

Data Science

Markus Haslinger

DBI/INSY

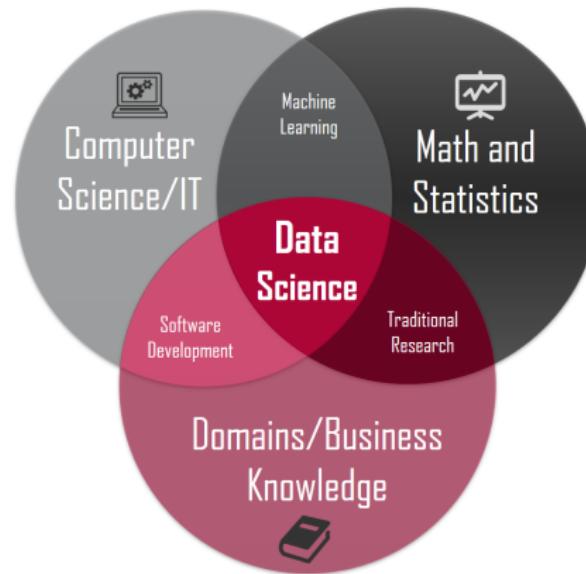
What is Data Science?

- Two parts:
 - Data – to be precise *raw data*
 - Science – a methodical approach

Definition

Data Science = Extraction of *knowledge* from data.

An interdisciplinary field



Source: <https://www.datasciencesociety.net/data-science-career-path-after-college>,
2020-02-20

An interdisciplinary field

- Machine Learning is *not* Data Science
- Programming is *not* Data Science
- Research & Statistics in *not* Data Science
- But for Data Science we need to apply and combine *all three* of them!
- Example:
 - R is a programming language
 - It features rich mathematical and statistical functionalities
 - We will apply those to different domains

Why is it currently so popular?

- More data than ever before available and created every day
- Large computing power is readily available (e.g. Cloud)
- New programming tools are available which are tailored to data processing (e.g. R)
- ⇒ Companies need data scientists to compete

Job Profile

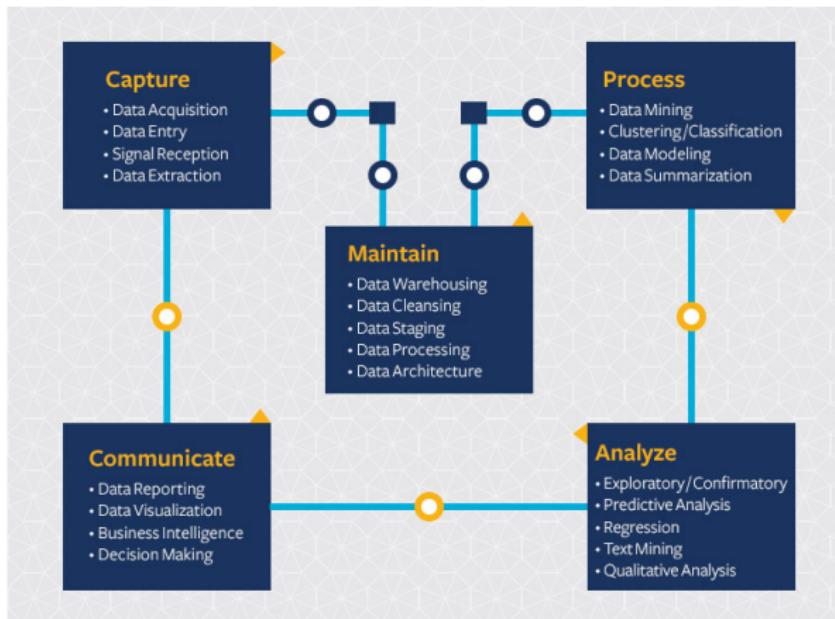
- Evolves rapidly (term 'data scientist' started to appear around the year 2008)
- Very promising (demand rising steadily)
- Very sought after (⇒ good salaries)
- But also demanding!
- Consists of:
 - Analyzing large amounts of data
 - Data mining
 - Programming / automation

Job Profile

“The ability to take data – to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it – that’s going to be a hugely important skill in the next decades.”

— Hal Varian, chief economist at Google

Data Science Life Cycle



Source: <https://datascience.berkeley.edu/about/what-is-data-science>, 2020-02-20

Data Structures
oooooooooooo

Recursive Relations
oooo

Generalization
oooooo

Temporal Dimension
oooo

Advanced Data Structures

Markus Haslinger

DBI

Agenda

1 Data Structures

2 Recursive Relations

3 Generalization

4 Temporal Dimension

Aggregation

- A step performed *after* Normalization
 - At least 3. NF
- Combines relations with identical primary keys to a single relation
- Example:
 - Rel1 (KA1, KA2, NKA1)
 - Rel2 (KA1, KA2, NKA2)
 - AggRel (KA1, KA2, NKA1, NKA2)
- The new relation *could* not be in 3. NF any more
 - Despite both starting relations being in 3. NF
 - Check and perform normalization if necessary

Aggregation – Simple Example

Relation	Att1	Att2	Att3	Att4
Person	PNo	Name	-	-
Address	PNo	-	City	-
Department	PNo	-	-	Sales
Person	PNo	Name	City	Sales

Overdetermined Data Structures

- An overdetermined system (a mathematical term) has more equations than unknowns
- In our context it means data structures which are represented more than necessary (redundant)
- Usually: relation attributes which are required based on the domain description, but can be safely removed with formal methods

Overdetermined Data Structures – Example I

- One trainer works at exactly one training institute
- One institute has many trainers
- A trainer can teach many classes
- Any class is held by a single trainer
- Any class is organized by exactly one institute
- Resulting relations:
 - Trainer (TrainerNo, ..., InstituteNo)
 - Institute (InstituteNo, ...)
 - Class (ClassNo, ..., InstituteNo, TrainerNo)

Overdetermined Data Structures – Example II

- The relation 'Class' hurts 3. NF (InstituteNo is functionally dependent on TrainerNo)
- Normalization leads to an additional relation:
 - TrainerInstitute (TrainerNo, InstituteNo)
- Applying aggregation we can combine the relations 'Trainer' and 'TrainerInstitute' (same PK)
- Final relations:
 - Trainer (TrainerNo, ..., InstituteNo)
 - Institute (InstituteNo, ...)
 - Class (ClassNo, ..., TrainerNo)
 - ⇒ InstituteNo in Class was redundant

No Redundancies – at all costs?

- Applying normalization and aggregation techniques allows to achieve redundancy free data structures
- However, do we really want that at all costs?
- Always consider the total cost of an application
 - Planning, Development & Maintenance
 - Storage space & Execution (system resources)
- Remember:
 - 1 Create a complete data model
 - 2 Create all required relations
 - 3 Perform normalization & aggregation
 - 4 Relax restrictions where and if reasonable
 - You need a solid understanding of the previous tasks to make sound decisions

Structure Rules

- The main structure rules¹ a data model has to comply with:

SR 1

Every table has to have a primary key.

SR 2

A database has to consist of tables which are at least in 3. NF and only contain global and local attributes.

- A global attribute is part of a primary key.
- A local attribute exists in only one relation and is not part of the primary key.

¹cf. Steiner, R. Grundkurs Relationale Datenbanken, Springer, p. 64f

Structure Rules

SR 3

- Local attributes have to have static value ranges (domains).
- Global attributes
 - Have to have a static value range in one table (where they are part of the PK).
 - Have a dynamic range in all other tables where they have to be foreign key attributes.

SR 4

Recursive relations are not allowed. Table B may only use a foreign key referencing table A if A can be defined independently of B

Structure Rules

SR 5

If in a generalized table there is no allowed intersection a discriminating attribute has to be added to differentiate between the specialized tables.

SR 6

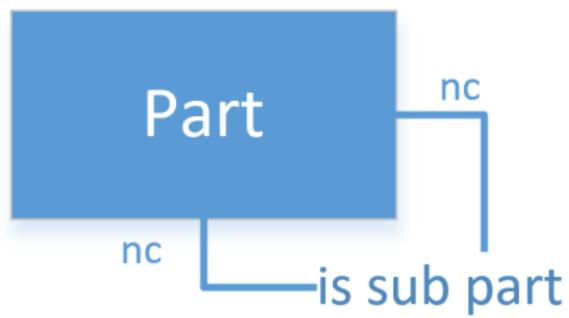
If a global attribute should be added as foreign key to a table use those tables as reference which have the smallest dynamic value range for the foreign key.

- e.g. do not extract LastName from a Person table; the new table (LN#, LastName) would contain too many entries.

Integrity Rules

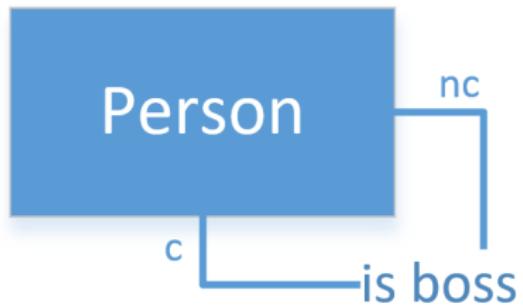
- Remember the major integrity rules/constraints:
 - Domain Integrity
 - Value ranges of data types (e.g. int)
 - Entity Integrity
 - The primary key
 - Referential Integrity
 - Foreign keys
 - User defined Integrity
 - e.g. UNIQUE constraints
 - and non-technical, business workflow controlled constraints

Recursive Relations



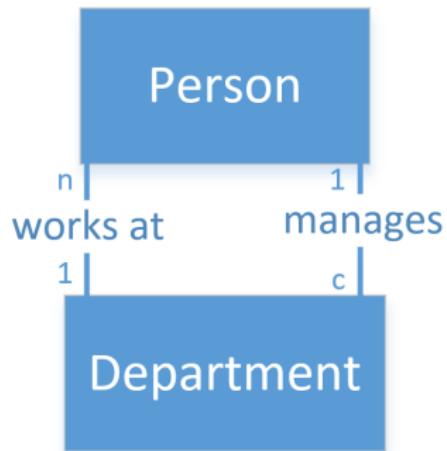
- Two relations:
 - Part (PartNo,...)
 - SubParts (PartNo, SubPartNo,...)
- Part can exist without sub parts
- Part can exist without being part of another
- We don't expect any issues

Recursive Relations



- Only one relation (Person)
- To insert an employee a boss is needed
- A boss is also an employee
- Can't insert anything!

Recursive Relations



- Two relations:
 - Department (DeptNo, MgrNo,...)
 - Person (PersNo, DeptNo,...)
- Can't insert department without a manager
- Cannot insert a person without a department
- Can't insert anything!

Recursive Relations – Possible solutions

- If foreign key columns can allow for NULL values it is possible to insert one row first, then the second and finally update the first one.
 - Sometimes the model does not allow for foreign keys to be NULL
- First fill tables with data, then add the foreign key constraint afterwards instead of during table creation.
 - But what happens if new data is added afterwards?

Generalization

- Some entities are closely related to each other
- May even be specific representations of the same real-world entity
- Think about inheritance in OOP → similar situations can arise in data models
 - We cannot share functionality
 - But we may share common data ⇒ reduces redundancy
- How to represent 'inheritance' with relations?

Related Entities – Disjunct

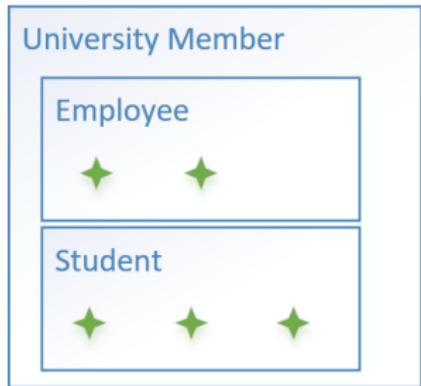


Figure: Complete



Figure: Incomplete

Related Entities – Overlapping

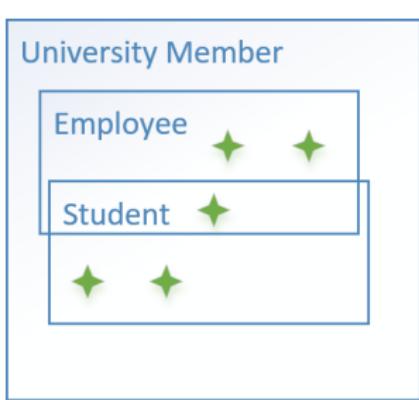


Figure: Complete

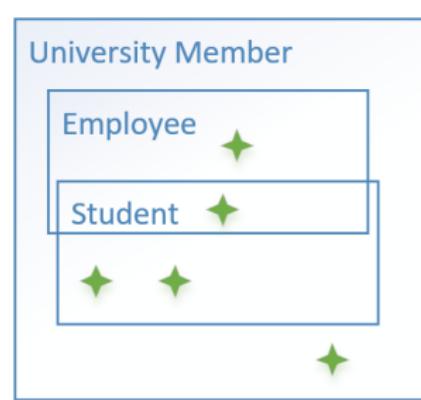


Figure: Incomplete

Generalization – Option I

- The most flexible approach: create one relation/table for every entity
- One for the supertype
 - Contains the shared data (e.g. name, address,...)
- One for each subtype
 - Contains the specific date (e.g. salary,...)
- Super- and subtype are linked with foreign key
- Be careful: there is a conditional relation between sub- and supertype
 - Subtype row cannot be inserted until the supertype row is present
- This can work well if there are many subtypes and/or many specific subtype attributes

Generalization – Option II

- Create a relation/table for each subtype
- Each of those contains all the 'shared' attributes
- Can be used if there are only few supertype attributes
- Difficulty: every update has to be made to *all* replicated instances of the data
- Also only makes sense for disjunct subtypes

Generalization – Option III

- Save all attributes of super- and all subtypes in a single table
- Can be useful if there are very few subtype attributes and few subtypes in general
- All attributes of subtypes need to be **NULLable**
 - Because they are only set in rows 'belonging' to the specific subtype
- A good practice is to add a so called 'discriminator' column
 - e.g. a string or enum-like value
 - Used to differentiate between different subtypes
 - Allows to quickly identify the subtype of each row
 - So you know which columns will be set and avoid a lot of **NULL** checks
 - Wastes space
- If all supertype entities are needed at once this approach is performance optimal (no joins / unions needed)

Temporal Dimension

- What does a row in a database represent?
- In the widest sense: a snapshot of an entity at some point in time
- Usually it is the last available snapshot, created after the last update
- So it represents the *current* state of an entity
- Often that is enough
- Sometimes it is important to know about previous states of entities
 - How did the price of a product change over the last months?
 - At which time of day does the stock of bread run out?

Storing Temporal Data

- In general temporal data means historic data (e.g. what happened in the past)
- However, it might often be necessary to 'plan' changes for the future
 - Imagine a promotion for a certain product lowering the price starting next Thursday
- In both cases it might not be enough to allow for a temporal dimension of one table
 - If a certain situation can only be represented by joining several tables historic/future data for all those tables is required
- In general two options:
 - Manually saving temporal information
 - Relying on features provided by the database system

Manual temporal data storage

- Can be a good option for either simple or very specific cases.
- It can be achieved by adding a column with a date/time data type
 - Be careful if date is enough if or time is needed as well
 - It may be necessary to add this column to the PK
- Three possibilities:
 - A point in time ⇒ a single date/datetime column
 - e.g. 'Hiredate' of employee
 - A duration without gaps ⇒ a single date/datetime column
 - e.g. 'ValidFrom' for product price
 - A duration with gaps ⇒ two date/datetime columns
 - e.g. 'ValidFrom' & 'ValidTo' for a product promotion

Automatic temporal data storage

- Most database systems provide features for temporal data storage
- Some more than others
 - Temporal data is often a use case of data warehouses
 - Thus a vendor's OLTP product may provide much less options than the OLAP product
- *Be aware, that also things like transaction time(outs) are related to temporal aspects of databases*
- See paper 'Time Concepts in Relational Database Systems'²

²Haslinger M. et al, 2013



SOWI

Social Sciences,
Economics and Business



INSTITUT FÜR WIRTSCHAFTSINFORMATIK

Time Concepts in Relational Database Systems

SEMINAR PAPER

in the field of study of

Master's programme

Business Informatics

Authors:

Markus Haslinger (k0955088)
Marcel Majer (k0755046)
Sabrina Pauli (k1257654)
Daniel Wiesinger (k0755502)

Institute:

Institut für Wirtschaftsinformatik - Data & Knowledge Engineering

Academic advisors:

o.Univ.Prof. Dipl.-Ing. Dr. Michael Schrefl
Mag. Dr. Eric Brewster, M.A.
Mag. Michael Huemer

Linz, June 2013

Abstract

Nearly every application nowadays relies on persistent data which is usually stored in a database. However, there are several use cases where it is necessary to store not only the latest valid data tuples, but also historical values or changes in a relational database and to do that sophisticated time concepts are required. The question is if the widely used commercial products Oracle 11g, Microsoft SQL Server 2012 and IBM DB2 support time concepts at all and if yes to what extent. This paper provides a general overview of the capabilities of each system and demonstrates both their practical applicability as well as their limitations by showing possible solutions to simulated use cases from the financial sector. We conclude which of the examined products offers the richest set of features concerning temporal concepts, also taking usability into account. The results can be used to facilitate the decision which system to deploy when a certain subset of temporal functions is required.

keywords: time concepts, relational database systems, oracle, ibm db2, ms sql

Contents

1	Introduction	1
2	Time Concepts (M. Majer)	3
2.1	Time	3
2.2	Historical Data	3
2.3	Temporal Support	3
2.4	Time Dimensions	4
2.5	Transaction Timeouts	5
2.6	Transaction Runtime	5
3	Case study (D. Wiesinger)	6
3.1	Data Model	7
3.2	Valid Time: Determining the Remaining Time to Maturity	8
3.3	Historical Data: Supervising a Bank Account	9
3.4	Transaction Time: Identifying a Specific Transaction	9
3.5	Transaction Timeouts: Automatic Interruption	10
3.6	Transaction Runtime: Performance optimization	10
4	Example Oracle 11g/12c (D. Wiesinger)	11
4.1	Valid Time	11
4.2	Historical Data	14
4.3	Transaction Time	16
4.4	Transaction Timeouts	20
4.5	Transaction Runtime Statistics	21
4.6	Performance Summary	22
5	Example MS SQL 2012 (M. Haslinger)	22
5.1	Valid Time	27
5.2	Historical Data	27
5.3	Transaction Time	28
5.4	Transaction Timeouts	32
5.5	Transaction Runtime Statistics	32
5.6	Summary of time concepts in MS SQL 2012	33
6	Example IBM DB2 (S. Pauli)	34
6.1	Valid Time	35
6.2	Historical Data	36
6.3	Transaction Time	38

6.4	Transaction Timeout	39
6.5	Transaction Time Measurement	39
6.6	Temporal Database	40
6.7	Summary of time concepts in IBM DB2	40
7	Comparison of Oracle, MS-SQL and DB2 – Strengths and weaknesses (M. Haslinger, S. Pauli)	42
8	Conclusion	44

List of Figures

1	Composition of the transaction time	5
2	Types of databases	7
3	Insert and Select on accounts	12
4	Version-enable a table	12
5	Select on accounts	13
6	Select on accounts	14
7	Check which accounts will be closed due the lack of funding	14
8	Insert and Select on accounts	14
9	Select on view accounts_hist	15
10	Select on view accounts_hist referencing a specific bank account	16
11	Select from owners including pseudo column ORA_ROWSCN	17
12	Select from owners, translating the pseudo column ORA_ROWSCN to a Timestamp value. All entries for May 2013 are shown	17
13	Select from owners, translating the pseudo column ORA_ROWSCN to a Timestamp value	18
14	Select from owners, translating the pseudo column ORA_ROWSCN to a timestamp value	18
15	Select from owners, showing the row-level timestamp value	19
16	Update of owners, showing the row-level timestamp value	19
17	Flashback of owners	20
18	Activation of Transaction Runtime Measurement	21
19	CDC usage by design	24
20	CDC process	24
21	CDC tables	26
22	SQL Server Agent tasks	26
23	CDC change log entry example	26
24	Data model	26
25	Account historical balance	26
26	Data model	28
27	Account historical balance	28
28	Transaction valid in the past	30
29	Transactions in a specific time frame	31
30	Transaction and query execution timeout settings in SQL Server Manage- ment Studio	32
31	Result of the valid time query	36
32	Results of the historical data query	38
33	Results of the transaction time query	39

34	Transaction Time Measurement in Toad for DB2	39
35	Result of Bi-Temporal Table query	41

List of Tables

1	Structure of owners	7
2	Structure of accounts	8
3	Structure of transactions	8
4	Summary of Oracle's Performance and Usability	22
5	Difference change data capture and change tracking	23
6	Different operation IDs in the log	25
7	Summary of Performance and Usability in Microsoft SQL Server 2012 . . .	34
8	Summary of Performance and Usability in IBM DB2	41
9	DBS comparision overview	43

List of Listings

1	Enable Versioning for accounts and transactions	13
2	Inserts into accounts including a Valid Time Period	13
3	Enabling the Collection of Historical Data	15
4	Inserts into accounts including a Valid Time Period	15
5	Update on Accounts	16
6	Create Table Command to enable Row-Level-Tracking	18
7	Setting for a 10-minute connection timeout	20
8	Enable CDC for database and each table	25
9	Show accounts which are valid less than two months from now	27
10	Transaction example	29
11	Stored procedure spDoTransaction	29
12	Query for historical balances of an account	29
13	Planning a transaction for the future	30
14	Selecting the current value stored in the database	30
15	Ignoring the transactions which are only valid in the future	30
16	Transactions in a given time frame	31
17	Rollback Database	31
18	Transaction runtime measurement	33
19	Valid time as Business Time in accounts table	35
20	Insert of data into accounts table	36
21	Query valid time from accounts table	36
22	Historical table implementation via System Time	37
23	Query of historical data	37
24	Transaction time implementation in transactions table	38
25	Transaction time implementation in transactions table	38
26	Bi-Temporal Table implementation	40
27	Update data and display Bi-Temporal table	40

1 Introduction

Nearly every application nowadays relies on persistent data which is usually stored in a database. However, there are several use cases where it is necessary to store not only the latest valid data tuples, but also historical values or changes in a relational database and to do that sophisticated time concepts are required. This is at least necessary when building a data warehouse but is of general use if changes of data have to be tracked, for example, for an earnings-over-time chart. One of the temporal aspects is to consider the difference between valid (or event [2]) time, which is the time period when data is valid in the real world, and transaction time, which is the time point data is stored in the database. Therefore, relational database management systems support different time concepts.

The high number of database management systems which implement time concepts forces developers to examine the difference between these implementations of concepts, in specific Oracle 11g/12c, Microsoft SQL Server 2012 and IBM DB2.

[3] is the primary literature source for the theory of this term paper. It concerns how to manage uni-temporal and bi-temporal data in relational databases in such a way that they can be seamlessly accessed together with current data. In contrast, the other literature, such as [4], [5], [1] and [2], describe temporal database systems. However the concepts of these systems can be used for the different relational databases which will be compared in the term paper.

[5] describes the fundamental time dimension concepts, like user-defined time, valid time and transaction time. Other times have also been proposed, such as the decision time [2]. [5] also gives a good illustration by an example where the time concepts will be shown.

In [1] Snodgrass and Ahn show four theoretical types of databases, which are snapshot, rollback, historical and temporal database. They are differentiated by their ability to represent temporal information and address different problems with respect to use cases and consumption of storage space and calculation time.

The implemented functionality to track data changes over time in Microsoft SQL Server 2012 is described in [6]. [7] and [8] provide tutorials for the activation of the so-called Change Data Capture (CDC) mechanism on a specific table for the purpose of storing historical values, whereas [9] and [10], in contrast, suggest a simple approach of just adding additional columns for valid time entries, which might be more costly concerning storage but more easy and efficient to use. [11] shows an overview of available stored procedures for CDC while [12] provides a corresponding tutorial for querying historical data which is the most relevant function due to the use cases discussed in this paper. The second built-in function to track changes is called Track Data Changes and only supposed to indicate a change without storing the old values as described in [13]. An

important question is if the recently introduced functions in the database system can be reasonably used for querying historical data.

The elementary literature which describes how time concepts are supported in IBM DB2 is [14]. It contains the clarification of the terms system time, business time and bitemporal tables. Moreover, it defines which steps have to be followed to implement these three concepts and illustrates it by small examples. An article identical to this is [15], which provides the same information. These definitions of the main terms that describe time concepts in DB2 are confirmed in [16], [17] and [18]. However, these papers focus on different issues. The main topic of [16] is the system time. [17] describes use cases and examples when to use which concept and gives advice on using them. Another approach regarding time concepts is given by [19]. It reports the history of SQL standards and leads to the current SQL:2011 standard which includes the definition of time concepts. The terms used in this literature lean on the ones defined in [14]. The end of this article it refers to DB2 and points out that IBM was the first vendor who implemented the time concepts of SQL:2011 standard.

The paper consists of two main aspects. Firstly, the creation of a state of the art overview of approaches for temporal databases is provided. More precisely it is illustrated how these approaches are available in a production-ready software and how it is discussed in literature. Secondly, the implementation of temporal features in current commercial first-class relational database management systems like Oracle 11g/12c, Microsoft SQL Server 2012 or IBM DB2, are examined, as roughly described in [20], [6] and [14]. The research contains the time concepts as well as a comparative overview and real-world examples with the functionality of the systems in use. A summary contains which of the three systems provide the best coverage of temporal concepts and is the easiest to use. Furthermore, the paper shows which subset of the methods described in literature is relevant for practical use nowadays.

2 Time Concepts (M. Majer)

In this chapter the theory of time concepts will be illustrated. To explain the relevance of time concepts, basic aspects have to be introduced to the reader. First, the explanation of time generally and then the importance of historical data and temporal support for database management systems are described.

To understand the term 'time' in a better way, we examine afterwards the time dimensions. Finally the aspects transaction runtime and query timeout will be explained, which are important for our example.

2.1 Time

Time is a relevant component in everyday life. Also relating to facts or data which need to be interpreted in the context of time. Time is a universal attribute in most information management applications and deserves special treatment [1].

Time representations like time points and time intervals. We need the following explanations to understand the time concepts in the next sections:

- Time point: A time point denotes either the start or the end of the lifespan of an object, relation, tuple or attribute [21].
- Time interval: A time interval is a directed duration of time, that is, an amount of time with a known length, but not specific starting or ending instants. A positive interval denotes forward motion of time, toward the future; a negative interval denotes motion toward the past [22].

2.2 Historical Data

Data is important to save. That is why we keep data in databases. Historical data can usually be found somewhere, of course, in archives and transaction logs if nowhere else. However, if it is important to be able to quickly and easily access data about what objects used to be like, either by itself or together with data about what those objects are like now, then keeping that data in the same table that also contains data about the current state of those objects makes a lot of sense. [23].

2.3 Temporal Support

With increasing sophistication of database management system applications, the lack of temporal support raises serious problems. The need to provide temporal support in database management systems has been recognized for at least one decade [1].

Conventional database management systems cannot support queries, where you get the historical data of an attribute. Without temporal support from the system, many applications have been forced to manage temporal information in an ad-hoc manner [1].

2.4 Time Dimensions

There are three kinds of time dimensions: the transaction time, the valid time and the user-defined time which will be described and explained by an example of this dimension on the bank sector. Other times have also been proposed, such as the decision time [2], but this time dimension is not so important for our paper.

- Valid time:

Valid time is also known as real-world time, intrinsic logical time and data time [24]. Valid time can span the past, present and the future of a fact. The value of valid time is a set of collected times. It can be a single time point, a set of time points, a time interval, a set of time intervals or even the entire time domain [5]. By definition, all facts have a valid time. However, the valid time may not necessarily be recorded in the database, for any of a number of reasons. For example, the valid time may not be known, or recording it may not be relevant for the applications supported by the database. If a database models different possible worlds, the database facts may have several valid times, one for each such world [4] For example, the valid time of a credit card is [2012-05-05, 2017-05-05]. This means that the credit card is issued at 2012-05-05 and expires five years later 2017-05-05. So, the valid time is from the first time point until the second time point stored in the database.

- Transaction time:

Transaction time is also known as registration time, extrinsic time and physical time [24]. The transaction time is the time point when the change of fact has occurred in the database. In this case, change means the insertion, deletion, or update transaction that changes the state of the database. It is automatically determined by the system time when the transaction occurs and is entirely application independent. In other words, the user cannot change its value [5]. For example, a bank customer deposits on November 22nd 2012 5,000.00 into his new account. On December 20th 2012 he withdraws 500.00. So, at the transaction time [2012-11-21] he has 0.-, at the transaction time [2012-11-22] he has 5,000.- and at the transaction time [2012-12-20] he has 4,500.00 on his account.

- User-defined time:

User-defined time is necessary when additional temporal information, not handled by transaction or valid time, is stored in the database [1]. In most cases, the value

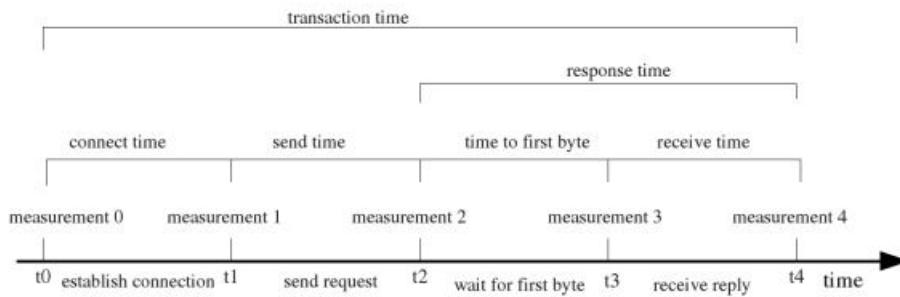


Figure 1: Composition of the transaction time

Source: [26]

of user-defined time is a time point. The database management system takes the user-defined time as a usual attribute. The value of user-defined time is managed by the users [5]. For example, an owner of a credit card is born February 1st 1986. So, the birthdate of the owner can be stored in the database with the user-defined time value [1986-02-01] to send him birthday greetings.

2.5 Transaction Timeouts

The transaction timeout specifies the amount of time that a transaction has to complete and helps to avoid system deadlocks. There are trade-offs in the amount of time given to a transaction to complete or abort. A higher value might result in fewer timeout exceptions but might not be too pleasing to the user, who might think the system is not responding. A lower timeout value might result in more timeout exceptions but would certainly let the user know what is happening [25]. For example, if the transaction timeout is set to a value of 1, the transaction must complete in one minute or be automatically aborted and rolled back.

2.6 Transaction Runtime

The transaction runtime is a measure of how quickly a user can complete a transaction. The transaction time is the total sum of three other time intervals, see Figure 1. Firstly, the 'connection time', which is the time when a connection with the server has been established. Then there exists the 'send time', which is the time when a request is sent out completely and third the 'response time'. This kind of time corresponds to the time the server needs to process and transmit the transaction [26].

3 Case study (D. Wiesinger)

This paper's example is based on the working tasks of the department for reporting of a large Austrian bank. The employees of this department create reports for various fields of applications for the bank itself and for the bank's commercial partners or customers. Cases in point for such reports are

- The running total of a specific bank account including a history of transactions
- A daily, weekly and monthly overview of the most important business ratios for the higher and middle management
- A numerousness of different reports for the business partners of the bank (e. g. an insurance company)

For the creation of these reports, the programmers take excessive usage of the time concepts of the used database. The following features of the database are very important and highly used:

- Valid-Time: Necessary to determine the remaining time to maturity of a bank account or a loan
- Historical Data: Necessary to supervise the accounting balance of unreliable customers
- Transaction Time: Necessary for the identification of a specific transaction
- Transaction Timeouts: Used for automatic interruption of intensive queries (in terms of running time)
- Transaction Runtime: Use of statistics for performance optimization

Currently, the bank has three different databases in use:

- Oracle 11g R2
- Microsoft SQL Server 2012
- IBM DB2

With the assistance of this example, these database management systems (DBMS) will be analyzed. It will be tested to which extent the different test cases are supported by the particular DBMS. Additionally, the functionality regarding the different types of databases developed by Richard Snodgrass is evaluated. Figure 2 shows the four types of databases according to Snodgrass [1]. This paper will not go into detail in snapshot functionality because every database system offers such functions. Thus, the following three types are interesting for this paper:

	No rollback	Rollback
Current queries	Snapshot	Rollback
Historical queries	Historical	Temporal

Figure 2: Types of databases

- Rollback: The functionality to get a past value of a field, seen from now
- Historical: The functionality to get a past valid value of a field
- Temporal: Combination of Rollback and Historical; the functionality to get a past valid value of a field seen from a past point in time

At the end of the paper, a comparative table with the strengths and weaknesses of the three different DBMS referencing the usability and the overall support for the test cases and the three types of databases will be built.

The used Data Model and each of the test cases are precisely described in the following subchapters.

3.1 Data Model

The following tables are used for the example. This model shows the data-columns only. Potential additional columns, e. g. for Transaction Time storage, will be needed. Since these columns are database-specific, they are not described in the model.

Table Owners:

COLUMN NAME	DATA TYPE	CONSTRAINT	DESCRIPTION
ID	Integer	Primary Key	ID for distinct identification
first_name	String	Not Null	Every person has a first name
last_name	String	Not Null	Every person has a last name

Table 1: Structure of owners

Table 1 stores the personal data of the bank's customers and is simplified to three columns.

Table Accounts:

COLUMN NAME	DATA TYPE	CONSTRAINT	DESCRIPTION
ID	Integer	Primary Key	ID for distinct identification
accountno	String	Not Null	Distinct number of the account
ownerid	Integer	Foreign Key (Owner)	Reference to Owner.ID
balance	Double	Not Null	Actual balance of the account

Table 2: Structure of accounts

Table 2 stores the necessary data for every bank account.

Table Transactions:

COLUMN NAME	DATA TYPE	CONSTRAINT	DESCRIPTION
fromAccount	Integer	Foreign Key (Account),Primary Key	Reference to Account.ID. The amount of the transaction will be debited to the referenced account
toAccount	Integer	Foreign Key (Account),Primary Key	Reference to Account.ID. The amount of the transaction will be credited to the referenced account
amount	Double	Not Null	Amount of the transaction

Table 3: Structure of transactions

Table 3 is the linking table between accounts and owners. Every tuple represents exactly one transaction.

3.2 Valid Time: Determining the Remaining Time to Maturity

The storage of Valid-Time-Parameters is very important in modern databases and data warehouses. For a bank, for instance, it is essential to know the remaining time to maturity of a fund or savings book.

The example shows another field of application. Usually, a bank account has a life time of one year. After that year, the customer has to pay an activity fee. Otherwise, the bank account will be closed. This example shows the methodology to create a query to select accounts that will be closed within two months' time due to the lack of funding.

To fulfill this task, it is essential to store a Valid Time Period for every single bank ac-

count. It is shown how to activate Valid Time support in the different DBMS and how to extract the necessary data for the task from the database.

3.3 Historical Data: Supervising a Bank Account

It is often necessary for clerks to check the bank account of a specific customer. For example, this activity is required if the customer needs a loan, a delayed payment for debts or simply a consultation what he should do with his money. Although it is important to check the actual balance, it is much more important to check the account's running total for the last couple of years. Only under consideration of this information is the bank assistant able to make a sound standing decision.

In order to fulfill this task, the clerk accesses the reporting services of the bank. He enters the account number and gets a detailed report of that account with all major payments and payoffs; simply every single balance of the last couple of years. Therefore, storage of Historical Data of the account within that time period is essential for this report.

Another very important reason for the storage of historical data is the Austrian legislature. Banks must store balances and transactions for at least ten years.

The example shows how to enable historical data in the different DBMS and compares usability and support of the different systems for this test case. Finally, it is shown how to request historical data of a bank account. The result is a list of historic balances, ordered by the date, of a specific bank account.

3.4 Transaction Time: Identifying a Specific Transaction

There are many different reasons for banks to supervise and store transactions for a specific bank account. Examples for such reasons are

- The tax fraud investigation of Austria needs to know account movements on specific dates to unmask potential tax evaders
- A customer wants to undo a wrongly made transaction
- Transactions on the statement for the bank account must have the correct chronological order

The most important part to fulfill all these reasons is the storage of the Transaction Time. These datasets provide a chronological insight into the bank account. In the example, the programmer has the task to create a report which shows all the payments and payoffs of the last month for a given bank account.

The example shows how to enable capturing of the Transaction Time and interpreting the stored data. Finally, it is shown how to request specific transactions for a given month out of the database.

3.5 Transaction Timeouts: Automatic Interruption

Transaction Timeouts are very useful in modern databases. The automatic interruption of long-running queries prevents blocking transactions and supports the availability for use of the database server.

Nowadays, the size of Data Warehouses of small or middle-sized companies is at least 50 Terabyte, big companies or large public organizations could have a data base of a Petabyte (= 1000 Terabyte) or more. Tables and views with several million entries which are joined in complex and complicated queries are no curiosities. Without system-level functions like automatic-query-cancellation, a simple mistake in the input parameters, e. g. a wrong date or a missing join, could block the server for hours.

In the bank-based example of this paper, with only a couple of datasets in each table and query runtimes of just a few microseconds, it is not possible to generate a Transaction Timeout.

3.6 Transaction Runtime: Performance optimization

In modern databases and data warehouses, the measurement of Transaction Runtimes is at least as important and useful as Transaction Timeouts. Such measurements are often the clue for the discovery of functions or queries with bad performance and are the basis for optimization.

As mentioned in chapter 3.5, banks or other large companies have to deal with an enormous amount of data. Tables with several million entries are no curiosities. In case of such huge amounts of data, performance is incredibly important. A missing join or a missing index on a field can result in Transaction Runtimes of an hour or more. Usually, such long runtimes are a hint for potential failures.

In this example, with runtimes of just a few microseconds, it will be difficult to obtain significant data, but not impossible. The example shows how to enable the runtime-measurement functions of the different DBMS and where the measured data is stored. Finally, the time needed for the selection of all accounts for a given owner is measured.

4 Example Oracle 11g/12c (D. Wiesinger)

Oracle 11g Release 2 Enterprise Edition is one of the most used database servers worldwide. As one of the biggest providers of professional database software, Oracle has several different types of such systems in their portfolio. Examples are

- Oracle 11g Release 2¹

Oracle's basic product with three different versions available:

- Standard Edition
- Standard Edition One
- Enterprise Edition

Every version has different features, and especially high professional functions like support of time-related concepts are mostly excluded in standard editions. For this reason, the database server used to conduct this paper's example is Oracle's 11g Release 2 Enterprise Edition

- Oracle 11g Express Edition ²

Free version of Oracle Database 11g with limited functionality

- MySQL ³

Probably the most popular and most important free database server

Enterprise-version databases made by Oracle support all major time concepts like Valid Time or Historical Data [27]. Nevertheless, the usage of Oracle's Data Warehouse System is recommended for this field of application.

4.1 Valid Time

Oracle 11g Release 2 Enterprise Edition offers explicit support of Valid Time functionality. It is possible to enhance every table or view with a system-level field to store the Valid Time Period for each data row within this table or view.

For this paper's example it is necessary to extend the table accounts with such a functionality. Assuming, that the three tables of the exemplary data model are already created and the table owners contains data, an insert in accounts is unproblematic.

Figure 3 shows that accounts has four fields and one row of data after the insert. The

¹<http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>

²<http://www.oracle.com/technetwork/products/express-edition/downloads/index.html>

³<http://www.mysql.com/downloads/>

```

insert into accounts values (1, 555.555, 1, 999.99);
commit;
select * from accounts;
1 Zeilen eingefügt
commit erfolgreich.

ID          ACCOUNTNR      OWNERID      BALANCE
-----      -----
1           555.555        1            999.99

1 rows selected

```

Figure 3: Insert and Select on accounts

```

DBMS_WM.EnableVersioning(
    table_name  IN VARCHAR2,
    hist         IN VARCHAR2 DEFAULT 'NONE',
    isTopology   IN BOOLEAN DEFAULT FALSE,
    validTime    IN BOOLEAN DEFAULT FALSE,
    undo_space   IN VARCHAR2 DEFAULT NULL);

```

Figure 4: Version-enable a table

next task is to add Valid Time support to this table. A precondition for this feature is a version-enabled table.

Figure 4 shows the syntax used to version-enable a table.

There are five parameters for this DBMS_WM command [28]:

1. **table_name**: The table which should be version-enabled. If there are any Foreign Key relationships with other tables, every table within these relationships must be version-enabled
2. **hist**: The history option for tracking modifications on table_name
3. **isTopology**: Indicates if the version-enabled table is part of an Oracle Spatial Topology
4. **validTime**: Indicates if Valid Time support should be included
5. **undo_space**: Indicates how much disk space should be available for undo-operations on this table

By the following command, the table accounts will be enhanced with a virtual system-level field, “WM_Valid”. This field stores the Valid Time Periods for each dataset. Since there is a connecting Foreign Key between accounts and transactions, both tables must be version-enabled. The history option should not be included here since section 4.2 has a detailed description how to add and use this option. Accounts is not a part of a Spatial Topology, and Valid Time support should be enabled. The last parameter is

Listing 1: Enable Versioning for accounts and transactions

```
1 EXECUTE DBMS_WM.EnableVersioning ('accounts,transactions', 'NONE', FALSE, TRUE);
```

```
select * from accounts;
-----
```

ID	ACCOUNTNR	OWNERID	BALANCE	WM_VALID
1	555.555	1	999.99	WMSYS.WM_PERIOD(2013-05-23 22:04:40.772,null)

1 rows selected

Figure 5: Select on accounts

missing because there should be no disk space available for possible undo-operations. After the implementation of Valid Time, all Datasets in accounts are updated with a standard Valid Time Period. This period lasts from the point in time when versioning was enabled until the next update of the dataset.

Figure 5 shows the single, and now enhanced with a Valid Time Period, dataset in accounts. A new dataset which should be inserted in a Valid Time enabled table must have a Valid Time Period because an insert without this parameter (as shown in Figure 3) is not possible any more [29]. Listing 2 shows four more inserts in accounts; each dataset has an own time period included.

On May 1, 2013, the first one of the four inserts has an expired Valid Time. As Figure 6 shows, it is not retrieved by a usual select-command on the table accounts.

That means Valid Time support helps the developer by preselecting the relevant datasets within a Valid Time enabled table. The programmer of this paper's example gets preselected data, so the task of creating a report which shows all accounts that will be closed due to the lack of funding within two months' time is made easy.

Figure 7 shows the query to select all accounts which will be closed within two months' time due to the lack of funding.

Listing 2: Inserts into accounts including a Valid Time Period

```
1 insert into accounts values (2, 123.456, 1, 500.00, WMSYS.WM_PERIOD(TO_DATE('03-03-2012', 'DD-MM-YYYY'),TO_DATE('02-02-2013', 'DD-MM-YYYY')));
2 insert into accounts values (3, 987.654, 1, -20.00, WMSYS.WM_PERIOD(TO_DATE('03-03-2013', 'DD-MM-YYYY'),TO_DATE('03-03-2014', 'DD-MM-YYYY')));
3 insert into accounts values (4, 777.777, 2, 100.00, WMSYS.WM_PERIOD(TO_DATE('07-07-2012', 'DD-MM-YYYY'),TO_DATE('07-07-2013', 'DD-MM-YYYY')));
4 insert into accounts values (5, 999.999, 3, 1000.00, WMSYS.WM_PERIOD(TO_DATE('04-04-2013', 'DD-MM-YYYY'),TO_DATE('04-04-2014', 'DD-MM-YYYY')));
```

```
select * from accounts;
ID      ACCOUNTNR   OWNERID    BALANCE    WM_VALID
-----  -----  -----  -----
1       555.555     1          999.99    UMSYS.WM_PERIOD(2013-05-23 22:04:40.772,null)
3       987.654     1          -20        UMSYS.WM_PERIOD(2013-03-03 00:00:00.0,2014-03-03 00:00:00.0)
4       777.777     2          100        UMSYS.WM_PERIOD(2012-07-07 00:00:00.0,2013-07-07 00:00:00.0)
5       999.999     3          1000       UMSYS.WM_PERIOD(2013-04-04 00:00:00.0,2014-04-04 00:00:00.0)

4 rows selected
```

Figure 6: Select on accounts

```
select * from accounts ac
WHERE WM_CONTAINS(wm_period(TO_DATE('01-01-1900', 'DD-MM-YYYY'),TO_DATE('23-07-2013', 'DD-MM-YYYY')), ac.wm_valid) = 1;
ID      ACCOUNTNR   OWNERID    BALANCE    WM_VALID
-----  -----  -----  -----
4       777.777     2          2000       UMSYS.WM_PERIOD(2013-05-23 22:31:03.412,2013-07-07 00:00:00.0)

1 rows selected
```

Figure 7: Check which accounts will be closed due the lack of funding

To sum it up, Oracle's 11g Release 2 Enterprise Edition database server has a full scale support for all Valid Time concepts. The feature is intuitional, fast and easy to use.

4.2 Historical Data

Storage of Historical Data for a table is a consolidation of the Valid Time concept. The starting point for this example is a version-disabled, Valid Time free data model with records in the table owners. As shown in section 4.1, the respective table has to be version-enabled by the command in figure 8.

```
DBMS_WM.EnableVersioning(
    table_name  IN VARCHAR2,
    hist        IN VARCHAR2 DEFAULT 'NONE',
    isTopology  IN BOOLEAN DEFAULT FALSE,
    validTime   IN BOOLEAN DEFAULT FALSE,
    undo_space  IN VARCHAR2 DEFAULT NULL);
```

Figure 8: Insert and Select on accounts

There is only one slight difference by using the command to enable the collection of historical data compared to the concept of Valid Time. The history option must not be 'NONE' to enable the Historical Data feature. Listing 3 shows how to enable history data for accounts. Compared to Listing 1, the parameter for the history option has changed. The following values are valid for this attribute [28]:

- **NONE**: This is the default value. No modifications of the respective table are tracked
- **VIEW_W_OVERWRITE**: The overwrite option. A view named

Listing 3: Enabling the Collection of Historical Data

```
1 EXECUTE DBMS_WM.EnableVersioning ('accounts,transactions', 'VIEW_WO_OVERWRITE', FALSE, TRUE
) ;
```

Listing 4: Inserts into accounts including a Valid Time Period

```
1 insert into accounts values (1, 555.555, 1, 999.99, WMSYS.WM_PERIOD(TO_DATE('03-03-2012', ,
DD-MM-YYYY'),null));
2 insert into accounts values (2, 123.456, 1, 500.00, WMSYS.WM_PERIOD(TO_DATE('03-03-2012', ,
DD-MM-YYYY'),TO_DATE('02-02-2013', 'DD-MM-YYYY')));
3 insert into accounts values (3, 987.654, 1, -20.00, WMSYS.WM_PERIOD(TO_DATE('03-03-2013', ,
DD-MM-YYYY'),TO_DATE('03-03-2014', 'DD-MM-YYYY')));
4 insert into accounts values (4, 777.777, 2, 100.00, WMSYS.WM_PERIOD(TO_DATE('07-07-2012', ,
DD-MM-YYYY'),TO_DATE('07-07-2013', 'DD-MM-YYYY')));
5 insert into accounts values (5, 999.999, 3, 1000.00, WMSYS.WM_PERIOD(TO_DATE('04-04-2013',
'DD-MM-YYYY'),TO_DATE('04-04-2014', 'DD-MM-YYYY')));
```

<table_name>_HIST is created and the most recent changes of the table <table_name> are tracked and stored

- **VIEW_WO_OVERWRITE:** The without-overwrite option. A view named <table_name>_HIST is created. This view stores all changes of the table table_name. This is the option which is used for this paper's example

After the execution of the command shown in Listing 3, the history view associated with accounts should be created [30]. In order to test its functionality, datasets are inserted into accounts, see Listing 4.

After the insert, the view accounts_hist contains five datasets with associated Valid Time Periods.

