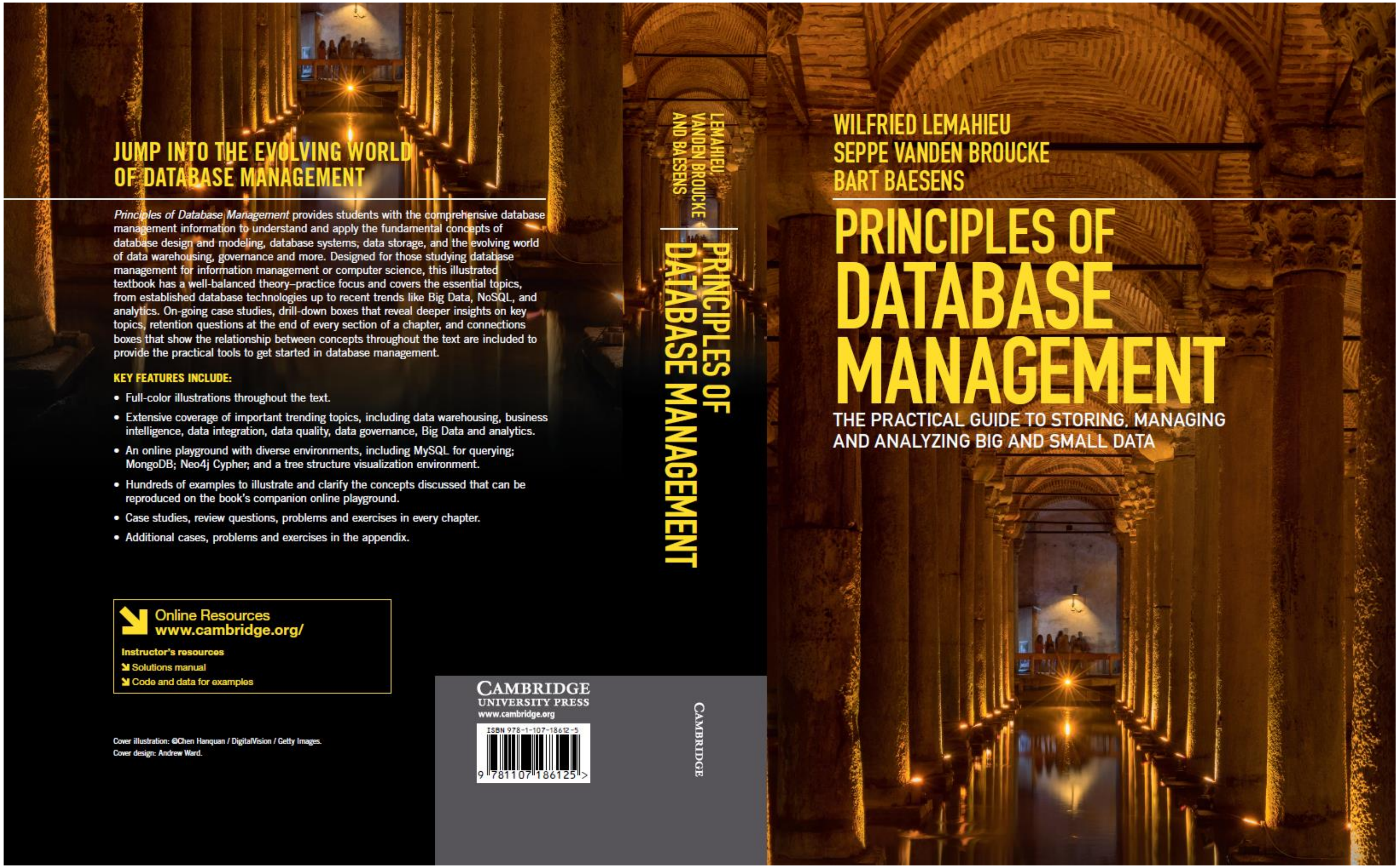
SSN	NAME	SALARY	MNGR	LEVEL
2	Baesens	2.000	3	3
5	vanden Broucke	3.000	3	3
6	Lemahieu	2.500	3	3

SSN	NAME	SALARY	MNGR	LEVEL
1	Jones	10.000	NULL	1
3	Adams	5.000	1	2
4	Smith	6.000	1	2
2	Baesens	2.000	3	3
5	vanden Broucke	3.000	3	3
6	Lemahieu	2.500	3	3

Conclusions

- Success of the relational model
- Limitations of the Relational Model
- Active RDBMS Extensions
- Object-Relational RDBMS extensions
- Recursive SQL queries

More information?



JUMP INTO THE EVOLVING WORLD
OF DATABASE MANAGEMENT

Principles of Database Management provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

KEY FEATURES INCLUDE:

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
www.cambridge.org/

Instructor's resources

- Solutions manual
- Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org
ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF
DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF
DATABASE
MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING
AND ANALYZING BIG AND SMALL DATA

www.pdbmbook.com