The view shows all columns of accounts and, additionally, the associated valid times. If an update is performed in accounts, the history view is updated with new valid time information of the respective row of data. The update of accounts results in two additional rows in accounts_hist.

select * from accounts_hist;				
ID	ACCOUNTNR	OWNERID	BALANCE	WM_VALID
1	555.555	1	999.99	WMSYS.WM_PERIOD(2012-03-03 00:00:00.0,null)
2	123.456	1	500	WMSYS.WM_PERIOD(2012-03-03 00:00:00.0,2013-02-02 00:00:00.0)
3	987.654	1	-20	WMSYS.WM_PERIOD(2013-03-03 00:00:00.0,2014-03-03 00:00:00.0)
4	777.777	2	100	WMSYS.WM_PERIOD(2012-07-07 00:00:00.0,2013-07-07 00:00:00.0)
5	999.999	3	1000	WMSYS.WM_PERIOD(2013-04-04 00:00:00.0,2014-04-04 00:00:00.0)

5 rows selected

Figure 9: Select on view accounts_hist

Listing 5: Update on Accounts

```
1 update accounts set balance = 2000 where ID = 4;
```

```
select * from accounts_hist where accountnr = '777.777'
ID      ACCOUNTNR    OWNERID      BALANCE      WM_VALID
-----  -----  -----  -----
4       777.777      2           100          WMSYS.WM_PERIOD(2012-07-07 00:00:00.0,2013-07-07 00:00:00.0)
4       777.777      2           2000         WMSYS.WM_PERIOD(2013-05-23 22:31:03.412,2013-07-07 00:00:00.0)
4       777.777      2           100          WMSYS.WM_PERIOD(2012-07-07 00:00:00.0,2013-05-23 22:31:03.412)

3 rows selected
```

Figure 10: Select on view accounts_hist referencing a specific bank account

The first row shown in Figure 10 is the one originally inserted. The balance is 100, and the time period starts on July 7, 2013 and ends on July 7, 2014. The next row contains the updated data. The balance has the updated value of 2000, and the Valid Time Period starts at that point in time the update of balance was performed (May 23, 2013). The end of the Valid Time Period of this dataset is the same as the end of the originally inserted time period. The last row of data contains the old value for balance and the expired Valid Time Period. The start of that period was at the same point in time than the start of the originally inserted period, and the end at the point in time the update of balance was performed.

If the value for the history option parameter in the command used for version-enabling accounts has been **VIEW_W_OVERWRITE** instead of **VIEW_WO_OVERWRITE**, row four would have been replaced by row five and six because just the most recent data would have been stored. Due to this fact, it is absolutely necessary for this paper's example to store Historical Data without overwriting it. Now, using the history view, it is easy for the programmer to build a report that shows the history of balances of a specific bank account.

In summary, Oracle's 11g Release 2 Enterprise Edition database server provides full support of Historical Data functionality. The solution of storing history data in a view (and not a table) provides clarity and speed. The feature, as an extension of Valid Time functionality, is easy to use and intuitional. Additionally, this feature represents the mechanism of a "Historical Data"-database of Snodgrass ([1]). Due to this functionality, it is possible to select past values of a field, seen from now.

4.3 Transaction Time

With the release of version 10g, Oracle introduced an automatically added timestamp for each row in each table [31]. This timestamp indicates the point in time the associ-

```

SELECT ora_rowscn, id, firstname, lastname from owners;
ORA_ROWSCN          ID        FIRSTNAME      LASTNAME
-----  -----
1151985            1          Joseph        Meyer
1151985            2          Jack          Knight
1151985            3          Scott         Fisher

3 rows selected

```

Figure 11: Select from owners including pseudo column ORA_ROWSCN

```

select scn_to_timestamp(ora_rowscn) as time_created, id, firstname, lastname from owners
where scn_to_timestamp(ora_rowscn) between
TO_TIMESTAMP ('01-May-13 00:00:00', 'DD-Mon-RR HH24:MI:SS') and
TO_TIMESTAMP ('31-May-13 23:59:59', 'DD-Mon-RR HH24:MI:SS')
TIME_CREATED          ID        FIRSTNAME      LASTNAME
-----  -----
01-MAY-13 22.43.28.000000000 1          Joseph        Meyer
01-MAY-13 22.43.28.000000000 2          Jack          Knight
01-MAY-13 22.43.28.000000000 3          Scott         Fisher

3 rows selected

```

Figure 12: Select from owners, translating the pseudo column ORA_ROWSCN to a Timestamp value. All entries for May 2013 are shown

ated row was changed the last time. It is not an absolutely precise timestamp because Oracle tracks the time by transaction committed for the block in which the row resides. That means, this pseudo column is merely useful for the approximate determination when a row was changed for the last time.

By default, Oracle tracks the System Change Number (SCN) just on table-, not on row-level. The following three subchapters explain how to handle SCN-tracking and how to flashback a table to a specific point in time.

Table-Level SCN-Tracking

Figure 11 shows the datasets inserted in table owners. The system-level column ORA_ROWSCN is not part of the data model for this paper's example and was automatically added when the table was created. The SCN-value indicates the point in time when the associated row was created.

The number stored in **ORA_ROWSCN** is clearly not meaningful for a human user, so the usage of the database function **SCN_TO_TIMESTAMP** is strongly recommended. [32].

Figure 12 shows that all three datasets were inserted at the same point in time. The query shows how to use the the pseudo column ORA_ROWSCN for data restriction (e.g. a given month). However, after an update of Joseph Meyer's first name to Joe, all data rows get the same new SCN. This SCN indicates the point in time when the up-

```

update owners set firstname = 'Joe' where ID = 1;
commit;
SELECT SCN_TO_TIMESTAMP(ora_rowscn) AS time_created, id, firstname, lastname FROM owners;

```

TIME_CREATED	ID	FIRSTNAME	LASTNAME
02-MAY-13 20.30.03.000000000 1		Joe	Meyer
02-MAY-13 20.30.03.000000000 2		Jack	Knight
02-MAY-13 20.30.03.000000000 3		Scott	Fisher

3 rows selected

Figure 13: Select from owners, translating the pseudo column ORA_ROWSCN to a Timestamp value

Listing 6: Create Table Command to enable Row-Level-Tracking

```

1 CREATE TABLE owners (
2   id number,
3   firstname varchar(20),
4   lastname varchar(20),
5   primary key (id)
6   ROWDEPENDENCIES;

```

date on Joseph Meyer's first name was performed and committed.

Row-Level SCN-Tracking

This subchapter explains how to enable row-level SCN-tracking in Oracle. It is important to know that row-level-tracking has to be enabled when a table is created; an alter table command to enable this feature later is not possible [33]. Listing 6 shows how to enable row-level-tracking by the command **ROWDEPENDENCIES**.

After an insert, all three datasets have the same value that indicates when they were changed the last time.

Now, Joseph Meyer's first name will be updated to Joe again. The difference to the example above is, as Figure 14 shows, that Oracle tracks changes on owners now on row-level.

```

select scn_to_timestamp(ora_rowscn) AS time_created, id, firstname, lastname FROM owners;

```

TIME_CREATED	ID	FIRSTNAME	LASTNAME
02-MAY-13 20.51.07.000000000 1		Joseph	Meyer
02-MAY-13 20.51.07.000000000 2		Jack	Knight
02-MAY-13 20.51.07.000000000 3		Scott	Fisher

3 rows selected

Figure 14: Select from owners, translating the pseudo column ORA_ROWSCN to a timestamp value

```

update owners set firstname = 'Joe' where id = 1;
commit;
select scn_to_timestamp(ora_rowscn) as time_created, id, firstname, lastname from owners;
TIME_CREATED          ID        FIRSTNAME      LASTNAME
-----  -----
02-MAY-13 20.58.01.000000000 1           Joe        Meyer
02-MAY-13 20.51.07.000000000 2           Jack       Knight
02-MAY-13 20.51.07.000000000 3           Scott      Fisher
3 rows selected

```

Figure 15: Select from owners, showing the row-level timestamp value

```

update owners set lastname = 'Smith' where id = 3;
commit;
select scn_to_timestamp(ora_rowscn) as time_created, id, firstname, lastname from owners;
TIME_CREATED          ID        FIRSTNAME      LASTNAME
-----  -----
02-MAY-13 20.58.01.000000000 1           Joe        Meyer
02-MAY-13 20.51.07.000000000 2           Jack       Knight
02-MAY-13 22.09.24.000000000 3           Scott      Smith
3 rows selected

```

Figure 16: Update of owners, showing the row-level timestamp value

Flashback Table

In this subchapter, the concept of flashback is described precisely on the basis of an example. It is important to mention that this functionality fulfils the requirements of Snodgrass' definition of a "Rollback"-database [1]. It is possible to reset a table to a past state as it is shown in the following example.

First of all, another update to change Scott Fisher's last name to Smith is performed. After this update, owners contains three rows of data with each row having a different timestamp to indicate the last change. This is the actual state of the table owners.

Through Oracle's flashback technology it is possible to recover a former state of a table. It is possible to flashback a table to a specific timestamp [31]. Figure 17 shows how to use the command to flashback owners to the point of time when Scott Smith' last name was Fisher (Note: If row movement is not active for owners, it must be enabled).

As the example shows, Oracle's Transaction Timestamp mechanism is incredible powerful and extraordinary helpful. Through the flashback technology, the programmer has the ability to build a report which shows all the payments and payoffs of the last month for a given bank account.

To sum it up, Oracle 11g Release 2 Enterprise Edition Database Server provides full support for automatic recording of Transaction Timestamps. Furthermore, the system offers the excellent flashback feature. Altogether, the mechanism is fast and easy to use.

```

alter table owners enable row movement;
commit;
FLASHBACK TABLE owners TO TIMESTAMP TO_TIMESTAMP('02-MAY-13 20.58.01.000000000');
select scn_to_timestamp(ora_rowscn) as time_created, id, firstname, lastname from owners;
FLASHBACK TABLE erfolgreich.
3.313 ms abgelaufen
TIME_CREATED           ID        FIRSTNAME      LASTNAME
-----
02-MAY-13 22.19.36.000000000 1        Joe        Meyer
02-MAY-13 22.19.36.000000000 2        Jack       Knight
02-MAY-13 22.19.36.000000000 3       Scott      Fisher
3 rows selected

```

Figure 17: Flashback of owners

Listing 7: Setting for a 10-minute connection timeout

```
1 SQLNET.EXPIRE_TIME=10
```

4.4 Transaction Timeouts

In Oracle's 11g Enterprise database server, there is no explicit setting for an overall Transaction Timeout. Once a query is started, it runs indefinitely except an abnormal client termination takes place. Reasons for such a termination could be

- Manual cancelation of the query by the user
- Lost connection to the client
- Crash of the database server system

All these reasons, except number one, are very unlikely to take place. Since long or indefinitely running queries could be a big problem in a database, Oracle solved that problem by delegating query-abortion handling to the client programs. Such client programs, mostly webservers like Apache or JBoss, almost always have a setting which controls Transaction Timeouts. If this time is exceeded, the query is canceled by the client program.

A possibility to give the database at least a little bit of control over query cancelation is the setting **SQLNET.EXPIRE_TIME**. This parameter in the profile configuration file **SQLNET.ORA** controls the time interval in minutes to verify that connections are still active. A time interval of ten minutes, for instance, results in the following setting [34].

```

set timing on;
select * from owners;

ID          FIRSTNAME      LASTNAME
-----
1           Joseph         Meyer
2           Jack           Knight
3           Scott          Fisher

3 rows selected

16 ms abgelaufen

```

Figure 18: Activation of Transaction Runtime Measurement

4.5 Transaction Runtime Statistics

Oracle 11g Release 2 Enterprise Edition provides support for runtime measurement on user- and administrator-level.

For a user who wants to know how long the runtime of an executed query was, the setting **set timing on** should be satisfactory. After the execution of a query like a select-, update- or delete-statement, the duration of the execution is shown [35].

This may be satisfactory for the standard database user, but it is far too little for an administrator. For effective query optimization, much more detailed statistics are needed. For this field of application, Oracle 11g Enterprise Edition has a built-in browser-based database management and analyses console. This management console, called ADDM (short for Automatic Database Diagnostic Monitor) shows an administrator all information needed for analyzing and tuning the database.

For this paper's example, it is not possible to get significant data for good statistics. The reason for this lack of information is the simplicity of the used database model and example. Oracle 11g Enterprise Edition is a highly professional system, and the analysis tools target at a usage as such a system. That means, single SQL-statements with a runtime of just a few microseconds could not be captured because automatic system-level statements consume much more time than the example's statements.

To sum it up, Oracle 11g Release 2 Enterprise Edition database server provides Runtime Statistics for both the usual database user and the database administrator. The provided tool for measuring and analyzing the database's performance is intuitional and the needed information can be found easily.

Use Case	Support	Usability
Valid Time	extensive	excellent
History Data	extensive	excellent
Transaction time	extensive	excellent
Query Timeouts	not supported	not supported
Transaction Runtime	very good	excellent
Snapshot	yes	excellent
Rollback	yes	very good
Historical	yes	excellent
Temporal	no	-

Table 4: Summary of Oracle's Performance and Usability

4.6 Performance Summary

To sum it up, the overall performance of Oracle's 11g Enterprise Edition Database Server is very good. Except for Query Timeouts, the most important time concepts are supported very well. Especially the support for Valid Time, History Data and Transaction Time is excellent.

In addition, two of three concepts of Snodgrass' typology of databases are also supported very well. The Historical Data functionality fulfils the requirements of a "Historical" type database and Transaction Time functionality mostly fulfils the requirements of a "Rollback" type database. Only the type "Temporal" is not supported. To use this functionality, an upgrade to a data warehouse is necessary.

5 Example MS SQL 2012 (M. Haslinger)

Microsoft SQL Server 2012 is a typical relational database system used by many small and large applications. As one of the widely used commercial database systems, SQL Server provides a broad variety of supported functionality. Since Version 2008 there is system level support for tracking changes of tuples available which had to be done manually before. These features enable the processing of some temporal operations. However, SQL Server can also be installed and configured as a Data Warehouse and Business Intelligence platform. Therefore, Microsoft encourages the usage of their DW & BI⁴ solution for historical data related tasks, which exceeds the scope of database audits based on transaction time. In the following sections we describe what functionality SQL Server 2012 offers in its database configuration.

⁴Data Warehouse and Business Intelligence

Change tracking functionality

As mentioned before, MS SQL Server now has system-level support for change tracking. This functionality consists of two different extensive change logging options, automatically generated tasks for the SQL Server agent, as well as auto generated functions for accessing the logged data. The main differences between the two options are shown in Table 5.

Feature	Change data capture	Change tracking
<i>Tracked changes</i>		
DML changes	Yes	Yes
<i>Tracked information</i>		
Historical data	Yes	No
Whether column was changed	Yes	Yes
DML type	Yes	Yes

Table 5: Difference change data capture and change tracking

Source: [6]

Change Tracking

Change tracking is a lightweight solution to flag changed data sets without manually writing triggers [36]. It can be used to synchronize changed data between two applications if SQL Server is the active database source [36].

According to [13], it is necessary to activate the function:

1. for the whole database;
2. for each table that has to be tracked;

and decide in which style the changes will be tracked, either

- as ROWVERSION value for each tuple that will change on every update including false-updates or
- as CHECKSUM value for each tuple that will be able to distinguish between real and false updates.

The stored data can be accessed through some simple functions as described in [37]. Since change tracking does not support historical data management, it is not applicable in the context of this paper and not further discussed.

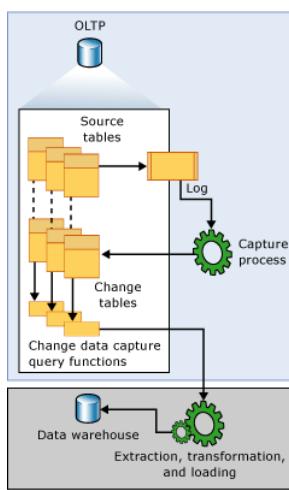


Figure 19: CDC usage by design

Source: [38]

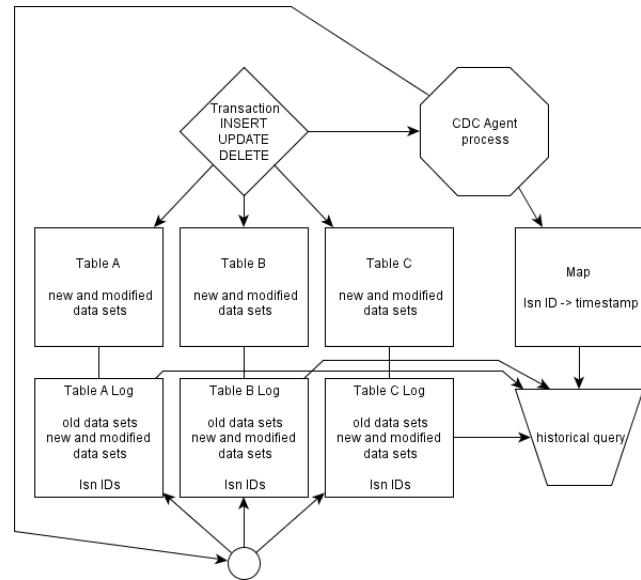


Figure 20: CDC process

Change Data Capture

A much more sophisticated approach than change tracking is change data capture (cdc). Contrary to change tracking, change data capture stores historical information. It operates at the table level and not only stores changes to existing tuples, but also inserts into and deletes from the table. For each update operation both the old and the new value are logged. If many operations are executed on the table, this may lead to performance problems and large storage needs, therefore, Microsoft insists on using online analytical processing(OLAP) instead of online transaction processing (OLTP) tools for working with historical data and limits the support in the SQL Server transactional database configuration, see Figure 19. It should also be noted that SQL Server agent by default cleans the log tables every two days, since it is assumed that the data is regularly transferred to the data warehouse. This reduces the storage requirements for the OLTP database. It is, however, still possible to solve simple problems without the use of a data warehouse at the database level by disabling the cleanup task.

Like change tracking, change data capture has to be activated for the whole database first and then for each table that has to be monitored. Due to the resource requirements it has to be carefully considered for which tables cdc is necessary, see Listing 8 line 1, 3 and 8 respectively. This activation will create several tables for storing the logged data, transaction IDs and associated timestamps, see Figure 21. Additionally, two tasks for the SQL Server agent are created: capture and cleanup (see Figure 22). The agent has to be active for cdc to work. Finally, data access functions are generated for each tracked table because it is discouraged to directly query system tables; these functions also handle any DDL changes on the source tables [38].

Listing 8: Enable CDC for database and each table

```

1  EXEC sys.sp_cdc_enable_db
2  GO
3  EXEC sys.sp_cdc_enable_table
4  @source_schema = N'dbo',
5  @source_name = N'accounts',
6  @role_name = N'CDCRole'
7  GO
8  EXEC sys.sp_cdc_enable_table
9  @source_schema = N'dbo',
10 @source_name = N'transactions',
11 @role_name = N'CDCRole'
```

operation id	operation	data stored
1	data set deleted	deleted value
2	data set inserted	new value
3	data set updated	old value
4	data set updated	new value

Table 6: Different operation IDs in the log

Source: [39]

After cdc has been activated, each change will force a new entry in the corresponding log table, see Figure 23. Each log entry is identified via its log sequence number (lsn) via start_lsn ID. The transaction timestamp for the specific lsn and vice versa can be queried by the system functions `sys.fn_cdc_map_lsn_to_time` respectively `sys.fn_cdc_map_time_to_lsn`. As shown by Figure 23, each column of the source table is copied with its content to the log table. Other important fields are operation and update_mask. There are four possible values for the field operation (see Table 6), while each update operation forces one '3' and one '4' entry with the same lsn ID. The update mask is only relevant for update operations. If a column has changed, the according bit in the mask is set. In Figure 23 the balance has changed which is the forth column in the log table, therefore, the forth bit is set, which results in a hex value of 0x08. If the primary key had changed too, the update mask value is 0x09.

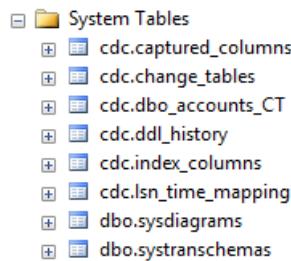


Figure 21: CDC tables

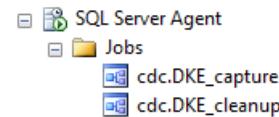


Figure 22: SQL Server Agent tasks

	<u>_start_lsn</u>	<u>_seqval</u>	<u>_operation</u>	<u>_update_mask</u>	<u>id</u>	<u>accountnr</u>	<u>ownerid</u>	<u>balance</u>	<u>validUntil</u>
105	0x0000004F000000410005	0x00000004F000000410003	4	0x08	5	000.001	3	1997...	2014-12-24

Figure 23: CDC change log entry example

Use Case Implementation

There are many use cases for temporal features in relational database systems. Some of them are described in Section 3 and will now be applied to the Microsoft SQL Server 2012 to examine which problems can be solved using the system support for change tracking.

Database Model

Since there is only support for logging transaction time timestamps but no explicit function for valid time, it is necessary to manually add and manage appropriate columns in the tables. For this example, a date column is added to the transactions table and a validUntil column to the accounts table, see Figure 24. Since cdc has been activated for these tables already, each operation will also copy the valid time values into the log table but without special treatment.

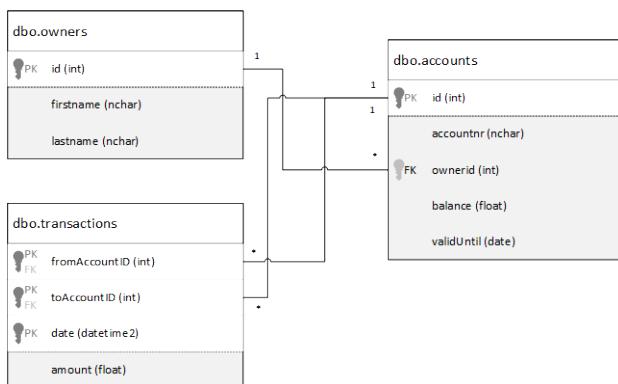


Figure 24: Data model

	<u>date</u>	<u>id</u>	<u>accountnr</u>	<u>balance</u>
1	2013-04-26 17:58:05.1770000	2	987.654	80
2	2013-04-26 17:58:05.1870000	2	987.654	-10
3	2013-04-26 17:58:05.1900000	2	987.654	-110
4	2013-04-26 18:31:51.9630000	2	987.654	90
5	2013-04-26 18:31:54.9670000	2	987.654	2590
6	2013-04-26 18:31:57.9730000	2	987.654	52590
7	2013-04-26 18:32:00.9770000	2	987.654	-7410

Figure 25: Account historical balance

Listing 9: Show accounts which are valid less than two months from now

```
1 SELECT id, accountnr, validUntil
2 FROM dbo.accounts
3 WHERE validUntil < DATEADD(month, 2, GETDATE());
```

5.1 Valid Time

As there is no explicit support or special treatment of valid time in Microsoft SQL Server 2012, there are two options left:

1. get all changes of the validUntil field from the cdc log table with the techniques shown before, or
2. query the dbo.accounts table to get the current value.

In the context of this example, there is no additional benefit for working with the cdc tables at all. The simple query shown in Listing 9 will create the desired result of accounts that will be closed in two months using SQL Server's DATEADD and GETDATE functions.

5.2 Historical Data

With cdc active and the necessary date columns available it is possible to show the historical balances of an account ordered by date. Due to the limitations of simulating elapsed time for this example, the query will only cover a few seconds instead of days or months as we place only a few seconds between each demo transaction, see Listings 10 and 11. However, the process is exactly the same for larger time scopes. First, it is necessary to set the two borders for the time frame, see Listing 12 lines 2 and 3. This time frame refers to the transaction time logged by the SQL Server and not any manually managed valid time. Second, system functions are used to map the date values to lsn IDs, see Listing 12 lines 4 and 5. The parameters 'smallest greater than' and 'largest less than or equal' define if a border has to be included or excluded from the result set.

While the log table for the accounts contains all past balance values since cdc has been activated, it is also necessary to get the corresponding time of the transaction. This does not refer to the transaction time from the database system but to the data field in the transactions table though. However, as the stored procedure spDoTransaction (see Listing 11), is designed to execute all insert and update operations in one single transaction, which is necessary for clean rollbacks, the update log entry in the accounts log and the insert log entry in the transactions log will have the same lsn ID. There-

	balance
1	1640

Figure 26: Data model

	date	id	accountnr	balance
1	2013-04-26 18:32:04.9830000	1	123.456	640

Figure 27: Account historical balance

fore, it is possible to join the two log tables on this lsn ID (see Listing 12 line 7). The historical account balance of the account with ID 2 can be retrieved with the query shown in Listing 12 (see Figure 25). As mentioned before it is discouraged to query the system generated and maintained cdc tables directly, instead access functions like `cdc.fn_cdc_get_all_changes_dbo_accounts` have to be used [7]. It would be possible to create this report without the cdc tables by calculating each balance value while going back through the transactions in the transactions table. Although this would reduce storage cost, it is less efficient.

Using SQL Server 2012 as a Historical Database as described by Snodgrass and Ahn in [1] is partially supported if the valid time is treated manually. Listing 13 shows a transaction which is committed today but will be valid at the end of June. The stored procedure `spDoTransactionFixedDate` is similar to the procedure in Listing 11 but sets the valid time value to a fixed date. However, as the transaction is committed immediately a simple Select on the accounts table, see Listing 14, will return the actual stored value, see Figure 26. To 'ignore' the future transaction and get the currently valid value a much more complicated query is necessary. Listing 15 shows a query which joins the change tables of the transactions and the accounts table like the historical balances query shown in Listing 12 and limits the results to transactions which are valid today (line 8). To only get the latest valid value from this account history result set the TOP 1 command (line 6) is used together with a DESC ordering (line 9), see query result in Figure 27. Even more difficult is adding transactions which are valid in the past, see Figure 28. While the current value would be correct it would be necessary to manually compute the balance based on all transactions if the valid balance from the point of view of a past date is required. As this applies even more to the concept of a temporal database as described in [1] it is not really possible to use SQL Server in this context as there is no dedicated support.

5.3 Transaction Time

While the support for historical queries and valid time may not be fully advanced, the logging of transaction time stamps is well supported by SQL Server 2012. With a quite similar approach to before Listing 16 shows a query to get transactions in a specific time frame which may be a day, a month or only few seconds. The results in Figure 29 show that the date value of each inserted data can in some situations slightly differ

Listing 10: Transaction example

```

1 BEGIN TRANSACTION
2 EXEC [dbo].[spDoTransaction]
3 @fAid = 5,
4 @tAid = 3,
5 @amount = 200
6 COMMIT TRANSACTION
7 GO
8 -- wait for 3 seconds
9 WAITFOR DELAY '00:00:03'
10 GO
11 BEGIN TRANSACTION
12 EXEC [dbo].[spDoTransaction]
13 @fAid = 5,
14 @tAid = 1,
15 @amount = 2500
16 COMMIT TRANSACTION

```

Listing 11: Stored procedure spDoTransaction

```

1 CREATE PROCEDURE [dbo].[spDoTransaction] @fAid int, @tAid int, @amount float(7)
2 BEGIN TRANSACTION
3 INSERT INTO dbo.transactions
4 VALUES (@fAid, @tAid, GETDATE(), @amount);
5 UPDATE dbo.accounts
6 SET balance = balance - @amount
7 WHERE id = @fAid;
8 UPDATE dbo.accounts
9 SET balance = balance + @amount
10 WHERE id = @tAid;
11 COMMIT TRANSACTION

```

Listing 12: Query for historical balances of an account

```

1 DECLARE @begin_time datetime, @end_time datetime, @begin_lsn binary(10), @end_lsn binary(10);
2 SET @begin_time = '2013-04-26 17:57:00.000';
3 SET @end_time = '2013-04-26 18:33:00.000';
4 SELECT @begin_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @begin_time);
5 SELECT @end_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time);
6 SELECT S1.date, S2.id, S2.accountnr, S2.balance
7 FROM (SELECT * FROM cdc.fn_cdc_get_all_changes_dbo_transactions(@begin_lsn, @end_lsn, 'all'
     ) as S1 JOIN (SELECT * FROM cdc.fn_cdc_get_all_changes_dbo_accounts(@begin_lsn,
     @end_lsn, 'all')) as S2 ON (S1.__$start_lsn = S2.__$start_lsn)
8 WHERE id = 2
9 ORDER BY S1.date;

```

Listing 13: Planning a transaction for the future

```

1 BEGIN TRANSACTION
2 EXEC [dbo].[spDoTransactionFixedDate]
3 @fAid = 5,
4 @tAid = 1,
5 @amount = 1000,
6 @date = "2013-06-25"
7 COMMIT TRANSACTION

```

Listing 14: Selecting the current value stored in the database

```

1 SELECT balance
2 FROM dbo.accounts
3 WHERE id=1;

```

Listing 15: Ignoring the transactions which are only valid in the future

```

1 DECLARE @begin_time datetime, @end_time datetime, @begin_lsn binary(10), @end_lsn binary
      (10);
2 SET @begin_time = '2013-04-26 17:57:00.000';
3 SET @end_time = '2013-05-25 00:00:00.000';
4 SELECT @begin_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @begin_time);
5 SELECT @end_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time);
6 SELECT TOP 1 S1.date, S2.id, S2.accountnr, S2.balance
7 FROM (SELECT * FROM cdc.fn_cdc_get_all_changes_dbo_transactions(@begin_lsn, @end_lsn, 'all'
     )) as S1 JOIN (SELECT * FROM cdc.fn_cdc_get_all_changes_dbo_accounts(@begin_lsn,
     @end_lsn, 'all')) as S2 ON (S1.__$start_lsn = S2.__$start_lsn)
8 WHERE id = 1 AND S1.date < '2013-05-25 00:00:00.000'
9 ORDER BY S1.date DESC;

```

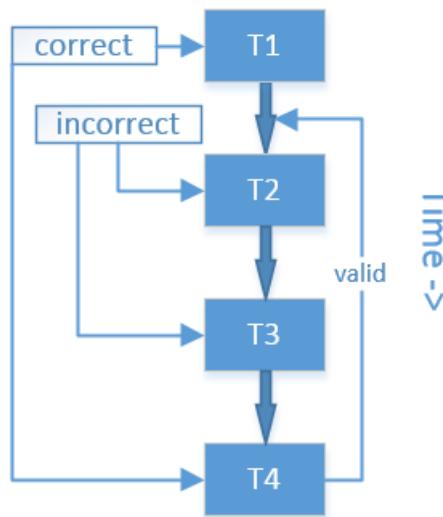


Figure 28: Transaction valid in the past

Listing 16: Transactions in a given time frame

```

1 DECLARE @begin_time datetime, @end_time datetime, @begin_lsn binary(10), @end_lsn binary
   (10);
2 SET @begin_time = '2013-04-26 17:57:00.000';
3 SET @end_time = '2013-04-26 18:33:00.000';
4 SELECT @begin_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @begin_time);
5 SELECT @end_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time);
6 SELECT __$operation, fromAccountID, toAccountID, date, amount, sys.fn_cdc_map_lsn_to_time(
      __$start_lsn) as transaction_timestamp
7 FROM cdc.fn_cdc_get_all_changes_dbo_transactions(@begin_lsn, @end_lsn, 'all')

```

Listing 17: Rollback Database

```

1 DECLARE @begin_time datetime, @end_time datetime, @begin_lsn binary(10), @end_lsn binary
   (10);
2 SET @begin_time = '2013-04-26 18:31:49.000';
3 SET @end_time = '2013-04-26 18:32:03.000';
4 SELECT @begin_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @begin_time);
5 SELECT @end_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time);
6 SELECT id as account_id, balance, sys.fn_cdc_map_lsn_to_time(__$start_lsn) as
      transaction_timestamp
7 FROM cdc.fn_cdc_get_all_changes_dbo_accounts(@begin_lsn, @end_lsn, 'all')
8 WHERE id=1

```

from the transaction time. This is due to the fact that date represents a kind of valid time which might be set to any value while the transaction time represents the system time when the operation was executed. This difference can be seen in Figure 29 at the twelfth data set where the milliseconds slightly differ.

Listing 17 can be used to get the current balance of account nr. 1 with a transaction time of 2013-04-26 18:31:54.967. Therefore, it is possible to use SQL Server 2012 as Rollback database as described in [1].

	__\$operation	fromAccountID	toAccountID	date	amount	transaction_timestamp
1	2	1	2	2013-04-26 17:58:05.1770000	100	2013-04-26 17:58:05.177
2	2	1	3	2013-04-26 17:58:05.1800000	50	2013-04-26 17:58:05.180
3	2	2	1	2013-04-26 17:58:05.1870000	90	2013-04-26 17:58:05.187
4	2	2	3	2013-04-26 17:58:05.1900000	100	2013-04-26 17:58:05.190
5	2	5	3	2013-04-26 18:31:45.9530000	200	2013-04-26 18:31:45.953
6	2	5	1	2013-04-26 18:31:48.9600000	2500	2013-04-26 18:31:48.960
7	2	5	2	2013-04-26 18:31:51.9630000	200	2013-04-26 18:31:51.963
8	2	1	2	2013-04-26 18:31:54.9670000	2500	2013-04-26 18:31:54.967
9	2	5	2	2013-04-26 18:31:57.9730000	50000	2013-04-26 18:31:57.973
10	2	2	3	2013-04-26 18:32:00.9770000	60000	2013-04-26 18:32:00.977
11	2	5	1	2013-04-26 18:32:03.9800000	100	2013-04-26 18:32:03.980
12	2	5	1	2013-04-26 18:32:04.9830000	100	2013-04-26 18:32:04.983

Figure 29: Transactions in a specific time frame

5.4 Transaction Timeouts

[40] describes the process to set transaction and query timeouts for SQL Server. There are two places in the options window where the timeout limits for query execution and transactions can be set, see Figure 30. Timeouts for remote operations can be set via `EXEC sp_configure 'remote query timeout', 0; GO RECONFIGURE;` while '0' seconds disable timeouts [41].

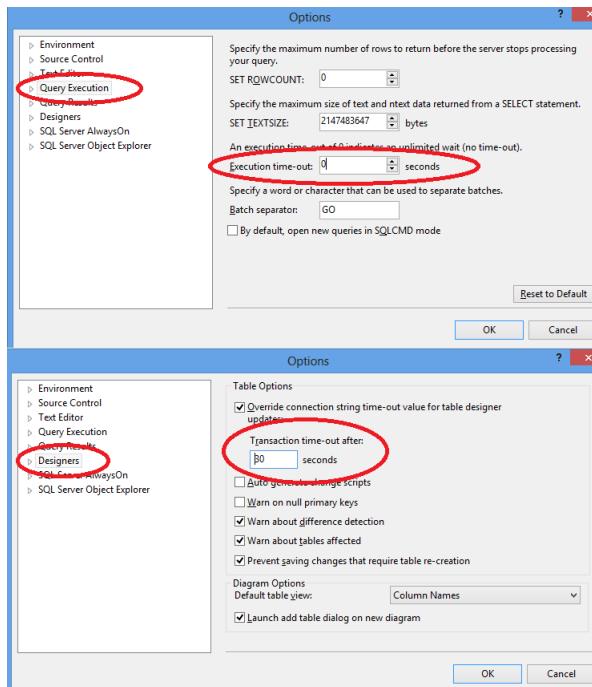


Figure 30: Transaction and query execution timeout settings in SQL Server Management Studio

5.5 Transaction Runtime Statistics

Measuring statement execution runtime is quite simple in SQL Server 2012. Listing 18 shows a query from one of the example with `STATISTICS TIME` commands. It results in the following output:

SQL Server parse and compile time: CPU time = 0 ms, elapsed time = 4 ms. [...] (12 row(s) affected) SQL Server Execution Times: CPU time = 0 ms, elapsed time = 3 ms.

Listing 18: Transaction runtime measurement

```
1 SET STATISTICS TIME ON;
2 GO
3 DECLARE @begin_time datetime, @end_time datetime, @begin_lsn binary(10), @end_lsn binary
   (10);
4 SET @begin_time = '2013-04-26 17:57:00.000';
5 SET @end_time = '2013-04-26 18:33:00.000';
6 SELECT @begin_lsn = sys.fn_cdc_map_time_to_lsn('smallest greater than', @begin_time);
7 SELECT @end_lsn = sys.fn_cdc_map_time_to_lsn('largest less than or equal', @end_time);
8 SELECT __$operation, fromAccountID, toAccountID, date, amount, sys.fn_cdc_map_lsn_to_time(
   __$start_lsn) as transaction_timestamp
9 FROM cdc.fn_cdc_get_all_changes_dbo_transactions(@begin_lsn, @end_lsn, 'all')
10 SET STATISTICS TIME OFF;
```

5.6 Summary of time concepts in MS SQL 2012

Microsoft seems to place the SQL Server database system as a simple data and changes source for the data warehouse in the context of historical and temporal data analysis. There is still a subset of functionality available that can be used to perform simple temporal operations without a data warehouse. The presented change data capture feature enables the logging of all operations performed on a specific table including transaction time stamps. However, there is no explicit support for valid time concepts. While it is quite impossible for a database system to, in contrast to the transaction time, determine a valid time of a data set without user input it would be possible to provide some flags for table columns marking them as valid time fields. This would allow system functions to query data sets in a specific valid time frame like it is already possible with transaction time.

The availability of the temporal concepts according to Snodgrass and Ahn in [1] is quite low as explained before in Sections 5.2 and 5.3, see Table 7.

The provided functionality will be sufficient for some simple historical queries. For any real temporal operations the manufacturer clearly wants its users to use a separate data warehouse installation, see Table 7.

Checkpoint	Support	Usability
Valid time	no	bad
History Data	indirect	bad
Transaction time	yes	good
Query Timeouts	yes	good
Transaction Runtime	yes	good
Snapshot	yes	excellent
Rollback	yes	good
Historical	limited	bad
Temporal	no	-

Table 7: Summary of Performance and Usability in Microsoft SQL Server 2012

6 Example IBM DB2 (S. Pauli)

[42] shows that IBM DB2 is a relational database management system which is offered in different versions to support the various needs and variations of sizes of companies:

- Advanced Enterprise Server Edition, which is a multi-workload database solution including data warehousing, transactional and analytics capabilities and ideal for enterprises.
- Advanced Workgroup Server Edition, which offers the same functionality as Advanced Enterprise Server Edition, but optimized for midsize companies.
- Enterprise Server Edition, which is a database server software solution for enterprises.
- DB2 Workgroup Server Edition, which offers the same functionality as Enterprise Server Edition, but optimized for small and midsize companies.
- Express Edition, which is an entry-level database system for small and midsize companies.
- DB2 Express-C, which is a no-charge community edition of the DB2 data server and ideal for small companies, developers, and students.

For exploring time concepts of this database management system, the current release 10.1.2 of the version DB2 Express-C is used, because it is free of charge and contains all core features like the other versions. [43]

[19] states, that IBM has implemented the specification of temporal data defined by the SQL:2011 standard. The verification of this statement is checked below by the compari-

Listing 19: Valid time as Business Time in accounts table

```

1 CREATE TABLE DB2ADMIN.ACCTS1 (
2     ID INTEGER NOT NULL,
3     ACCOUNTNR VARCHAR(15) NOT NULL,
4     OWNERID INTEGER NOT NULL,
5     BALANCE DOUBLE NOT NULL,
6     VALIDSINCE DATE NOT NULL,
7     VALIDUNTIL DATE NOT NULL,
8     PERIOD BUSINESS_TIME(VALIDSINCE, VALIDUNTIL),
9     PRIMARY KEY (ID, BUSINESS_TIME WITHOUT OVERLAPS),
10    FOREIGN KEY OWN (OWNERID) REFERENCES OWNERS
11 ) ;

```

son if the use cases defined in chapter three are implemented in DB2.

6.1 Valid Time

”[B]usiness time involves tracking when certain business conditions are, were, or will be valid.” [14, 10]

[14, 11-15] signifies that valid time is the equivalent to business time, which is the term used in DB2. There are several requirements to implement business time. First, define the valid start and end columns as data type DATE. Secondly, add the expression PERIOD BUSINESS_TIME with the two dates in parentheses, so DB2 tracks the validity period. Thirdly, add BUSINESS_TIME WITHOUT OVERLAPS as primary key to ensure that not two rows exists which have the same identifier and an overlapping validation period. When to insert data, the valid start and end dates are appended like the other data. In addition, it is checked if a row with the same id exists which covers an already existent valid period. The data manipulation is rejected if this is true.

To query data with temporal conditions, [14, 10] declares which three different clauses are supported. FOR BUSINESS_TIME AS OF queries data at a specific moment. FOR BUSINESS_TIME FROM TO returns data from a specified date including this point in time, to a certain point in time when the data of the row is not valid anymore. To query a range, where start and end time is included, FOR BUSINESS_TIME BETWEEN AND is used.

For the use case where a bank manager checks which accounts will be closed within two months time, the structure of accounts as it has been illustrated in Table 2 will be extended. As described above and illustrated in Listing 19, two date columns are appended, VALIDSINCE and VALIDUNTIL. These are defined as PERIOD and as PRIMARY KEY, which is WITHOUT OVERLAPS.

As Listing 20 shows, when data is inserted into the accounts table, VALIDSINCE and

Listing 20: Insert of data into accounts table

```

1 INSERT INTO ACCOUNTS1 (ID, ACCOUNTNR, OWNERID, BALANCE, VALIDSINCE, VALIDUNTIL)
2 VALUES (1, 555.555, 1, 999.99, '2012-10-10','2013-10-10'),
3 (2, 123.456, 1, 500.00,'2012-02-02','2013-02-02'),
4 (3, 987.654, 1, -20.00,'2013-03-03','2014-03-03'),
5 (4, 777.777, 2, 100.00,'2012-07-07','2013-07-07'),
6 (5, 999.999, 3, 1000.00,'2013-04-04','2014-04-04');

```

Listing 21: Query valid time from accounts table

```

1 SELECT * FROM ACCOUNTS1
2 FOR BUSINESS_TIME FROM CURRENT TIMESTAMP TO CURRENT TIMESTAMP + 2 MONTH
3 WHERE VALIDUNTIL < (CURRENT TIMESTAMP + 2 MONTH);

```

VALIDUNTIL dates have to be specified, too.

The query which accounts will be closed within two months time is listed in Listing 21. The BUSINESS_TIME FROM TO temporal condition is used to check which accounts are valid the next two months. Because of the WHERE condition those accounts are returned which are no longer valid than two months from now on. Figure 31 shows the result of this query.

As mentioned in [1], historical databases support valid time. Due to the fact that IBM DB2 implements valid time, it also supports this type of database.

6.2 Historical Data

”The support in DB2 for system time enables [...] to automatically capture changes made to the state of [a] table and to save ’old’ rows in a history table[.]” [14, 4f.] Three steps are required to implement system time functionality for a table. First of all, add a system start and end column as well as a transaction start time column. Furthermore, this three columns are of data type TIMESTAMP and defined as GENERATED ALWAYS, so DB2 generates a timestamp when the data of a table is altered. To define the system start and end column as system period the statement PERIOD SYSTEM_TIME is used. Another point is to create a table with the same structure to store the historical data within. Lastly, enable versioning by the phrase ADD VERSIONING

ID *	ACCOUNTNR *	OWNERID *	BALANCE *	VALIDSINCE *	VALIDUNTIL *
4	777.777	2	100	07.07.2012	07.07.2013

Figure 31: Result of the valid time query

Listing 22: Historical table implementation via System Time

```

1  CREATE TABLE DB2ADMIN.ACCTS2 (
2      ID INTEGER NOT NULL,
3      ACCOUNTNR VARCHAR(15) NOT NULL,
4      OWNERID INTEGER NOT NULL,
5      BALANCE DOUBLE NOT NULL,
6      SYSTEMBEGIN TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN NOT NULL,
7      SYSTEMEND TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END NOT NULL,
8      TRANSSTART TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID NOT NULL,
9      PERIOD SYSTEM_TIME (SYSTEMBEGIN, SYSTEMEND),
10     PRIMARY KEY (ID),
11     FOREIGN KEY OWN (OWNERID) REFERENCES DB2ADMIN.OWNERS
12 );
13 CREATE TABLE DB2ADMIN.ACCTS2_HIST LIKE ACCTS2;
14 ALTER TABLE ACCTS2 ADD VERSIONING USE HISTORY TABLE ACCTS2_HIST;

```

Listing 23: Query of historical data

```

1  SELECT ID, ACCOUNTNR, OWNERID, BALANCE, SYSTEMBEGIN, SYSTEMEND, TRANSSTART
2  FROM ACCTS2 FOR SYSTEM_TIME FROM '0001-01-01' TO '9999-12-30'
3  WHERE ACCOUNTNR = '777.777' ;

```

USE HISTORY TABLE. [16, 5-11]

As [17, 7] mentions, the system time periods, which are stored after these configurations of a table, are inclusive-exclusive. This means that the start time column defines the first point in time when data is inserted or updated. Moreover, the end time column gives information about the point in time, when the data is changed and another row in the table illustrates the state of that time.

The use case for historical data is to list historical balances of a specific bank account. In order to get this list, the above described steps have to be implemented, see Listing 22. The accounts table gets three more TIMESTAMP columns, whose names are SYSTEMBEGIN, SYSTEMEND and TRANSSTART. SYSTEMBEGIN is generated AS ROW BEGIN, SYSTEMEND is generated AS ROW END. Moreover, the PERIOD SYSTEM_TIME is defined, which involves these two columns. As a second step, the history table with same structure is created. Finally, ACCTS2_HIST is declared as HISTORY TABLE of ACCTS2.

To be able to query historical data, the balance of the account with number 777.777 is updated. How the balance has been changed, can be queried, as Listing 23 and Figure 32 illustrates. The row with the date 30.12.9999 shows the current balance, because it has no valid end date yet.

ID *	ACCOUNTNR *	OWNERID *	BALANCE *	SYSTEMBEGIN *	SYSTEMEND *	TRANSSTART *
4	777.777	2	100	26.05.2013 01:43:19	26.05.2013 01:43:37	26.05.2013 01:43:19
4	777.777	2	2000	26.05.2013 01:43:37	30.12.9999 00:00:00	26.05.2013 01:43:37

Figure 32: Results of the historical data query

Listing 24: Transaction time implementation in transactions table

```

1 CREATE TABLE DB2ADMIN.TRANSACTIONS (
2   FROMACCOUNTID INTEGER NOT NULL,
3   TOACCOUNTID INTEGER NOT NULL,
4   AMOUNT DOUBLE NOT NULL,
5   TRANSDATE TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID NOT NULL,
6   PRIMARY KEY (FROMACCOUNTID, TOACCOUNTID, TRANSDATE),
7   FOREIGN KEY FROMACC (FROMACCOUNTID) REFERENCES ACCOUNTS2,
8   FOREIGN KEY TOACC (TOACCOUNTID) REFERENCES ACCOUNTS2
9 );

```

6.3 Transaction Time

”A transaction-start-ID column is required for a system-period temporal table. If defined as NOT NULL, the value corresponds to the start time associated with the most recent transaction (the value of the AS ROW BEGIN column).” [44]

Accordingly, this column gives information about the transaction time when the data of a row has been changed.

The use case regarding transaction time is to show all transactions of the last month. The structure for the transactions table illustrates Listing 24. It includes the TRANSDATE which is generated AS TRANSACTION START ID. Furthermore, it is important that this column is defined as NOT NULL, otherwise the value is always null.

Listing 25 is the query to see which payments and payoffs for a given bank account has been executed in the last month. Figure 33 displays the result.

The table which is shown in Figure 32 also includes the transaction time. It can be recognized as one of the types of databases mentioned in [1], to be specific, the type is Rollback.

Listing 25: Transaction time implementation in transactions table

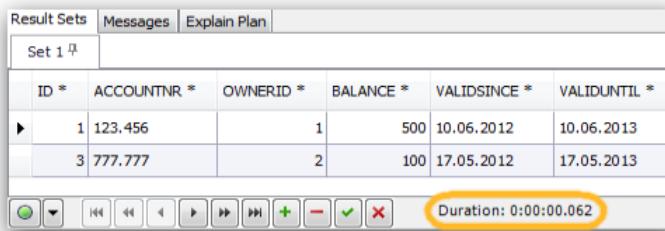
```

1 SELECT * FROM TRANSACTIONS
2 WHERE TRANSDATE > (CURRENT TIMESTAMP - 1 MONTH);

```

FROMACCOUNTID *	TOACCOUNTID *	AMOUNT *	TRANSDATE *
1	2	100	26.05.2013 01:20:54
1	3	50	26.05.2013 01:20:54
2	1	90	26.05.2013 01:20:54
2	3	100	26.05.2013 01:20:54
5	3	200	26.05.2013 01:20:54
5	1	2500	26.05.2013 01:20:54
5	2	200	26.05.2013 01:20:54
1	2	2500	26.05.2013 01:23:25
5	2	50000	26.05.2013 01:23:25
2	3	60000	26.05.2013 01:23:25
5	1	100	26.05.2013 01:23:25
5	1	100	26.05.2013 01:23:47

Figure 33: Results of the transaction time query



The screenshot shows a database query results window in Toad for DB2. The results are displayed in a grid with columns: ID *, ACCOUNTNR *, OWNERID *, BALANCE *, VALIDSINCE *, and VALIDUNTIL *. There are two rows of data: one with ID 1 and another with ID 3. At the bottom of the window, there is a toolbar with various icons and a status bar that displays the duration of the transaction as "Duration: 0:00:00.062".

ID *	ACCOUNTNR *	OWNERID *	BALANCE *	VALIDSINCE *	VALIDUNTIL *
1	123.456	1	500	10.06.2012	10.06.2013
3	777.777	2	100	17.05.2012	17.05.2013

Figure 34: Transaction Time Measurement in Toad for DB2

6.4 Transaction Timeout

[45] contains some information about transaction timeout. The database configuration parameter locktimeout defines the limit given in seconds, how long an application waits for a lock to end. The SQL statement SET CURRENT LOCK TIMEOUT can change this configuration parameter. It sets the locktimeout value to wait for the specified seconds before to return an error.

6.5 Transaction Time Measurement

The duration for which an SQL statement is processed is called response time in DB2. The elapsed times can be recorded for example in Tivoli OMEGAMON accounting reports. Tivoli is not supported in every DB2 product. Although Advanced Enterprise Server Edition includes this functionality, DB2 Express-C does not. [45]

Nevertheless, there is a possibility to see how long the duration of one transaction is if a tool for example IBM Data Studio or Toad for DB2 is used. Whenever a SQL statement is executed, a line with the duration is displayed as you can see in figure 34.

Listing 26: Bi-Temporal Table implementation

```

1  CREATE TABLE DB2ADMIN.ACCTS (
2      ID INTEGER NOT NULL,
3      ACCOUNTNR VARCHAR(15) NOT NULL,
4      OWNERID INTEGER NOT NULL,
5      BALANCE DOUBLE NOT NULL,
6      VALIDSINCE DATE NOT NULL,
7      VALIDUNTIL DATE NOT NULL,
8      SYSTEMBEGIN TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN NOT NULL,
9      SYSTEMEND TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END NOT NULL,
10     TRANSSTART TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID NOT NULL,
11     PERIOD BUSINESS_TIME(VALIDSINCE, VALIDUNTIL),
12     PERIOD SYSTEM_TIME (SYSTEMBEGIN, SYSTEMEND),
13     PRIMARY KEY (ID, BUSINESS_TIME WITHOUT OVERLAPS),
14     FOREIGN KEY OWN (OWNERID) REFERENCES DB2ADMIN.OWNERS
15 );
16  CREATE TABLE DB2ADMIN.ACCTS_HIST LIKE ACCTS;
17  ALTER TABLE ACCTS ADD VERSIONING USE HISTORY TABLE ACCTS_HIST;

```

Listing 27: Update data and display Bi-Temporal table

```

1  UPDATE ACCTS
2  SET VALIDSINCE = '2013-10-10', VALIDUNTIL = '2014-10-10'
3  WHERE id = 1;
4  COMMIT;
5  SELECT ID, ACCOUNTNR, OWNERID, BALANCE, VALIDSINCE, VALIDUNTIL, TRANSSTART
6  FROM ACCTS FOR SYSTEM_TIME FROM '0001-01-01' TO '9999-12-31'
7  WHERE ID=1;

```

6.6 Temporal Database

DB2 provides an additional time concept which is not covered by a use case in chapter 3. "A bi-temporal table is a table that combines the historical tracking of a system temporal table with the time-specific data storage capabilities of an application temporal table." [15]

In conclusion, the structure of the accounts table of the two use cases of the sections 6.1 and 6.2 can be merged to one. As a result, it is called bi-temporal table. Listing 26 shows the structure. If the valid time is changed and the query is executed as shown in Listing 27, it results in figure 35. This table represents the Temporal type of database which is defined in [1].

6.7 Summary of time concepts in IBM DB2

To sum up, every single use case defined in Chapter 3 has been implemented using the relational database management system IBM DB2. Moreover, the statement of [19] is

ID *	ACCOUNTNR *	OWNERID *	BALANCE *	VALIDSINCE *	VALIDUNTIL *	TRANSSTART *
1	555.555	1	999,99	10.10.2012	10.10.2013	26.05.2013 02:16:26
1	555.555	1	999,99	10.10.2013	10.10.2014	26.05.2013 02:16:36

Figure 35: Result of Bi-Temporal Table query

Checkpoint	Support	Usability
Valid time	yes	excellent
History Data	yes	excellent
Transaction time	yes	excellent
Query Timeouts	yes	good
Transaction Runtime	depends on edition	depends on edition
Snapshot	yes	excellent
Rollback	yes	excellent
Historical	yes	excellent
Temporal	yes	excellent

Table 8: Summary of Performance and Usability in IBM DB2

confirmed that IBM has implemented the specification of temporal data defined by the SQL:2011 standard.

Furthermore, [14, 10] emphasizes that not only application logic decreases and handling of time-related events is more consistent, it also saves time and money. Additionally, the four types of databases defined in [1] are supported by DB2. A detailed description on research of the types Rollback, Historical and Temporal has been given in sections 6.1, 6.3 and 6.6.

The results of all checkpoints have been collected in table 8.

7 Comparison of Oracle, MS-SQL and DB2 – Strengths and weaknesses (M. Haslinger, S. Pauli)

The decision if a concrete functionality regarding time concepts is available in a database system can be made objectively, whereas it is much more difficult to find a reasonable grade for the usability. Therefore, we discussed the different implementation approaches as well as our experience we got when we have worked with the system. Hence, each system for each discussed use case has been assigned a rating for support and usability of the feature. For a not supported function or a bad usability it has been assigned only one point. For an excellent functionality, which means that the time concept is supported respectively it provides a good usability, it has been awarded three points.

The results with the individual and accumulated points for each database system are shown in Table 9. In the sections historical data and valid time Oracle and DB2 get better grades than MS SQL. Regarding transaction time and transaction runtime the three database systems seem to implement the concepts with nearly equivalent support and usability. Query timeouts is better supported in MS SQL and DB2 than in Oracle. The four time concepts described by Snodgrass and Ahn in [1] (Snapshot, Rollback, Historical, Temporal) have also been included in Table 9.

To conclude, there are various grades in the diverse sections. To get an overview, these have been summed up. The most significant factor is that MS SQL Server comes far behind on the last place, because it does not provide support for essential time concepts. The Oracle and IBM database systems are nearly on the same level with a quite good support in general. However, the best solution concerning our use cases which includes objective as well as subjective point of views is IBM DB2.

Use Case	Oracle 11g/12c	Microsoft SQL Server 2012	IBM DB2
<i>Historical Data</i>			
Support	3	2	3
Usability	3	1	3
<i>Transaction time</i>			
Support	3	3	3
Usability	3	2	3
<i>Valid time</i>			
Support	3	1	3
Usability	3	1	3
<i>Query Timeouts</i>			
Support	1	3	3
Usability	1	2	2
<i>Transaction Runtime</i>			
Support	3	3	2
Usability	3	2	2
<i>Snapshot Database</i>			
Support	3	3	3
Usability	3	3	3
<i>Rollback Database</i>			
Support	3	3	3
Usability	3	2	3
<i>Historical Database</i>			
Support	3	2	3
Usability	3	1	3
<i>Temporal Database</i>			
Support	1	1	3
Usability	1	1	3
Total			
Rank	2	3	1

Table 9: DBS comparision overview

8 Conclusion

In this paper, the three different relational database management systems Oracle, MS SQL and IBM DB2 have been studied if and to what extent they support time concepts. Hence, it has been examined if they implement valid time, historical data, transaction time, query timeouts and transaction time measurement. It has been illustrated by use cases which could occur in a large Austrian bank. Finally, a comparison overview is shown where you can see which database system supports which time concept. Moreover, a personal opinion about the usability to implement the time concepts has been provided in the comparison table.

To sum it up, the database management system IBM DB2 has the best result in comparison to Oracle and MS SQL. Depending on the edition, it supports all time concepts. Microsoft SQL Server 2012 supports in fact only transaction time as an explicit time concepts which can be misused to imitate some other aspects. However, it is by far the last capable system concerning temporal aspects. Oracle only misses a transaction timeout functionality but implements all other time concepts and can be rated as a very good second choice. All in all, it can be said that every relational database management system has its own strengths and weaknesses with respect to historical data.

All of the manufacturers and especially Microsoft push their customers to use their data warehouse/business intelligence product. Therefore, for these use cases it can be an interesting point to compare the three ecosystems not on the pure database but on the OLAP level maybe in cooperation with the according database management system in future research work.

References

- [1] R. Snodgrass and I. Ahn, "Temporal Databases," *IEEE Computer*, vol. 19, pp. 35–42, 1986.
 - [2] S. Chakravarthy and S.-K. Kim, "Resolution of time concepts in temporal databases," *Information Sciences*, vol. 80, no. 1-2, pp. 91–125, Sep. 1994. [Online]. Available: [http://dx.doi.org/10.1016/0020-0255\(94\)90059-0](http://dx.doi.org/10.1016/0020-0255(94)90059-0)
 - [3] T. Johnston and R. Weis, *Managing Time in Relational Databases*. Burlington, MA, USA: Morgan Kaufmann, 2010.
 - [4] C. Jensen and R. Snodgrass, "Temporal data management," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, pp. 36–44, 1999.
 - [5] C. Mao, H. Ma, Y. Tang, and L. Yao, *Temporal Information Processing Technology and Its Application*. Springer Berlin Heidelberg, 2010.
 - [6] Microsoft MSDN, "Track Data Changes (SQL Server)," 2012. [Online]. Available: <http://msdn.microsoft.com/en-us/library/bb933994.aspx>
 - [7] B. McGehee, "An Introduction to SQL Server 2008 Change Data Capture," 2010. [Online]. Available: <http://www.bradmcgehee.com/2010/04/an-introduction-to-sql-server-2008-change-data-capture/>
 - [8] D. Pinal, "Introduction to Change Data Capture (CDC) in SQL Server 2008," 2009. [Online]. Available: [https://www.simple-talk.com/sql/learn-sql-server/introduction-to-change-data-capture-\(cdc\)-in-sql-server-2008/](https://www.simple-talk.com/sql/learn-sql-server/introduction-to-change-data-capture-(cdc)-in-sql-server-2008/)
 - [9] D. Kawliche, "Time After Time: Creating a Valid Time Sql Server Audit Table," 2012. [Online]. Available: <http://www.restfuldevelopment.net/time-after-time-creating-a-valid-time-sql-server-audit-table/>
 - [10] I. Ben-Gan, "Query Temporal Data," 2008. [Online]. Available: <http://www.sqlmag.com/article/sql-server/query-temporal-data-99874>
 - [11] Microsoft MSDN, "Change Data Capture Stored Procedures (Transact-SQL)," 2013. [Online]. Available: <http://msdn.microsoft.com/en-us/library/bb500244.aspx>
 - [12] Programming4Us, "Transact-SQL in SQL Server 2008 : Change Data Capture (part 2) - Querying the CDC Tables," 2012. [Online]. Available: <http://programming4.us/database/7230.aspx>
 - [13] S. Agarwal, "Track Data Changes In SQL Server 2012," 2013. [Online]. Available: <http://www.c-sharpcorner.com/UploadFile/ae6b35/track-data-changes-in-sql-server-2012/>
-

- [14] C. Saracco, M. Nicola, and L. Gandhi, “A matter of time: Temporal data management in DB2 10,” 2012. [Online]. Available: <http://www.ibm.com/developerworks/data/library/techarticle/dm-1204db2temporaldatabase/index.html>
- [15] R. Matchett, “Temporal Tables in DB2,” 2012. [Online]. Available: <http://ibmdatamag.com/2012/04/temporal-tables-in-db2/>
- [16] J.-E. Michels and M. Nicola, “Adopting temporal tables in DB2, Part 1: Basic migration scenarios for system-period tables,” 2012. [Online]. Available: <http://www.ibm.com/developerworks/data/library/techarticle/dm-1210temporaltablesdb2/dm-1210temporaltablesdb2-pdf.pdf>
- [17] M. Nicola, “Best Practices Temporal Data Management with DB2,” 2012. [Online]. Available: https://www.ibm.com/developerworks/mydeveloperworks/wikis/form/anonymous/api/library/0fc2f498-7b3e-4285-8881-2b6c0490ceb9/document/7d8b3282-b3d3-42a2-b225-884a86429360/attachment/43b6d119-228b-4998-a436-dc301881d8de/media/DB2BP_Temporal_Data_Management_1012.pdf
- [18] ——, “Managing Time in DB2 with temporal consistency,” 2012. [Online]. Available: <http://www.ibm.com/developerworks/data/library/techarticle/dm-1207db2temporalintegrity/dm-1207db2temporalintegrity-pdf.pdf>
- [19] D. Petkovic, “Was lange währt, wird endlich gut: Temporale Daten im SQL-Standard,” 2013. [Online]. Available: <http://link.springer.com/article/10.1007/s13222-013-0120-3>
- [20] B. Beauregard and B. Speckhard, “Oracle Database Workspace Manager Developer’s Guide,” 2012. [Online]. Available: http://docs.oracle.com/cd/E11882_01/appdev.112/e11826.pdf
- [21] N. Mahmood, A. Burney, and K. Ahsan, “A Logical Temporal Relational Data Model,” vol. 7, no. 1, pp. 1–9, 2010.
- [22] R. T. Snodgrass, “Temporal databases,” 1986, pp. 1–8.
- [23] T. Johnston and R. Weis, *Managing time in relational databases: How to design, update and query temporal data.* Amsterdam [u.a.]: Elsevier Morgan Kaufmann, 2010.
- [24] C. S. Jensen, J. Clifford, S. I. Gadia, P. Hayes, S. Jajodia, K. Wolfgang, N. Kline, A. Montanari, J. F. Roddick, N. L. Sarda, M. Rita, S. Arie, R. T. Snodgrass, M. D. Soo, and A. Tansel, “Glossary of Temporal Database Concepts * Curtis Dyreson Nikos Lorentzos Paolo Tiberio Relevance ria for the and Evaluation Glossary Crite- Concepts t erest of General Database In-,” vol. 23, no. 1, 1994.

- [25] S. Klein, *Professional WCF Programming: .NET Development with the Windows Communication Foundation.* Birmingham, UK, UK: Wrox Press Ltd., 2007.
- [26] P. Kemper and W. H. Sanders, *Computer Performance Evaluations, Modelling Techniques and Tools. 13th International Conference, TOOLS 2003, Urbana, IL, USA, September 2-5, 2003, Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2003, vol. 2794.
- [27] Oracle Corporation, “Oracle Database 11g Workspace Manager Overview ,” 2009. [Online]. Available: <http://www.oracle.com/technetwork/database/twp-appdev-workspace-manager-11g-128289.pdf>
- [28] ——, “Enable Versioning for Tables,” 2004. [Online]. Available: http://docs.oracle.com/cd/B12037_01/appdev.101/b10824/long_ref.htm#i80309
- [29] ——, “Oracle® Database Workspace Manager Developer’s Guide 11g Release 1 (11.1),” 2013. [Online]. Available: http://docs.oracle.com/cd/B28359_01/appdev.111/b28396/long_vt.htm
- [30] ——, “Workspace Manger - HIST Views,” 2003. [Online]. Available: http://docs.oracle.com/cd/B12037_01/appdev.101/b10824/long_views.htm#i87609
- [31] Burleson Consulting, “Oracle enable row movement tips,” 2012. [Online]. Available: http://www.dba-oracle.com/t_enable_row_movement.htm
- [32] Oracle Corporation, “Oracle® Database 2 Day DBA 11g Release 2 - Monitoring and Tuning the Database, url = http://docs.oracle.com/cd/E25178_01/server.1111/e10897/montune.htm, urldate = 05.05.2013, year = 2012.”
- [33] ——, “Oracle® Database SQL Reference 10g Release 2 - SCN_TO_TIMESTAMP, url = http://docs.oracle.com/cd/B19306_01/server.102/b14200/functions142.htm, urldate = 05.05.2013, year = 2005.”
- [34] ——, “Oracle® Database Net Services Reference 11g Release 2 - SQLNET.EXPIRE_TIME, url = http://docs.oracle.com/cd/E11882_01/network.112/e10835/sqlnet.htm#NETRF209, urldate = 05.05.2013, year = 2012.”
- [35] Burleson Consulting.
- [36] Microsoft MSDN, “About Change Tracking (SQL Server).” [Online]. Available: <http://msdn.microsoft.com/en-us/library/bb933875.aspx>
- [37] ——, “Change Tracking Functions (Transact-SQL).” [Online]. Available: <http://msdn.microsoft.com/en-us/library/bb964727.aspx>

- [38] ——, “About Change Data Capture (SQL Server),” 2008. [Online]. Available: <http://msdn.microsoft.com/en-us/library/cc645937.aspx>
- [39] ——, “cdc.fn_cdc_get_all_changes_capture_instance (Transact-SQL),” 2013. [Online]. Available: <http://msdn.microsoft.com/en-us/library/bb510627.aspx>
- [40] Hosting.com, “Configure Query Timeout Period in SQL Management Studio,” 2013. [Online]. Available: <http://www.hosting.com/support/sql/configure-query-timeout-period-in-sql-management-studio>
- [41] Microsoft MSDN, “Configure the remote query timeout Server Configuration Option,” 2013. [Online]. Available: <http://msdn.microsoft.com/en-us/library/ms189040.aspx>
- [42] IBM, “DB2 software editions.” [Online]. Available: <http://www-01.ibm.com/software/data/db2/linux-unix-windows/index.html>
- [43] ——, “What is DB2 Express-C?” [Online]. Available: <http://www-01.ibm.com/software/data/db2/express-c/about.html>
- [44] S. Rameshkumar, “Go back in time. Get started with temporal data, a timely new DB2 feature.” [Online]. Available: http://www.ibm.com/developerworks/data/library/dmmag/DMMag_2011_Issue3/TemporalData/
- [45] IBM, “locktimeout.” [Online]. Available: <http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp>

Data Warehousing & Data Mining

Markus Haslinger

DBI/INSY

Agenda

1 Basics

2 Data Warehousing

3 Data Mining

Database Usage Scenarios

- There are basically two use case types for databases:
 - 1 OLTP
 - 2 OLAP
 - Online Transaction Processing:
 - The operative day-to-day business (e.g. orders, bookings,...)
 - Performing a lot of update (insert) operations
 - Usually only small amounts of data are processed at once within a transaction
 - Operating on the latest and currently valid state (of the world)

Database Usage Scenarios

■ Online Analytical Processing

- Performing a lot of query operations
 - Often working with huge amounts of data at once
 - Concerned mostly with historic data
 - The goal is to draw conclusions about the development of KPIs
 - Tries to answer questions like:
 - How have sales of a specific product category progressed in a specific sales region over the last three years?
 - Meant to support the strategic company planning
 - ⇒ Part of Decision Support Systems (DSS) or Management Information Systems (MIS)

Database Usage Scenarios

- Due to the highly different requirements the two operation types should not be performed on the same data basis
 - Different data model, logical and physical optimization
 - Also not on the same DBS due to incompatible resource usage
- So if OLAP use cases emerge ⇒ Creation of a *Data Warehouse*

Data Warehouse

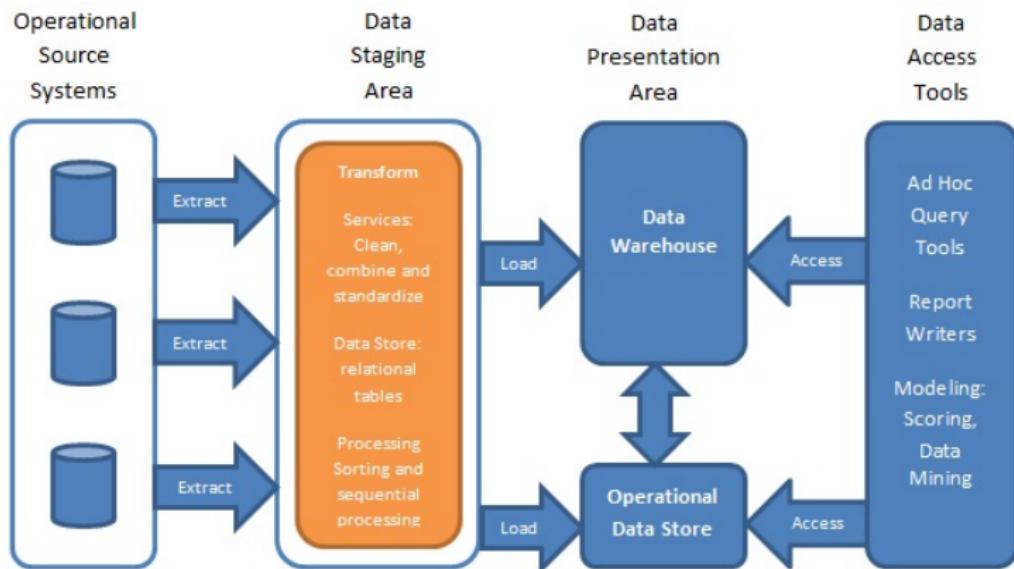
- A dedicated DBS which stores and collects the data needed by DSS
- The data is stored:
 - De-normalized¹
 - Structured
- William Inmon has defined the following properties DW data should have:
 - Topic related (e.g. sales)
 - Consolidated (from different sources and formats)
 - Static / Non-volatile (data once imported never changes)
 - Historic / Time variant (Focus on changes over time)

¹Mind: the *opposite* of normalization

ETL Processes

- Data is loaded and stored initially
 - Then, in defined intervals, additional data is added
 - But existing data is *not* updated
 - Data needs to be prepared for the Data Warehouse
 - For example increasing density via aggregation
 - These processes are called:
 - Extraction
 - Transformation
 - Loading

ETL Processes

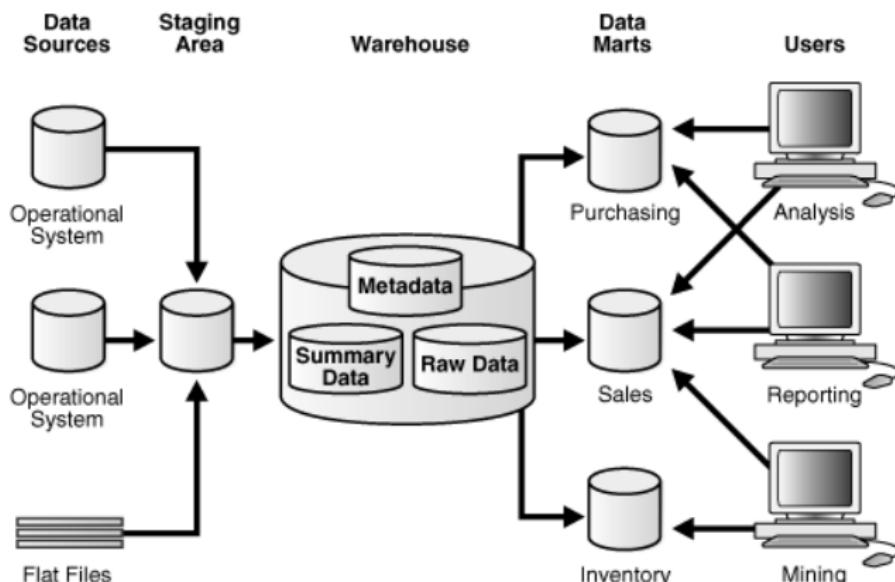


Source: <https://datenbanken-verstehen.de/data-warehouse/data-warehouse-grundlagen/data-warehouse>, 2021-02-14

Data Mart

- A Data Mart can be considered to be an intermediary between the 'main' Data Warehouse and the applications using it
- It is focused on a single functional area of an organization
- Thus, it contains a subset of all the data stored in the Data Warehouse
- It is designed for use by a specific department or business function
 - A department might even control its own Data Mart
- Due to the smaller size and more focused data they are a little easier to use

Data Mart



Source: https://docs.oracle.com/cd/E11882_01/server.112/e25554/concept.htm#DWHSG8075,
2021-02-14

Schema (Data Modelling)

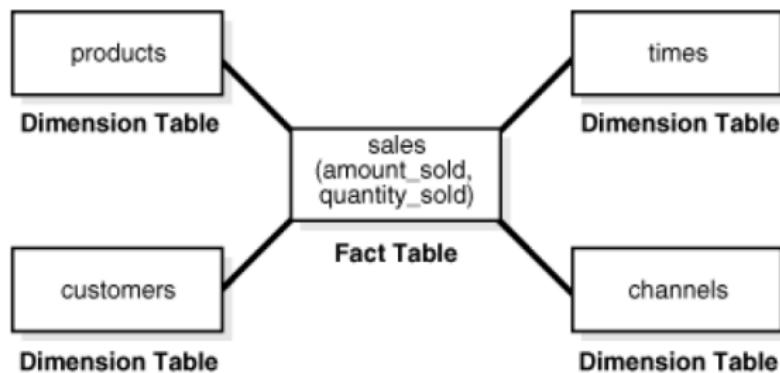
- Three different approaches can be taken when designing the data model of a Data Warehouse:
 - 3NF Schema
 - Star Schema
 - Snowflake Schema

3NF Schema

- The 'usual' schema for OLTP databases
 - Benefit: possibly fewer transformations steps during ETL necessary
- Not that common in Data Warehouses
- As always with normalization the goal is to remove redundancies
- Best suited for very big Data Warehouses

Star Schema

- Most common schema for a Data Warehouse
- Consist of **one fact table** and several **dimension tables**
 - Joined via foreign keys
 - Those dimension tables usually contain less important data
 - Also they are typically not normalized



Star Schema – Example

- Sales(SalesDate, StoreNo, ProdNo, CustNo, EmpNo,
Amount)
- Store(StoreNo, Country, District,...)
- Customer(CustNo, Name, Age,...)
- Salesman(EmpNo, Name, AreaOfExpertise, Manager,...)
- Time(Date, Day, Month, Year, Quarter, Weekday, Season,...)
- Product(ProdNo, Type, Group, Manufacturer,...)



Star Schema – Example

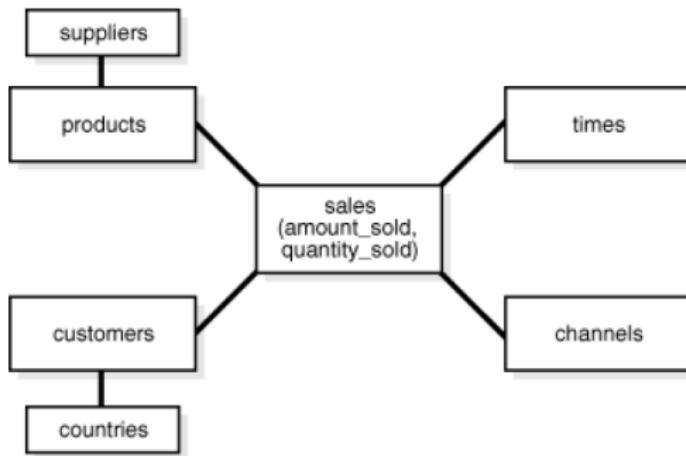
- A typical OLAP query: 'Phones of which manuf. did young cust. in Linz buy for Christmas?'
- Mind that we group to increase the *density* of the data

OLAP Example Query

```
SELECT SUM(s.Amount), p.Manufacturer
FROM Sales s, Store st, Product p, Time t, Customer c
WHERE t.Season = 'Christmas' AND t.Year = 2020
      AND c.Age < 30
      AND p.Type = 'Phone' AND st.District = 'Linz'
      AND s.SalesDate = t.Date AND s.Product = p.ProductNo
      AND s.StoreNo = st.StoreNo AND s.CustNo = c.CustNo
GROUP BY p.Manufacturer;
```

Snowflake Schema

- A more complex version of the Star schema
- The goal is to reduce redundancies by normalizing the dimension tables
- ⇒ leads to more complex queries (more joins)



Increasing Data Density

- Date stored in a Data Warehouse can only be meaningfully interpreted if its density is increased by grouping and aggregating.
- The density level can be influenced by the number of attributes in the Group By clause
 - Fewer attributes → higher density
 - More attributes → lower density

Drill Down

- When more attributes are added to the Group By clause this is called *drill down*
- Data is less dense, because there are more individual groups

Drill Down Example Query

```
SELECT SUM(s.Amount), p.Manufacturer, t.Year
FROM Sales s, Product p, Time t
WHERE s.SalesDate = t.Date AND s.Product = p.ProductNo
      AND p.Type = 'Phone'
GROUP BY p.Manufacturer, t.Year; -- added year
```

Roll Up

- When fewer attributes are used in the Group By clause this is called *roll up*
- This usually leads to the aggregation of one or more dimensions of the Star schema
- The data (result) becomes more dense

Data Cube

- DSS rely strongly on query results of varying density
- If those results are recalculated every time it would be quite costly²
- For example the data created by the previous example query could be represented as the following table³:

Manufacturer/Year	2019	2020	2021	Total
Man1	2000	3000	3500	8500
Man2	1000	1000	1500	3500
Man3	500	1000	1500	3000
Total	3500	5000	6500	15000

²Remember, that data does not change once it has been put into the Data Warehouse.

³Which is an n-dimensional (2 dimensions in this case) data cube!

Data Cube – Table

- Thus, it makes sense to persist commonly used aggregates.
 - Mind: NULL values show the dimension used for aggregation

Manufacturer	Year	Amount
Man1	2019	2000
Man1	2020	3000
Man1	2021	3500
...
Man1	NULL	8500
Man2	NULL	3500
...
NULL	2019	3500
NULL	2020	5000
NULL	2021	6500
...

Data Cube – Table – Example

Persisted Aggregates for Sales by Manufacturer and Year

```
CREATE TABLE SD2DCube (
    Manufacturer VARCHAR2(20),
    Year NUMBER(4),
    Amount NUMBER(5)
)
```

- Inserts: 2^n ($n=\text{dimensions}$)

Data Cube – Table – Example

Insert Aggregates for Sales by Manufacturer and Year

```
INSERT INTO SD2DCube (
    SELECT p.Manufacturer, t.Year, SUM(s.Amount)
    FROM Sales s, Product p, Time t
    WHERE s.Product = p.ProductNo AND p.Type = 'Phone'
        AND s.SalesDate = t.Date
    GROUP BY t.Year, p.Manufacturer)
UNION (
    SELECT p.Manufacturer, NULL, SUM(s.Amount)
    FROM Sales s, Product p
    WHERE s.Product = p.ProductNo AND p.Type = 'Phone'
    GROUP BY p.Manufacturer)
UNION (
    SELECT NULL, t.Year, SUM(s.Amount)
    FROM Sales s, Product p, Time t
    WHERE s.Product = p.ProductNo AND p.Type = 'Phone'
        AND s.SalesDate = t.Date
    GROUP BY t.Year)
UNION (
    SELECT NULL, NULL, SUM(s.Amount)
    FROM Sales s, Product p
    WHERE s.Product = p.ProductNo AND p.Type = 'Phone');
```

DING

Grouping Set – CUBE

- Instead of manually writing 2^n statements a special operator can be used in the Group By clause: CUBE
- This makes not only the query simpler but is also more performant, because the same data does not have to be read multiple times
 - Check the execution plan to verify
- For example: with 3 attributes in the Group By clause the CUBE operator would create the following combinations:
 - $(a, b, c) \Rightarrow \{(), (a), (a, b), (a, c), (b), (b, c), (c), (a, b, c)\}$

Grouping Set – CUBE

CUBE Example

```
SELECT p.Manufacturer, t.Year, st.Country, SUM(s.Amount)
FROM Sales s, Store st, Product p, Time t
WHERE s.Product = p.ProductNo AND p.Type = 'Phone'
      AND s.SalesDate = t.Date AND s.StoreNo = st.StoreNo
GROUP BY CUBE(t.Year, p.Manufacturer, st.Country)
```

Grouping Set – ROLLUP

- Another Group By clause operator is ROLLUP
- It creates the following combinations:
 - $(a, b, c) \Rightarrow \{(), (a), (a, b), (a, b, c)\}$

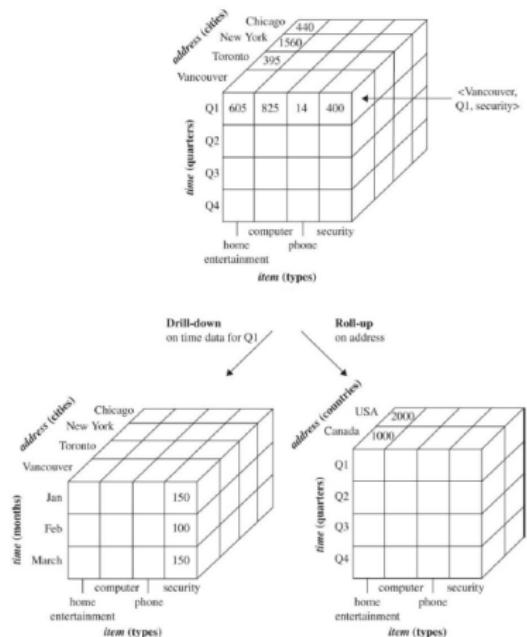
ROLLUP Example

```
SELECT t.Year, t.Quarter, SUM(s.Amount)
FROM Sales s, Time t
WHERE s.SalesDate = t.Date
GROUP BY ROLLUP(t.Year, t.Quarter)
```

Grouping Set – ROLLUP – Example Result

Year	Quarter	Sum
2019	1	13600
2019	2	23167
2019	3	35747
2019	4	55496
2019	NULL	128010
2020	1	56655
...
NULL	NULL	898422

Data Cube – ROLLUP & DRILL-DOWN



Source: Dr. Michael Karlinger, 2012

Grouping Operator

- Both ROLLUP & CUBE use NULL as a placeholder to show grouping dimension
- If you want to differentiate between NULL as a placeholder and an *actual* NULL value use the GROUPING function
 - Returning '0' if the value is actually NULL
 - Returning '1' if the value is a placeholder

GROUPING Example

```
SELECT DECODE(GROUPING(t.Year),1,'Total',t.Year),
       DECODE(GROUPING(t.Quarter),1,'Sum',t.Quarter),
       SUM(s.Amount)
  FROM Sales s, Time t
 WHERE s.SalesDate = t.Date
 GROUP BY ROLLUP(t.Year, t.Quarter)
```

DING

Ranking

- Often it is necessary to perform a ranking of results based on certain conditions
- There is an actual operator – RANK – to help with that

RANK Example

```
SELECT st.StoreNo, SUM(s.Amount), RANK() OVER
    (ORDER BY SUM(s.Amount)/st.EmployeeCount)
FROM Sales s
    INNER JOIN Store st ON (s.StoreNo = st.StoreNo)
GROUP BY st.StoreNo, st.EmployeeCount
ORDER BY st.StoreNo
```

Ranking – Example Result

StoreNo	SUM(s.Amount)	Rank
1	123956	3
2	126924	7
3	133023	2
4	128259	5
5	131322	1

Motivation

- Over the last decade we moved from terabyte scale to petabyte scale when talking about data
- Data is not only collected but also made available
- ⇒ *We are drowning in data but starving for knowledge!*
- Thus, an automated analysis of massive data sets is a necessity
- And Data Mining is the solution

What is Data Mining?

- Knowledge discovery in data is the (semi-)automatic process of extracting knowledge from interesting data
 - Interesting data is: valid, previously unknown and potentially useful
- Different terminology in literature: knowledge extraction, data/pattern analysis, data archeology, information harvesting,...
- But *not* everything is Data Mining, e.g. simple search and query processing

Data to be mined

- The type of data determines:
 - The technologies required to access the data
 - The kind of knowledge that can be (easily) extracted
- Most relevant kinds of data:
 - Relational Data (OLTP)
 - Aggregated Data (OLAP)
 - Transactional Data

Type	Components	Query Language	Typically mined knowledge
Relational Data	tables tuples attributes	SQL	diverse kinds of patterns trends
Aggregated Data	data cubes dimensions cells	SQL + OLAP drill down roll up	similar to relational data but multiple levels of abstraction
Transactional Data	transactions items	depends on storage choice	frequent item sets

Other kinds of data

- Temporal data
 - For example stock quotes
 - The temporal sequence is important
 - Data streams
 - For example sensor data
 - Main challenge: short time for processing
 - Spatial data⁴
 - For example frequently visited natural landmarks
 - Multimedia data
 - For example categorizing movies by analyzing video and audio tracks

⁴c.f. <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/spatial-data>, 2021-02-20

Knowledge to be extracted

- Knowledge = interesting patterns
 - Either descriptive or predictive (goal)
 - Data Mining functionalities:
 - Characterization and Discrimination
 - Association and Correlation
 - Classification and Regression
 - Cluster and Outlier Analysis
 - Sequential Pattern and Trend Analysis

Characterization & Discrimination

■ Characterization

- Summarize properties of a class of objects
- Example:

Big Spenders	
Age	40-50
Employed	yes

■ Discrimination

- Compare one class of objects to another
- Example:

Regular Customers	Random Customers
Age	20-40
Education	university

Association & Correlation

- Frequent patterns
 - Example: Which items are frequently purchased together?
- Association
 - Diaper → Beer
 - In 5% of all purchases, diapers and beer are bought together
 - In 75% of all diaper purchases, beer was also bought
- Causality
 - Is buying diapers the reason for buying beer? Probably not!
- Correlation
 - Do people buy beer more often when they do not buy diapers?
 - If so, purchasing diapers and beer is negatively correlated

Classification & Regression

■ Classification

- Construct models (functions) based on some training examples
- Describe and distinguish classes or concepts for future prediction
- Predict unknown class labels
- Instance of supervised learning, i.e. classes are known
- Example: Classify items based on responses to a sales campaign (more sales, unchanged sales) using features like price, brand,...
- Typical Classification Algorithms: Decision trees, naïve Bayesian classification,...

Classification & Regression

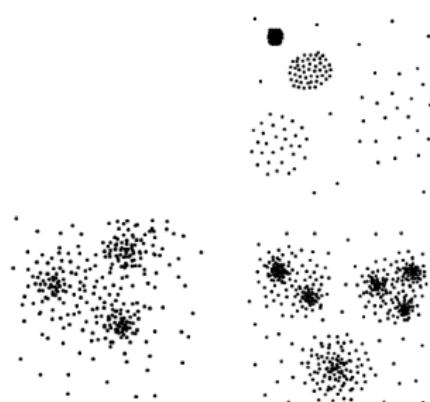
■ Regression

- Classification with countably infinite set of classes
- Statistical method
- Example: Classify amount of revenue of items based on responses to a sales campaign using features like price, brand, . . .

Cluster & Outlier Analysis

■ Clustering

- Group data to form new categories, called clusters
- Example: Cluster houses to distribution patterns
- Principle:
 - Maximizing intra-class similarity
 - Minimizing interclass similarity
- Typical clustering algorithms:
 - k-Means (centroid based)
 - BIRCH (hierarchical clustering)
 - DBSCAN (density based)



Cluster & Outlier Analysis

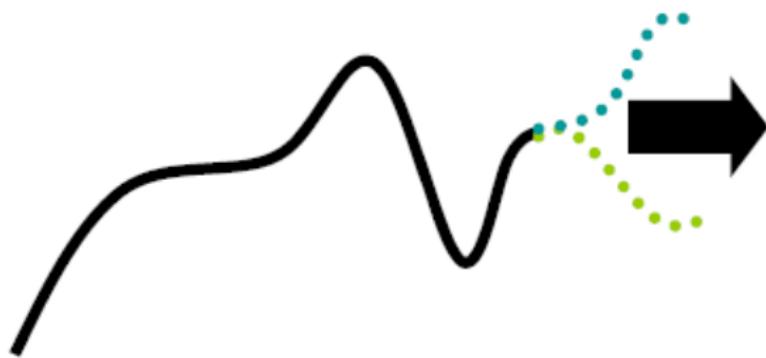
■ Outlier Analysis

- Outliers are data objects outside any cluster, i.e. those which does not comply to the general structure of data
 - By-product of clustering analysis
 - Applied as:
 - The actual aim of data mining process or
 - A technique for cleaning data (preprocessing)
 - Example: Uncover fraudulent usage of credit cards by detecting purchases of unusually large amounts

Sequential Pattern & Trend Analysis

- Sequential Pattern = Frequent Pattern with some ordering (e.g. time)
- Example:
 - First one buys a digital camera, then large SD memory card
 - Contrast to Association: digital cameras and SD cards are frequently bought together
- Applications:
 - Periodicity analysis
 - Biological sequence analysis
- Challenge: Mining in ordered, time-varying, potentially infinite data streams

Sequential Pattern & Trend Analysis



Source: Dr. Michael Karlinger, 2012

Evaluation of Knowledge

- Are all mined patterns interesting?
 - One can mine tremendous amount of 'patterns' and knowledge
 - Some may fit only certain dimension space (time, location,...)
 - Some may not be representative, may be transient,...
- Goals:
 - Completeness: Find all interesting patterns
 - Soundness: Find only interesting patterns
- Objective criteria of interestingness
 - Based on statistics underlying the patterns
 - Dependent of data mining functionality applied
 - Example: support and confidence for association rules
- Subjective criteria of interestingness
 - Based on user beliefs in data
 - Example: unexpectedness of a pattern

Database Basics

DI Markus Haslinger MSc

DBI/INSY

Agenda

1 Introduction

- Introduction
- Historical development
- Requirements and components

2 DBMS Workflow

- DBMS Workflow

3 3 Level Architecture

- 3 Level Architecture

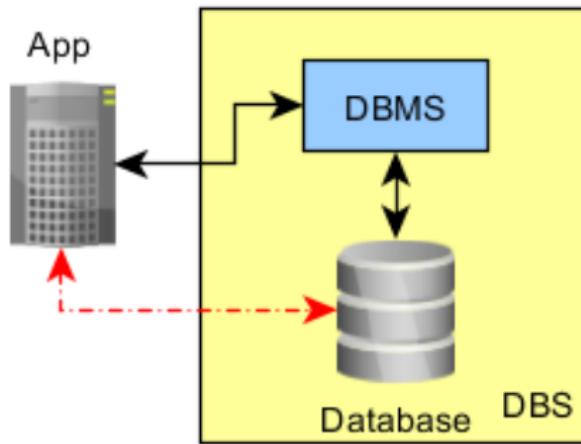
Why use a database?

- The situation:
 - Huge amounts of data in different formats
 - Data required by a multitude of users
 - Data required in different ways
- Consequences:
 - Redundancies ⇒ Inconsistencies
 - Data loss
 - Increased effort for accessing data
 - Dependency on data file structure
 - **Correct data not available when and where needed**

What is a database system?

- Not **just** data (file system vs. database system)
- Four components:
 - Persistent data
 - Hardware (storage and computation)
 - Software managing the data (DBMS)
 - User (directly or via an application)
- A computer supported system for managing and providing **information**

What is a database system?



What is a database system?

- Database (DB): The persistent, structured data maintained by the DBMS
- DBMS: Database management system
 - Interface between physical data and applications/users
 - All activity on the database is controlled and monitored

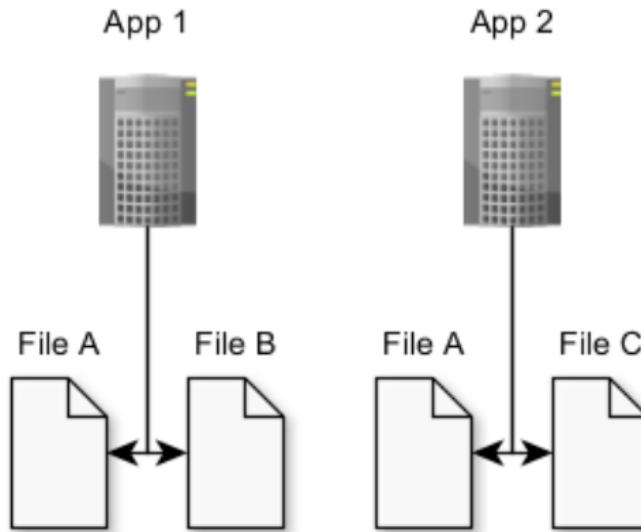


Historical development

- 1 Separate data storage
- 2 Shared data storage
- 3 Hierarchical database systems
- 4 Network based database systems
- 5 **Relational database systems**
- 6 Non-relational database systems

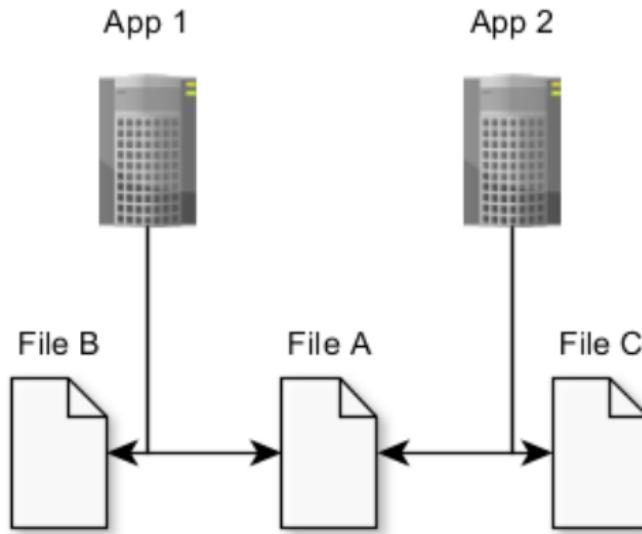


Separate data storage



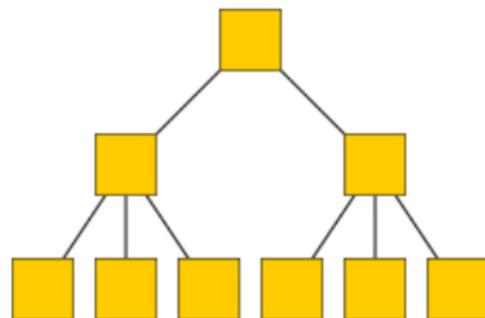


Shared data storage



Hierarchical database systems

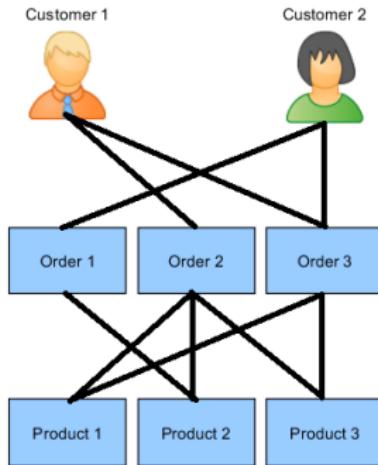
- Tree structure
 - Root node
 - Child nodes
 - Siblings
 - Leaves
- Allows only 1:1 and 1:n relations
 - m:n via replication
- Loose relation: OS file system
- **XML**





Network based database systems

- A generalization of a hierarchical database
- Allows 1:1, 1:n and m:n relations
- Different paths can lead to the same result
- **Not** a graph database



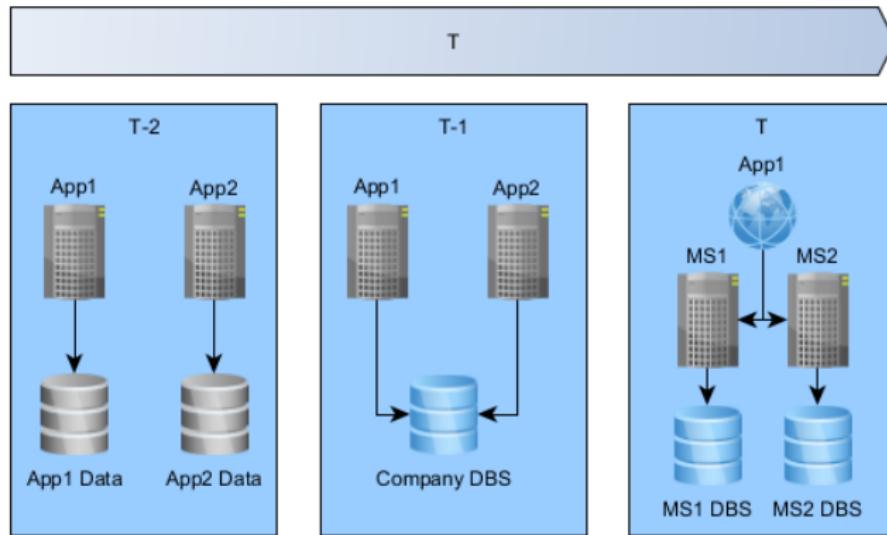


Relational database systems

- Entities and relations
- Usually one table per entity (type)
 - Columns = attributes
 - Rows (records) = data for one entity
 - Should contain a unique key
- Tables are also called relations – be careful not to mix it up with relations between entities
- Entity relations are usually represented with keys
- 'Special', dedicated tables can be used to represent complex entity relations



Historical development





Benefits of a database system

- Reduction of data redundancies
- Ensuring data integrity
 - Correctness
 - Completeness
 - Consistency
- Data security
- Data independence
 - Changing physical organization of data does not require application change
- Specific views on data
- Efficiency



Tasks of a database system

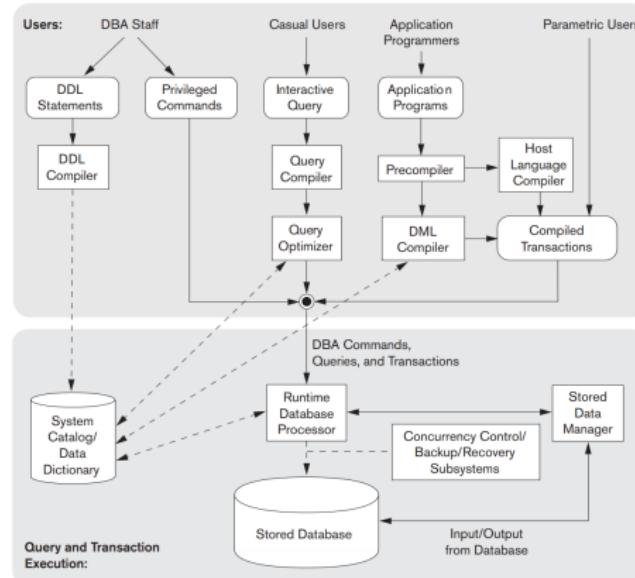
- Storing data
- Managing data
- Access and modify data
- Controlling concurrent access
- Performing optimizations
- Ensuring data security and data integrity



Components of a database system

- Data Dictionary (DD)
 - Contains structural type definitions of application data
 - Contains information about the organization of the data
 - Knows the file system paths
 - Contains view definitions
- SQL Interpreter: Converts SQL queries into disk access operations and creates the result set
- Optimizer: Attempts to reduce the number of disk access operations (cache, query plan)
- Manipulated via:
 - Data Manipulation Language (DML)
 - Data Definition Language (DDL)

Components of a database system



Source: Elmasri, Navathe, Fundamentals of Database Systems, 6th ed.



Requirements (E. Codd)

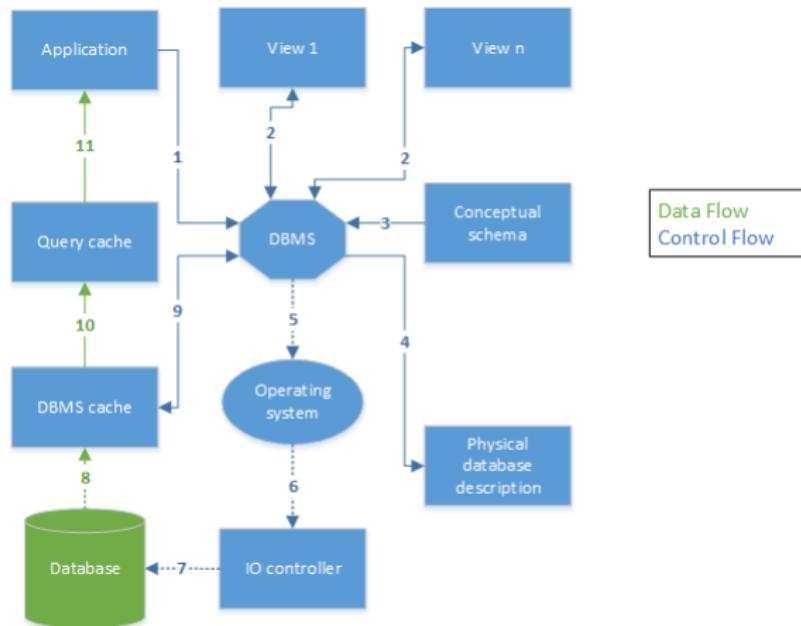
- Integration
 - Uniform, redundancy free administration of all data used by applications
- Operations
 - Storing, modifying and searching for data
- Catalog
 - Metadata (Data Dictionary)
- User views
 - Based on a single application's needs and DBMS controlled
- Consistency checks
 - Ensuring integrity of stored data also after modifications



Requirements (E. Codd)

- Data protection
 - Only authorized access to data (cf. privacy)
- Transactions
 - Grouping of functionally related operations (Commit & Rollback)
- Synchronization
 - Concurrent transactions of different users must not interfere with each other
- Backup & Restore
 - Features for creating and restoring backups
 - Best case: even preventing data loss

DBMS Workflow



DBMS Workflow

- 1 A query is sent to the DBMS
- 2 Before executing the DBMS verifies the validity of the request
- 3 The DBMS determines the type of the requested logical data rows
- 4 The DBMS determines the physical data rows which have to be read
- 5 Data file request is sent to the OS
- 6 The OS verifies the IO commands
- 7 The IO operations are relayed to the database

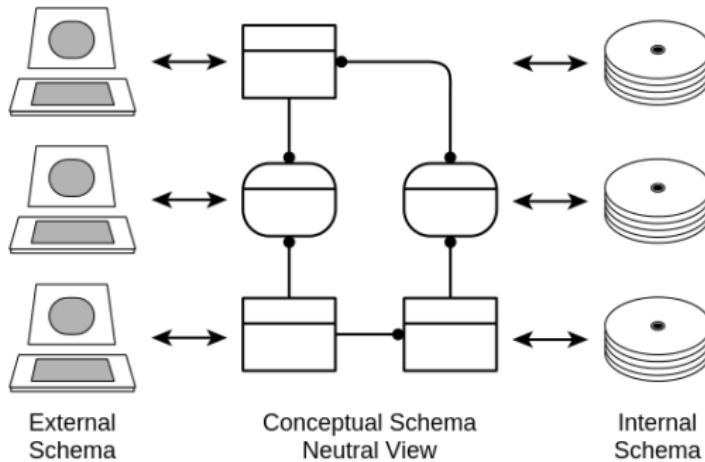
DBMS Workflow

- 8 The requested data is sent to the system cache
- 9 The DBMS sends data from the system cache to the application (which may also have a cache). The data is structured in the way the application requested it.
- 10 See (9)
- 11 See (9)

3 Level Architecture

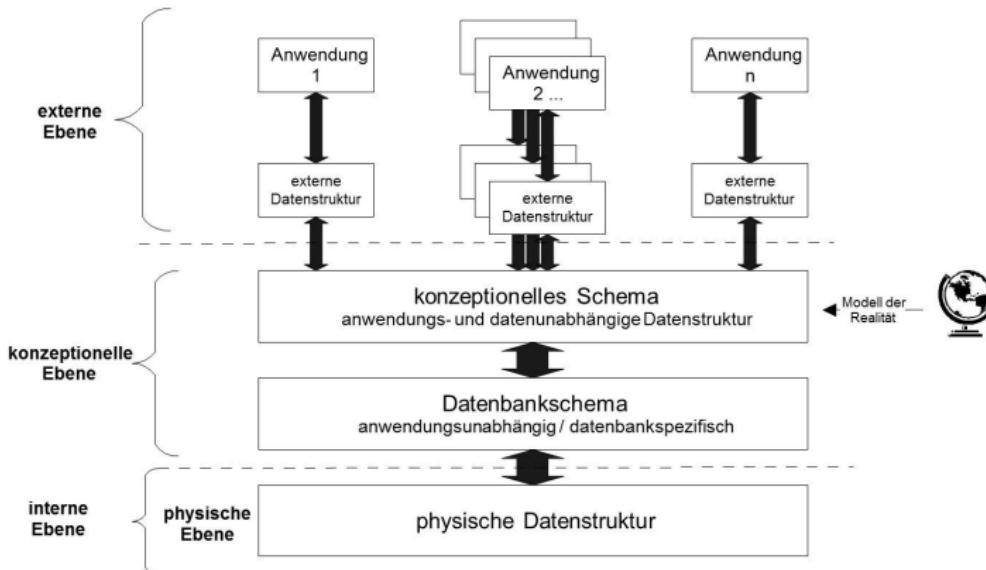
- Data contained in a database is described on three different levels
- Each level is based on a different point of view:
 - The first level represents the real world
 - The second level is the **neutral** link between real world and physical storage
 - The third level represents the 'computer view'

3 Level Architecture



Source: https://en.wikipedia.org/wiki/Three-schema_approach, 2018-09-23

3 Level Architecture



Source: http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/ANSI-3-Ebenenmodell, 2018-08-18

External schema

- A DBS is usually
 - Big and complex
 - Shared by many users
- Provides views on the data (= small section)
- Views can be completely independent or overlapping
- Each user has access to one or more views relevant to him
 - Example: App 1 can only access PersNo and Name, but not the Salary

Conceptual schema

- Describes the logical files and contained record types
 - Example: an employee has a PersNo a Name and a Salary
- Describes the fields present in a row of a specific type
 - Example: PersNo is an Integer, Name has at most 20 characters
- Describes the relations between logically linked record types
 - Example: Book is part of a library
- Describes valid values for specific fields
 - Example: an age value can only be in the range [0-150]
- Represents the whole logical structure

Internal schema

- Representation of attribute values
 - How are numeric values represented (binary, decimal, precision/scale)?
 - How are chars represented (ASCII, Unicode)?
 - Encoding: how is a colour represented (number, reference table, string)?
 - Fixed or variable length
- Structure of persisted rows
 - Field order
 - Lengths
 - Pointer positions

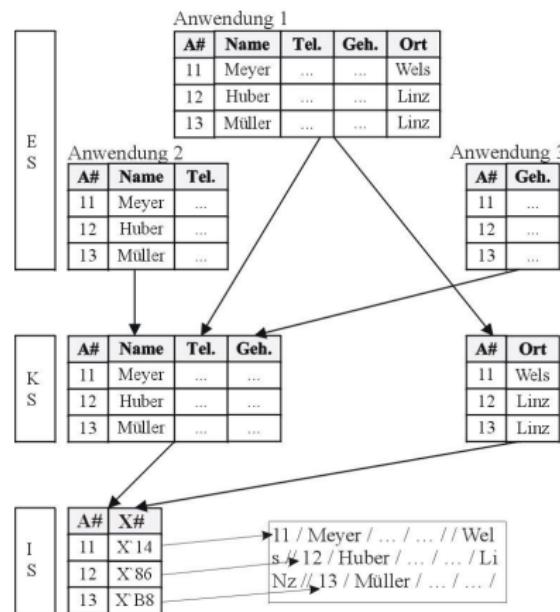
Internal schema

- Strategies for persisting rows
 - Hashing
 - Index
 - direct/indirect access
 - index sequential storage (or not)
 - Clustering: grouping logically related data rows
- Access paths
 - Primary/secondary index
 - Pointer
- Represents the implementation dependent properties

Relations between the levels

- Conceptual schema ↔ External schema
 - Supports logical data independence
 - How well is an application isolated from changes to the conceptional schema?
- Conceptual schema ↔ Internal schema
 - Supports physical data independence
 - How well is an application isolated from changes to the physical database

Example



Data independence

- Does *not* mean the schema is unrelated to the data it should hold
- The goal is the independence of a user (application) from the physical structure, storage medium and storage location
- ⇒ **Data abstraction**

DDL/DML/DCL

Markus Haslinger

DBI/INSY 3rd year

Agenda

1 Tools

2 DDL

3 DML

4 Transactions

■ Transactions

5 DCL

SQL*Plus

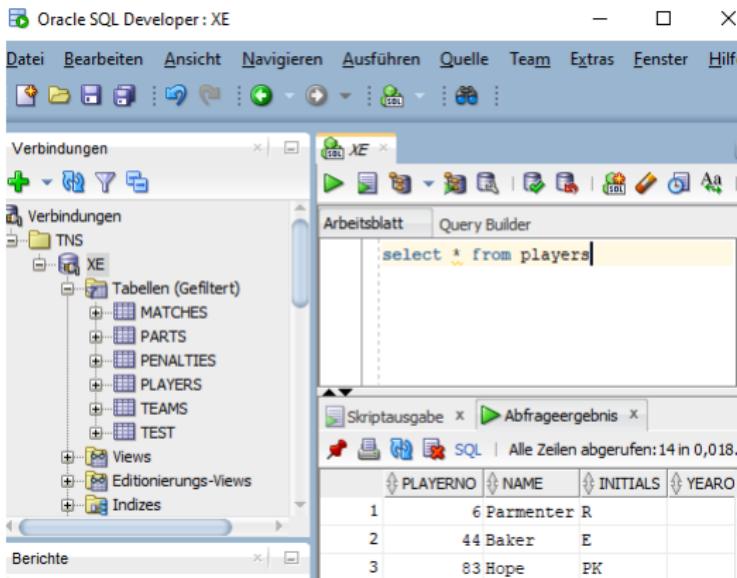
- Console application for sending commands for execution to the DBMS
 - Start via sqlplus command
 - Commands:
 - exit [| quit]: exit SQL*Plus
 - run: re-show and re-submit last statement
 - list: re-show last statement
 - get: open file
 - start: execute
 - /: re-submit last statement
 - @: open and run SQL script

SQL Developer

- IDE for creating SQL and PL/SQL scripts
- Allows to manage and configure databases
- Java based, from Oracle

- Will be used for all examples during this class

SQL Developer



Data Definition Language (DDL)

- Contains commands used to define the database schema
 - CREATE
 - DROP
 - ALTER
 - ...
- Also allows to modify an existing schema
 - Consider possible problems when tables already contain data

CREATE TABLE

- Table name max. 30 characters [Oracle]
- Table name unique within the schema/owner

Syntax

```
CREATE TABLE table_name (
    column_name data_type [(size)] [default_expr] [column_integrity_rule]
    [, ...]
)
```

CREATE TABLE

Example

```
CREATE TABLE Student (
    MatNo NUMBER(10) NOT NULL,
    FirstName VARCHAR2(20),
    LastName VARCHAR2(20) NULL,
    DateOfBirth DATE NULL,
    IsActive CHAR(1) DEFAULT('1') NOT NULL
);
```

- Integrity: **NOT NULL**
- Default values
- Boolean substitute [Oracle]

Data Types – Numeric

- There is only a single (underlying) numeric data type:
NUMBER(P,S) [Oracle]
 - ANSI data types like Integer or Float could be used but are internally represented as NUMBER
- P defines the precision (= total number of digits)
- S defines the scale (= decimal places)
- Example: **NUMBER(6,2) = 0000.00**

Data Types – Alphanumeric

Data Type	Description
CHAR(n)	string with a length of n (space padding)
VARCHAR2(n)	string with a max. length of n
NCHAR/NVARCHAR2	allows to use a different charset (unicode), otherwise equal to CHAR/VARCHAR2
CLOB	Character Large Object (max. 4GB)
LONG	stores VARCHAR values up to 2GB (backwards comp. only)

Data Types – Date

Data Type	Description
DATE	stores date and time (1s precision)
TIMESTAMP(p)	stores date and time p defines the fractional second places

Data Types – Binary

Data Type	Description
BLOB	Binary Large Object (max. 4GB)
BFILE	Reference to external file
RAW	1-2000 raw bytes (e.g. picture)
LONG RAW	max. 2GB, raw bytes
ROWID	fixed length binary value every row has a ROWID (pseudo column)

DESC[RIBE] table

Example

```
DESC Student;
```

Name	Null?	Typ
MATNO	NOT NULL	NUMBER(10)
FIRSTNAME		VARCHAR2(20)
LASTNAME		VARCHAR2(20)
DATEOFBIRTH		DATE
ISACTIVE	NOT NULL	CHAR(1)

DROP TABLE

- Syntax: **DROP TABLE** table_name [**CASCADE CONSTRAINTS**] [**PURGE**]
- PURGE deletes the table right away (no 'recycle bin') [Oracle]
- **CASCADE CONSTRAINTS** drops all foreign key constraints

- Example: **DROP TABLE** Student PURGE;

TRUNCATE TABLE

- Syntax: TRUNCATE **TABLE** table_name
- Empties the table
- Does not allow for rollback ⇒ no undo space needed
- Contrary to **DELETE** no triggers are executed ⇒ better performance

- Example: TRUNCATE **TABLE** Student;

Copy Table

- Creates a new table and fills it with the result of the **SELECT** statement
- Different column names can be used
- The data types are defined by the **SELECT** statement

Syntax

```
CREATE TABLE table_name [(  
    column_name [column_integrity_rule]  
    [, ...])]  
AS SELECT column_name [, column_name [, ...]] FROM table_name
```

ALTER TABLE – MODIFY

- Allows to change data type and integrity rule of columns if conditions are met:
 - Max. length can always be increased, but only decreased if NULL is allowed
 - Changes to the data type are only possible if NULL is allowed
 - NOT NULL can always be removed, but only added if no value is actually NULL

Syntax

```
ALTER TABLE table_name  
    MODIFY (column_name [data_type] [integrity_rule]);
```

ALTER TABLE – MODIFY

Example

```
ALTER TABLE Student
```

```
    MODIFY (FirstName VARCHAR2(30))
```

```
    MODIFY (LastName VARCHAR2(50) NOT NULL);
```

Name	Null?	Typ
MATNO	NOT NULL	NUMBER(10)
FIRSTNAME		VARCHAR2(30)
LASTNAME	NOT NULL	VARCHAR2(50)
DATEOFBIRTH		DATE
ISACTIVE	NOT NULL	CHAR(1)

ALTER TABLE – ADD

- Allows to add columns to a table
- Table names need to be unique

Syntax

```
ALTER TABLE table_name  
ADD (column_name data_type [default_expr] [integrity_rule]);
```

ALTER TABLE – ADD

Example

ALTER TABLE Student

```
ADD (StartDate DATE DEFAULT TO_DATE('01.10.2018', 'DD.  
MM.YYYY') NOT NULL,  
Country VARCHAR2(30));
```

Name	Null?	Typ
MATNO	NOT NULL	NUMBER(10)
FIRSTNAME		VARCHAR2(30)
LASTNAME	NOT NULL	VARCHAR2(50)
DATEOFBIRTH		DATE
ISACTIVE	NOT NULL	CHAR(1)
STARTDATE	NOT NULL	DATE
COUNTRY		VARCHAR2(30)

ALTER TABLE – DROP

- Allows to remove columns from a table

Syntax

```
ALTER TABLE table_name DROP COLUMN column_name;  
ALTER TABLE table_name DROP (column_name [, column_name,  
...]);
```

Example

```
ALTER TABLE Student DROP (StartDate, Country);
```

CREATE SYNONYM

- Creates an alias for a table to shorten the access path (usually user_space.table_name)
- Maintained centrally by the DBA

Syntax

```
CREATE [PUBLIC] SYNONYM synonym_name FOR TABLE
table_name
```

Data Manipulation Language (DML)

- Contains commands used to
 - **INSERT** data rows
 - **UPDATE** data rows
 - **DELETE** data rows
- Operates on existing tables

INSERT INTO

- Allows to insert data rows into a table
- Column names can be omitted if the value order is correct
- For missing values NULL is inserted – if NULL is not allowed an error is raised

Syntax

```
INSERT INTO table_name [(column_name_1, ...)]  
VALUES (value_1, ...);
```

INSERT INTO

Example

```
INSERT INTO Student  
VALUES (1234, 'Max', 'Muster',  
        TO_DATE('01.01.1990', 'dd.mm.yyyy'), '1');
```

Example

```
INSERT INTO Student (MatNo, LastName)  
VALUES (2345, 'Mayer');
```

INSERT INTO – Parameters

- Opens dialog for values in SQL Developer (cf. prepared statements)

Syntax

```
INSERT INTO table_name [(column_name_1, ...)] VALUES (:1, ...);
```

Example

```
INSERT INTO Student (MatNo, LastName) VALUES (:1, :2);
```

BULK INSERT

- Insert multiple rows at once
- Value source is a SELECT statement
- Data types must be compatible

Syntax

```
INSERT INTO table_name [(column_name_1, ...)]  
SELECT ...
```

BULK INSERT – WITH method [Oracle]

Syntax

```
insert into Penalties (PenaltyNo, PlayerNo, PenaltyDate, Amount)
with foo as (
    select 1, 6, to_date('1980-12-08','yyyy-mm-dd'), 100 from dual
    union all
    select 2, 44, to_date('1981-05-05','yyyy-mm-dd'), 75 from dual
    union all
    select 5, 44, to_date('1980-12-08','yyyy-mm-dd'), 25 from dual)
select * from foo;
```

UPDATE

- Allows to update one or more data rows in a table
- The **WHERE** condition defines which rows are affected

Syntax

```
UPDATE table_name
    SET column_name_1 = expression | subquery
        [, column_name_2 = expression | subquery, ...]
[WHERE condition]
```

UPDATE

Example

```
UPDATE Student  
  SET IsActive = '0'  
  WHERE MatNo = 3456;
```

Example (Advanced)

```
UPDATE Student  
  SET MatNo = (SELECT MAX(MatNo) + 1 from Student)  
  Where MatNo = 2345
```

DELETE

- Allows to delete rows in a table
- Difference to TRUNCATE:
 - Triggers are executed
 - Undo space is used
 - A condition can be specified

Syntax

```
DELETE FROM table_name  
[WHERE condition]
```

DELETE

Example

```
DELETE FROM Student  
WHERE IsActive <> '1';
```

Transaction Basics

- Transactions allow us to encapsulate changes to the database
- This has multiple benefits:
 - Changes are only visible to other users/processes when they are complete
 - Imagine changes to several tables depending on each other
 - If anything goes wrong the whole block of changes can be rolled back
- There are always three steps:
 - 1 Begin Transaction
 - 2 Make 0..n changes
 - 3 Commit/Rollback Transaction (based on the successful outcome of the changes)

ACID

- **Atomicity:** All tasks of a transaction are performed or none of them are ⇒ one unit
- **Consistency:** The transaction takes the database from one consistent state to another consistent state.
- **Isolation:** The effect of a transaction is not visible to other transactions until the transaction is committed.
- **Durability:** Changes made by committed transactions are permanent (ensured by the DBMS).

Oracle Transactions

- The first edit to any row starts the implicit transaction ⇒
There is no explicit *BEGIN TRANSACTION* command
 - More options are available for PL/SQL
- Once all required changes have been made the transaction is either
 - Committed if the changes should be persisted: **COMMIT**;
 - Rolled back if the changes should be discarded: **ROLLBACK**;

Data Control Language (DCL)

- Contains commands used to
 - **GRANT** rights on database objects
 - **REVOKE** rights on database objects
- Usually a tool for the database administrator
- Different levels of granularity

GRANT

- Syntax: **GRANT <right> [ON] <target> [TO] <user>**
- Slightly different syntax for different tasks

Examples

```
GRANT EXECUTE ON process_pay TO Fred;
GRANT SELECT ON Customer TO Fred;
GRANT SELECT any table public;
```

REVOKE

- Syntax: **REVOKE <right> [ON] <target> [FROM] <user>**
- Slightly different syntax for different tasks

Examples

```
REVOKE EXECUTE ON update_pay FROM Fred;
REVOKE SELECT ON Customer FROM Fred;
REVOKE dba FROM Fred;
```

DDL Advanced

Markus Haslinger

DBI/INSY 3rd year

Agenda

1 Sequence

- Sequence

2 View

- View

3 Constraints

- Constraints

Introduction

- When using a surrogate key for a table the need for an automatically incremented index arises
- To accomplish this there is the concept of SEQUENCES
- A SEQUENCE is created independent of a table
 - It could be used by multiple tables
 - But then there is no guaranteed index order for each table, so usually you don't want to do that

SEQUENCE – Usage

- After the SEQUENCE has been created it can be used e.g. in an INSERT
- Two pseudocolumns:
 - NEXTVAL: Returns the next value of the SEQUENCE
 - CURRVAL: Returns the current (last) value of the SEQUENCE

SEQUENCE – Options

- INCREMENT BY: The stepwidth, default 1, negative values possible
- START WITH: The initial value (if omitted MINVALUE/MAXVALUE)
- NOMAXVALUE: Counter will run up to the technical max. of 10^{27}
- NOMINVALUE: Counter will run down to the technical min. of -10^{26}

SEQUENCE – Options

- CYCLE: When MAXVALUE is reached SEQUENCE starts again at the beginning
 - Opposite: NOCYLC
 - Usually CYCLE is not desirable, because index collisions can be the result
- ORDER: Generates numbers in order (but this is not guaranteed!) – opposite NOORDER
- CACHE: Creates several values at once and keeps them in memory for a minuscule performance increase – usually not a good idea

SEQUENCE – Example

Example

```
CREATE SEQUENCE Dept_DeptNo
    INCREMENT BY 10
    START WITH 50
    MAXVALUE 120
    NOCACHE
    NOCYCLE;

INSERT INTO Dept(DeptNo, DName, Loc)
    VALUES (Dept_DeptNo.NEXTVAL, 'Marketing', 'San Diego');
```

Simple Autoincrement

- If only a simple auto increment index is required a *generated identity* option can be used

Syntax & Example

```
-- Syntax: GENERATED BY DEFAULT AS IDENTITY (START WITH 1)
```

```
CREATE TABLE qname
(
    qname_id integer GENERATED BY DEFAULT AS IDENTITY (START WITH 1)
        NOT NULL PRIMARY KEY,
    qname VARCHAR2(40) NOT NULL
);
```

View

- A *view* on a specific subset of the data available in the database
 - Compare the application views of the external level
- Created as a database object (like a table)
- Encapsulates a SQL query which is hidden
 - Similar to an interface
 - Views are read-only, you cannot modify the data in the view because it is 'just' a SQL statement
- Can be used to:
 - Re-use complex queries
 - Limit the access of a user to a read-only subset
 - Provide different views from the same data

View

Syntax

```
ALTER TABLE Census
    ADD CONSTRAINT UNQ_city_state
    UNIQUE (city, state);
```

Introduction

- You've already worked with several constraints:
 - NOT NULL
 - PRIMARY KEY
 - FOREIGN KEY
- Now we introduce a couple more:
 - UNIQUE
 - ON DELETE
 - CHECK

UNIQUE

- Allows to ensure uniqueness of values in a column
- Also allows to ensure uniqueness of a value combination in several columns
- Independent from the PRIMARY KEY
- Contrary to the PRIMARY KEY, NULL values are allowed

UNIQUE – Example

Example

```
ALTER TABLE Census
    ADD CONSTRAINT UNQ_city_state
        UNIQUE (city, state);
```

ON DELETE

- An optional extension of the FOREIGN KEY constraint
- Defines what should happen to the row(s) referencing a PRIMARY KEY in another table if the referenced row is deleted
- Two options:
 - SET NULL: if the referenced row is deleted, the foreign key value is set to NULL
 - This requires the FOREIGN KEY column to *not* have a NOT NULL constraint!
 - CASCADE: if the referenced row is deleted, those rows referencing it are deleted as well

ON DELETE – Example

Example

```
CREATE TABLE Emp(  
    ...  
    DeptNo NUMBER(2) CONSTRAINT FK_Emp_Dept  
        REFERENCES Dept(DeptNo)  
        ON DELETE CASCADE  
)
```

CHECK

- Defines a condition values in a column have to fulfill
- Can use other columns in the same table
- Cannot use:
 - Columns in another table
 - SYSDATE, UID, USER, USERENV
 - CURRVAL, NEXTVAL, LEVEL, ROWNUM
 - Subqueries

CHECK – Example

Example

```
CREATE TABLE Person(  
    ...  
    Age NUMBER NOT NULL CONSTRAINT check_age  
        CHECK (Age BETWEEN 0 AND 120)  
);
```

Database Design

Markus Haslinger

DBI/INSY 5th term

Agenda

1 Introduction

- Introduction
- Example

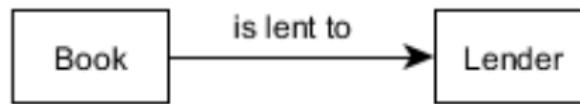
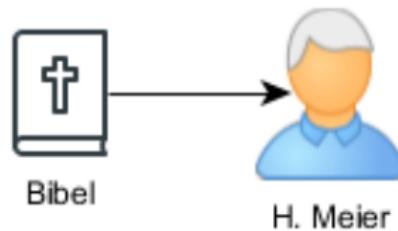
2 ERD

- Entities & Relations
- ERD

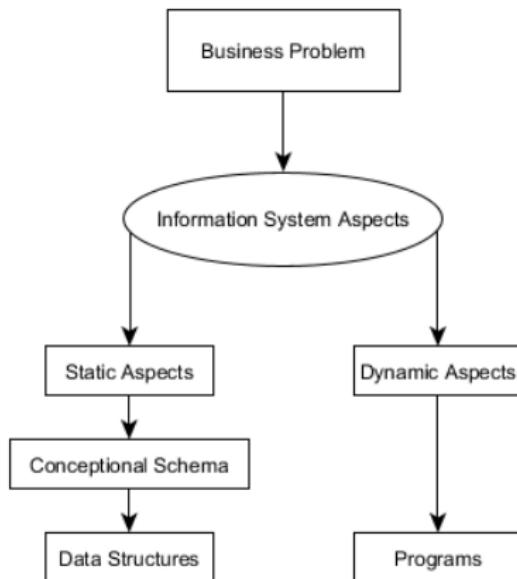
3 Relational Model

- Relational Model
- Keys
- ERD transformation

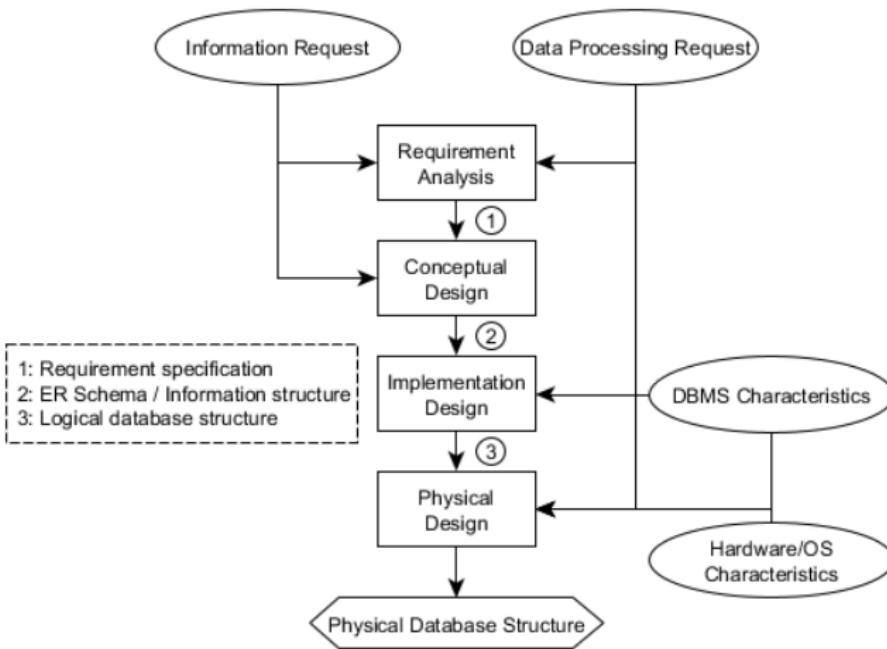
A business problem



Approaching a business problem



Database Design Stages





Sample Data

- Data about employees and currently ongoing projects are stored

PersNo	Name	City	DivNo	DivName	ProjectNo	ProjectName	Time
100	Susi	Linz	20	Org	23c	Accounting	40
101	Sabine	Wels	20	Org	23y 30a	OpenPos List Exchange	50 5
102	Franz	Steyr	21	SW Dev.	23x 30a	Account Balancing Exchange	30 10
103	Otto	Linz	22	Testing	23x 23y 30a	Account Balancing OpenPos List Exchange	10 10 10



Sample Data – Issues

- Multiple records for the DivNo and DivName relation
- Multiple records for the ProjNo and ProjName relation

- How to add a new row (e.g. new division)?
- If data row for person 100 is deleted Division 23c information is lost

Sample Data – Restructuring

ProjNo	ProjName
23c	Accounting
23y	OpenPos List
23x	Account Balancing
30a	Exchange

PersNo	Name	City	DivNo
100	Susi	Linz	20
101	Sabine	Wels	20
102	Franz	Steyr	21
103	Otto	Linz	22

PersNo	ProjNo	Time
100	23c	40
101	23y	50
101	30a	5
102	23x	30
102	30a	10
103	23x	10
103	23y	10
103	30a	10

Database Design Cornerstones

- Goal: Creation of a *correct, complete and consistent* representation of a real world domain
- In the end an application must be able to use it
- Difficult to describe in prose \Rightarrow formal methods required
- All kinds of information can be structured and managed
 - Only by combining (raw) data with semantics structured information is gained

Entities

Definition

Entity: something that exists apart from other things, having its own independent existence¹

- An entity in the context of databases is an individual and *identifiable* exemplar of things, persons or concepts
- A relation between entities can be considered an entity
- Examples:
 - John Doe
 - The IEEE Proceedings Issue from July 2018
 - Book with ISBN 1234546789
 - The marriage between persons A & B

¹<https://dictionary.cambridge.org/dictionary/english/entity>, 2018-08-26

Entity Sets

- We cannot and do not represent each entity individually
- Entities are grouped together with others which share the same (relevant) properties ⇒ these groups are sets of entities
- Examples:
 - Employees of a company
 - Books in a library
 - All marriages in a county

Entity Sets/Types

- Disjunct entity sets: each entity belongs to exactly one set
- Overlapping/Joined entity sets: entities may belong to several sets at once (e.g. a student can at the same time work as tutor)

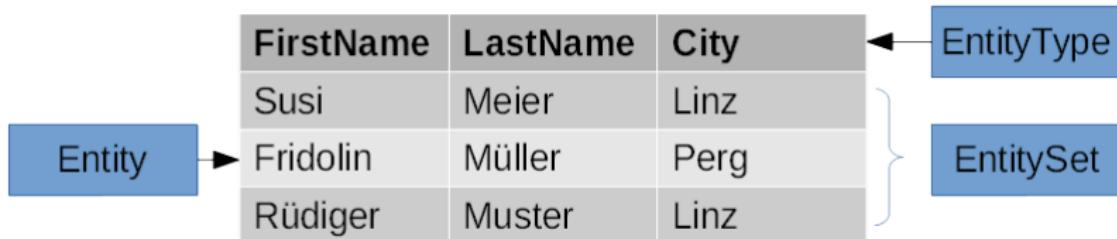
- Fundamental entities: describe a situation/facts independently of other entities
- Dependant entities: only provide meaningful data in conjunction with data from the related/parent entity

Entity Type

Definition

EntityType: describes an entity set which elements have the same (relevant) properties.

Every EntityType is defined by its name and properties.



Relations (Associations)

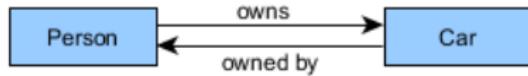
- Syntax: association(ES1,ES2)
- An association between entity sets ES1 & ES2 defines how many entities from ES2 are (possibly) associated with one entity from ES1
- These definitions are often quite different, thus min and max values are declared

Relation Types

Association(ES1,ES2)	ES2 entities assigned to ES1
1: simple association	exactly one (1)
c: conditional association	none or one (0,1)
m: multiple associations	at least one (≥ 1)
mc: multiple, conditional association	none, one or multiple (≥ 0)

Relations

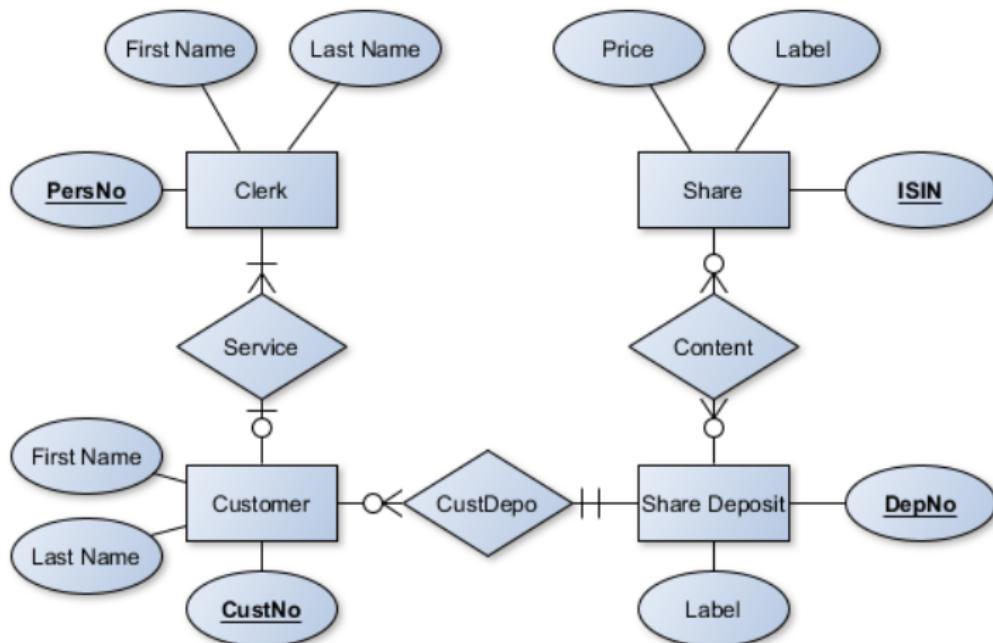
- Combining an association (ES1,ES2) with its counter association (ES2,ES1) produces the relation between the two entity sets



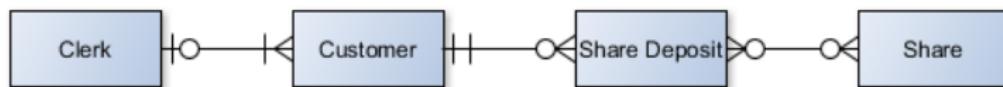
Relations

Entity Set 1	Entity Set 2	Relation Type	Relation
Wife	Husband	1 - 1	Marriage
Division	Personnel	c - 1	Division Head
Personnel	Division	m - 1	Working at a division
Cell	Body	mc - 1	Parts of a Body
Women	Men	c - c	Marriage (pol. incor.)
People	pol. Party	m - c	Party member
Employee	Employee	mc - c	Is boss of
Place	Place	m - m	Distance
Lecture	Student	mc - m	Enrollment
People	People	mc - mc	Friendships

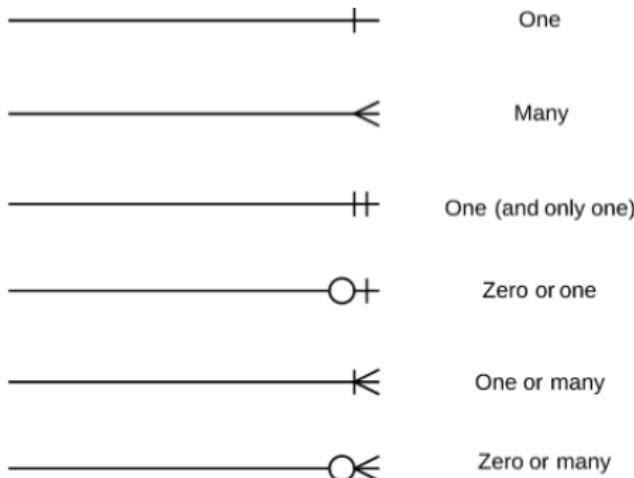
ERD – Example Stock Shares



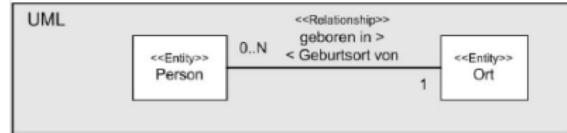
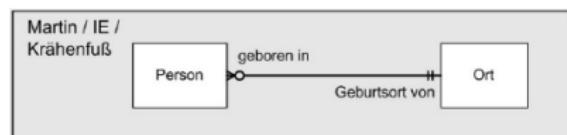
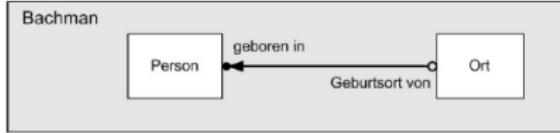
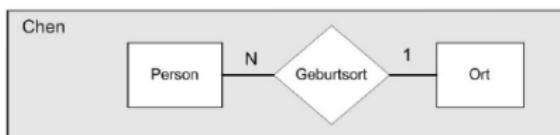
ERD – Example Stock Shares



Relation notations (Martin)



Notations



Domains

- A Domain represents a set of valid/allowed values of a specific data type
- Examples:
 - Whole numbers (int)
 - All ASCII characters
 - Weekdays ('Mon', 'Tue',...)
 - 1..23
 - Strings of length 10

Attributes & Values

- Every entity has a number of attributes
- For every attribute only values of a specific domain are allowed

Person		Attribute
Name	Eye Color	Age
Maier	Brown	12
Bauer	Green	24
...

Values

Domain

Formatted Values

- If values can only come from a specific domain (e.g. days of the week) they are called 'formatted'
- If no specific domain is required or a domain cannot be defined they are called 'unformatted'
 - Example: A 'special characteristics' attribute could contain a 'freetext' value like 'scar above the left eye'
 - Note: the data type is still specified ⇒ domains can sometimes only be enforced by organizational rules

Naming

- Within one model (possibly even within one company) each entity/relation should be unique
- Entities:
 - Use singular
 - Avoid abbreviations
 - Use business terms, not technical terms ⇒ target audience is the business
- Relations:
 - Verbs
 - Remember: a relation has two directions (associations)

Naming

	Synonym	Homonym	Äquipollenzen	Vagheiten	falsche Bezeichner
Intension (Inhalt der Begriffe)					
Zeichenebene (Bezeichner)					
Extension (Umfang der Begriffe)					
Beispiel	Firma, Unternehmen, Unternehmung	Wertschöpfungskosten	Zweckaufwand, Grundkosten	SGF, SGE	relativer Marktanteil
Aktion	Kontrollieren	Beseitigen	Aufdecken	Klären	Ersetzen

Definitions

- To prevent misunderstandings an exact definition of entities and relations is paramount
- Moderating this communication process between business and technicians is the task of the designer

- 'A customer is anyone who buys something from us'
 - 'anyone': can employees also be customers?
 - 'something': what if we sell an old office building?
 - 'buys': when is something sold, at shipment or payment?
 - If someone hasn't bought anything yet but is interested in our products, is he already a customer?

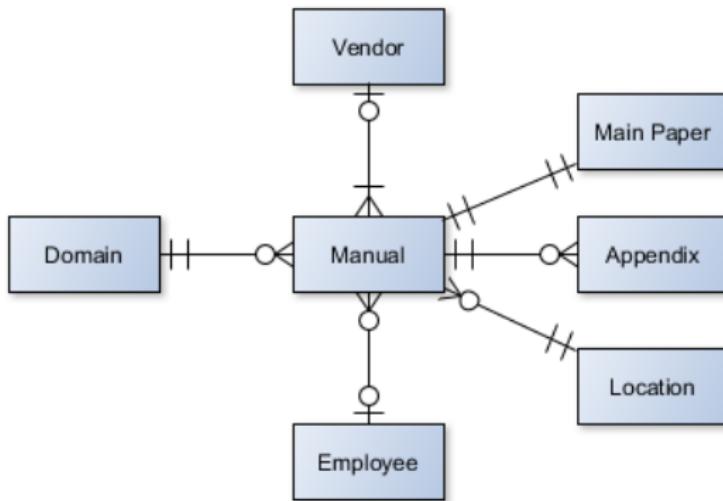
Requirements for Definitions

- Be clear and concise
- Make sure the model is complete
- Be precise
- Be consistent
 - Example: customer vs. reseller

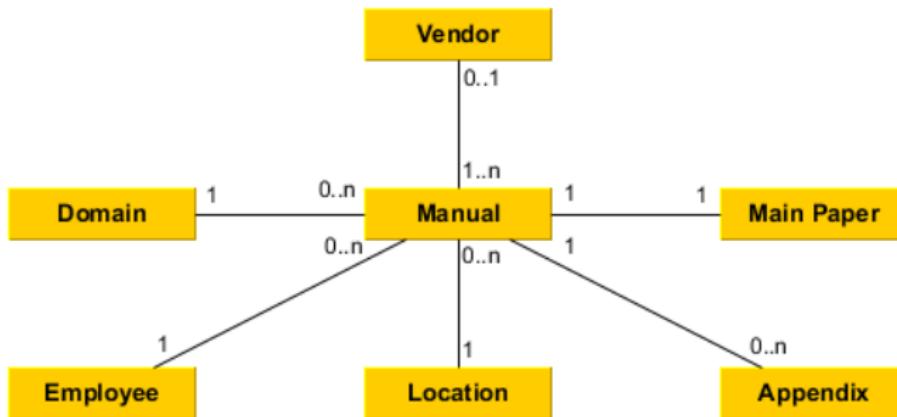
Example: a technical manual

- A manual consists of a main paper and possibly several appendices
- A manual relates to a certain (technical) domain
- Each manual is stored at a specific location
- A manual can be borrowed by employees
- A manual is shipped by a vendor

Example: a technical manual



Example: a technical manual



The relational model

- Transition from the abstract level of entity sets to the more data related level of relations
 - Be careful: relation between entities != (table)relation
- Every entity set is represented by a relation and set of attributes within
- Principles:
 - Differentiate between logical and physical aspects
 - Structural simplicity
 - tables are easy to understand
 - tables are easy to talk about
 - Set oriented processing

Relational schema

- If a finite set of attributes (A_1, A_2, \dots, A_n) has been defined
- This set is called a relational schema R which consists of these attributes
- $R = \{A_1, A_2, \dots, A_n\}$

Domain

- Each attribute is based on a non-empty set $D_i (1 \leq i \leq n)$
 - This set is the range of attribute A_i and called $\text{Dom}(A_i)$
 - The domain D is defined as the union of all attribute ranges
-
- $D = D_1 \cup D_2 \cup \dots \cup D_n$

Relation & Tuple

- We define a relation r of the relational schema R as a finite set of assignments (t_1, t_2, \dots, t_k) of R in D
- The individual assignments t_k are called tuple or n-tuple
- For each tuple the value in a specific row A_i , called $t(A_i)$ has to be an element of the range of A_i

Relation

- In addition to the definition of the assignment of tuples to relations a relation can be defined in more general terms
- Based on n ranges D_i , which are not necessarily disjoint, a relation R is a subset of the following cartesian product:
 - $R \subset D_1 \times D_2 \times \dots \times D_n$
- Consider:
 - The subset does not contain duplicates
 - Every row of a relation is such a tuple

Keys – Introduction

- Keys are a fundamental concept of the relational model
- They are a way of locating specific tuples during a table search
 - This could be achieved by other means as well...
 - ...but keys represent a reliable way by providing *uniqueness*
- Remember that by definition there must not be two identical rows
 - All tuples in a relation have to have (some) different values
 - Are there situations in which this rule doesn't apply?
- Usually a subset of attributes exists which fulfills this requirement (no two tuples with the exact same combination of values)

Superkeys

- For each attribute subset (SK), $t_1(SK) \neq t_2(SK)$ is true for every two tuples t_1 & t_2
- These attribute subsets are called *Superkey* of a relational schema R
 - They consist of one or more attributes
 - The value combination is unique
 - They allow for an explicit access of a single, specific row
- A Superkey can contain redundant attributes \Rightarrow a *Key* has a better usability

Keys

- A key K of a relational schema R is a superkey of R with one additional property: if one attribute is excluded the remaining set K' is no longer a superkey
- Thus, a key fulfills the following requirements:
 - Uniqueness
 - Minimal

Keys

Example

Relation Students: $S = \{SVNR, Name, Age\}$

$\{SVNR, Name, Age\}$: superkey, but no key

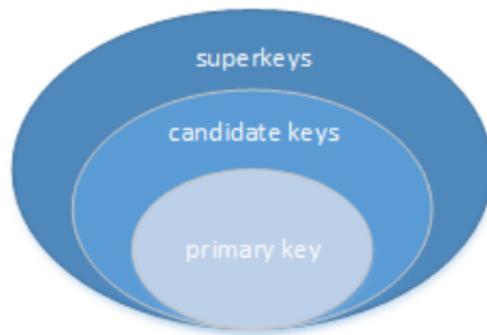
$\{SVNR\}$: superkey *and* key

Primary Keys

- Every table can contain multiple keys which are called *candidate keys*
- The *primary key* is one selected candidate key
 - Consider clustering
- Those candidate keys not selected for the primary key are called *alternate keys*
- Attributes of the primary key must not be NULL(able)!

Primary Keys

- $\text{Superkeys} \supseteq \text{candidate keys}$
- $\text{primary key} \in \text{candidate keys}$



Foreign Keys

- A set of attributes which represent the primary key of another table are called *foreign key*
- Two conditions must be true:
 - The foreign key and the corresponding primary key are defined in the same domain
 - Foreign key values are either null (if allowed) or have to equal the primary key values
- A relation can contain multiple foreign keys
- Foreign keys can also be part of a primary key

Foreign Keys

Example

Relation Class: $C = \{ClassID, \dots\}$

Relation Student: $S = \{ClassID, StudentNo, Name, \dots\}$

- Class:

- Primary Key: ClassID
- Foreign Key: –

- Student:

- Primary Key: (ClassID, StudentNo)
- Foreign Key: ClassID

ERD Transformation

- During the design stage the first step for the database architecture is creating an ERD diagram
- Finally, the database has to be populated with tables representing the identified entity sets
- To achieve this goal, the ERD is transformed into a relational model

ERD Transformation – Step 1

- The first step is to ensure that every relation has a primary key

Example

Entity set Employees → Employee(EmployeeNo,...)

Entity set Projects → Project(ProjectNo,...)

Entity set Customers → Customer(CustNo,...)

ERD Transformation – Step 2

- The second step is to represent 1:n relations with foreign keys
- The primary key of the master relation is added as foreign key attribute in the dependant relation

Example

Relation Project-Customer → Project(. . . , CustNo, . . .)
▷ Why not Customer(. . . , ProjectNo, . . .)?



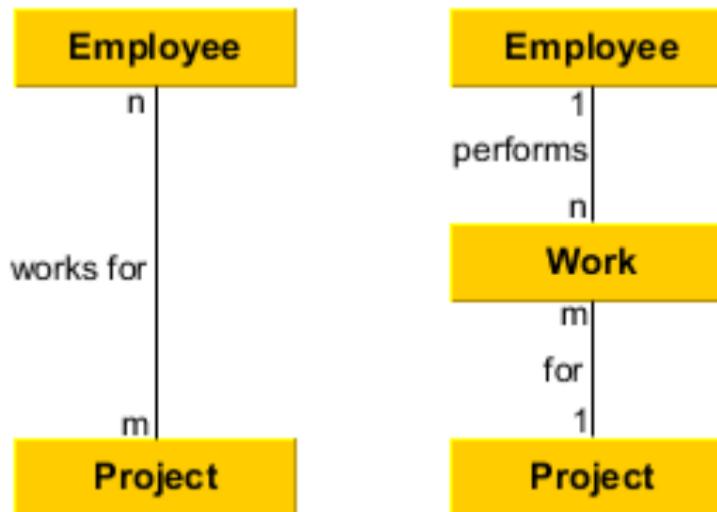
ERD Transformation – Step 3

- The third step is to represent m:n relations with assoziation tables
- The new relation contains the primary key of both related relations
- The primary key of the new relation is either a combination of the foreign keys or a new, virtual surrogate key

Example

Relation Project-Employee → Work(. . . , ProjNo, EmpNo, . . .)

ERD Transformation – Step 3



ERD Transformation – Step 4

- The fourth step is to represent properties as attributes (columns)
- The property values are then put as values into the columns
 - Remember: domain

Example

Employee(. . . , FirstName, LastName, Address, ZIP, City, . . .)

Project(. . . , Name, StartDate, EndDate, . . .)

Customer(. . . , FirstName, LastName, Address, Revenue, . . .)

ERD Transformation – Steps 5-7

- 5 Every relation becomes a foreign key
- 6 Normalization
- 7 Aggregation

Database Concurrency

Markus Haslinger

DBI/INSY

Agenda

- 1 Introduction**
 - Introduction
 - Lock Motivation
- 2 Locks**
 - Locks
 - Deadlocks
- 3 Consistency**
 - ACID
 - Excursus: Oracle Consistent Read

Introduction

- One of the major benefits of a DBMS compared to a file based data storage is multi user support
- However, with concurrent use several issues arise
- It is necessary to control how shared resources are accessed to prevent problems
 - Like inconsistencies, incorrect data returned,...
- To this end we use: Transactions, a Logical Unit of Work (LUW) and Locks

Reasons for a Transaction failure

- System Crash (hardware, software, network)
- Transaction error: some operations failed (overflow, div by zero,...)
- Logical operation impossible (required data is missing, constraints are hurt,...)
- Concurrency control: The component responsible for supervising concurrent operations has to abort a transaction (deadlock)

Logical Unit of Work

- Remember: a transaction leads the database from one consistent state to another

A Banking Transaction

```
UPDATE Acc SET Balance = Balance - 500 WHERE AccNo = 4711;  
UPDATE Acc SET Balance = Balance + 500 WHERE AccNo = 1174;
```

- If one of the operations above fails the bank is either loosing money or betraying a customer
- So it is paramount that related operations are executed either all or none
- This principle we call a Logical Unit of Work

Ensuring consistency

- During the single steps of a Logical Unit of Work the database goes through inconsistent states
- But, at the end the state has to be consistent again
- A transaction helps us to ensure that
 - By encapsulating related operations
 - By allowing us to *commit* or *rollback* related operations as one (atomic) unit
- Transactions ensure the 'all or nothing' principle we need
 - In a way that takes into account the issues discussed previously
 - So you don't have to care about e.g. a power outage, the DBMS will ensure that ongoing transactions are cancelled/rolled back

Motivation for Locks

- The transaction concept discussed before allows to ensure consistency
- But it is not sufficient to deal with all issues of concurrency
- In a typical DBMS we have a multi user setup \Rightarrow multiple transactions are running at the same time
- These transactions (might) access the same data records
- So we need a way to ensure every transaction can either complete successfully or has to be terminated to allow another to succeed
- This solution are different kinds of locks

Problem – Lost Update

Time	Transaction A	Price	Transaction B
1	Reads price value	100	-
2	Analyzes price value	100	Reads price value
3	Increases price value by 10%	110	Analyzes price value
4	-	120	Increases price value by 20%

- Both transactions want to increase the price by a certain amount
- They both run at approx. the same time
- But because Transaction B starts a little later the effect of Transaction A's update is lost (120 instead of 132)
- Despite all transactions completing successfully, the database is in a logically inconsistent state

Problem – Lost Update – Pessimistic Solution

- Assumption: Transaction B bases its calculation on invalid data, because A is currently reading
- Thus, the DBMS ensures, that B has to wait until A is done
- The value (row or table) has to be *locked* for everyone else
- How does the DBMS know that A actually wants to edit the data and not just read it?
 - An additional hint from the developer (e.g. 'for update') is required

Problem – Lost Update – Optimistic Solution

- Assumption: Instead of assuming the worst case we expect everything to work out in most cases and only act if we actually detect a problem
- Right before executing the actual update statement the originally read value is compared to the current value in the database
- If these values differ we rollback the current transaction
- So to increase overall performance we will sometimes sacrifice a transaction which would have successfully completed with the pessimistic strategy
- A good check attribute is one updated every time a row is modified with the current timestamp

Problem – Uncommitted Dependency

Time	Transaction A	Price	Transaction B
1	-	100	-
2	Updates price to 120	120	-
3	Throws an exception	120	Reads the price value
4	Performs rollback	100	Analyzes the value
5	-	100	Still thinks value is 120!

- Transaction B does not realize the value it read has been rolled back in the meantime
- Reading updated values before these are committed is called a *dirty read*
- This can be prevented by setting the transaction level to something like 'read committed'

Problem – Inconsistent Analysis

Time	Transaction A	Acc1	Acc2	Transaction B
1	-	100	200	-
2	Reads Acc1 value	100	200	-
3	-	100	200	Reads Acc2 value
4	-	100	190	Updates Acc2 value
5	-	100	190	Reads Acc1 value
6	-	110	190	Updates Acc1 value
7	-	110	190	Commits changes
8	Reads Acc2 value	110	190	-
9	Returns sum 290!	110	190	-

- Transaction A returns a wrong sum, despite only reading committed values

Problem – Inconsistent Analysis

- To prevent this issue all rows (tables) A needs to read would have to be locked for editing transactions
- Imagine the operations A performs are more complicated than a simple sum
- Yet locking all tables touched during this transaction will block most of the normal business operation
- So it might be prudent to perform such an operation (reporting!) outside of business hours

Problem – Inconsistent Analysis – Phantom Problem

- A special case is the so called *Phantom Problem*
- It occurs when during a read transaction other transaction add or remove rows
- For example the same SQL statement performing a COUNT based on some criteria might return different results during the same transaction
- See <https://vladmihalcea.com/phantom-read/>¹ for a phantom read example

¹last accessed 2019-05-01

Lock Basics

- Basic idea: objects a transaction A needs are locked until A is complete so that other transactions cannot modify/access these objects in the meantime
- This ensures that every transaction can operate on 'stable', reliable data
- Be careful: No database access without a lock!

Lock Types

- Exclusive Lock (X Lock): If transaction A locks the row R (object O) with an X lock then all locking attempts of other transactions are denied
 - No matter if the other transaction requests a X lock or S lock
 - The other transaction has to wait
- Shared Lock (S Lock): If transaction A holds a S lock on row R (object O) then X lock requests of other transactions are denied
 - But multiple S locks on the same object are allowed!
- A locked object is also called a *critical resource*
 - cf. 'critical region' in application code

Lock Compatibility Matrix

Lock Request	Object State		
	X Lock	S Lock	No Lock
X Lock	X	X	✓
S Lock	X	✓	✓

- Above table shows when a lock can be granted by the DBMS
- An extensive use of X locks degrades performance considerably
- We want to lock as little as possible while ensuring consistency
- Most DBMS create locks automatically:
 - Read creates a S lock
 - Update creates a X lock

Lock Release

- We now know why and when locks are created
- But when is it safe to remove them again?
- We usually cannot release locks after a successful read or update
- Instead we have to wait for the LUW to be complete
- Otherwise e.g. a later rollback could lead to problems once again

- Waiting for the LUW to complete has other transactions waiting ⇒ many long lived locks degrade performance

Optimizing Locking

- We established that locks are needed to avoid problems
- X locks are worse for performance than S locks
- Locks that are held for a long time are worse than those held shortly
- To not have our database slow down too much we need to come up with a *locking strategy*

Locking Strategy

- When planning a transaction always keep the execution order in mind
- Do not perform updates before all reads are done (except if absolutely necessary)
 - S locks before X locks
- Perform single table operations before multi table ones
 - Locks less objects right away
- Keep transactions as small as possible
 - Releases locks as soon as possible
- But: *Performance always comes second to correctness of a business process!*

Locking Strategy – Example

- At the beginning of a lengthy operation a timestamp is updated in the application wide configuration table
- Updating the configuration table creates a X lock
- All transactions requiring the configuration have to wait
 - Even those just reading which is very common
- This X lock is now held for a very long time, blocking most other operations
- Performing the timestamp update at the end of the transaction would lead to pretty much the same result with a much smaller performance impact

Problem Lost Update – Solved?

- TA gets X lock at t1 and reads price
- TB request X lock at t2, but has to wait
- TA changes value at t3, commits and releases lock
- TB receives X lock at t4 and reads correct value
- ...
- Operations successful!

Problem Uncommitted Dependency – Solved?

- Assumption: 'for update' hint supplied
- TA receives X lock at t2 (update) and changes the price
- TB requests S lock at t3 but has to wait
- TA rolls back and releases lock at t4
- TB receives S lock at t5 and reads correct price
- ...
- Operation (TB) successful!

Problem Inconsistent Analysis – Solved?

- Assumption: 'for update' hint supplied; table, not row locking active
- TA receives S lock on the Acc table (reads Acc1 row) at t2
- TB requests X lock on the Acc table but has to wait at t3
- TA finishes reading and processing at t4-t9
- TB receives X lock at t10
- ...
- Operations successful!

Problems – Conclusion

- We need to plan & execute transactions with care
 - 'for update' flag supplied
 - Table instead of row locking active
- The total time the operations took increased in all cases
- But all problems have been solved thanks to locks

- It seems that locks are a silver bullet – what do you think?

Deadlocks

- Surprisingly locks are not without drawbacks
- By introducing locks to solve our problems we created a new one: *deadlocks*
- Now we might run into situations where all transactions are waiting forever on each other

T	Transaction A		Price	Transaction B	
1	SELECT	S Lock	100	-	-
2	-	S Lock	100	SELECT	S Lock
3	UPDATE 10%	Req. X Lock	100 (110)	-	S Lock
4	-	Waiting	100 (132)	UPDATE 20%	Req. X Lock
5	-	Waiting	100 (132)	-	Waiting

Two Phase Locking

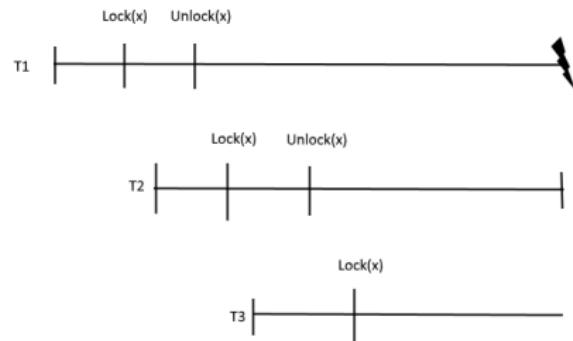
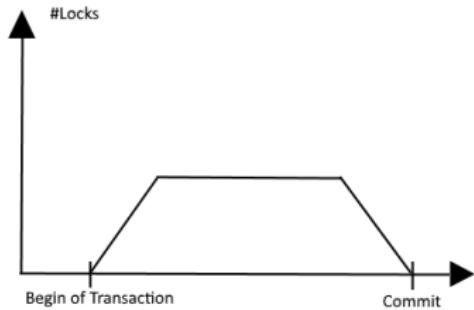


Figure: Cascading Rollback

Figure: Locking Profile

Two Phase Locking

- As the name suggests: there are two phases
- Locks are requested as required (growing phase)
- Locks are released when no longer required (shrinking phase)
- So another transaction can use objects previously locked despite the first transaction not being complete
- If the first transaction fails later transactions need to be rolled back as well (cascading rollback)
 - Because this could lead to a dirty read again

Strict Two Phase Locking

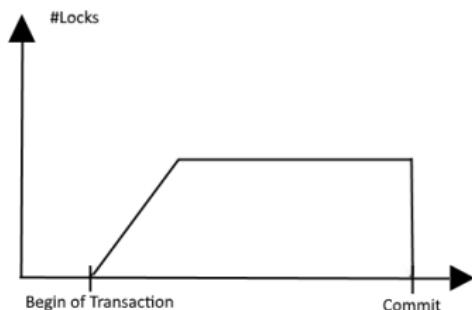


Figure: Strict 2P Locking

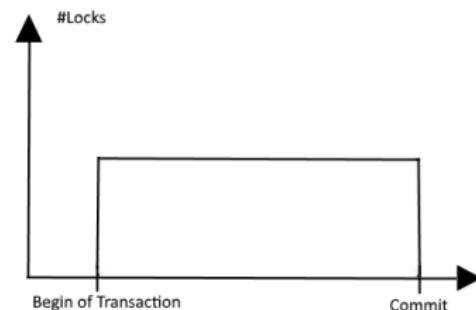


Figure: Preclaiming Locking

Strict Two Phase Locking

- To prevent dirty reads we can perform a strict two phase locking
- All locks are released at the end of the transaction \Rightarrow no dependent transactions, no cascading rollbacks
- No deadlocks can occur if all transactions:
 - 1 Lock all required objects right at the start
 - 2 Abort if this is not possible
- But to perform preclaiming all required objects need to be known at the start which is often not possible
- Also all transactions would have to be independent of each other (no logical order)

Transaction Scheduling

- This strategy allows only independent transactions to run concurrently
- Independent transactions do not use the same data objects
- Similar to preclaiming it is necessary to know all required objects right at the beginning
- That's hard to determine, so often a pessimistic approach is taken:
 - Whole tables instead of actually needed rows are locked
 - Transactions which could in fact run in parallel aren't allowed to do so

Timestamp Strategy

- Each transaction receives a timestamp (transaction start time) as a unique identifier
- This timestamp also determines the priority
 - Older or younger transactions receive higher priority
- Conflicts are resolved by a combination of:
 - Waiting
 - Restarting (Rollback & Retry)

Timestamp Strategy – Conflict Resolution

- Situation: Transaction A requests an object already locked by transaction B
- Wait-Die:
 - If A is older than B then A waits
 - Else A is restarted
- Wound-Wait:
 - If A is older than B then B is restarted
 - Else A waits

Timestamp Strategy – Wait-Die Example

T	Transaction A (TS1)	Price	Transaction B (TS2)
1	Reads Price; S Lock	100	-
2	-	100	Reads Price; S Lock
3	Update Price; X Lock wait	100	-
4	-	100	Update Price; Restarts
5	Gets Lock; done	110	-
6	-	110	Starts again with TS6

- A has to wait because it is older
- When B requests a X lock as well it has to restarted
- This allows A to continue
- Then B starts again for the next attempt

Timestamp Strategy – Wound-Wait Example

T	Transaction A (TS1)	Price	Transaction B (TS2)
1	Reads Price; S Lock	100	-
2	-	100	Reads Price; S Lock
3	Update Price; Req. X Lock	100	forced restart
4	done	110	-
5	-	110	Starts again with TS5

- A requests an X lock
- B blocks this attempt, but A is older, so B is restarted
- This allows A to continue
- Then B starts again for the next attempt
- Both strategies prefer the older transaction, but Wound-Wait even more so leading to a better performance

Timestamp Strategy – Plausibility Considerations

- Every transaction has a specific priority
- These with lower priority are sacrificed
- Thus, no cyclic waiting is possible \Rightarrow no deadlock possible

Timeout Strategy

- Each transaction gets a max. time allotted
- If this time runs out it is terminated
 - Because the system assumes that it is deadlocked
- Allows for deadlocks to exist
- Resolves existing deadlocks
- Problems:
 - Depending on the timeout some (long) transaction may never be executed
 - A short timeout can lead to a lot of restarts

ACID Properties

- A transaction is an abstraction for a sequence of operations representing an occurrence in the real world
- Transactions must not reduce the correctness of the database
- To ensure this they have to comply to four rules:
 - Atomicity
 - Consistency
 - Isolation
 - Durability

ACID Properties

- Atomicity:
 - A transaction is executed in total or not at all
- Consistency:
 - A transaction has to ensure logical and physical consistency
 - Logical:
 - The values represent the real world
 - The values comply with semantic rules (domains, keys, . . .)
 - Physical:
 - Data is stored correctly (on disk)

ACID Properties

- Isolation:

- Every transaction operates independent of other Transactions
- The DBMS ensure a 'logical single user mode'

- Durability:

- Once a transaction has completed (commit successful) the changes are reliably and permanently persisted

Oracle Consistent Read – Motivation

- Remember that a simple SELECT without a lock can lead to inconsistencies already
- Imagine the following situation:
 - A long running process is reading data for a big report
 - In the meanwhile another process is changing data
 - Because they can run concurrently the first process might at the end have read some old and some updated values
 - That is not a consistent basis to make decisions upon
- This is called a *dirty read*
- It allows for more concurrency and higher performance at the risk of (partially) incorrect data

Oracle Consistent Read – Motivation

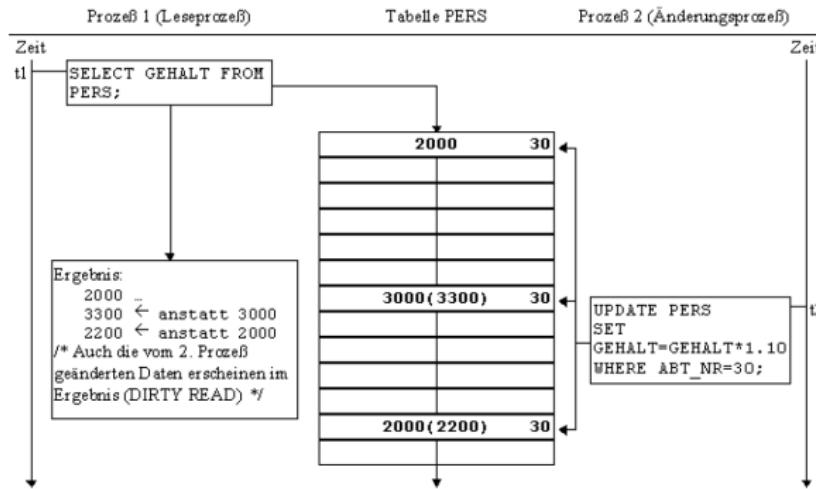


Figure: Inconsistent Read²

ANSI Process

- The problem is solved by creating a S lock on a table when reading
 - The reading transaction has to wait until no editing transaction uses the table any more
 - While the reading transaction is active no changes to the table are possible
- This is the recommended default behavior: secure but slow
 - Imagine a bank with a table for accounts
 - Would someone run a report over this table during the day all (millions) of transactions would be blocked
- So reports should either be created outside of office hours or a different database design is necessary (mirrors,...)

Oracle Read-Consistency-Model

- Oracle decided not to comply with the standard
 - They created a process which should allow for:
 - Correct read results and
 - High amount of concurrency
 - To achieve this they are using so called Rollback-Segments
 - These are usually used for rollbacks, but in this case they provide the last consistent database state
 - Thus, no table locking for read transactions is necessary
 - This allows (theoretically) for any number of read and write transactions working on the same table at the same time
 - However, performance issues occur in case of a collision

Oracle Read-Consistency-Model

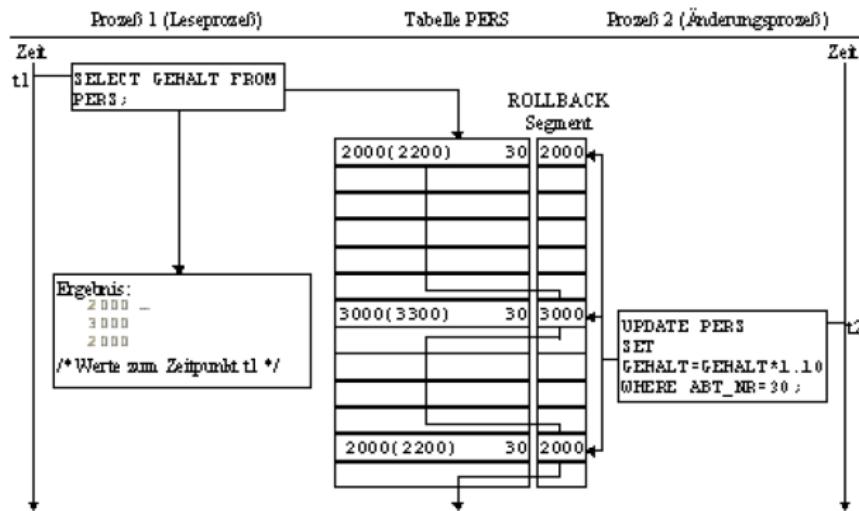


Figure: Consistent Read³

3 © T. Stütz 2001

Java Persistence API

Markus Haslinger

DBI/INSY 4th year

Agenda

1 Introduction

- ORM
- JPA

2 Basics

- Model Definition & Inheritance
- Relations

3 CRUD & Queries

- Transactions & CRUD
- Queries

ORM

- **ORM = Object Relational Mapper**
- In the (relational) database entities are stored in tables (as rows)
 - They may even be split across several tables
- In our application entities exist as objects
 - Assuming OOP
- The goal of an ORM is to map between those two worlds

Why ORM?

- It is not absolutely necessary to use an ORM
- All programming languages allow a more or less raw access to a relational database
 - For Java that can be done using JDBC (almost) directly
- Some people¹ argue that ORMs cause insufferable performance bottlenecks and everything should be done manually instead

¹Often those who also like C and vi.

No ORM – Insert a Person

Insert entity with JDBC

```
public int insertPerson(Person person) {
    final String stmt = "insert into person values (?,?,?,?,?,?)";
    return jdbcTemplate.execute(stmt, new PreparedStatementCallback<Integer>() {
        @Override
        public Integer doInPreparedStatement(PreparedStatement ps) throws SQLException,
            DataAccessException {
            ps.setString(1, person.getSSN());
            ps.setDate(2, Date.valueOf(person.getDateOfBirth()));
            ps.setString(3, person.getFirstName());
            ps.setString(4, person.getLastName());
            ps.setBoolean(5, person.isAwesome());
            ps.setFloat(6, person.getAwesomeness());
            ps.setBigDecimal(7, person.getWealth());
            return ps.executeUpdate();
        }
    });
}
```

No ORM – Load Person entities

Load entities with JDBC

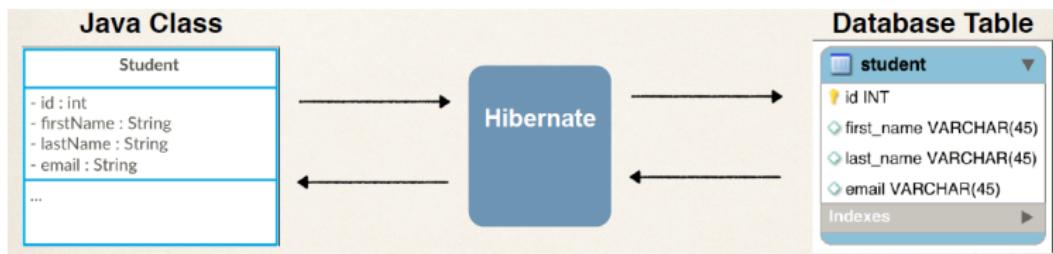
```
public List<Person> getAllPeople() {  
    return jdbcTemplate.query("select * from person", (rs, rowNum) -> new Person(  
        rs.getString("SSN"),  
        rs.getDate("date_of_birth").toLocalDate(),  
        rs.getString("first_name"),  
        rs.getString("last_name"),  
        rs.getBoolean("is_awesome"),  
        rs.getFloat("awesomeness"),  
        rs.getBigDecimal("wealth")  
    ));  
}
```

- Can you imagine writing (and maintaining!) such queries and statements all day long?

What is an ORM doing differently

- Fact is: some mapping like in the past two examples has to happen
- But: this is a very linear process which lends itself towards automation
- ⇒ that is the goal of an ORM
- We define our entities as objects (classes)
 - To help the ORM understand our intent we often have to give it some hints
- Then the ORM analyzes the code and generates those mapping functions for us (usually in memory)
- This can save a lot of labor and helps to avoid errors (like assigning a wrong index somewhere)

ORM Process



- Hibernate is an example for an ORM

Source: Fadatare R., <https://dzone.com/articles/what-is-the-difference-between-hibernate-and-spring-1>, 2019-11-01

Downsides of an ORM

- Of course ORMs are not perfect
- Common downsides are:
 - The ORM has to analyze the code and generate mapping logic
⇒ this increases startup time
 - The ORM may make mistakes ⇒ It can be hard to figure out where the problem is (e.g. DB Tracing necessary)
 - The ORM may not find an optimal solution ⇒ Queries run slower
 - You have to define the model twice: in the database and in the code ⇒ those two models have to be kept in sync

Ways to deal with ORM shortcomings

- Startup time is not always a big concern, e.g. for a server application which should restart maybe every couple weeks or months
- It helps to know the quirks of a specific ORM to quickly spot the problem based on (often not very helpful) error messages
- The database has to have proper constraints in place, this will prevent the ORM from generating inconsistent data by mistake
- For slow queries only manual (low level) functions can be created if other optimizations fail
- Many ORMs provide ways to sync the database and the code model

Syncing the models

- Tools to make the process of keeping database and code models in sync easier (and less prone for errors):
 - Database-First: Generate entity classes based on existing DB model
 - Code-First: Generate database tables and constraints based on existing object model
 - Migrations: Apply changes in either model to the other (similar to version control)
- Be aware that many DBAs frown upon such tools and will not grant an application the rights to make changes to the database
 - e.g. Script Branding

Performance of generated Queries

- Queries generated by an ORM are often very verbose and huge
- This *can* make them slower than a query created by a human
- But: because an ORM is often *provider aware* it may have generated the query in a way that is optimal for the SQL interpreter of the specific database to process
 - They are meant to be machine readable, because humans usually don't have to care about them
- So if you are doubting the performance of a query always look at the execution plan and query statistics and compare those with your own query's results before making changes

Java Persistence API

- JPA² is a *specification*
 - Not an implementation
 - An example for an implementation of the JPA is Hibernate
 - It does not *do* anything by itself!
- It is concerned with *persistence*
 - Persistence means that objects (can) outlive the application process which created them
- It allows you to define *which* and *how* objects are persisted

²Now also called Jakarta Persistence, see <https://projects.eclipse.org/projects/ee4j.jpa>, 2019-11-02

JPA as 'Interface'

- JPA is *not* a tool or framework
- It just defines a set of concepts that can be implemented
 - Similar to a class interface
- JPA was originally based on Hibernate³
 - Thus tuned for object-relational mapping
- Now JPA has evolved to support NoSQL as well
 - The current reference implementation EclipseLink⁴ supports JPA with NoSQL

³see <https://hibernate.org/>

⁴see <https://www.eclipse.org/eclipselink/>

What JPA specifies

- Bootstrapping
 - Managing DB connections, enabling transactions
- Basic entity mappings
 - Properties, DataTypes & Key(s)
- Mapping relations & associations
 - Foreign Keys, Multiplicities, Cascade operations

What JPA specifies

- Its own query language *JPQL*
 - Very similar to SQL
 - Smaller feature set and no DB specific commands (like CONNECT BY)
 - Option to use *Native Query* which is basically hard coded SQL
 - Allows you to use all features
 - Makes you responsible for writing a correct & safe query and always update it if the model changes
 - Makes database portability (= switching databases) much harder
- Support for custom DataTypes (e.g. enums) when implementing a converter

Differences between Hibernate & EclipseLink

- Both implement the JPA standard ⇒ interchangeable
 - Except when you use specific custom features ⇒ something to be avoided when possible
- EclipseLink
 - Current reference implementation
 - Newer
 - Smaller community
 - Support for NoSQL
- Hibernate
 - Most popular implementation
 - Mature
 - Many proprietary features
 - Focused on relational databases

Configuration Options

- We will be using Hibernate as provider for JPA
 - There are many others but Hibernate is widely used, fully featured and a solid choice which should serve you well
- Hibernate can be configured in two ways:
 - 1 Via XML
 - 2 Via annotations on classes and fields
- We will only look at the annotation variant, because it is much easier to understand and we don't have to leave the code to configure our entities

Bootstrapping

- 1 Make sure your database is running and you can connect to it
- 2 Include the required dependencies in your project
- 3 Configure the *persistence unit* in the persistence.xml⁵
- 4 Create an EntityManager instance
- 5 Begin Transaction
- 6 Perform operations
- 7 Commit/Rollback and close the connection

⁵We'll cover that in a practical session

The EntityManager

- The EntityManager is the most important interface in JPA
- It allows you to:
 - Persist & Remove entities
 - Find existing entities by their key
 - Create queries
 - Use Transactions
- It is created via an EntityManagerFactory which needs a configured persistence unit

The EntityManager

EntityManager Usage

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-
    persistence-unit");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();

// perform insert/update/delete/query

em.getTransaction().commit(); // or em.getTransaction().rollback();
em.close();
```

Defining an Entity

- An entity in JPA is just a POJO
- It has private fields
 - These should have primitive data types to allow mapping to database columns
- It has getters and setters
- It has additional annotations which declare it as an entity:
 - The class is annotated with `@Entity`
 - The primary key field is annotated with `@Id`
- For simple entities this is already enough information for Hibernate to perform the necessary mappings

Customizing the Mapping

- The `@Column` annotation⁶ for fields allows to define
 - The column name (if different from the field name)
 - If the column is nullable
 - If the column can be updated (updateable)
 - Example:
`@Column(name = "id", updatable = false, nullable = false)`
- The `@GeneratedValue` annotation for keys allows to define a key generation strategy
 - Either `GenerationType.SEQUENCE`
 - Or `GenerationType.IDENTITY` (= auto increment)
 - Example:
`@GeneratedValue(strategy = GenerationType.IDENTITY)`

⁶There are more options and more annotations for JPA – only the most common ones are shown here.

Customizing the Mapping

- The `@Enumerated` annotation for enum fields allows to define
 - If the enum is stored as string representation or
 - If the enum is stored by its ordinal value (the default)
 - Example: `@Enumerated(EnumType.STRING)`
- The `@Temporal` annotation for Date field allows to
 - Either store only the date part `TemporalType.DATE`
 - Or date and time `TemporalType.TIME[STAMP]`
 - Example: `@Temporal(TemporalType.DATE)`
 - For Java ≥ 8 use the new Date & Time API which is supported by Hibernate and allows for much better mapping
- The `@Table` annotation for an entity class allows to define a table name (if different from the class name)
 - Example: `@Table(name = "person")`

Composite Keys

- Remember that some entities have a composite primary key
 - For example a ReceiptPosition might have a key consisting of ReceiptNo and PosNo
- In these cases the @Id annotation alone is not enough
- Two steps are necessary to express such a construct in JPA:
 - 1 We have to create a class which represents this key (= contains the key attributes as fields)
 - 2 We need to tell JPA to use this special class as key for our entity

Composite Keys – IdClass

- The first option is to use an IdClass
 - More flexible but also a little more complicated
- Contains the fields necessary for the composite key
- Has to implement Serializable

Example

```
public class ReceiptPosId implements Serializable {  
    private int receiptNo;  
    private int posNo;  
  
    public ReceiptPosId(int receiptNo, int posNo) {  
        this.receiptNo = receiptNo;  
        this.posNo = posNo;  
    }  
  
    // equals() and hashCode()  
}
```

Composite Keys – IdClass

- Set IdClass at Entity class

Example

```
@Entity
@IdClass(ReceiptPosId.class)
public class ReceiptPos {
    @Id
    private int receiptNo;
    @Id
    private int posNo;
    // other fields
    // getter & setter
}
```

Composite Keys – EmbeddedId

- The second option is to use Embeddable
 - Simpler but creates more verbose queries and allows for less control
- Otherwise very similar to the IdClass

Example

```
@Embeddable
public class ReceiptPosId implements Serializable {
    private int receiptNo;
    private int posNo;

    public ReceiptPosId(int receiptNo, int posNo) {
        this.receiptNo = receiptNo;
        this.posNo = posNo;
    }

    // equals() and hashCode()
}
```

Composite Keys – EmbeddedId

- Set EmbeddedId at Entity class

Example

```
@Entity
public class ReceiptPos {
    @EmbeddedId
    private ReceiptPosId receiptPosId;
    // other fields

    // getter & setter
}
```

Composite Keys – EmbeddedId vs. IdClass

- When using IdClass we have to specify the columns twice
- EmbeddedId creates an additional indirection in queries (e.g. receiptPos.receiptPosId.receiptNo)
- If we need to access parts of the composite key individually IdClass can be better
 - Otherwise the getters & setters have to redirect to those of the ID class
- In the wild EmbeddedId is in general preferred thanks to its less verbose structure (= not defining the columns twice).

Inheritance

- Inheritance is an important concept in OOP⁷
- It is not directly supported in relational databases
- There are three strategies to represent inheritance:
 - 1 Single Table Strategy (Table per Hierarchy (TPH))
 - 2 Joined Table Strategy (Table per Type (TPT))
 - 3 Table per Class Strategy (Table per Concrete Class (TPC))
- All those strategies are supported by JPA
- Keep in mind: inheritance in the data model often causes more problems than it solves so use it responsibly
- *For the next examples we assume that a Person can be employee or customer*

⁷Despite composition being the superior model in most cases.

Inheritance – Single Table Strategy

- All entities of the whole class hierarchy are stored in one table differentiated by a so called *discriminator* column
- DiscriminatorType can be STRING or INT

Example

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="person_type", discriminatorType = DiscriminatorType.STRING)
public class Person {

    @Entity
    @DiscriminatorValue(value="Emp") // value to identify rows of this type
    public class Employee extends Person {
```

Inheritance – Joined Table Strategy

- For every entity a table is created
- Shared properties are stored in a base class table
- Inheriting entities have a foreign key to the base table

Example

```
@Entity  
@Inheritance(strategy = InheritanceType.JOINED)  
public class Person {  
  
    @Entity  
    public class Employee extends Person {
```

Inheritance – Table per Class Strategy

- Every entity gets its own table
 - Possibly including the base entity but this table will be empty
 - The @Id column should only be specified in the base class
- Shared attributes are duplicated, Discriminator is not needed
- You have to query each table individually (no discriminator)

Example

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class Person {

    @Entity
    public class Employee extends Person {
```

Relations

- In general you should always model bi-directional relationships
- So both sides of the relation have a *navigation property* to the other
- This can be achieved by annotating fields
- The fields have the class of the other entity as data type
- *To Many* relations are represented with Lists of Entities
- Sometimes it might be necessary to specify the foreign key column

One To One

- Annotate the navigation property with @OneToOne
- Define the join column at most at one side of the relationship (= in one entity, not in both and only if necessary)

Example

```
@Entity
public class Client {
    @OneToOne
    @JoinColumn(name = "shipping_address_id")
    private ShippingAddress shippingAddress;
    // other fields (with ID!)
}

@Entity
public class ShippingAddress {
    @OneToOne(mappedBy = "shippingAddress")
    private Client client;
    // other fields (with ID!)
}
```

DING

One To Many

- The most common relation type
- Again: define the FK column only once and reference the mapping on the other side

Example

```
@Entity
public class OrderItem {
    @ManyToOne
    @JoinColumn(name = "order_id")
    private Order order;
}
@Entity
public class Order {
    @OneToMany(mappedBy = "order")
    private List<OrderItem> items = new ArrayList<OrderItem>();
}
```

Many To Many

- Many to many (n:m) relations are a special case
- We know that to represent those we need an additional table in the database: an association table
- This association table usually has a composite primary key consisting of the primary keys of the two related entities
 - Those are foreign keys at the same time of course
- JPA allows you to hide this association table
- Yet hiding an integral part of the model often leads to problems down the road, so you shouldn't do that
 - Instead just create two 1:n relations between each entity and the association table
- The process is shown in the following anyway to benefit your total mastery of JPA

Many To Many

- Don't forget to update the collection on both ends when inserting/deleting!

Example

```
@Entity
class Student {
    // @Id...
    @ManyToMany
    @JoinTable(
        name = "course_like",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id"))
    Set<Course> likedCourses;
}
@Entity
class Course {
    // @Id...
    @ManyToMany
    Set<Student> likes;
}
```

DING

Many To Many – Modeling the association table

- Using a composite key entity the join table can be modeled (no background magic) with @MapId

Example

```
@Entity
class CourseRating {
    @EmbeddedId
    CourseRatingKey id;
    @ManyToOne
    @MapId("student_id")
    @JoinColumn(name = "student_id")
    Student student;
    @ManyToOne
    @MapId("course_id")
    @JoinColumn(name = "course_id")
    Course course;
    int rating;
}
```

Many To Many – Modeling the association table

- Then reference the join table in both entities

Example

```
@Entity
class Student {
    // ...
    @OneToMany(mappedBy = "student")
    Set<CourseRating> ratings;
}

@Entity
class Course {
    // ...
    @OneToMany(mappedBy = "course")
    Set<CourseRating> ratings;
}
```

FETCH Types

- Often (almost always) there is a relationship between entities
 - e.g. a Class has several Students
- This is represented as an object reference or even collection in the entities
 - A Class has a list of Student objects
 - A Student has a reference to the Class object
- We can decide how those related entities are loaded:
 - Either *eager*: when we load the class all related students are loaded as well
 - Or *lazy*: when we load the class the related students are not loaded until the getter is called

FETCH Types

- EAGER
 - PRO
 - Lower chance for unexpected errors
 - Less IO operations
 - CON
 - Longer initial query execution time
 - Higher memory usage
- LAZY
 - PRO
 - Faster initial query execution time
 - Lower memory usage
 - CON
 - Higher chance for unexpected errors
 - More IO operations

FETCH Types – The n-Query Problem

- When using lazy loading you can run into a problem where n queries are sent which can very quickly become a major performance issue if n is not guaranteed to be very small.

Lazy Load (worst case)

```
Company company = // load 1 company entity — 1 query
for (Customer customer : company.getCustomers()) { // load n customers — 1 query
    for (Order order : customer.getOrders()) { // load m orders — n queries
        for (OrderItem orderItem : order.getOrderItems()) { // load j order items — n*m queries
            Product product = orderItem.getProduct(); // load 1 product — n*m*j queries
            // do something with the product
        }
    }
}
```

FETCH Types – The n-Query Problem

n	m	j	$2 + n + mn + jmn$ Queries	Duration
1	1	1	5	Seconds
1	2	2	9	Seconds
2	3	3	28	Minutes
3	4	4	65	Minutes
12	6	4	374	Minutes
200	6	4	6202	Hours
1800	6	4	55802	Days

- Remember: You do IO for each query and *IO is dead slow!*

FETCH Types – General Guidelines

- Always do eager loading!
- When creating a query think in advance which data you need
- Load as few things as possible but everything that is needed
- If the query becomes to slow try optimizing or batch processing
- That way you receive a predictable and consistent performance
- Consider lazy loading if your ORM does not support eager loading well

FETCH Types – Hibernate

- Contrary to EF JPA/Hibernate does not allow us to do conditional includes of related entities⁸ on a per-query basis
- Given that usually all entities in our model are (transitively) connected to all others (think: graph):
 - We'd have to set up eager loading for all relations
 - So we would load the entire graph of related entities with each query
 - We would *not* load the whole database, but just all entities which are in some way related to the one we are interested in
- For this reason eager loading is considered a bad practice when using Hibernate
 - ⇒ We will usually be using lazy loading

⁸ And recursively their related entities.

Transactions

- One big benefit of a DBMS are transactions, so we want to use them
- While they are necessary to group several related operations it is good practice to wrap at least every write operation
 - A read won't be committed or rolled back, so custom transaction is usually not needed (the DBMS often creates a R lock by itself)
 - You may add other operations later on and if the transaction is already in place no bad surprises can happen

Transactions

- Open and commit or rollback a transaction always at the top level (e.g. in the controller of a WebAPI)
 - This ensures that lower level services can rely on executing in a transaction
 - It also automatically groups all operations belonging to this API call, no matter how internal operation changes
- If you commit in a lower level service you also cannot easily reuse it in other API calls

Transactions

- Transactions are created by the EntityManager

Transaction usage

```
EntityManager em = factory.createEntityManager();
try{
    em.getTransaction().begin();
    // operations like em.persist(anEntity);
    em.getTransaction().commit();
} catch (Exception e) { // a more specific Exception wouldn't hurt...
    // log exception
    em.getTransaction().rollback();
}
em.close();
```

CRUD

- With JPA we can
 - Create: persist entity
 - Read: find entity by id
 - Update: update entity fields
 - Delete: remove entity
- All of those operations (except maybe the Read) will be encapsulated in a transaction
- For none of those do we have to write a SQL statement

Persisting an entity

- 1 Instantiate an instance of the entity class
- 2 Set the properties (you may want to use a factory method)
- 3 Use the persist method of the EntityManager⁹

Insert Entity

```
Person person = new Person();
person.setSSN("1234010190");
person.setName("John Doe");

em.persist(person);
```

⁹Don't forget to commit!

Persisting child entities

- When persisting a master-detail combination you'll usually
 - 1 Create the master entity
 - 2 Create and add the child entities to the master's collection
 - 3 Persist the master entity
- To tell JPA to also persist the child entities in such a situation you need to give it a CascadeType hint
- Be careful to add this hint to the master side of the relation (= in the annotation of the collection)

Example

```
@Entity
public class Order {
    @OneToMany(mappedBy = "order", cascade = CascadeType.PERSIST)
    private List<OrderItem> items = new ArrayList<OrderItem>();
}
```

DING

Reading an entity

- With the EntityManager we can get a previously persisted entity by its key
- For this we use the find method which takes the desired class and key
 - Yes, it really is not a generic method...

Find Entity

```
Person person = em.find(Person.class, "1234010190");
```

Updating an entity

- This is one of the neatest features of a good ORM¹⁰
- Entities which we loaded via the EntityManager are *tracked*
- Hibernate will perform change detection and create a proper update statement for us (if necessary)

Find Entity

```
Person person = em.find(Person.class, "1234010190");
person.setName("Jane Doe");
// optional: other operations
em.getTransaction().commit(); // executes update for the name of the person
```

¹⁰Some ORMs are lightweight on purpose to be used in a specific scenario ⇒ missing feature not always bad.

Deleting an entity

- Once again we retrieve an entity via the EntityManagers find method
- Then we can remove it from the database ⇒ creates a delete statement

Find Entity

```
Person person = em.find(Person.class, "1234010190");
em.remove(person);
// optional: other operations
em.getTransaction().commit(); // executes delete for the person
```

JPQL

- Until now we only retrieved previously persisted entities with `find`
- Of course it is also possible to define search criteria which may return several entities
- For this task JPA provides the *JPQL* query language which is very similar to SQL
- If a specific query cannot be expressed with (DB independent) JPQL we can fall back to issuing native SQL commands

JPQL – Simple Query

- To create a simple Query object we can use the EntityManagers createQuery method
- Then we can retrieve the resulting list of entities (which might be empty) using getResultList¹¹

Simple Query

```
Query query = em.createQuery("select p from Person p where age between 10  
and 20");  
List<Person> persons = query.getResultList();
```

¹¹As usual with Java all of these operations are synchronous which is bad practice for IO operations.

JPQL – Parametrized Query

- Most SQL queries have one or more parameters
- We can define placeholders in the query and then set the actual values for the query object

Parametrized Query

```
Query query = em.createQuery("select p from Person p where name like :  
    name");  
query.setParameter("name", "D%");  
List<Person> persons = query.getResultList();
```

JPQL – Named Query

- Allows to define queries with names for an entity class using the `@NamedQuery` annotation
- It is paramount that you define those query names as string constants
- We will use the superior *repository pattern* instead

Named Query

```
@Entity  
@NamedQuery(name=QUERIES.FIND_BY_NAME, query = "select p from Person p where name like :name")  
public class Person {  
    [...]  
    Query query = em.createNamedQuery(QUERIES.FIND_BY_NAME);  
    query.setParameter("name", "D%");  
    List<Person> persons = query.getResultList();
```

JPQL – Single Result

- There is also an option to retrieve a single result:
`getSingleResult`
- Usually used together with scalar functions

Single Result

```
Query query = em.createQuery("select count(p) from Person p where name  
like :name");  
query.setParameter("name", "D%");  
Long count = (Long) query.getSingleResult();
```

JPQL – Joins

- You can use typical SQL joins like inner join
- But outside of native queries you are limited to your model
- And e.g. a one-to-many relation will not necessarily have the foreign key as actual field
- In such a case ('adding' related entities) you can use a special syntax: `JOIN <entity>.<field>`

Example: Joining related entities

```
SELECT a.city FROM Person p JOIN p.addresses a
```

JPQL – Select DTO

- Sometimes you'll need to select objects which are not entities
 - e.g. just a subset of properties or data from several tables for a report
- You can then define a POJO with fields of the same data type and name as in the select
- Provide it in the JPQL query together with the custom NEW operator
 - Make sure to provide the FQDN in the query or else Hibernate will not know which class to use

JPQL – Select DTOs

Example: Selecting DTOs

```
public class CityPeopleCount {  
    private String cityName;  
    private Long peopleCount;  
  
    public CityPeopleCount(String cityName, Long peopleCount){  
        this.cityName = cityName;  
        this.peopleCount = peopleCount;  
    }  
  
    // getter  
}  
  
// usage  
Query q = em.createQuery("SELECT NEW <FQDN>.CityPeopleCount(a.city, COUNT(p.SSN)) FROM Person p  
    JOIN p.addresses a GROUP BY a.city");  
List<CityPeopleCount> result = q.getResultList();
```

JPQL – Various Quirks

- When using a bool variable in a condition you have to provide an equals condition
 - `f.isBar = true` instead of just `f.isBar`
- If you want to assign an alias for a column use foo as bar without quotation marks
- If you want to check if a property has children you can check if the collection has values
 - e.g. `Query findPoliticians = em.createQuery("SELECT p from Person p where p.brainCells IS EMPTY")`
- *Subqueries are only allowed in a WHERE clause!*
- Be careful that you are not selecting from a table but a defined entity (class starts uppercase)

Repository Pattern

- You should separate database entities and business objects
 - You could implement your business logic in the entities but this will get confusing fast
 - You might not need to expose all fields (as writeable)
- Solution: database entities as *State* of a business object
- We also need a way to group our queries:
 - NamedQueries are really messy
 - The DAO pattern is outdated
- We are going to create a *Repository* for each entity
 - The repository provides methods which return business objects
 - The queries are contained within the repository
 - For dependent (child) entities we might not need a dedicated repository

Introduction

oooooooo
ooooooo

Usage

oooooooooooooooooooo
oooooooooooooooooooo

Performance & Consistency

oooooooooooooo
oooo

Application Integration

oooooo
oooooooooooooooooooo

MongoDB

Markus Haslinger

DBI/INSY

Introduction

oooooooo
ooooooo

Usage

oooooooooooooooooooo
oooooooooooooooooooo

Performance & Consistency

oooooooooooooo
oooo

Application Integration

ooooo
oooooooooooooooooooo

Agenda

1 Introduction

2 Usage

3 Performance & Consistency

4 Application Integration

What is NoSQL?

- Two interpretations are common:
 - Non SQL
 - Not only SQL
- 'SQL' is used as a stand-in for 'relational database'
 - NoSQL databases *can* store relational information
 - NoSQL databases often have SQL inspired query dialects or even support SQL flat out
- But the general consensus is that a *NoSQL database stores data in a format other than relational tables*

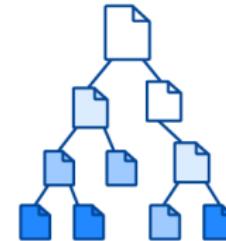
NoSQL format differences

- Four common formats:
 - Document (e.g. MongoDB)
 - Key-value (e.g. Redis)
 - Graph (e.g. Neo4j)
 - Wide-column (e.g. Cassandra)
- Important differences:
 - Related data is (often) not split between tables
 - This is also supported by nesting
 - ⇒ e.g. children/details are stored directly with the parent/master
- These features:
 - Can make modelling data and relations easier
 - Can also lead to weaker models fostering inconsistencies

Document Database

- Data is stored in documents
 - Each document contains key-value pairs
 - Matching closely an OOP object
 - These documents often use JSON notation
 - Related documents can be:
 - Nested or
 - Referenced

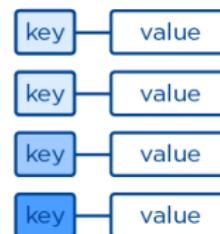
Document



Key-Value Database

- Data is stored as value with a unique key referencing it
 - This allows for fast and easy retrieval of data with a given key
 - Complex queries are usually not recommended or supported
 - The value should have a basic datatype but can also be a complex object

Key-Value



Graph Database

- Data is stored in:
 - Nodes (values)
 - Edges (relationships)
 - Good when searching for patterns
 - e.g. social networks

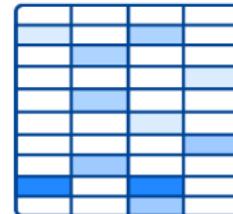
Graph



Wide-Column Database

- At first glance similar to relational DB
- But not every row has to have the same columns:
 - Different count
 - Different names
 - Different data types
- Do not confuse with a column-store database!

Wide-column



Why MongoDB?

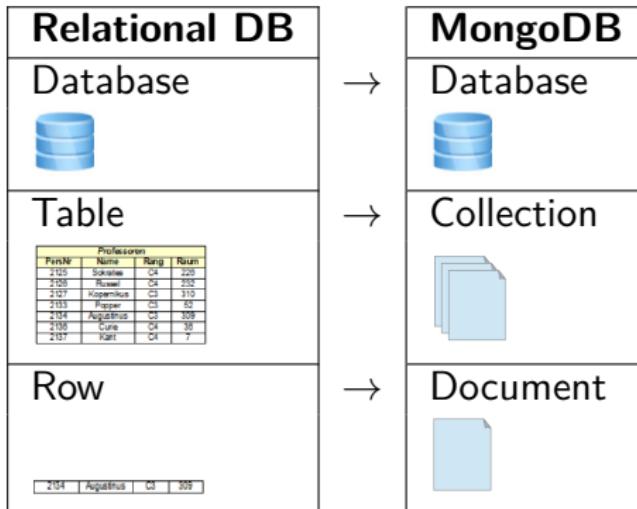
- A Document DB is the most flexible (=applicable to many different scenarios) type of NoSQL database
- It is a very good match for OOP applications
- MongoDB is one of the most commonly used and feature rich Document DBs
 - Plus it has a free community edition and is open source
- Another powerful option for a Document DB would be Raven DB (e.g. used by JetBrains) but with more Enterprise focus and a limited free edition

What is MongoDB?

- Name is a derivate of 'humongous' ('gigantisch')
- Document oriented NoSQL database
- Data is stored in the BSON (Binary JSON) format
 - But interaction usually happens with JSON which is then converted by the database
- Engine written in C++ with the first release in 2009
- One of the most widely used NoSQL databases
 - Dual licensed (open source community and proprietary enterprise)

MongoDB vs. Relational DB – Concepts

- As a rough approximation you can use the following 'translation' of terms:

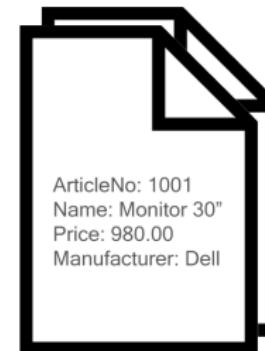


MongoDB vs. Relational DB – Concepts

- Data split into several tables in a relational model is contained within one document

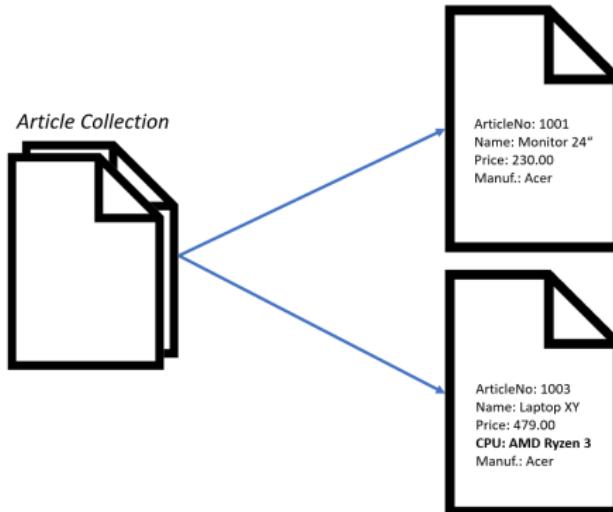
Article			
ArticleNo	Name	Price	ManufacturerID
1001	Monitor 24"	230.00	1
1002	Monitor 30"	980.00	2

Manufacturer	
ManufacturerID	Description
1	Acer
2	Dell



MongoDB – Collections

- Groups similar data (documents)
- Documents are not necessarily identical (different fields)!



MongoDB – Connection – Shell

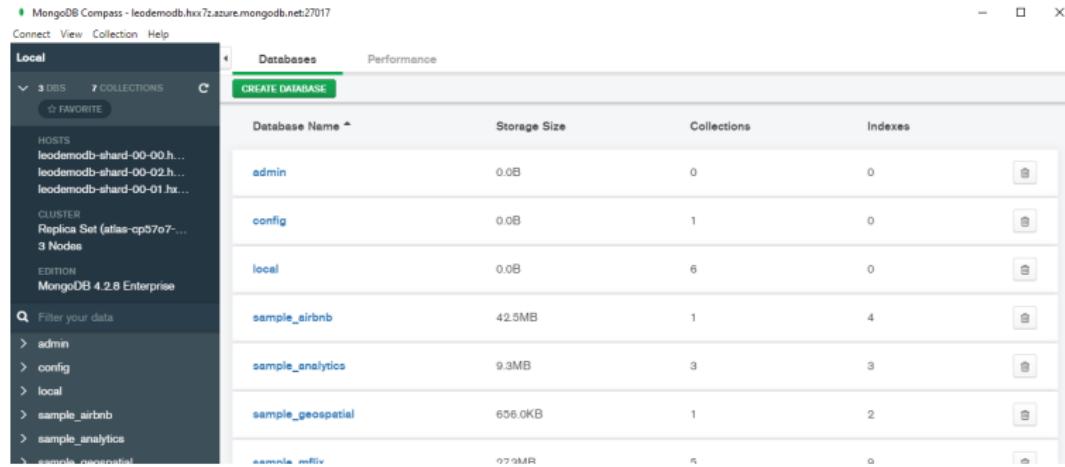
- Using the MongoDB Shell a connection to the database can be established
- Then you can use CRUD operations and issue queries
- *See the Atlas tutorial for details.*

```
Welcome to the MongoDB shell.  
For interactive help, type "help".  
For more comprehensive documentation, see  
    https://docs.mongodb.com/  
Questions? Try the MongoDB Developer Community Forums  
    https://community.mongodb.com  
MongoDB Enterprise atlas-cp57o7-shard-0:PRIMARY> db.adminCommand( { listDatabases: 1 } )  
{  
    "databases" : [  
        {  
            "name" : "sample_airbnb",  
            "sizeOnDisk" : 54296576,  
            "empty" : false  
        },  
    ]  
}
```

MongoDB Basics

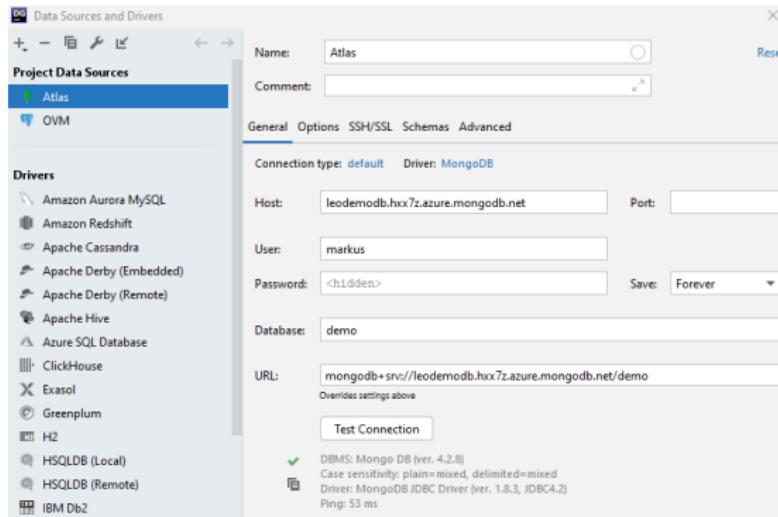
MongoDB – Connection – Compass

- The MongoDB Compass application provides a simple GUI for database management
 - With it you can connect to and then view, search/filter and CRUD data



MongoDB – Connection – DataGrip

- It is also possible to use DataGrip to connect to the MongoDB instance and issue CRUD and query commands



Basic Shell Commands

Command	Effect
help	shows available commands ¹
show dbs	lists available databases
use <dbname>	switches to the given database will create a new one if it does not exist! ²
db	shows the current database
show collections	shows the available collections

¹See also: `db.help()`

²After first write command.

Insert

- Documents need to be inserted into a collection
 - If the collection does not exist yet it is automatically created

Insert Example

```
> db.article.insert(  
... {  
... "ArticleNo": 1001,  
... "Name": "AwesomeThing",  
... "InStock": true  
... })  
WriteResult({ "nInserted" : 1 }) # result = 1 entry inserted
```

Insert – Data Types

- In general 'JSON Data Types' can be used
 - To insert a date object use a JS Date: new Date(2020, 08, 12) (will be saved as ISO Date)
 - To insert a collection use an array: ["foo", 12.34, false]
 - You can nest documents in other documents (children do not need their own ID)

Embedded Insert Example

```
{  
    "Foo": "bar",  
    "Baz": {  
        "FooBar": true  
    }  
}
```

Basic Usage

Retrieve all documents

- Every document needs a unique ID
 - If none is provided it will be automatically generated

Find Example

```
> db.article.find()
{
    "_id" : ObjectId("5f33a3d945b1d7f6d14a4eae"),
    "ArticleNo" : 1001,
    "Name" : "AwesomeThing",
    "InStock" : true
}
```

Search for specific documents

- The `find`³ command can be used with selection parameters
 - Always provide key-value pairs

Find Example

```
> db.article.find({"InStock":true, "ArticleNo": 1001})  
{  
    "_id" : ObjectId("5f33a3d945b1d7f6d14a4eae"),  
    "ArticleNo" : 1001,  
    "Name" : "AwesomeThing",  
    "InStock" : true  
}
```

³Note: find actually returns a cursor, not a collection of results.

Search – Arrays & Embedded

- Array entries can be queried like separate fields
- Embedded documents can be accessed with the dot-notation

Array Find Example

```
db.article.find({"Tags":"Monitor"})
```

Embedded Find Example

```
db.article.find({"PowerSpec.Watts": 65})
```

Basic Usage

Delete

- Documents can be removed from a collection⁴ with the `remove`⁵ function
 - The function takes a search pattern as parameter
 - Use the `deleteOne` function to remove only the first match

Delete Examples

```
db.article.deleteOne({"Manufacturer":"Acer"}) # removes only first  
match  
db.article.remove({"Manufacturer":"Acer"}) # removes all matches
```

⁴Use db.<collection>.drop() to drop a whole collection.

⁵Or deleteMany

Update

- Updates are much less straight forward than inserts and deletes
- They are performed using the `update` function
- The function accepts several parameters, the first is always a search pattern (query)
 - Supplying `{}` matches *all* documents
- Additional parameters control different update modes

Update – Simple Update

- A single document can be updated using the \$set operator
 - All update operators start with a '\$'
 - If the field does not exist \$set adds it to the document
 - *Only the first matching document is updated!*
 - This can also be accomplished using the updateOne function

Simple Update Example

```
db.article.update({Tags:"Monitor"}, {$set:{"Price":189}})
```

Update – Operators

Update Operator	Effect
\$currentDate	Sets the value of the field to the current date
\$inc	Increments the value of the field by the given amount
\$min	Only updates if new value is lower than existing
\$max	Only updates if new value is greater than existing
\$mul	Multiplies field value by given amount
\$rename	Renames a field
\$set	Sets the value of the field
\$setOnInsert	Sets the field value only if 'upsert' option caused insert
\$unset	Removes the specified field from a document

Update – Multiple Documents

- To update multiple documents there are two options:
 - Add the {"multi":true} option to the update function⁶
 - Or use the updateMany function

Multi Document Update Examples

```
db.article.update({Tags:"Monitor"}, {$set:{"Price":189}}, {"multi":true})
```

```
db.article.updateMany({Tags:"Monitor"}, {$set:{"Price":189}}) # same effect
```

⁶If not in the shell (but e.g. DataGrip) these 'overloads' may not work.

Update – Upsert

- When supplying the `{"upsert":true}` option a document will be inserted if it does not exist
 - Even the collection will be created if it does not exist!

Upsert Examples

```
db.review.update({ArticleNo:1001}, {$inc:{count:1}}, {upsert:true})
```

Update – Replacement

- To replace a document you can use the `replaceOne` function
 - The ID stays the same but the whole content is replaced

Replacement Example

```
db.article.replaceOne({Tags:"Monitor"}, {Price:189})
```

	_id	Price	ArticleNo	Manufacturer	Pov
1	5f33afcc45b1d7f6d14a4eaf	189	<unset>	<unset>	<unse

Update – Array – Updating Value

- Single values in an array can be updated using the dot-notation
 - Using the zero based index of the array
 - Be careful to use quotation marks for the key (= array field name)

Array Update Example

```
db.article.update({ArticleNo:1003}, {$set: {"Tags.0": "Notebook"}})
```

Update – Array – Add Value

- There are two options to add a value to an array:
 - `$push` adds the value to the end of the array
 - `$addToSet` adds the value to the end only if it does not already exist in the array

Array Add Item Example

```
db.article.update({ArticleNo:1002}, {$push: {Tags: "Monitor"}})  
db.article.update({ArticleNo:1002}, {$addToSet: {Tags: "Monitor"}}) #  
  won't add it a second time
```

Update – Array – Remove Value

- With the \$pop operator the first and last value of an array can be removed
 - 1 removes the last element
 - -1 removes the first element

Array Remove Item Example

```
db.article.update({ArticleNo:1002}, {$pop: {Tags: 1}})  
db.article.update({ArticleNo:1002}, {$pop: {Tags: -1}})
```

Update – Array – Remove Values

- With the \$pull operator all occurrences of a value can be removed from an array
 - Independent of their position in the array

Array Remove Items Example

```
db.article.update({ArticleNo:1002}, {$pull: {Tags: "Monitor"}}) # will  
remove 0, 1 or n elements
```

Queries – Multiple Criteria

- A query can contain several criteria
- These will be linked with an *AND* condition

Multi Conditional Query Example

```
db.article.find({Tags: "Monitor", Manufacturer: "Acer"})
```

Query – Comparison Operators

Comparison Operator	Meaning
\$gt	greater than
\$gte	greater than or equal
\$lt	lower than
\$lte	lower than or equal
\$ne	not equal

Comparison Operator Example

```
db.article.find({Price: {$lte: 189}})  
db.article.find({Price: {$lt: 200, $gt: 180}}) # between query
```

Query – Array Range Query

LED TV Document

```
{  
  Desc: "LED TV",  
  Sizes: [40, 47, 48]  
}
```

Plasma TV Document

```
{  
  Desc: "Plasma TV",  
  Sizes: [32, 34, 47]  
}
```

Array Range Query Example

```
db.tv.find({sizes: {$gt: 34, $lt: 47}}) # checks every element, each has  
to pass any condition (OR)
```

Query – Array Range Query – \$elemMatch

LED TV Document

```
{  
  Desc: "LED TV",  
  Sizes: [40, 47, 48]  
}
```

Plasma TV Document

```
{  
  Desc: "Plasma TV",  
  Sizes: [32, 34, 47]  
}
```

Array Range Query Example

```
db.tv.find({sizes: {$elemMatch: {$gt: 34, $lt: 47}}}) # at least one  
element has to fulfill all conditions (AND)
```

Query – Projection

- The `find` function takes a document as second parameters specifying the fields to include in the result
- The field name is supplied as key and the value as bool⁷ with two different meanings:
 - If using `true` only the supplied fields are selected
 - If using `false` the supplied fields are deselected and the others are shown

Projection Examples

```
db.article.find({}, {"Desc": true, "Price": true})  
db.article.find({}, {"Manufacturer": false, "PowerSpec": false})
```

⁷Do not mix `true` and `false`!

Query – Count

- The `count` function can be used with the `find` function
- It returns the count of results found (= matches)

Count Example

```
db.article.find({}, {"Desc": true, "Price": true}).count()  
# DataGrip uses the Java Driver so you have to write db.article.count  
({<searchPattern>}) instead
```

Query – Sort

- The `sort` function can be used with the `find` function
- It sorts the results based on the given attributes:
 - ascending: `1`
 - descending: `-1`

Sort Examples

```
db.article.find({}, {"Price": true}).sort({"Foo": 1})
```

```
db.article.find({}, {"Price": true}).sort({"Foo": 1, "Price": -1}) #  
multiple sort fields
```

Query – Pagination

- To allow pagination (or batch processing if no updates happen in between) the following functions can be used:
 - `skip` skips the first n elements in the result set
 - `limit` takes n elements from the result set⁸

Skip and Limit Example

```
db.article.find({}).skip(6).limit(3)
```

⁸c.f. Take in LINQ

Embed or Reference Documents?

- Related documents can be linked in two different ways:
 - Embedded
 - Referenced
- There is no normalization, so the rules for relational databases don't apply
- But we have to keep one limit in mind: 16MB per document (including those embedded) and 100 levels of nesting
- How to decide which method to use when?

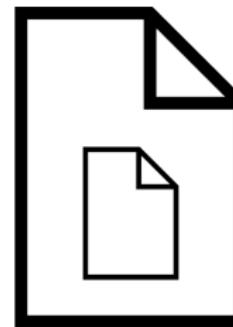
Embedding Documents

- Pro:

- Retrieval with single query
- Atomic write operation

- Con:

- Redundancies
- Possible inconsistencies



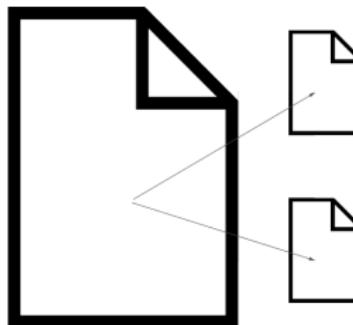
Referencing Documents

- Pro:

- Less Redundancy
- Documents exist independent of each other

- Con:

- Requires 2+ queries
- Documents exist independent of each other



Embed or Reference – Decision Matrix

<i>Is the data commonly used together?</i>			
	Always	Sometimes	Rarely
Embed	✓	✓	
Reference		✓	✓
<i>How big is the data?</i>			
	Small	Medium	Large
Embed	✓	✓	
Reference		✓	✓
<i>How often does the data change?</i>			
	Rarely	Sometimes	Often
Embed	✓	✓	
Reference		✓	✓

Aggregate

- Generally speaking, the aggregate function allows grouping of data
- Once the data has been split into groups statistical functions can be used

Aggregate Syntax

```
db.<collection>.aggregate(  
  [  
    {<operator>: {<key>: <value>}}  
  ]  
)
```

Aggregate – \$group

- There are two different kinds of '\$' strings:
 - Operators used as keys (do not require quotation marks)
 - Field-Paths used as values (require quotation marks)
- \$group is the basic operator used to group a collection by a specific field

Group Example

```
db.article.aggregate([{$group: {"_id": "$Manufacturer"} }])
```

Aggregate – \$sum

- After grouping it is possible to use accumulators
- `$sum` counts the elements in the group

Sum Example

```
db.article.aggregate([{$group: {"_id": "$Manufacturer",  
    "Count": {"$sum : 1}}}] )
```

	{}_id	Count
1	Acer	2
2	Asus	1

Aggregate – \$avg

- \$avg calculates the average for the given field within each group

AVG Example

```
db.article.aggregate([{$group: {"_id": "$Manufacturer",  
"Count": {"$sum : 1},  
"AvgWatts": {"$avg: "$PowerSpec.Watts"} }}])
```

	_id	AvgWatts	Count
1	Acer	35	2
2	Asus	65	1

Aggregate – \$min & \$max

- \$min & \$max calculate the min and max value respectively
- The same field can be used for multiple accumulators

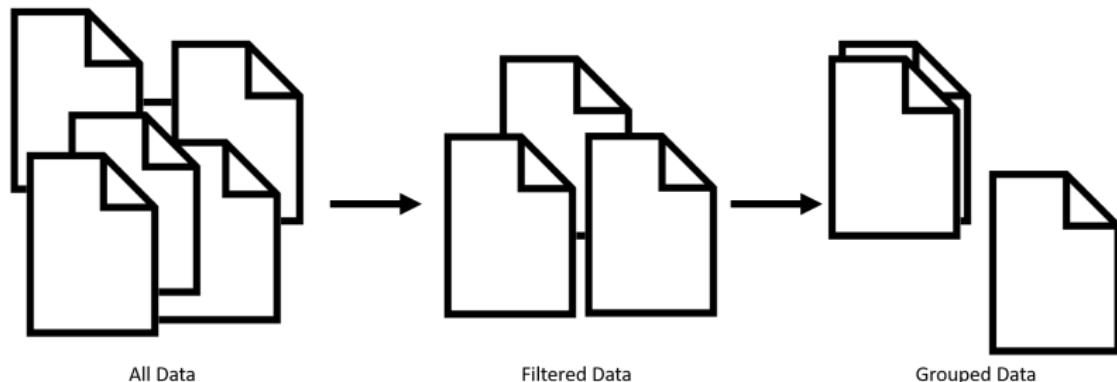
Min & Max Example

```
db.article.aggregate([{$group: {"_id": "$Manufacturer",
    "Count": {"$sum : 1},
    "MinWatts": {"$min: "$PowerSpec.Watts"},
    "MaxWatts": {"$max: "$PowerSpec.Watts"}}}])
```

	_id	Count	MaxWatts	MinWatts
1	Acer	2	45	25
2	Asus	1	65	65

Aggregate – Pipeline

- The aggregate function can work like a pipeline
 - Data is queried (filtered), then grouped, then possible accumulators are applied



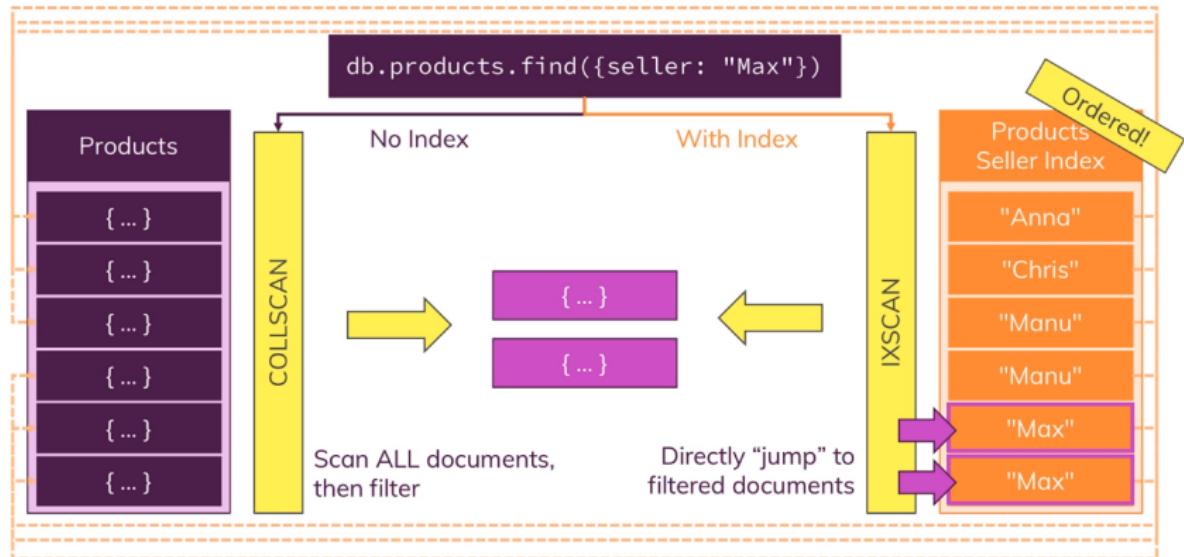
Index – General

- Indices are additional structures
 - They do not replace a collection, but exist in addition to them
- When used correctly they can greatly speed up query performance
- But a bad index design can also lead to slower query performance
- They take up space so only add them where actually required

Index – General

- Imagine that we want to find products sold by 'Max'
 - Without an index on the 'seller' field the database has to do a full *collection scan* (COLLSCAN)
 - For huge collections this can take a long time
 - If an index for the 'seller' field has been defined then the database performs an *index scan* (IXSCAN)
 - Reading through an index is faster, because the values are *sorted* (by index value vs. primary key)
 - Thus the process can 'jump' to the correct region and vastly reduce search duration
 - The index contains a 'pointer' to the document in the collection
 - It kind of works like a sorted key-value store with field value as key and document location as value

Index – General



Source: M. Schwarzmüller 2018

Index – For every Field?

- If an index speeds up query performance when filtering by a specific field, then why don't we add an index for every single field of every single collection?
 - There are two reasons:
 - An index takes up additional space
 - While an index speeds up queries it incurs a performance *cost* on insert
 - Because not only the document is inserted but then also the index updated (extended and possibly resorted)
 - **Remember:** indices don't come for free – you have to weight cost against benefit when deciding where to use one

Default Index

- Every document has a unique ObjectId
- That is used as the primary key
- MongoDB automatically creates and maintains an index for the ID field
 - This ensures uniqueness
 - And it defines the default order in which documents are fetched (cf. clustering)

Explain

- We have established that we want to use as many indices as needed and as few as possible
- To help us make these decisions we can use the `explain` function
- This gives you information about the *plan* MongoDB used to perform the query

Explain Example

```
db.airport.explain().find({RunwayLength: {$lt: 8000}})
```

Explain

Plan Explained

```
"plannerVersion" : 1,  
"namespace" : "flights.airport", # collection we are searching  
"indexFilterSet" : false, # no index  
"parsedQuery" : { # query information  
    "RunwayLength" : {  
        "$lt" : 8000  
    }  
},  
"queryHash" : "461D1D6C",  
"planCacheKey" : "461D1D6C",  
"winningPlan" : { # plan used  
    "stage" : "COLLSCAN", # collection scan was performed  
    "filter" : {  
        "RunwayLength" : {  
            "$lt" : 8000  
        }  
    },  
    "direction" : "forward" # started at the beginning (vs. end)  
},  
"rejectedPlans" : [] # simple query and no index => only one possible plan
```

DING

Explain – Execution Statistics

- Using the `executionStats` verbosity option for the `explain` function returns detailed execution performance data

Execution Stats Explained

```
# db.airport.explain("executionStats").find({RunwayLength: {$lt: 8000}})  
# snip  
"nReturned" : 947, # documents found  
"executionTimeMillis" : 0, # my new PC is damn fast  
"totalKeysExamined" : 0, # keys processed during index scan  
"totalDocsExamined" : 1612, # all documents scanned (= whole collection)  
# snip  
"works" : 1614, # 'work units', e.g. fetching 1 document or applying a projection to 1 document  
"advanced" : 947, # documents returned by the stage  
"needTime" : 666, # 'lost' work cycles (= work that did not directly return a document, but may have helped)  
"needYield" : 0, # number of times query had to pause to release locks  
"saveState" : 1, # times state was saved (e.g. prior to releasing locks)  
"restoreState" : 1, # times state was restored (e.g. after resuming)  
"isEOF" : 1, # 0 if there might be additional results (cf. limit)  
# snip
```

DING

Single Field Index

- The simplest type of an index is one covering only a single field
- It is basically a (sorted!) key-value collection
- The key is the field value, the value is the (original) document location
- Create using⁹

```
db.<collection>.createIndex({"<field>": 1})
```

 - 1 = ascending order
 - -1 = descending order
- This is possible for
 - top level field
 - embedded fields

⁹Remove using `dropIndex`.

Single Field Index – Effects

- Winning Plan is now a combination of IXSCAN & FETCH
 - IXSCAN providing document locations
 - FETCH loading these (already filtered) documents
- Fewer total documents are processed
- The filter criteria is translated into an index range ($[-\inf, 0, 8000.0]$)
- The index is used, so keys are examined
- Fewer work units are needed
 - Execution time is faster¹⁰

¹⁰For small data sets it will not be noticeable.

Single Field Index – Effects

Execution Stats Explained

```
# db.airport.explain("executionStats").find({RunwayLength: {$lt: 8000}})  
# snip  
"totalKeysExamined" : 947,  
"totalDocsExamined" : 947,  
"executionStages" : {  
    "stage" : "FETCH",  
    "nReturned" : 947,  
    "executionTimeMillisEstimate" : 0,  
    "works" : 948,  
    "advanced" : 947,  
    "# snip"  
    "docsExamined" : 947,  
    "alreadyHasObj" : 0,  
    "inputStage" : {  
        "stage" : "IXSCAN",  
        "# snip"  
        "indexBounds" : {  
            "RunwayLength" : [  
                "[‐inf.0, 8000.0)"  
            ]  
        }  
    }  
    "# snip"
```

DING

Unique Index

- Every document receives a unique ObjectId as primary key
- If you want the database to ensure uniqueness for other fields as well use a unique index
- Create it using

```
db.<collection>.createIndex({"<field>": 1},  
{unique: true})
```

Compound Index

- For certain scenarios you might want to include multiple fields in one index
- Create it using

```
db.<collection>.createIndex({"<field1>": 1,  
    "<field2>": -1})
```
- The order in which you put these fields *is* important, because the index will be stored in this (sorted) order

Text Index

- When searching (frequently!) within a text (stored in a field) you can use a special text index
- This index breaks up the text into (key)words and *automatically removes stop words*
- Create using `db.<collection>.createIndex({<field>: "text"})` (do *not* use 1 or -1)

This product is a must-buy for all fans of modern fiction!



Transactions in MongoDB

- Operations on a single document are guaranteed to be *atomic*
 - This includes arrays and embedded documents
- Transactions are required:
 - When using reference relations
 - When updating more than one document
- If you use them regularly you can't forget about them later if operations change
- Multi document ACID transactions are part of MongoDB since version 4
 - Using *Snapshot* isolation
 - **But only available for replication set and sharded instances!**

Transaction Usage

- A transaction belongs to a *session*
- The session object provides four important functions:
 - 1 `session.startTransaction()`: transaction begin
 - 2 `session.getDatabase("<DBNAME>").<COLLECTION>`: retrieves a collection
 - Only operations on collections retrieved via the session are part of the transaction!
 - 3 `session.commitTransaction()`: transaction commit
 - 4 `session.abortTransaction()`: transaction rollback

Using a Transaction – Shell

Transaction Example Shell

```
const session = db.getMongo().startSession() # open session
session.startTransaction() # begin a transaction
const users = session.getDatabase("blog").user # get user collection
const posts = session.getDatabase("blog").post # get post collection
# be careful to ONLY use collections retrieved via the session, a 'db.user'.
    insertOne' would NOT be within the transaction at this point!
users.deleteOne({_id: ObjectId("<ID>")}) # only deleted in the session —
    not yet committed!
posts.deleteMany({user: ObjectId("<ID>")}) # second operation within the
    transaction
session.commitTransaction() # commit changes
```

Using a Transaction – Driver

Transaction Example Driver

```
using var session = await db.Client.StartSessionAsync(); // open session
session.StartTransaction(); // begin transaction

var customers = database.GetCollection<Customer>("customer"); //
    collections are retrieved independent of the session
var c1 = new Customer{Name = "Horst"};
await customers.InsertOneAsync(session, c1); // you HAVE to pass the session
    object for every operation!

await session.CommitTransactionAsync(); // commit changes
```

Database

- MongoDB is generally meant to be used for big amounts of data
- It supports features like *sharding* and *replica sets*
 - Sharding distributes data across multiple machines increasing throughput and supporting very large data sets
 - Replica sets are multiple instances which each hold the same data to provide redundancy
- You can use a single node instance as well
 - Which we will do to keep administration efforts low

Connection

- Typically the database will run on one or more external servers
- Luckily MongoDB now includes support for SSL connections
 - Using certificates
 - In the past that feature was disabled in the community edition and one had to build from source with a flag set to get SSL
- Depending on the setup (single node, sharded, replica set) there are some differences but the general principle stays the same
- You need a connection string [+ username & password]

Driver

- Integration of the database into a platform happens via a *driver*
- There are drivers available for all major platforms:
 - C/C++
 - C#
 - Java
 - Node.js
 - Python
 - Rust
- Every driver tries to follow the specific platform coding style
 - More or less successful

Do we need an ORM?



- MongoDB is not a relational database
- Its document approach already allows to persist objects
- This means that we do *not* need an ORM
 - Less configuration required
 - But we also lose some features powerful ORMs provide (e.g. change tracking, ambient transactions)

Async/Await

- Database operations are I/O operations
- I/O operations should almost always be asynchronous
- Luckily we are using a platform with native support for asynchronous operations¹¹
 - ⇒ most of the time the async variant of a method is used
- Proper asynchronous operation makes the necessity of transactions even more evident

¹¹If you are locked into an old platform like Java at least try to use continuation callbacks.

Usage

Entity Attributes

- Not much is needed to turn a class into a database entity
 - There are no base classes you need to inherit from
 - Also the amount of attributes is very limited
 - The most important attributes are:
 - BsonId defines the ID field¹² – typically of type ObjectId
 - BsonRepresentation(BsonType.ObjectId) allows to use a string field as ID
 - BsonElement allows to override field name
 - BsonIgnore ignores the field
 - BsonDefaultValue defines a default value
 - BsonDateTimeOptions better control of DateTime serialization
 - BsonRequired serialization fails if field is null

¹²Instead of fields, properties can be used as well in all of these cases.

Entities

- We are able to use POCOs with the driver API
- So no need to work with JSON/BSON documents directly
- References to other objects (documents) *need* to be stored as ObjectId
 - No 'magic' references like with EF

Entity Example

```
public sealed class Foo {  
    [BsonId]  
    public ObjectId Id { get; set; }  
    public int SomeVal { get; set; }  
    public ObjectId Bar { get; set; }  
}
```

DING

Entities – Embedded

- Embedded objects can be put directly in the POCO
 - From the application perspective they are one entity
- Embedded arrays can be represented as lists¹³

Entity Embed Example

```
public sealed class Foo {  
    [BsonId]  
    public ObjectId Id { get; set; }  
    public SomeOtherClass EmbeddedObject { get; set; }  
    public List<YetAnotherClass> EmbeddedArray { get; set; }  
}
```

¹³In this specific case I recommend using an actual List and not IList or IEnumerable.

Class Maps

- Instead of using attributes you can also define a class map¹⁴

Class Map Example

```
BsonClassMap.RegisterClassMap<MyClass>(cm => {  
    cm.AutoMap();  
    cm.GetMemberMap(c => c.X).SetIsRequired(true);  
    cm.GetMemberMap(c => c.Y).SetSerializer(new  
        MyCustomSerializer());  
    cm.GetMemberMap(c => c.Z).SetDefaultValue("abc");  
});
```

¹⁴Also see <https://mongodb.github.io/mongo-csharp-driver/2.11/reference/bson/>

Class Polymorphism

- When using a class hierarchy where all entities of the subclasses are stored in one collection
- There is a default setting for using the class name as *discriminator*
 - Which can be overridden
- Important for the deserialization is that the *known types* have to be specified

Known Types

```
[BsonKnownTypes(typeof(Cat), typeof(Dog))]  
public class Animal {}
```

References

- In the relational world references are important, because we have to normalize
- Tools like EF provide to load a referenced entity (Include)
- MongoDB relies much more on the embedding approach
- There is no built-in support for references
- Instead you simply store the ObjectId of the referenced entity in a field and load it with a second query (or n queries for n references)
- Be aware that 'foreign keys' only exist in our model, not in the database
 - So things like a cascade delete or an error due to trying to delete a referenced entity do not exist either

Database Connection

- To connect to the MongoDB database from your application you first need a `MongoClient`
 - Which requires a connection string
 - And possibly a `MongoCredential` object for authentication
- Then you can retrieve an `IMongoDatabase`

Database Connection Example

```
var client = new MongoClient("mongodb://localhost"); // pass  
connection string  
var database = client.GetDatabase("<DATABASE NAME>");
```

Collections

- `IMongoCollection<T>` is retrieved from the `IMongoDatabase` object
 - A collection which not yet exists is usually created at first write
 - But you can ensure the existence manually

Collection Example

```
var database = client.GetDatabase("<DATABASE NAME>");  
await database.DropCollectionAsync("<COLLECTION>");  
await database.CreateCollectionAsync("<COLLECTION>");  
var entities = database.GetCollection< TEntity >("<COLLECTION>");  
    // use proper class for T
```

Insert

- First a collection is needed
- Then you can use
 - `InsertOneAsync` to insert a single document
 - `InsertManyAsync` to insert multiple documents at once

Insert Examples

```
var entities = database.GetCollection< TEntity >("<COLLECTION>");  
    // use proper class for T  
await entities.InsertOneAsync(new Foo { Bar = "Baz" });  
await entities.InsertOneAsync(session, new Foo { Bar = "Baz" }); //  
    uses transaction!  
await entities.InsertManyAsync(new[] { new Foo { Bar = "Baz" }, new  
    Foo { Bar = "Foobar" } });
```

DING

Introduction
○○○○○○
○○○○○○○○

Usage
○○○○○○○○○○○○○○○○
○○○○○○○○○○○○○○○○

Performance & Consistency
○○○○○○○○○○○○○○
○○○

Application Integration
○○○○
○○○○○○○●○○○○○○

Usage

Query

- You have two options for querying data
 - Filter (+ Builder)
 - Uses more or less the same query operators and style we used in the shell
 - Is sometimes required for using Update and Delete operations
 - LINQ
 - Much easier to write and understand
 - Can also be used with transactions
 - But only limited use for Update/Delete
- It is possible to mix both approaches
 - LINQ for queries
 - Filter for write operations

Filter

- The Filter approach uses Builders
- You can use the operators known from the shell (eq, gte, in)
- Such a FilterDefinition can be used with Update & Delete as well

Filter Examples

```
var entities = database.GetCollection<X>("<COLLECTION>");  
await entities.FindAsync(Builders<X>.Filter.Eq(x => x.Title, "Foo"));  
await entities.ReplaceOneAsync(Builders<X>.Filter.AnyGt(x => x.  
    Value, 2));  
await entities.DeleteManyAsync(Builders<X>.Filter.In(x => x.  
    DecimalVal, new[]{2.1d, 4.5d}));
```

Filter – Update

- Performing updates instead of replacing can be more performant
- This requires the use of Builders together with the Update and Filter methods
- It is also possible to update values contained in arrays

Update Example

```
var entities = database.GetCollection<X>("<COLLECTION>");  
var filter = Builders<X>.Find.Eq(x => x.City, "Foo");  
var update = Builders<X>.Update.Set(x => x.Value, 3);  
await entities.UpdateManyAsync(filter, update);
```

LINQ – Query

- You need a collection first
- Then call the `AsQueryable` method
 - Pass the session if the query should be part of a transaction
- Use `Where` as usual
- Just like with `Find` we get an `IAsyncCursor` which is then materialized into a `List`

LINQ Query Example

```
var fooColl = database.GetCollection<Foo>("<COLLECTION>");  
var foos = await fooColl.AsQueryable()  
    .Where(f => f.Grade == 1)  
    .ToListAsync();
```

DING

LINQ – Query – Advanced

- In addition to Where you can also use
 - OrderBy for sorting
 - GroupBy for aggregation

LINQ Advanced Query Example

```
var fooColl = database.GetCollection<Foo>("<COLLECTION>");  
var foos = await fooColl.AsQueryable(session) // optional transaction  
    .Where(f => f.Grade == 1)  
    .OrderBy(f => f.LastName)  
    .ToListAsync();  
  
await fooColl.AsQueryable().GroupBy(o => o.Customer).CountAsync();
```

Lambda instead of Filter

- In most cases you can use a lambda instead of a `FilterDefinition` for update and delete operations¹⁵ and simple queries
- This makes the code less verbose and easier to read

Lambda Examples

```
var fooColl = database.GetCollection<Foo>("<COLLECTION>");  
await fooColl.FindAsync(d => d.Id == document.Id);  
await fooColl.ReplaceOneAsync(session, d => d.Id == document.Id,  
    document);  
await fooColl.DeleteOneAsync(session, d => d.Id == document.Id);
```

¹⁵Mind that modifying operations should always use a transaction.

Index Configuration

- Indexes can be configured via `IMongoIndexManager<TDocument>`
- This manager is collection dependent and can be accessed via the `IMongoCollection<TDocument>.Indexes` property
- The two important methods are:
 - `CreateOneAsync` to add an index¹⁶
 - `DropOneAsync` drops an index
 - Requires the *name* of the index returned by the `CreateOneAsync` call

¹⁶You can check with the `ListAsync` method which indexes already exist.

Index Configuration

Index Configuration Example

```
var collection = database.GetCollection<Person>("person");
var indexOptions = new CreateIndexOptions();
    // see API for options (e.g. unique)
var indexKeys = Builders<Person>
    .IndexKeys.Ascending(person => person.Age);
var indexModel = new CreateIndexModel<Person>(indexKeys, indexOptions);
await collection.Indexes.CreateOneAsync(indexModel);
```

Normalization

Markus Haslinger

DBI/INSY 3rd year

Agenda

1 Introduction

- Redundancies & Anomalies
- Functional Dependencies

2 Normal Forms

- 1., 2. & 3. NF

3 Advanced Normal Forms

- BCNF, 4. & 5. NF

Why normalize data?

- Two reasons:
 - 1 To prevent redundancies
 - Which helps save storage space
 - 2 To prevent anomalies
 - Which makes programs simpler
 - Which prevents issues

Redundancy

Definition

Redundancy in data is then present when a (= the redundant) part of the stored data can be deleted without any information loss.

- On the conceptual level redundancy is always to be avoided
 - Wastes storage space
 - Allows for anomalies
 - Changes have to be applied in multiple locations
(cf. duplicate code)
 - How to be sure one copy of the data is the correct one?
⇒ single point of truth paradigm
- Do not mistake this with redundancy for things like backups, georedundancy, . . .

Anomalies

- If data is stored redundant this will lead certainly lead to anomalies
- E. Codd described three types of anomalies:
 - Update Anomaly
 - Insertion Anomaly
 - Deletion Anomaly

<i>Example Table 'Teaching'</i>				
Stu#	Class#	StuName	StuAddress	ClassName
S21	8725	Josef	Linz	DBI
S21	8730	Josef	Linz	SEW
S33	8725	Maria	Wels	DBI

Example – Update Anomaly

- Changing address of Student S21
- Two affected rows
 - We could have overlooked one!

Stu#	Class#	StuName	StuAddress	ClassName
S21	8725	Josef	Leonding	DBI
S21	8730	Josef	Leonding	SEW
S33	8725	Maria	Wels	DBI

Example – Insertion Anomaly

- A new student is added
 - The new student does not attend any classes at this point
 - But the key demands the Class# not to be NULL
- A new class is added
 - The new class has not been booked by any students at this point
 - But the key demands the Stud# not to be NULL
- In both cases the insert is not possible!

Example – Deletion Anomaly

- Student S21 is dropping out, so we want to delete his data
- Yet he is the only one attending class 8730
- So when deleting the student data we would loose all information about the class!

Stu#	Class#	StuName	StuAddress	ClassName
S21	8725	Josef	Leonding	DBI
S21	8730	Josef	Leonding	SEW
S33	8725	Maria	Wels	DBI

Issues of Normalization

- A collection of rules meant to reduce redundancies and anomalies
 - Only reduces but not definitely prevents them
- Normalization only looks at isolated entity sets
 - Thus cannot detect redundancies spanning multiple entity sets
 - Not a complete set of rules for all eventualities
- Happens on the conceptual level
 - Sometimes reverted when creating the internal model ⇒ Performance considerations

Functional Dependency

- Representation: $\alpha \rightarrow \beta$
- α and β are each sets of attributes defined as $\alpha \subseteq R$
- For all tuple pairs $r, t \in R$ with $r.\alpha = t.\alpha$ has to be also true $r.\beta = t.\beta$
 - If two tuple have equal values for all attributes in α then they also have to have equal values for all attributes in β
- α values determine the β values, thus β values are functional dependent on α values
- If α is a composite key and β an attribute set of the same relation R .
Then β is fully functional dependent on α when it is functional dependent on α but not any subset of α

Functional Dependency – Example

<i>R</i>			
A	B	C	D
a4	b2	c4	d3
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c3	d2
a3	b2	c4	d3

■ True:

- $\{A\} \rightarrow \{B\}$
- $\{A\} \rightarrow \{C\}$
- $\{C, D\} \rightarrow \{B\}$

■ False:

- $\{B\} \rightarrow \{C\}$

Transitive Dependency

- An attribute C is transitively functional dependent on an attribute A
 - When there is an attribute B which is functional dependent on A **and**
 - When C is functional dependent on B
- A determines B & B determines C
 - $A \rightarrow B \rightarrow C \Rightarrow A \rightarrow C$

Armstrong Axioms

- Axiom of reflexivity
 - If X is a set of attributes and Y is a subset of X , then X holds Y . Hereby, X holds Y means that X functionally determines Y
 - If $Y \subseteq X$ then $X \rightarrow Y$
- Axiom of augmentation
 - If X holds Y and Z is a set of attributes, then XZ holds YZ
 - If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Axiom of transitivity
 - If X holds Y and Y holds Z , then X holds Z
 - If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

FD – Example

Desc	Type	Tires	HP
Käfer	PKW	4	45
Hanomag II	LKW	4	250
Vespa	Roller	2	4
Suzuki GX	Motorrad	2	150
Ford Focus GT	PKW	4	65

- Which functional dependencies exist?

FD – Example – Solution

- $Desc \rightarrow \{Type, Tires, HP\}$
 - All non-key attributes are functional dependent on the key
- $Desc \rightarrow Tires$
 - We know that $Desc \rightarrow Type$
 - We can deduct that $Type \rightarrow Tires$
 - We conclude $Desc \rightarrow Tires$
 - This transitive functional dependency exist *in addition* to the first one

Summary

- For practical reasons:
 - Keys determine non-key attributes
 - So non-key attributes depend on keys
 - Further dependencies (especially transitive ones) can be deducted
- Functional dependencies are primarily based on logical dependencies
- If the table structure does not comply with the logical dependencies that is an indication that something is odd

1. NF

- A relation is in 1. NF if the domains of all attributes are scalar
- Scalar means no repetition \Rightarrow atomic

- Some DBS allow this on purpose for performance reasons
- But usually all attributes should be scalar

Examples for atomic attributes

- Address
- Split into multiple columns
- Maybe even add street address

Cust#	Name	Address
1	Karl	Limesstr. 12-14 4060 Leonding

Cust#	Name	Street	ZIP	City
1	Karl	Limesstr. 12-14	4060	Leonding

Examples for atomic attributes

- E-Mail Addresses
- Split into multiple columns
- Or use an extra table for n addresses

CustNo	Name	E-Mail
1	Karl	karl123@gmy.at, karl.cool@dmail.com

CustNo	Name	E-Mail1	E-Mail2
1	Karl	karl123@gmy.at	karl.cool@dmail.com

CustNo	E-Mail
1	karl123@gmy.at
1	karl.cool@dmail.com

2. NF

- A relation is in 2. NF if it is in 1. NF *and*
- if all non-key attributes are fully functional dependent on the (composite) primary key

- A relation can only *not* be in 2. NF if the primary key is a composite key and the relation contains non-key attributes...
- ...because then a non-key attribute could be dependent on a subset of the primary key
- 2. NF is achieved by splitting the entity type and set

2. NF – Splitting

- n key attributes $\Rightarrow 2^n - 1$ possible combinations
- Process:
 - 1 Determine possible key combinations
 - 2 Assign non-key attributes to the key combination it is dependent on
 - 3 Key combinations which determine at least one non-key attribute are keys of new entity sets
 - 4 Key combinations which do not determine at least one non-key attribute can possibly be omitted
 - 5 Naming of the new entity sets / entity types
- Knowledge of the logical structure necessary

2. NF – Example

CDNo	AlbumTitle	Release	TrackNo	TrackTitle
4001	Swagger	2000	1	Salty Dog
4001	Swagger	2000	2	Selfish Man
4002	Drunken Lullabies	2002	9	Death Valley Queen

- TrackTitle depends on TrackNo and CDNo ⇒ ok
- AlbumTitle and Release only depend on CDNo ⇒ not in 2. NF

2. NF – Example – Solution

<u>CD#</u>	<u>AlbumTitle</u>	<u>Release</u>
4001	Swagger	2000
4002	Drunken Lullabies	2002

<u>CD#</u>	<u>Track#</u>	<u>Tracktitle</u>
4001	1	Salty Dog
4001	2	Selfish Man
4002	9	Death Valley Queen

3. NF

- A relation is in 3. NF if it is in 2. NF *and*
- if no non-key attribute is transitive dependent on a candidate key

- Transitive dependent attributes are moved to separate tables, because they are only indirectly dependent on the candidate key

3. NF – Example

Desc	Type	Tires	HP
Käfer	PKW	4	45
Hanomag II	LKW	4	250
Vespa	Roller	2	4
Suzuki GX	Motorrad	2	150
Ford Focus GT	PKW	4	65

- How to achieve 3. NF?

3. NF – Example – Solution

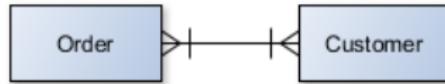
<u>Desc</u>	Type	HP
Käfer	PKW	45
Hanomag II	LKW	250
Vespa	Roller	4
Suzuki GX	Motorrad	150
Ford Focus GT	PKW	65

<u>Type</u>	Tires
PKW	4
LKW	4
Roller	2
Motorrad	2

Example Table

Order#	Prod#	ProdDesc	Amount	OrderDate	Cust#	Name	ZIP	City
1	4, 5	HDMI-Cable, TV	15, 550	2016-01-02	1	Maier	4020	Linz
2	5, 9	TV, Receiver	550, 250	2016-01-05	1	Maier	4020	Linz
3	4	HDMI-Cable	15	2016-01-07	2	Mueller	5020	Salzburg

- 3 Orders
- 3 Products
- 2 Customers
- All in one table



Example Table – Normalized – I

Order		
<u>Order#</u>	<u>Cust#</u>	OrderDate
1	1	2016-01-02
2	1	2016-01-05
3	2	2016-01-07

Customer		
<u>Cust#</u>	Name	<u>ZIP</u>
1	Maier	4020
2	Mueller	5020

City	
<u>ZIP</u>	City
4020	Linz
5020	Salzburg

Product		
<u>Prod#</u>	Name	Price
4	HDMI-Cable	15
5	TV	550
9	Receiver	250

Example Table – Normalized – II

<i>OrderItem</i>		
<u>Order#</u>	<u>Prod#</u>	<u>Amount</u>
1	4	1
1	5	1
2	5	1
2	9	1
3	4	1

- Amount has been added to allow for multiple items of the same product in the same order
- Note: Amount vs. Price

Boyce Codd NF (BCNF)

- A relation is in BCNF if it is in 3. NF *and*
- if there are no non-trivial functional dependencies of attributes on anything other than a superset of a candidate key

- Basically an extension of the 3. NF
- ⇒ we want to eliminate all overlapping candidate keys

BCNF – Determinates

Definition

An attribute A is called a determinate if another attribute B is fully functional dependent on A

- A relation is in BCNF if every attribute which is a determinate can be used as a key
- A relation is in BCNF if all determinates are also candidate keys

BCNF – Example

<i>BillPos</i>			
<u>Bill#</u>	<u>CostType</u>	<u>CostType#</u>	<u>Amount</u>
1	Material	2	600
1	Personnel	1	1200

- Candidate keys:
 - Bill# & CostType
 - Bill# & CostType#
- CostType is fully functional dependent on CostType# \Rightarrow CostType# is a determinate
 - But the determinate is *not* a candidate key, but only part of one

BCNF – Example – Solution

<i>BillPos</i>		
<u>Bill#</u>	<u>CostType</u>	<u>Amount</u>
1	Material	600
1	Personnel	1200

<i>CostTypes</i>	
<u>CostType</u>	<u>CostType#</u>
Material	2
Personnel	1

4. NF

- A relation is in 4. NF if it is in BCNF *and*
- if it contains at most one non-trivial multi valued dependency

- Multi valued dependency:
 - $X \twoheadrightarrow Y$
 - If for two attribute sets X & Y it is true that a subset of values of Y is assigned one value of X independently of other values there exists a multi valued dependency between X and Y
- A multi valued dependency of Y from X is trivial if Y is part of X or if the relation consists only of X & Y

4. NF – Example

<i>StudentInfo</i>		
<u>Student#</u>	<u>Subject</u>	<u>Activity</u>
100	Music	Swimming
100	Accounting	Swimming
100	Music	Tennis
100	Accounting	Tennis
110	Maths	Jogging

- All attributes are part of the primary key
- If student 100 takes up skiing as well, we'd have to insert two rows:
 - Music - Skiing
 - Accounting – Skiing

4. NF – Example – Solution

<i>StudentSubjects</i>	
<u>Student#</u>	<u>Subject</u>
100	Music
100	Accounting
110	Maths

<i>StudentActivities</i>	
<u>Student#</u>	<u>Activity</u>
100	Swimming
100	Tennis
110	Jogging

5. NF

- A relation is in 5. NF if it is in 4. NF *and*
- if for each join dependency (R_1, \dots, R_n) it is true that
 - The join dependency is trivial *or*
 - Every R_i of (R_1, \dots, R_n) is a candidate key
- A join dependency exists if a table can be 'recomposed' by joining together tables it has been split up into
- Simplified: A relation is in 5. NF if it is in 4. NF and cannot be split up any further without losing information

Minimality

- During normalization we decomposed our tables into sub tables
- If there are a lot of relations to start with this can lead to many tables after normalization
- Yet an increased number of tables leads to a higher complexity

- So we aim for minimality: the minimal number of tables which all fulfill the normal forms
 - ⇒ Do not create unnecessary tables

Charateristics

Schema property	Characteristic
1. NF	only atomic attributes
2. NF	no partial dependency of a non-key attribute from a key
3. NF	no transitive dependency of a non-key attribute from a key
BCNF	no transitive dependency of an attribute from a key
Minimality	no unnecessary relations while fulfilling the NFs

Summary

- 1., 2., 3. & BCNF are relevant
 - 1. NF: functional dependent on the superkey
 - 2. NF: no functional dependencies on parts of the key
 - 3. NF: no functional dependencies on non-key attributes
- 4. & 5. are nice to have

Principle

'Each attribute is placed in an entity where it is dependent on the key, the whole key and nothing but the key ... so help me, Codd!'

The inner workings of Oracle

Markus Haslinger

DBI



Agenda

1 Instances & Memory Structures

2 User Management

3 Error Handling

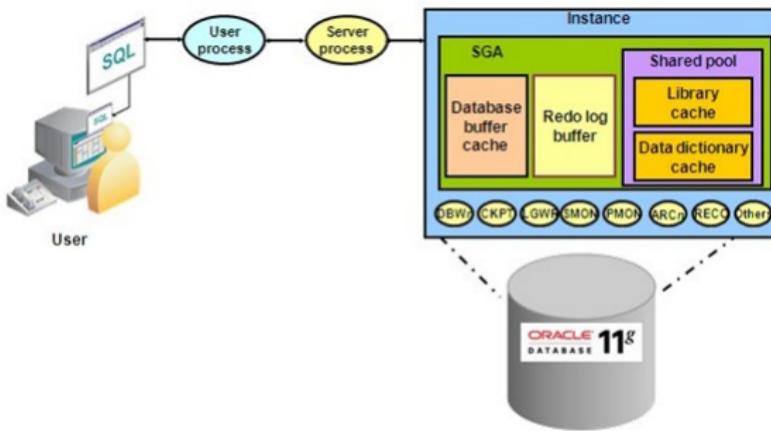
4 Backup & Recovery

Connection & Session

- *Disclaimer: Most of the following information relates to the OracleXE version which is free to use*
- Connection
 - The physical communication channel between client and server
 - Network (usually) or IPC
- Session
 - Logical unit in the memory of a DB instance
 - Represents current state of a user login
 - Exists from login until connection is ended
- One Connection can be used for 0-n sessions
- Sessions are separated from each other no matter which connection is used

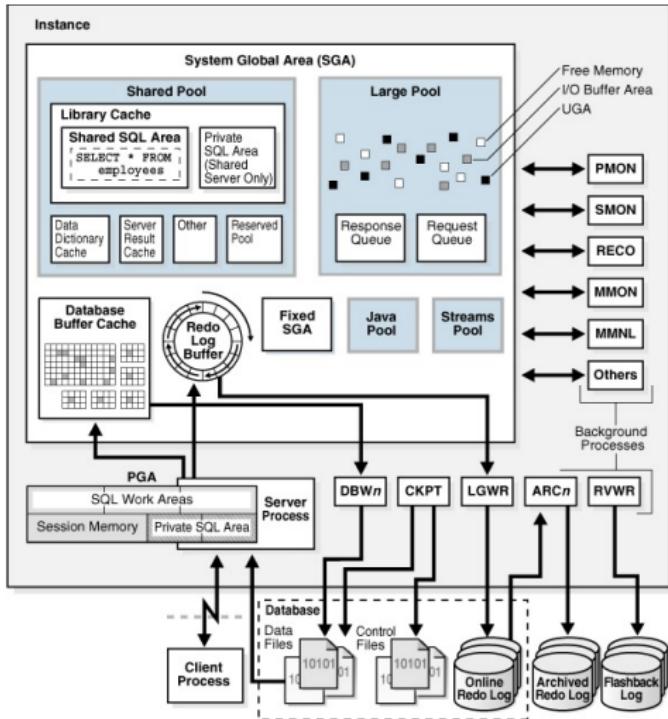
Instance & Database

- The database contains the physical data files
- Several instances can share one database



Source: <http://itsiti.com/interacting-with-an-oracle-database>, 2020-05-18

Instances & Memory Structures



Source:

https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT914,
2020-05-18

Definitions

■ System Global Area (SGA)

- Collection of shared memory structures containing data and control information for a DB instance
- The SGA is shared between all server and background processes
- Contains cached data blocks and shared SQL areas

■ Program Global Area (PGA)

- A memory area which provides data and control information for a server or background process
- PGA is not shared, each process gets its own

■ Large Pool

- Optional component
- Configurable memory area for:
 - Transactions which are distributed over several databases
 - IO-Server processes, Backup & Restore,...

Definitions

■ Database Buffer Cache

- Contains copies of data blocks which were read from data files
- If a user requests data the buffer cache is hit first
- If data is not available in the cache it will be loaded
- The buffer is managed with complex algorithms like Least Recently Used (LRU) or Touch Count

■ Redo Log Buffer

- Contains information about changes to the data
- Used to recover after a crash
- When changes are written to the buffer cache they are also automatically written to the redo log buffer
- The LGWR background process writes the redo log to the file system

Shared Pool Components

- Data Dictionary Cache
 - A collection of tables, views and their structure and users
 - Used while parsing SQL statements, used often ⇒ cached
- Library Cache
 - Every executed SQL statement is stored in the shared SQL area
 - Contains parse tree and execution plan for the statement
- SQL Query Result Cache
- PL/SQL Function Result Cache

Server Process

■ Server process

- Started as counterpart for a user process
- Tasks:
 - Parse and execute SQL statements
 - Read data blocks and save them to the buffer cache
 - Return results

Background Processes

■ Database Writer Process (DBWn)

- Writes the content of the buffer cache to the file system
- In the SGA the *Redo Byte Address* (RBA) is stored which is a pointer to the position at which a recovery would have to start. This pointer is written to the control files by the CKPT process.

■ Log Writer Process (LGWR)

- Manages the redo log buffer and writes its content into the redo log file
- The log is written to the file system if:
 - When a transaction is committed
 - When the redo log buffer is filled $\geq \frac{1}{3}$
 - Before DBWn writes changes files to the disk (which always waits for LGWR to complete)
 - Every three seconds

Background Processes

- Checkpoint Process (CKPT)
 - Defines a system change number (SCN) which is stored in the control files and in the data file headers
 - When the DBWn process writes to files it updates the SCN in the header, thus confirming that all changes up to this point are present in the data file
- System Monitor Process (SMON)
 - Performs a recovery at the start of an instance if necessary
 - Clears temporary memory segments which are no longer needed
- Process Monitor Process (PMON)
 - Responsible for cleaning the various database buffer caches and other resources if a user process fails
 - Also checks for idle session timeouts
- Recoverer Process (RECO)
 - Required for distributed (multiple systems) databases

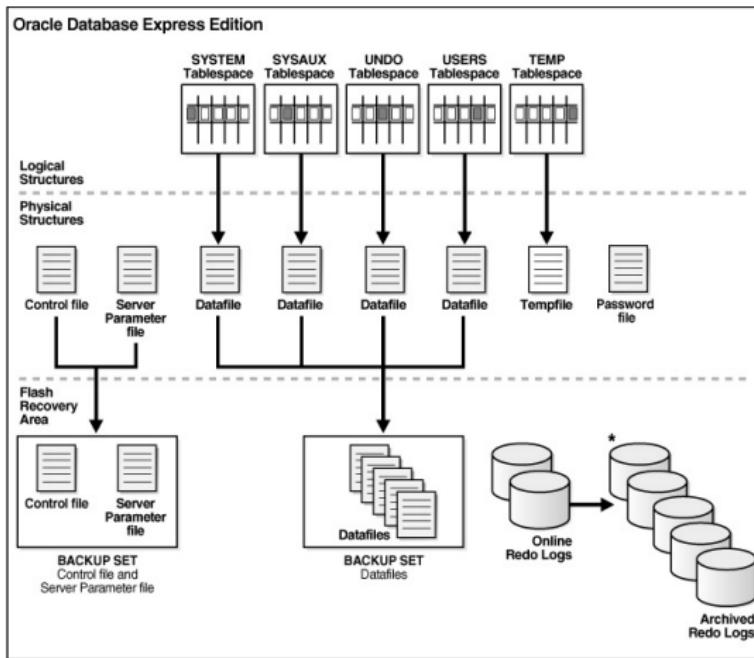
Background Processes

- Archiver Process (ARCn)
 - Copies redo log file after a log switch
 - Only active if ARCHIVELOG mode is enabled
- Manageability Monitor Process (MMON)
 - Notifies if various thresholds are hit
 - Creates statistical information for changes SQL objects
- Lightweight Manageability Monitor Process (MMNL)
 - Performs regular tasks like calculating statistics, session history,...
- Memory Manager Process (MMAN)
- Rebalance Process (RBAL)
- Automatic Management Process (ASMB)
- Queue Monitor Process (QMNC)

Memory Structures

- Logical Structures
 - e.g. Tablespace (only known to the DB, not the OS)
- Physical Structures
 - Files containing the data; visible to the OS
- Recovery Structures
 - e.g. Redo-Logs, DB-Backups
 - Used to recover data if a fault occurs
 - Automatically managed in a so called Flash Recovery Area within the memory

Instances & Memory Structures



* Archived Redo Logs present only after turning on log archiving (ARCHIVELOG mode)

Source:

http://docs.oracle.com/cd/E17781_01/server.112/e18804/storage.htm#ADMQS185,
2020-05-18

Definitions

■ Database

- Collection of logical and physical structures
- Contains all data and metadata

■ Tablespace

- 1 DB consists of 1..n tablespaces
- Logical grouping of one or more datafiles (tempfiles)
- Three types
 - Permanent (e.g. USERS)
 - Temporary (e.g. used for a SORT)
 - Undo (e.g. for Rollback, Read Consistency)
- Examples
 - SYSTEM (e.g. DataDictionary, Admin Tables)
 - SYSAUX, TEMP, UNDO, USERS

Definitions

- Datafile & Tempfile
 - Files in the file system.
 - Use a proprietary format
- Control File
 - Contains the names and paths of the physical DB components
 - Also contains some admin information, e.g. for Backup-Files
- Server Parameter File (SPFILE)
 - Contains initialization parameters
 - Uses a binary format
 - Controlled via ALTER SYSTEM commands
- Password File
 - Contains the password of the SYS user

Flash Recovery Area – Content

■ Backups

- Oracle's Backup & Recovery strategy is based on physical files and not on database objects (like tables)

■ Online Redo Logs

- Contain all changes in the database (~ Transaction Log)
- Used to recover data in a fault situation (e.g. power loss)

■ Archived Redo Logs

- Redo Logs can only contain a certain amount of data
- It is possible to archive them before recycling
- Online and Archived Redo Logs together contain all changes since the last backup

Redo Logs

- Important for recovery functionality
- Every change of data in the database is logged
 - e.g. INSERT, UPDATE, DELETE *not* SELECT
- There is a certain amount of redo log files
- If one file is full (= reached a certain size) the next one is used
- Once all are full content of the first one is overwritten
- It is possible to use *multiplexing* to keep identical redo logs on different disks to increase security

Archived Redo Logs

- If enabled the Archiving background process creates copies of the redo logs once they are full (ARCHIVELOG Mode)
- Allows for recovery in a fault scenario even if the last backup has been created some time ago
- This setting (if enabled) allows for online backups, otherwise the database has to be shut down to create a backup!

Database Startup

- The startup of a database moves through several states
 - 1 SHUTDOWN: Database is stopped
 - 2 NOMOUNT: Instance has been started
 - 3 MOUNT: Control files has been loaded and database has been mounted
 - 4 OPEN: All files are open, connections are possible

Database Shutdown

- There are four different ways to shutdown the database:
 - `shutdown`; (normal)
 - `shutdown transactional`;
 - `shutdown immediate`;
 - `shutdown abort`; (careful - does *not* abort a shutdown!)
- The abort option leads to an inconsistent database state ⇒ recovery necessary for next startup

	Normal	Transactional	Immediate	Abort
Allows new connections	✗	✗	✗	✗
Waits for sessions to complete	✓	✗	✗	✗
Waits for transactions to complete	✓	✓	✗	✗
Creates checkpoint and closes files	✓	✓	✓	✗

User Account

- Used to manage rights (grants)
- A user can also be an application
- Every account has the following attributes:
 - Unique user name (max. 30 chars)
 - Authentication method, e.g. password (or wallet)
 - Default tablespace for database objects
 - Default temporary tablespace
 - Profile, privileges and roles
 - Account status

Create User – Example

Create User

```
create user testuser
identified by geheim -- password
default tablespace testts
temporary tablespace temp
profile default
account unlock -- the account is not locked
quota 1M on sjm; -- the user may use 1MB of (disk) space
```

User Schema

- For each user a schema is created automatically
- A schema is a logical container for database objects (tables, trigger, functions,...)
- When accessing schema objects of another user the schema name has to be prefixed
 - Of course appropriate rights are required
 - This allows every object to have a schema-unique name

Privileges

- A privilege is a single right (grant)
- There are two types:
 - 1 System Privileges
 - Rights to perform a specific operation
 - For example: create table, delete row,...
 - 2 Object Privileges
 - Rights to perform operations on specific schema objects
 - For example: deleting data from table Employee
- There are more than 100 privileges
 - See <https://docs.oracle.com/database/121/TTSQL/privileges.htm#TTSQL338> (2020-05-18)

Privileges – Statements

Grant & Revoke

```
GRANT CONNECT TO testuser; — grants connect privilege to testuser
REVOKE CREATE TABLESPACE FROM testuser; — removes create tablespace
grant from testuser
```

System & Object Privileges

```
select * from user_sys_privs;
select * from dba_sys_privs;
select * from session_privs;
select * from all_tab_privs;
select * from user_tab_privs;
select * from dba_tab_privs;
```

DING

Roles

- A role is a group of privileges
- With the intent of easing administration
- There are many predefined roles
 - DBA
 - AUTHENTICATEDUSER
 - ...

Using Roles

Create and assign Role

```
create role app01_def;
grant select on a01.revenue to app01_def;
grant select, delete on a01.log to app01_def;
grant app01_def to testuser;
```

Selecting Role Information

```
select * from dba_roles;    -- Which roles are there?
select * from role_sys_privs; -- Which privileges does the role have?
select * from role_tab_privs;
select * from role_role_privs;
```

DING

Administrative Accounts

■ SYSTEM

- Used for all administrative tasks *except start & shutdown*

■ SYS

- The schema contains all tables and views of the Data Dictionary
 - Never modify!
- SYSDBA (connect sys as sysdba)
 - A right granted only to SYS which allows to start and stop the database

Error Categories

- The following types of errors can occur during database operation:
 - Erroneous Statements
 - Faulty User Processes
 - Network Errors
 - User Errors
 - Faulty Instances
 - IO Errors

Erroneous Statements

- Attempt to insert invalid data
 - Resolve together with developer/user
- Insufficient privileges
 - Modify rights (grants/roles)
- Memory allocation fails
 - Increase quota
 - Enlarge tablespace
- Logical programming error
 - Resolve together with developer

Faulty User Processes

- Connection interruptions
- Sessions not closed correctly
 - Possibly caused by a programming error
- The PMON process checks regularly if there are still open connections for registered sessions.
 - Usually the DBA does not have to intervene if the problem does not occur regularly/often

Network Errors

■ Issue with the Listener

- The Oracle listener is a service that runs on the database host and receives requests from Oracle clients
- Possibly configure a backup listener

■ NIC Issues

- Replace card or provide a second one as fallback

■ Network connection fails

- Try to find faults and bottlenecks in the network
- Possibly install a redundant backup line

User Errors

- User changed or deleted data by mistake
 - Rollback transactions
 - Reset tables (Flashback technology)
- User deleted table
 - Recover (Recycle Bin¹ or even Backup)

¹See <https://oracle-base.com/articles/10g/flashback-table-to-before-drop-10g>, 2020-05-18

Faulty Instances

- Causes:
 - Power failure
 - Hardware failure
 - Error in critical background process
 - Emergency Shutdown
- Resolutions:
 - Restart instance with STARTUP
 - Automatic Recovery using Redo Logs

Faulty Instances – Checkpoint

- At least every three seconds the CKPT writes data to the control files to document which information the DBWn has already persisted from the SGA to data files (checkpoint info in header and update of control files with checkpoint info)
- At these sections a possible recovery can start

Faulty Instances – Recovery

- During startup of an instance the system verifies if the SCN in the datafiles equals the one in the control files
- If they do *not* match the instance reads the redo logs and recreates the transactions until the data files are up to date
- Even open transactions (uncommitted) are restored, but immediately rollbacked (so these changes are not actually restored)

IO Errors

- Causes:
 - Disk Error
 - Faulty disk controller
 - Deletion/Corruption of database files
- Resolution:
 - Restore affected files from backup
 - Configure new storage locations
 - Recover data with redo information
- Multiplexing of control files is a good idea!

Offline Backup – NOARCHIVELOG

- Also called *COLD BACKUP*
- All required files are copied from the main disk to a backup disk (OS level)
- Requires the database to have been shutdown normally
 - Only possible if no 24/7 operation is necessary
- What has to be included in the backup:
 - Control file(s) (*.ctl or control.dbf)
 - Data files (*.dbf, tablespace files)

Offline Backup – NOARCHIVELOG – Recovery

- Recovery Options:

- Copy files back from the backup media
- Restore database to the state it had when the backup was taken

- What happens to changes since the last backup?

- If changes are still in the redo log recovery can be performed
- If not changes since the last backup are *lost!*

Offline Backup – ARCHIVELOG

- Activate ARCHIVELOG mode
 - `alter database archivelog;`
 - Check log settings using `select name, log_mode from v$database`
- Now the archived redo logs will contain all transactions since the last backup
 - Full recovery possible!

Online Backup

- Usually 24/7 operation is required
- Thus, no shutdown of the database for backups is possible
- Solution: HOT backups
- Only possible in the ARCHIVELOG mode
- Backup is created per tablespace
- While in Backup mode no changes are written to the tablespaces backing files

Backup Tablespace Example

```
ALTER TABLESPACE users01 BEGIN BACKUP;  
host  
cp /u01/app/oracle/..../users01.dbf /backup/  
exit  
ALTER TABLESPACE users01 END BACKUP;
```

DING

Online Backup – Recovery

- To perform an online recovery the following steps are required:
 - 1 Shutdown instance
 - 2 Copy back necessary files (from backup location)
 - 3 Startup mount;
 - recover automatic database; or
 - recover from '/backuppPath/' database;

Control File Backup

- If the tablespace configuration has changed (e.g. new data file added) the control file should be backed up so that the new datafile is known in case of a recovery

Backup Control File

```
ALTER DATABASE BACKUP CONTROLFILE TO '/backup/control.bkt';
```

Recovery

```
SHUTDOWN IMMEDIATE;  
-- copy file(s)  
STARTUP MOUNT;  
RECOVER DATABASE USING BACKUP CONTROL FILE;  
ALTER DATABASE OPEN RESETLOGS;
```

DING

RMAN – Recovery Manager

- Powerful utility² for backup & recovery of databases
- Shipped with Oracle
- Allows for incremental backups:
 - Level 0: complete backup
 - Level 1: all blocks which have been changed since last Level 0 or Level 1 backup
 - Level 2: all blocks changed since last Level 0, 1 or 2 backup
 - Level 3: all blocks changed since last Level 0, 1, 2 or 3 backup

²See https://docs.oracle.com/cd/E11882_01/backup.112/e10642/toc.htm, 2020-05-18

3.3.5 Die ORACLE-Hintergrundprozesse

Beim Hochfahren einer ORACLE-Instanz werden zahlreiche Prozesse gestartet, die ihre Arbeit im Hintergrund verrichten. Die folgende Aufzählung gibt einen Überblick über diese Prozesse. Für die späteren Untersuchungen sind nur diejenigen interessant, die hinsichtlich ihres Einflusses auf die Performance konfigurierbar sind, dazu zählen der DBWR-, der LGWR-, der CKPT- und der ARCH-Prozeß. Insbesondere die ersten drei der genannten Prozesse stellen die Verbindung zwischen der SGA und den Dateien auf der Festplatte her. Während jeder einzelne Serverprozeß (s. S.44) für das Lesen aus den Daten-Dateien und das Einlagern in die SGA verantwortlich ist, übernehmen nun die Hintergrundprozesse das Auslagern und damit das Schreiben in die Daten-Dateien und auch der Redo-Log-Dateien gemeinsam unabhängig von den jeweiligen Benutzerprozessen.

Der DBWR-Prozeß

Der Database Writer-Prozeß (DBWR) ist für alle Schreiboperationen vom DB Buffer Cache in die Daten-Dateien verantwortlich. Alle Blöcke, die sich im Puffer befinden und die verändert wurden (sog. *dirty blocks*), müssen zurück in die Daten-Dateien geschrieben werden, wenn Pufferplatz benötigt wird.

Dies geschieht nach einem Least Recently Used (LRU) Algorithmus, d.h. diejenigen Blöcke, die am längsten nicht mehr referenziert wurden, werden aus dem Puffer entfernt. Der DBWR-Prozeß arbeitet unabhängig vom Status der mit den einzelnen Blöcken assoziierten Transaktionen, d.h. es werden auch Blöcke und damit Daten, deren Zustand ungewiß ist (engl. *uncommitted data*), auf die Platte geschrieben. Die Sicherung der Konsistenz wird durch Rollback-Segmente und eine Markierung der entsprechenden Blöcke auf der Platte realisiert.

Datenblöcke werden vom DBWR auf die Platte geschrieben, wenn eine der folgenden Bedingungen erfüllt ist:

- Ein Serverprozeß meldet, daß sich mehr als DB_BLOCK_WRITE_BATCH / 2 Puffer in der dirty list befinden.
- Ein Serverprozeß untersucht DB_BLOCK_MAX_SCAN_CNT Puffer, ohne einen freien zu finden.
- Ein Time-Out findet statt (dies passiert alle 3 Sekunden)⁷.
- Ein Checkpoint (s. Abschnitt 3.3.5) findet statt, und der LGWR (s. Abschnitt 3.3.5) über gibt eine Liste zu schreibender Puffer an den DBWR. Jeder Checkpoint hat eine eindeutige Nummer, die sog. *System Checkpoint Number (SCN)*.

Arbeitet der DBWR nicht schnell bzw. effizient genug, so entstehen Konflikte und Wartezeiten beim Zugriff auf den DB Buffer Cache, sollten andere Prozesse freien Speicherplatz allokiieren wollen.

Der LGWR-Prozeß

Der Log Writer-Prozeß (LGWR) dient — wie der Name schon sagt — dazu, Teile des Redo Log Buffer in die Redo-Log-Dateien auf Platte zu schreiben. Der LGWR schreibt

- Redo-Puffer, wenn ein Benutzer eine Transaktion durch ein Commit permanent macht.
- Redo-Puffer alle 3 Sekunden, wenn ein Time Out auftritt.
- Redo-Puffer, wenn der Redo Log Buffer zu einem Drittel gefüllt ist.

⁷Bis jetzt haben wir keine Möglichkeit gefunden, diese Zeitspanne zu beeinflussen, obwohl das sicherlich ganz interessant wäre.

- Redo-Puffer, wenn der DBWR schreibt.

Der LGWR schreibt zur gleichen Zeit in die aktive gespiegelte Gruppe von Online-Redo-Log-Dateien. Sollten alle Dateien einer Gruppe defekt sein oder ist die Gruppe wegen fehlender Archivierung nicht verfügbar, wird der LGWR abgebrochen. Wenn manchmal mehr Pufferplatz benötigt wird, schreibt der LGWR Redo-Log-Einträge, bevor eine Transaktion festgeschrieben wurde. Diese Einträge werden nur dann dauerhaft gespeichert, wenn die Transaktion später festgeschrieben wird (mit einem *commit*).

Ist der Checkpoint-Prozeß (s. Abschnitt 3.3.5) nicht aktiviert, so ist der LGWR auch dafür verantwortlich, bei einem Checkpoint die Header der Daten-Dateien zu ändern. Hierbei wird die aktuelle SCN im Header vermerkt, um bei einer eventuellen Rekonstruktion das Alter der Daten ermitteln zu können.

Der CKPT-Prozeß

Ist der Checkpoint-Prozeß (CKPT) aktiviert, so übernimmt im Falle eines Checkpoints CKPT die Arbeit des LGWR, die Header der Daten-Dateien zu modifizieren, dies kann zu einer erheblichen Entlastung des LGWR und damit zu Performance-Steigerungen führen. Und es wird möglicherweise ein Datenbank- Stillstand vermieden, falls der LGWR mit dieser Tätigkeit überfordert wäre.

Checkpoints sorgen dafür, daß sämtliche modifizierten Blöcke regelmäßig aus dem DB Buffer Cache auf Platte in die Daten-Dateien geschrieben werden. Sonst könnte es nämlich passieren, daß sehr häufig referenzierte Blöcke auf Grund des LRU-Algorithmus des DBWR im Puffer „überwintern“, ohne auf Platte geschrieben zu werden.

Da alle Änderungen der Daten, die vor dem Checkpoint stattfanden, in den Daten-Dateien vermerkt sind, müssen beim Ausfall der Datenbank nur die Redo-Log-Einträge angewendet werden, die nach dem Checkpoint erzeugt werden.

Checkpoints treten in den folgenden Fällen auf:

- Es geschieht ein sogenannter *Log Switch*, d.h. es wird von einer Redo-Log-Datei auf die nächste gewechselt.
- Es wurden seit dem letzten Checkpoint mehr als LOG_CHECKPOINT_INTERVAL Blöcke auf Platte geschrieben.
- Seit dem letzten Checkpoint sind LOG_CHECKPOINT_TIMEOUT Sekunden verstrichen.
- Wird ein online Tablespace-Backup ausgeführt, so findet der Checkpoint nur für die Daten-Dateien dieses Tablespace statt. Dasselbe passiert, wenn der DBA einen Tablespace im normalen oder temporären Modus *offline* nimmt, d.h. den Benutzern den Zugriff auf diesen Tablespace verwehrt, z.B. im Rahmen von Wartungs- oder Backup-Maßnahmen.
- Schließt ein DBA die Instanz im Normal- oder Immediate-Modus — genauereres dazu in [Fee96] —, so findet ein Checkpoint statt.
- Ebenso kann der DBA selbst einen Checkpoint erzwingen, z.B. durch den entsprechenden Menüpunkt des ORACLE Server-Managers.

Der ARCH-Prozeß

Wird die Datenbank im ARCHIVELOG-Modus betrieben und ist außerdem automatisches Archivieren aktiviert, so werden gefüllte Redo-Log-Dateien durch den Archivierungs-Prozeß (ARCH) an einem festgelegten Ort gespeichert. Diese archivierten Redo-Log-Dateien können dann beim eventuellen Verlust von Daten-Dateien zusammen mit einem kompletten Backup der Datenbank zu einem früheren Zeitpunkt zur Wiederherstellung der Datenbank benutzt werden.

Sonst werden die benutzten Redo-Log-Dateien wieder überschrieben, wenn die anderen Dateien gefüllt sind. Die in ihnen gespeicherte Information geht somit unwiderbringlich verloren. Der ARCH-Prozeß, der sich eigentlich nur mit Dateien befaßt, die sich auf Sekundärspeichermedien befinden, benutzt für den Archivierungsvorgang einen Teilbereich des Hauptspeichers, den Log Archive Buffer (s. dazu S.39).

Der SMON-Prozeß

Der System-Monitor-Prozeß (SMON) führt die Wiederherstellung einer Instanz beim Hochfahren durch, falls diese ausgefallen sein sollte. Der SMON hat auch Einfluß auf die interne Speicherverwaltung, da er für das Aufräumen temporärer Segmente, die nicht mehr im Gebrauch sind, zuständig ist; zudem führt er benachbarte, freie Speicherbereiche zusammen, um so größere Bereiche mit freiem Speicherplatz zur Verfügung zu stellen. Der SMON-Prozeß wacht regelmäßig auf, um zu prüfen, ob er benötigt wird, und kann aufgerufen werden, wenn ein anderer Prozeß die Notwendigkeit des Einsatzes von SMON feststellt.

Der PMON-Prozeß

Der Process-Monitor-Prozeß (PMON) kommt zum Einsatz, falls ein Benutzerprozeß aus irgend-einem Grunde ausfallen sollte. Er führt dann die Prozeß-Wiederherstellung durch, räumt den Cache auf und gibt die Ressourcen frei, die dieser Prozeß benutzt hat. Die Aktivierung des PMON erfolgt auf analoge Weise wie die des SMON.

3.4 Die Abarbeitung einer SQL-Anweisung

Wir wollen nun anhand eines Beispiels einer SQL-Anweisung verdeutlichen, welche Prozesse innerhalb der Datenbank ablaufen, um diese Anweisung zu verarbeiten.

Wir werden zunächst die Benutzung der Strukturen des Primärspeichers erläutern, um dann auf die Benutzung des Sekundärspeichers einzugehen. Die einzelnen Schritte der Abarbeitung finden sich auch in Abbildung 3.12.

Als Voraussetzung gelte hier, daß ein Benutzer in einer Session angemeldet ist, d.h. dem der Session zugehörige Serverprozeß ist eine PGA im Speicher zugewiesen worden, deren Größe von der Anzahl der geöffneten Cursor bzw. der Sort Area abhängt. Der Benutzer schickt nun eine SQL-Anweisung an die Datenbank.

3.4.1 Die Benutzung des Primärspeichers während der Ausführung

1. Zunächst übernimmt der dem Benutzerprozeß zugeordnete Serverprozeß die Verarbeitung dieser Anweisung.
2. **OPEN** Es wird ein Cursor im PGA-Speicherbereich geöffnet.

Storage Media
oooooooo
oooooooooo

Secondary Storage
oo
ooooooo

Index
ooo
oooooo

B Tree
oooooooo
oooo

Hashing & Clustering
ooooooo
oooooo

R Tree
oooooo

Physical Data Organisation

Markus Haslinger

DBI/INSY

Storage Media
oooooooo
oooooooo

Secondary Storage
oo
ooooooo

Index
ooo
oooooo

B Tree
oooooooo
oooo

Hashing & Clustering
ooooooo
oooooo

R Tree
oooooo

Agenda

- 1 Storage Media
- 2 Secondary Storage
- 3 Index
- 4 B Tree
- 5 Hashing & Clustering
- 6 R Tree

Storage Media

●○○○○○○○
○○○○○○○○○○

Secondary Storage

○○
○○○○○○○

Index

○○○
○○○○○○○○

B Tree

○○○○○○○○
○○○○

Hashing & Clustering

○○○○○○
○○○○○○

R Tree

○○○○○○

Storage Media

Storage Levels

- 1 Primary Storage
- 2 Secondary Storage
- 3 Archive Storage

Primary Storage

- ⇒ RAM (system memory)
- Expensive
- Very fast
- Small size (compared to secondary storage)
 - Yet some databases demand huge amounts to run (mostly) in RAM only, e.g. HANA, which means immense investments
- Direct Access to individual blocks possible
- Not protected against faults
 - ECC (Error-correcting code) at most

Secondary Storage

■ Disks

- Assuming we are still using rotating HDD
- SSD storage closes the gap to RAM somewhat
- There might also be a level between RAM and HDD filled by SSDs
- Slower (ca. factor 10^5)
- Much bigger
- Access with low granularity (blocks)
- Relatively fault protected
 - Think RAID,...

Storage Media

○○○●○○○
○○○○○○○○

Secondary Storage

○○
○○○○○○

Index

○○○
○○○○○○

B Tree

○○○○○○○○
○○○○

Hashing & Clustering

○○○○○○
○○○○○○

R Tree

○○○○○○

Storage Media

Archive Storage

- Tapes
 - Yes, they are still in use
 - Cheap
 - Slow
 - Only sequential read/write
 - Important for backups

Storage Media

○○○●○○
○○○○○○○○

Secondary Storage

○○
○○○○○○

Index

○○○
○○○○○○

B Tree

○○○○○○○○
○○○○

Hashing & Clustering

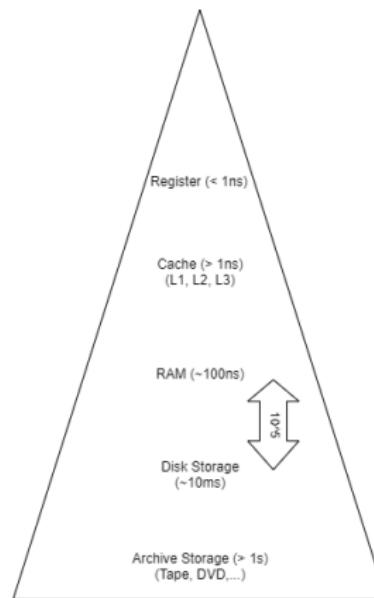
○○○○○○
○○○○○○

R Tree

○○○○○○

Storage Media

Storage Levels



Storage Media
○○○○○●○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

Storage Media

Latency Times

- Examples for data storage latencies¹:
 - L1 cache reference 1ns
 - Main memory reference 100ns
 - Reading 1MB sequentially from memory 150μs
 - SSD random read 17μs
 - Reading 1MB sequentially from SSD 2ms
 - Disk seek time 8ms
 - Reading 1MB sequentially from disk 9ms
 - Packet round trip within data center 500μs
 - Packet round trip Europe/USA 150ms

¹https://colin-scott.github.io/personal_website/research/interactive_latency.html, 2020-10-18

Disk Access

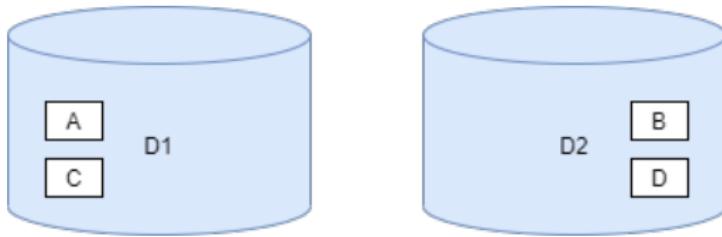
- Read times for rotating disks:
 - Seek Time $\sim 4ms$
 - Move arm to correct track
 - Latency (rotational delay) $\sim 3ms$
 - Wait until correct block moves by the header
 - $\frac{1}{2}$ rotation (10000 U/min)
 - Transfer data to primary memory (RAM) $\sim 150MB/s$

What is RAID?

- RAID = **R**edundant **A**rray of **I**nexpensive **D**isks
 - Inexpensive array is the key here ⇒ instead of using one expensive high performance drive we combine several slower but also cheaper ones
- Internally multiple drives are used in parallel
- To the outside world they are represented as a single drive (volume)
- We differentiate between 8 RAID Levels
 - Those are optimized for different scenarios
 - e.g. performance or reliability

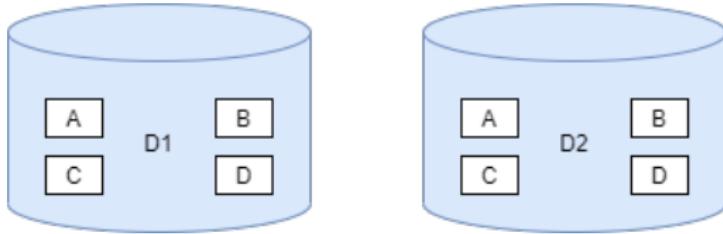
RAID 0 – Striping

- Data is split (in block-wise rotation) on multiple drives
- Reads are done in parallel, Controller combines data
- Very performant but also very low fault tolerance



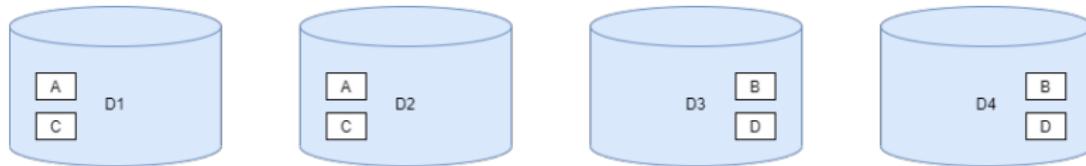
RAID 1 – Mirror

- Data is stored on all drives
- If one piece of data gets corrupted it can be read from the other disk
 - Very fault resistant
- To improve performance reads can be distributed
- Write operations are done in parallel



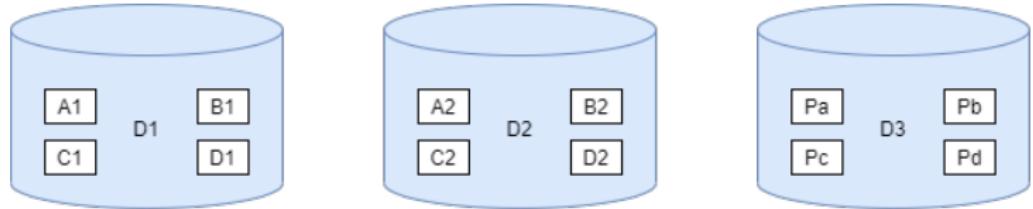
RAID 0+1 (RAID 10) – Striping + Mirror

- Data is stored on multiple drives
- Data is split on multiple drives
- Fast and very fault tolerant
- But twice as much storage space required



RAID 2 – Striping with Hamming Correction

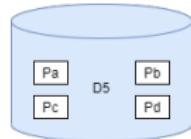
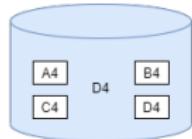
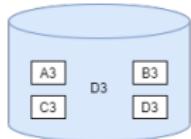
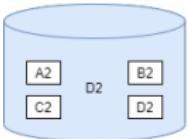
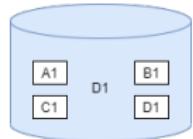
- Bitwise Striping with *Hamming Code*² error correction
- This configuration requires *parity* data
- For data which is split over multiple disks a checksum is calculated and stored
 - ⇒ requires additional disks for parity information storage
- If one drive dies the missing piece of data can be restored via the remaining pieces and the checksum
- *Rarely in use nowadays!*



²See https://en.wikipedia.org/wiki/Hamming_code

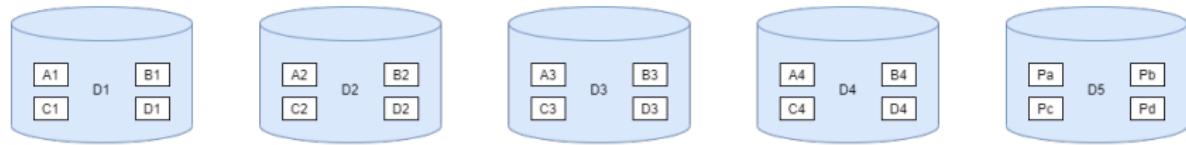
RAID 3 – Striping with Correction

- Bitwise or Bytewise Striping (e.g. below mod 4)
- This configuration requires *parity* data
- An additional disk is used to store parity information
 - XOR applied
 - 1 = odd number of bits
 - 0 = even number of bits
- During read (usually!) all disks except the parity disk are used
- During writes all disks are used



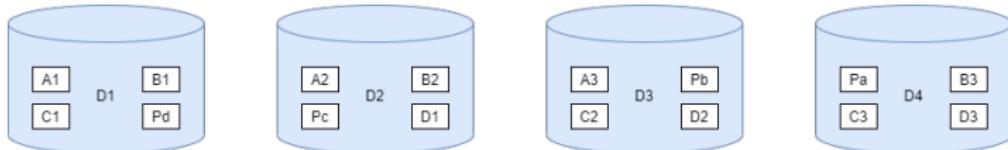
RAID 4 – Blockwise Striping

- Blockwise Striping
- More efficient when reading small pieces of data
- Write operation process:
 - 1 Read (existing) data block content
 - 2 Write new data block content
 - 3 Update parity
 - ⇒ all write operations need the *one* parity disk (bottleneck)



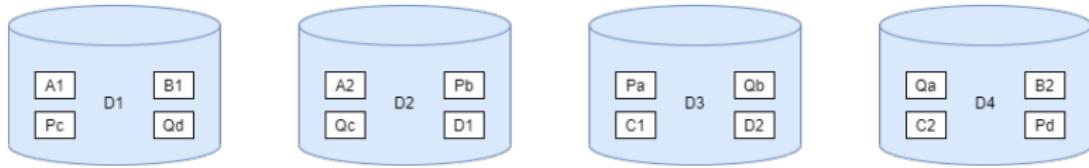
RAID 5 – Distributed Parity

- Similar to RAID 4
- But parity information is spread over all disks
- Write operation still expensive:
 - 1 Read (existing) data block content and related parity block
 - 2 Calculate new parity information based on old and new data
 - 3 Write data and parity blocks
 - But there is now not only a single parity disk ⇒ reduced bottleneck



RAID 6 – Double Distributed Parity

- Similar to RAID 5
 - But parity information is kept at two different disks
 - This allows up to two disks to fail at once



Database Buffer

- A database does not use data directly from the disk but transfers it to a *buffer* storage within the primary memory (RAM) first
 - Remember: data is stored and loaded in *pages*
- A page should be kept in primary memory as long as possible
 - Because it is *so* much faster
 - If a page is changed the update has to be written to the disk
- Once the buffer is full pages are replaced

Storage Media
○○○○○○○○
○○○○○○○○○

Secondary Storage
○●
○○○○○○○○

Index
○○○
○○○○○○○○

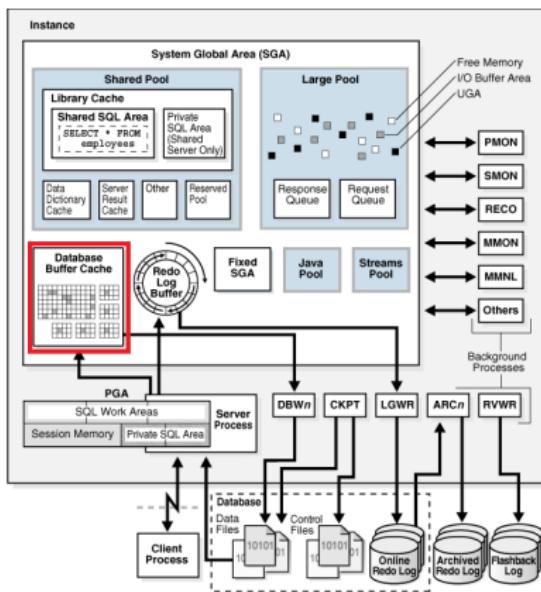
B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

Buffer

Database Buffer



Source: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT88788,

2020-10-24

HTL LEONDING

Storing Relations

- For a relation³ several pages are grouped into one file
- Tuples are stored in such a way that they do not spread over multiple pages
 - If there is not enough space left in a page a whole new one is created
- Every page contains a data row table with references to all tuples on this page

³Not between entities, but one entity set.

Storing Relations

- To uniquely identify a tuple⁴ a tuple identifier (TID) is used
 - It contains the page number and the row number in the reference table (of the page)
 - These TIDs can then also be used in an index
- This system of indirect references makes it easier to reorganize the stored data

⁴Remember: one tuple is a unique combination of attributes representing one entity.

Page Structure

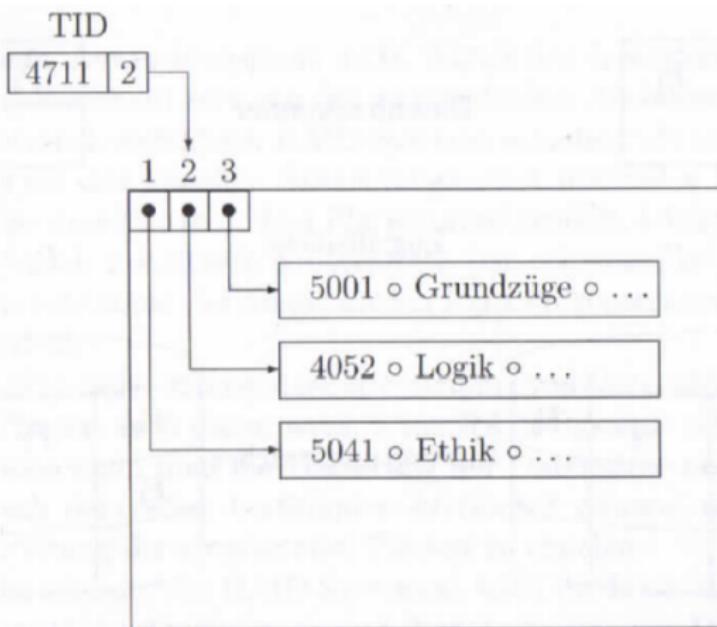
- Look at the following table⁵:

<i>Lectures</i>		
LectureID	Title	Syllabus
5001	Grundzüge	...
4052	Logik	...
5041	Ethik	...

- This relation has to be stored in (a) page(s) on the disk

⁵The following sample and some images have been taken from Kemper, A. & Eickler, A., Datenbanksysteme, 9th Edition, Oldenbourg Verlag, 2013, S. 220ff

Page – Relation



Seite 4711

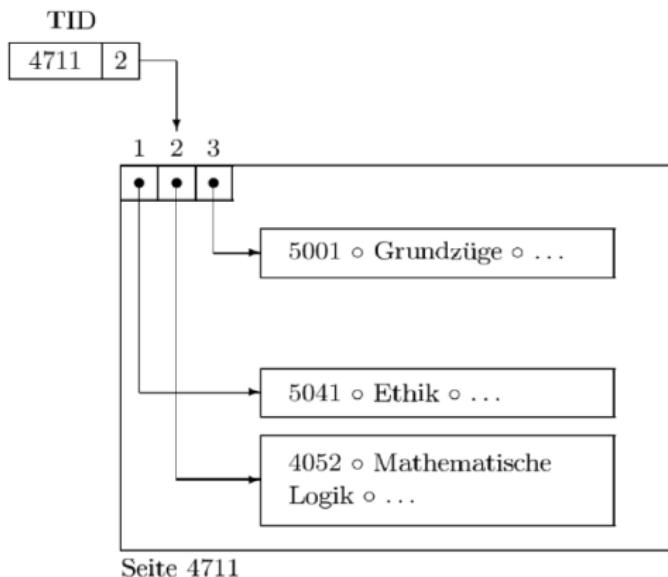
HTL LEONDING

Page – Move tuple within page

- Content of one tuple increases
- A tuple cannot grow in place
- If there is still enough space left on the page it can be moved there
 - This will update the row reference table of the page, but *not* the TID!

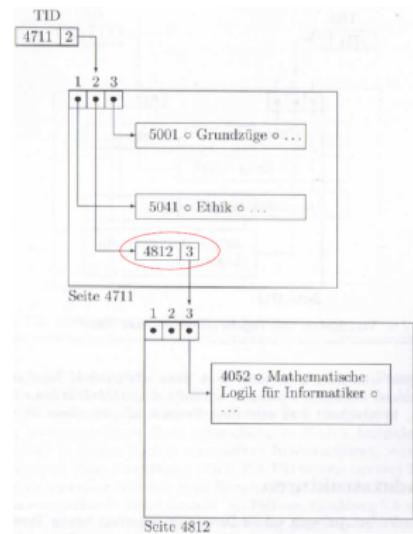
Block 1	Block 2-3	Block 4-n	Block n+1	Block n+2-n+3	Block n+3-m	Block m+1
4052	Logik	...	5041	Ethik

Page – Move tuple within page



Page – Move tuple to different page

- If there is not enough space left on the page the tuple is moved to a new page
- At the old position a reference to the new position is created
- Would the tuple move to yet another page the initial (first) reference is updated \Rightarrow no chained references



Index – General

- Rarely all tuples of a relation are required at once
- In these cases – to find the ones we need – the whole file(s) has to be searched
 - 'Full Table Scan'
- With an index a smaller, properly ordered ('jumps' possible) search is performed
 - Here we leverage the:
 - The direct access capabilities of the secondary storage
 - The TID reference structure
- However, an index requires space and cycles (creation & updates)

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○●○
○○○○○○

B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

Index

Index – Example

Index

Search Key	Pointer
Database indexes	
Intro to computers	
Intro to databases	
Intro to software	

Table

Title	Writer	Date
Intro to databases	Michele Clark	Dec 2, 2017
Database indexes	Adam Cambel	Nov, 14, 2016
Intro to computers	Nickolas Homes	Feb 5, 2018
Intro to software	Nicholas Robin	Feb 7, 2018

Source:

<https://quizlet.com/ca/306062283/sd-database-indexing-flash-cards/>,
2020-10-25

HTL LEONDING

Index – Primary & Secondary

- We differentiate between:
 - Primary Index⁶
 - Determines the order in which tuples are physically stored (clustering)
 - Secondary Index
 - There can be none, one or several
 - They duplicate some attributes taking up storage (in addition to the index structures)
 - Can be a 'simple' search index or a 'special' one like an unique index

⁶Often: Primary Key

Index-Sequential Access Method

- Groups data into related areas
- Via index the search jumps to the proper area then performs a detail search in it
- Index and data rows are stored in (sorted) order
- Index pages are stored in sequence on secondary storage
- The index pages contain references to the data page
- e.g. MySQL MyISAM-Index (default up to version 5.1)

Storage Media
○○○○○○○○
○○○○○○○○○○

Secondary Storage
○○
○○○○○○○○

Index
○○○
○●○○○○

B Tree
○○○○○○○○
○○○○

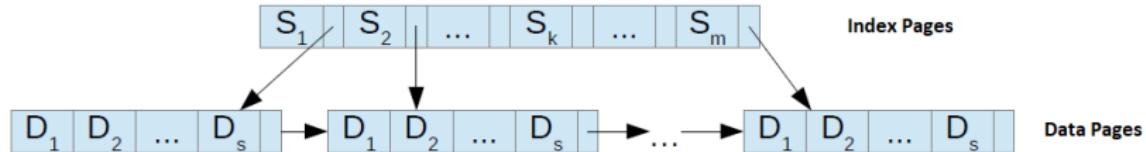
Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

ISAM

ISAM – Search

- 1 Binary Search within the index
- 2 Follow reference to data page
- 3 Search sequentially from there



Storage Media
○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○●○○○

B Tree
○○○○○○○
○○○○

Hashing & Clustering
○○○○○
○○○○○

R Tree
○○○○○

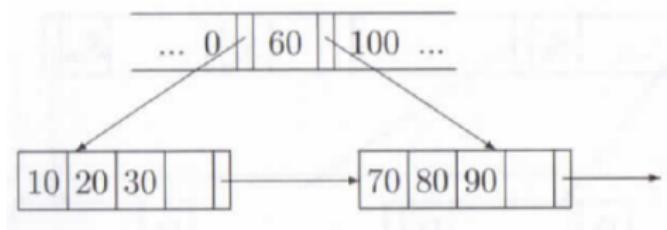
ISAM

ISAM – Insert

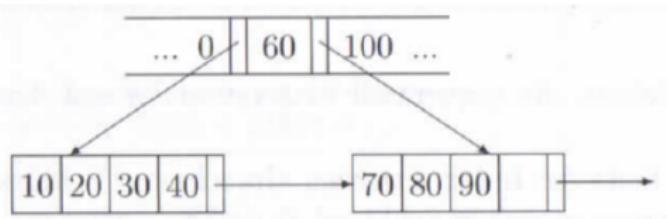
- Can be very complex if page is already full
- System has to balance between neighboring pages
- A new page has to be added if necessary
 - Which requires data to be moved around

ISAM – Insert – Example

■ Start State

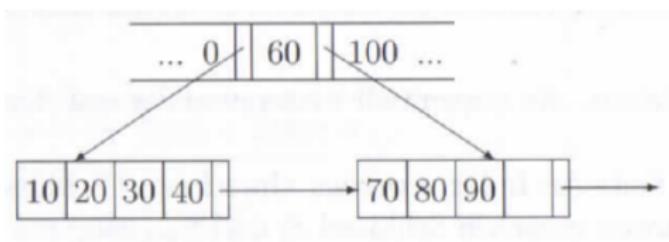


■ Insert 40

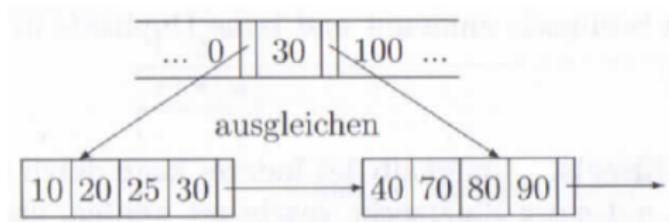


ISAM – Insert – Example

■ Start State



■ Insert 25



Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○●

B Tree
○○○○○○○○
○○○○

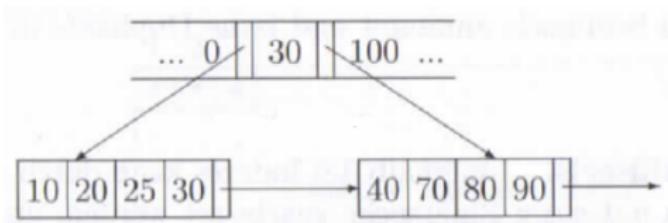
Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○

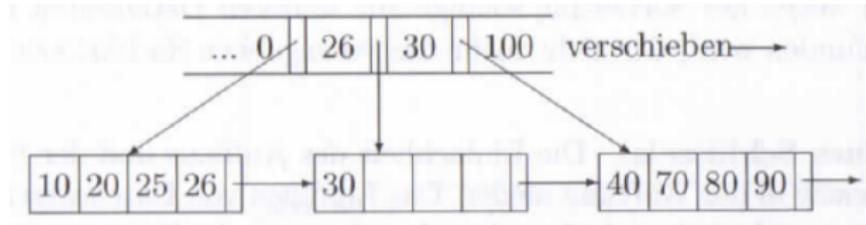
ISAM

ISAM – Insert – Example

■ Start State



■ Insert 26



B Tree

- B Tree is a different *index structure*
- The tree structure is optimized for secondary storage
- Each node has the same size as a page (1 node = 1 page)
- Page switch only required if following an edge
 - Max. number of page accesses equals height of the tree
- The tree is balanced
 - ⇒ every track between root and leaf has the same length
- Every entry consists of key S_i and data D_i
 - The data would be a TID in a secondary index
- Each node also contains a reference V_{i-1} to a node with smaller keys and V_i to a node with bigger keys

Storage Media

○○○○○○○
○○○○○○○○

Secondary Storage

○○
○○○○○○

Index

○○○
○○○○○○

B Tree

○●○○○○○○
○○○○

Hashing & Clustering

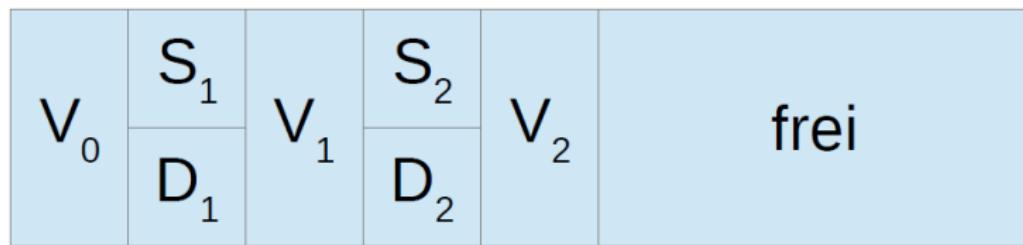
○○○○○○
○○○○○○

R Tree

○○○○○○

B Tree

B Tree – Memory Structure



B Tree – k Level Properties

- Balanced
- Every node – except the root – has at least k and max. $2k$ entries
- The root has one to $2k$ entries
- Entries are stored in sorted order within each node
- Every node with n entries – except for leafs – has $n+1$ children
- If S_1, \dots, S_n are the keys of a node with $n+1$ children and V_0, V_1, \dots, V_n the references to these children, then:
 - V_0 references the sub tree with keys smaller than S_1
 - $V_{i(i=1,\dots,n-1)}$ references the sub tree with keys between S_i and S_{i+1}
 - V_n references the sub tree with keys bigger than S_n
 - The leafs have uninitialized pointers

B Tree – Insert

- 1 Perform a search for the key
 - Will end at insert position
- 2 Insert key
- 3 If the node is now overfull we need to split:
 - Create a new node and put the keys of the overfull node which are bigger than the median
 - Move the median from the overfull node to its parent node
 - Set the right (bigger) reference of the parent node to the new node
- 4 If the parent node is now overfull:
 - If it is the root create a new root
 - Else repeat step 3 for the parent node

B Tree - Delete

- If the entry is in a leaf it can simply be deleted
- If it is within an inner node the next bigger (or smaller) key is located and put at its place
- If a node is then underpopulated it will be either balanced or merged with a neighboring node
 - A merge is only possible if both nodes are minimally populated
 - A merge results in a node which – in addition to the entries of the merged nodes – contains the a key of the parent node
 - This might result in an underpopulation of the parent node which then needs to be resolved

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○●○
○○○○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○

B Tree

B Tree – Secondary Storage

- Nodes in the tree are represented by pages
- Pages (at least of the inner nodes) should be kept in primary memory (buffer)
- The order is not deterministic (because the tree can mutate over time)
- A bit vector indicating free memory blocks is kept for performance reasons

Storage Media
oooooooo
oooooooo

Secondary Storage
oo
ooooooo

Index
ooo
oooooo

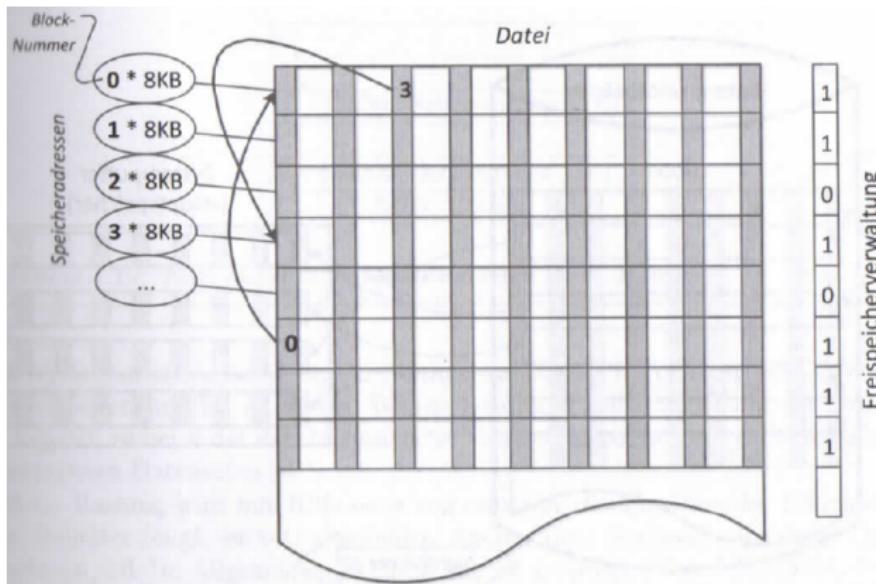
B Tree
oooooooo●
oooooo

Hashing & Clustering
ooooooo
oooooo

R Tree
oooooo

B Tree

B Tree – Secondary Storage



Source: Kemper, A. & Eickler, A.,
Datenbanksysteme, 9th Edition, Oldenbourg Verlag, 2013, S. 231

B+ Tree

- The number of page accesses is dependent on the number of branches in a tree
 - The more branches the smaller the height
- The B+ tree reduces its height by only storing data within leaf nodes
 - The inner nodes only contain keys
- Every search has to reach a leaf node
- To improve sequential operations the leaf nodes are (in addition to the tree structure) linked with the previous and next leaf nodes
- For the inner nodes reference keys (which are not actual keys of stored data) can be used which only need to be removed when nodes are merged

B+ Tree – Properties

- Every track from root to leaf has the same length
- Inner nodes (k) and leaf nodes (k^*) have a different number of entries
 - Called a (k,k^*) tree
- Every inner node has at least k and at most $2k$ entries
- Leafs have at least k^* and at most $2k^*$ entries
- The root has either at most $2k$ entries or it is a leaf at the same time with at most $2k^*$ entries
- Every node with n entries – except leafs – has $n+1$ children

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○●○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

B+ Tree

B+ Tree – Properties

- If R_1, \dots, R_n are the reference keys of an inner node (can be the root) with $n+1$ children and if V_0, V_1, \dots, V_n are the references to these children, then:
 - V_0 references the sub tree with keys smaller than or equal to R_1
 - $V_{i(i=1,\dots,n-1)}$ references a sub tree which has a key between R_i and R_{i+1} (including R_{i+1})
 - V_n references the sub tree with keys bigger than R_n

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○●

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○○○

B+ Tree

B+ Tree – Prefix

- One optimization for a B+ tree is to use key prefixes instead of complete keys
- A key prefix only has to divide left and right side of a tree without being a complete key
- The goal is to 'create' more keys, thus increasing the number of branches
- For example: use just the first letter instead of the whole string for alphanumeric keys

Hashing

- The goal is to find data with 1-2 page loads on average (in a DB context)
- Similar to a (Hash)Map implementation
- The hash function ($h : S \rightarrow B$) projects keys to buckets
 - S... Key set of any size
 - B... Index of n buckets $\rightarrow [0..n]$
 - $|S| >> |B| \Rightarrow h$ is not an injective function⁷
- We want to achieve a homogeneous distribution across all buckets
 - Often used: modulo

⁷https://en.wikipedia.org/wiki/Injective_function

Static Hashing

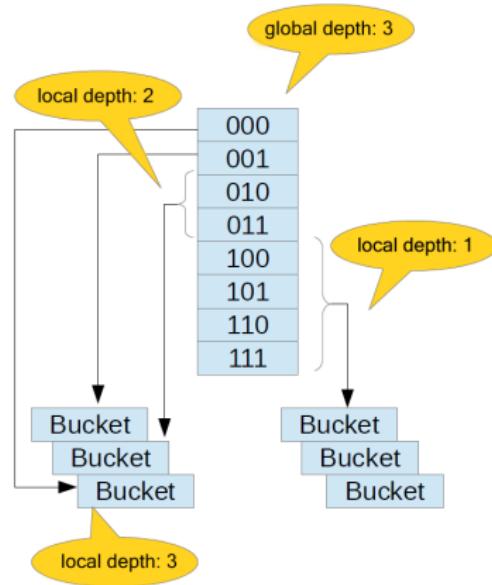
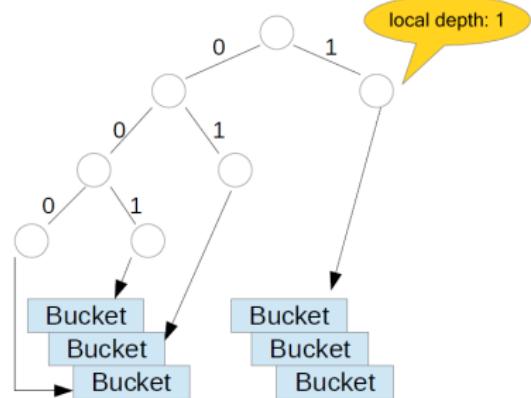
- At the beginning every bucket contains one page
- Once the page overflows an additional page is added to the bucket (reference link)
- If a bucket contains several additional pages the search becomes more expensive
 - $O(1) \rightarrow O(n)$
- If too many buckets are used to begin with space is wasted
- Solutions:
 - 1 Enlarging the hash table
 - Requires rehashing of all entries \Rightarrow very expensive for large amounts of data
 - 2 Extendable Hashing

Extendable Hashing

- The hash function is changed in such a way that it projects on a significantly larger area than the number of buckets
- The result of the hash function is represented in a binary notation
 - Only a prefix is used: $h(x) = dp$ (d... position, p... unused)
- Access to the buckets can be represented as (unbalanced) binary decision tree

- Global Depth: currently used number of bits in the hash values
- Local Depth: length of the path pointing to a specific bucket

Extendable Hashing



Extendable Hashing – Hash Function

- $h : KeySet \rightarrow \{0, 1\}^*$
- The Bit string has to be long enough to project all objects on their respective bucket
- We start with a short prefix (few bits)
- When the hash table grows longer and longer prefixes are required
 - Dictionary doubles in size every time

Hashing in Oracle

- 150 = Number of hash values
 - Will be automatically rounded up to the next prime

Cluster Example

```
Create CLUSTER trial_cluster (trailno NUMBER(5,0))
TABLESPACE users
STORAGE (
    INITIAL 250K NEXT 50K
    MINEXTENTS 1 MAXEXTENTS 3
    PCTINCREASE 0)
HASH IS trailno HASHKEYS 150;
```

Clustering

- Data which is regularly used together should be stored closely together on the secondary storage (in a perfect world: on a single page)
- Fewer page loads required
- Fewer pages in the buffer
- Typically (automatically) clustering by primary key
 - Doesn't help when selecting a single piece of data
 - Improves performance when selecting a (sequential) range

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

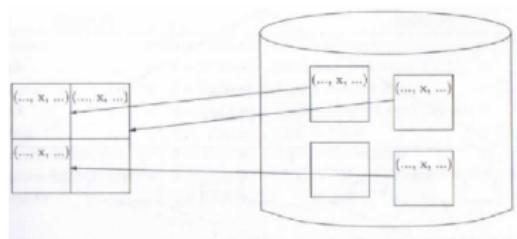
Hashing & Clustering
○○○○○○
○●○○○○

R Tree
○○○○○

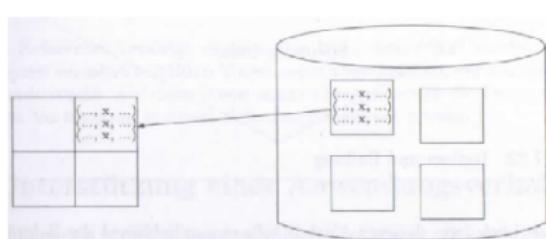
Clustering

Clustering

Bad RAM utilization



Good RAM utilization



Clustering – Indices

- In the primary index data rows are put directly into the leaves
⇒ defines clustering
- Secondary indices use TIDs ⇒ usually no clustering effects
 - One key can have several references to different pages
- When splitting a leaf node half of the data rows are moved ⇒ we would have to place 'forwarding' references which is too much trouble
- Instead we could use TIDs for the primary index as well and check for references of surrounding TIDs during insert
 - Then insert where those point
- Or we could create the primary index once most of the data is already inserted to prevent reorganization
 - Rarely practical

Clustering – Relations

- Another option would be to store data together with related data
- For example: storing lectures and lecturers intertwined:
 - 1 *Lecturer1*
 - 2 *Lecture1*
 - 3 *Lecture2*
 - 4 *Lecturer2*
 - 5 *Lecture3*

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○●○

R Tree
○○○○○

Clustering

Selecting an Access Method

- Depends on the use case!
 - For an exact match a hash index is usually faster than a B+ tree
 - For a range selection a hash index is significantly worse than a B+ tree
 - With many results also a B+ tree gets worse (due to random I/O)
 - Inserts happen fastest in an unordered file
- Good compromise: B+ tree

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○●

R Tree
○○○○○

Clustering

Access Methods

- Heap File (unordered or no suitable index)
- Sorted File
- Clustered Index
- Non-Clustered Index
- Common use cases:
 - Full file (table) scan
 - Exact match (one result)
 - Range selection
 - Insert
 - Delete

Multidimensional Index Structures

- Often you filter on several attributes at once
- A simple solution:
 - 1 Search for TIDs fulfilling attribute 1 condition (e.g. age)
 - 2 Search for TIDs fulfilling attribute 2 condition (e.g. salary)
 - 3 Calculate intersection between the two TID sets
 - 4 Load those
- Multidimensional index structures are meant to solve that problem more efficiently

Storage Media
○○○○○○○○
○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○●○○○

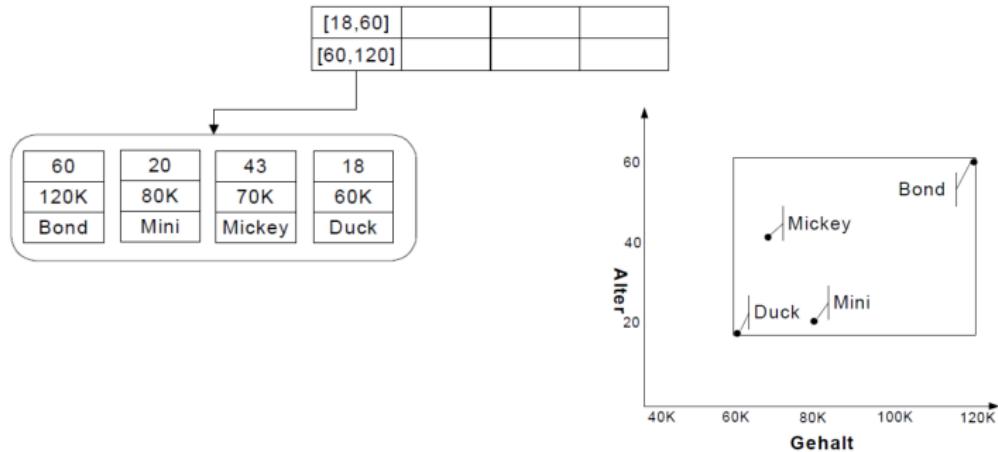
R Tree

R Tree

- Balanced tree
- Inner nodes just used for navigation, data is contained in the leaves (cf. B+ tree)
- Inner nodes consist of:
 - n dimensional region (box)
 - Reference to successor (either another inner node or a leaf)
- All data points (or boxes) of the successors have to be within the box of the predecessor

R Tree

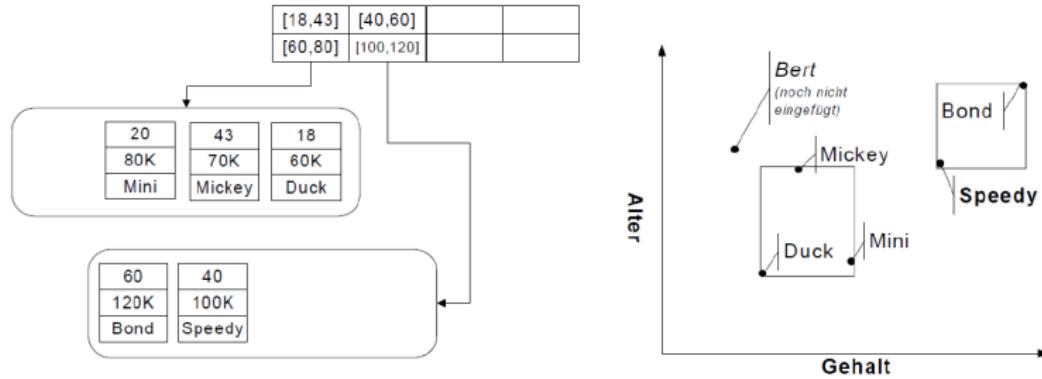
- To keep it simple we'll only use two dimensional data – more dimensions would be possible



Source: Kemper, A. & Eickler, A., Datenbanksysteme, 9th Edition, Oldenbourg Verlag, 2013, p. 240ff

R Tree – Insert

- If a leaf overflows we have to balance
- To do that we need to find a suitable distribution
 - For this task heuristics are used, because all possibilities cannot be checked



Storage Media
○○○○○○○○
○○○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

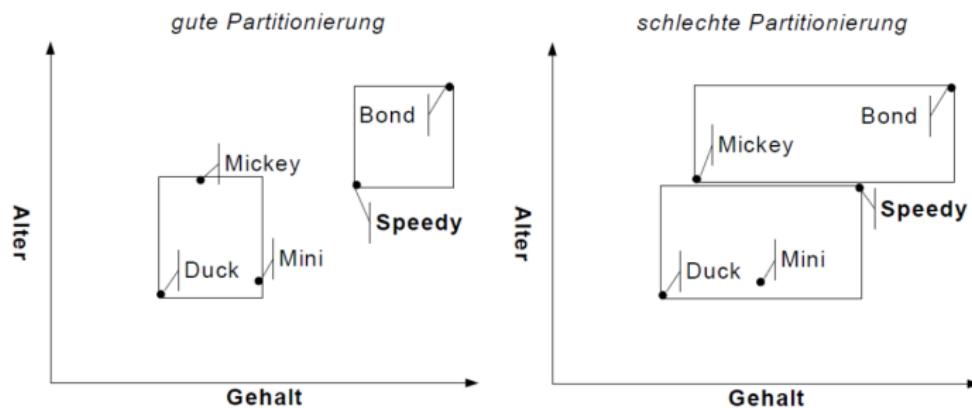
Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○●

R Tree

R Tree – Partitioning

- With a good distribution the resulting boxes are small and only slightly overlap (or even not at all)



Storage Media
○○○○○○○○
○○○○○○○○○○

Secondary Storage
○○
○○○○○○

Index
○○○
○○○○○○

B Tree
○○○○○○○○
○○○○

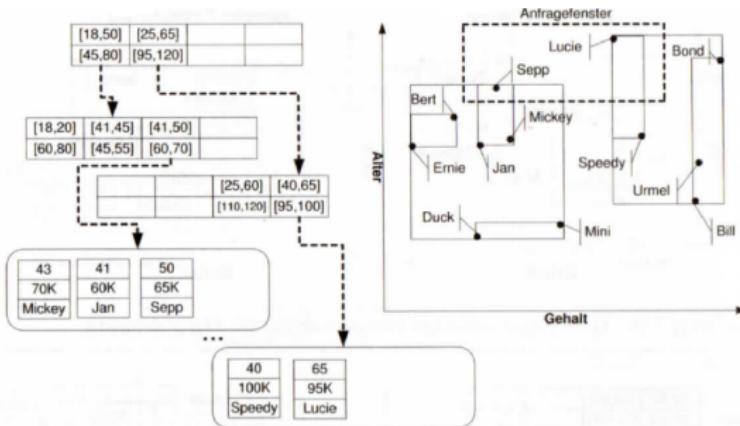
Hashing & Clustering
○○○○○○
○○○○○○

R Tree
○○○○●

R Tree

R Tree – Range Query

- Every query also defines a box
- We start at the root and follow *all* paths which have boxes intersecting with the query box



PL/SQL

Markus Haslinger

DBI/INSY 3rd/4th year

Agenda

- 1** Introduction
 - Introduction
- 2** Programming Basics
 - Basics
 - Advanced Variables & Flow Control
- 3** Advanced Features
 - Cursor & Records
 - Collections & Exceptions
- 4** Advanced Programming
 - Procedures, Functions & Packages
 - Trigger

Motivation

Definition

An application that uses Oracle Database is worthless unless only correct and complete data is persisted. The time-honored way to ensure this is to expose the database only via an interface that hides the implementation details – the tables and the SQL statements that operate on these. This approach is generally called the thick database paradigm, because PL/SQL subprograms inside the database issue the SQL statements from code that implements the surrounding business logic; and because the data can be changed and viewed only through a PL/SQL interface.¹

¹<https://www.oracle.com/technetwork/database/features/plsql/index.html>, 2019-02-06

Motivation Discussion

- Access to data should be restricted ⇒ true
- Only correct and complete data should be persisted ⇒ true
- User input has to be validated against business rules ⇒ true
- This business logic should live in the database ⇒ maybe
 - In general these functions are to be performed by the business application...
 - ...because we are using SQL in our application to abstract the type of database used in the backend away
 - **But in certain scenarios PL/SQL can lead to much better performance**

What is PL/SQL?

- PL/SQL is a procedural (programming) language
 - Variables
 - Control structures (loops, branches)
 - Reusability (functions)
 - Exception Handling
- It focuses on intertwining SQL statements and program code
- Execution (and compilation) happens on the DB(MS)
- Can be used everywhere where Oracle is installed

Introduction

PL/SQL Sample

Worksheet | Query Builder

```
SET SERVEROUTPUT ON
---

DECLARE
    v_name VARCHAR2(20);
BEGIN
    SELECT ename INTO v_name
    FROM emp
    WHERE empno = 7844;

    DBMS_OUTPUT.PUT_LINE('Name of employee with No 7844 is ' || v_name);
END;
```

Script Output X

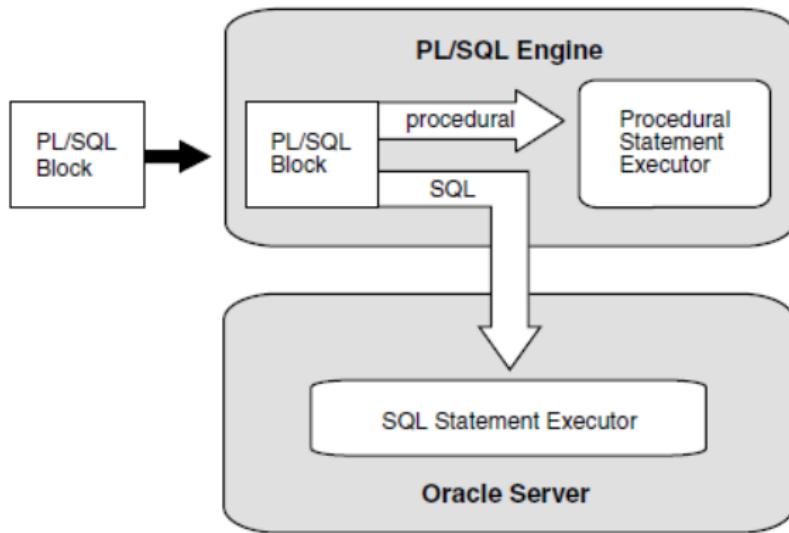
PL/SQL procedure successfully completed.

Name of employee with No 7844 is TURNER

Interlude: PL/SQL Output in SQLDeveloper

- Use 'SET SERVEROUTPUT ON' or
- Activate DBMS Output View

PL/SQL Execution



Benefits of PL/SQL (vs. pure SQL)

- SQL just defines *what* has to be done, not *how*
- Performance can be improved by reusing results
- Integration in (Oracle) tools (reports, forms,...)
- Exception Handling

Block Structure

1 DECLARE (optional)

- Variables, Cursors, user defined Exceptions

2 BEGIN

- SQL statements
- PL/SQL statements

3 EXCEPTION (optional)

- Exception handling

4 END;

Block Types

Procedure

```
PROCEDURE name
IS
BEGIN
--- statements
[EXCEPTION]
END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
--- statements
RETURN value;
[EXCEPTION]
END;
```

Anonymous

```
[DECLARE]
BEGIN
--- statements
[EXCEPTION]
END;
```

Variables

- Syntax: identifier [CONSTANT] datatype [**NOT NULL**] [:= | **DEFAULT** expr];
- Identifier
 - Have to start with a letter
 - Can contain letters and numbers
 - Can contain special characters (\$,...)
 - Have a max. length of 30
 - Must not be reserved words

Variable Examples

Variables

```
v_name VARCHAR2(20);  
v_name VARCHAR2(20) := 'Max';  
v_name VARCHAR2(20) DEFAULT 'Susi';  
c_pi CONSTANT NUMBER(3,2) := 3.14;
```

Datatypes

- CHAR [(length)]
- VARCHAR2 (max_length)
- NUMBER [(precision, scale)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- BINARY_FLOAT
- BINARY_DOUBLE
- DATE
- TIMESTAMP

%TYPE datatype declaration

- Allows to automatically assume datatype of a column or another variable
- Datatype changes with the other ⇒ this only works as long as the new datatype is still valid for the performed operations (technically & logically)
- Syntax Examples:
 - identifier **table.column_name**%TYPE;
 - **v_balance NUMBER(7,2);**
 - **v_min_balance v_balance%TYPE := 500;**

Strings

- Single apostrophe
 - Example: 'Hello World'
- Escaping:
 - [1] double quotes: 'Sorry Dave, I can't do that'
 - [2] q-notation: q'<Delimiter>...<Delimiter>'
 - q'!Sorry Dave, I can't do that!'
 - q'[Sorry Dave, I can't do that]'

SQL Functions

- These functions can be used in PL/SQL (outside of a SQL statement)
- Datatype conversions
 - TO_CHAR
 - TO_DATE
 - TO_NUMBER
 - TO_TIMESTAMP
- Sequences
 - var := my_seq.NEXTVAL;
- String functions
 - LENGTH,...
- Numerical functions
 - MONTHS_BETWEEN,...
- **No** aggregation functions (AVG, MIN, ...)

Hello World Example

Example

```
SET SERVEROUTPUT ON

DECLARE
    name VARCHAR2(10) := '<your_name>';
    birth DATE := date '<your_birthday>';
    age NUMBER;
BEGIN
    age := FLOOR(MONTHS_BETWEEN(SYSDATE, birth)/12);
    dbms_output.put_line('Hello, my name is ' || name || ' and I am ' || age || '
        years old');
END;
```

SELECT in PL/SQL

- INTO clause
- End each query with a semicolon
- Queries must only return a single result (otherwise a CURSOR is needed)

Syntax

```
SELECT <what>
INTO <variable>
FROM <table>
[WHERE <condition>];
```

SELECT in PL/SQL – Example

Example

```
SET SERVEROUTPUT ON
DECLARE
    v_hiredate emp.hiredate%TYPE;
    v_salary emp.sal%TYPE;
BEGIN
    SELECT hiredate, sal
    INTO v_hiredate, v_salary
    FROM emp
    WHERE empno = 7788;
    dbms_output.put_line('Hiredate: ' ||
        v_hiredate);
    dbms_output.put_line('Salary: ' ||
        v_salary);
END;
```

Script Output X | Query Result X
Task completed in 0,359 seconds

Hiredate: 09.12.82
Salary: 3000

PL/SQL procedure successfully completed.

DML in PL/SQL – Examples

INSERT

```
BEGIN
    INSERT INTO emp (empno, ename, job, mgr, hiredate, sal, comm, deptno)
    VALUES (9999, 'DMC', 'Magician', 7839, date '2019-02-22', 1337, null, 40);
END;
```

UPDATE

```
DECLARE
    v_raise emp.sal%TYPE := 200;
BEGIN
    UPDATE emp SET sal = sal + v_raise WHERE empno = 9999;
END;
```

DML in PL/SQL – Examples

DELETE

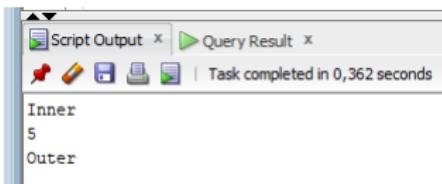
```
DECLARE
    v_deptno emp.deptno%TYPE := 40;
BEGIN
    DELETE FROM emp WHERE deptno = v_deptno;
END;
```

Nested Scopes/Blocks

- Blocks can be nested
- Variable lookup start at the innermost block

Example

```
SET SERVEROUTPUT ON
DECLARE
    v_pseudo_global VARCHAR2(10) := 'Outer';
BEGIN
    DECLARE
        v_pseudo_global NUMBER := 5;
        v_inner VARCHAR2(10) := 'Inner';
    BEGIN
        dbms_output.put_line(v_inner);
        dbms_output.put_line(v_pseudo_global);
    END;
    dbms_output.put_line(v_pseudo_global);
END;
```



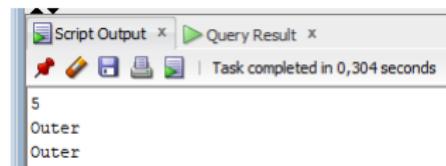
```
Script Output X | Query Result X
| Task completed in 0,362 seconds
Inner
5
Outer
```

Nested Scopes/Blocks – Qualifier

- Access outer variables explicitly

Example

```
SET SERVEROUTPUT ON
BEGIN <<OUTER>>
  DECLARE
    v_pseudo_global VARCHAR2(10) := 'Outer';
  BEGIN
    DECLARE
      v_pseudo_global NUMBER := 5;
    BEGIN
      dbms_output.put_line(v_pseudo_global);
      dbms_output.put_line(OUTER.v_pseudo_global);
    END;
    dbms_output.put_line(v_pseudo_global);
  END;
END OUTER;
```



Variables – BIND

- Syntax: VARIABLE name type
- Also called Hostvariables (global variables)
- Referenced by a leading colon
- Can be used in SQL statements and PL/SQL blocks
- Still available after execution of the PL/SQL block
- Output via PRINT
 - **SET AUTOPRINT ON**

Variables – BIND – Example

Example

```
SET SERVEROUTPUT ON

VARIABLE b_avgsal NUMBER
BEGIN
    SELECT AVG(sal) INTO :b_avgsal
    FROM emp;
END;
/
BEGIN
    dbms_output.put_line(:b_avgsal);
END;
/
SELECT ename FROM emp
WHERE sal > :b_avgsal;
```

PL/SQL procedure successfully completed.

2073,214285714285714285714285714285714286

PL/SQL procedure successfully completed.

ENAME

JONES
BLAKE
CLARK
SCOTT
KING
FORD

6 rows selected.

IF Conditional

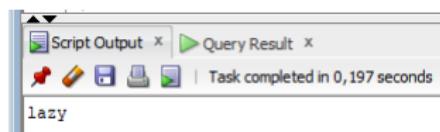
Syntax

```
IF <condition> THEN
    <statements>;
[ELSIF <condition> THEN
    <statements>];
[ELSE
    <statements>]
END IF;
```

IF Conditional – Example

Syntax

```
DECLARE
    v_score INTEGER := 55;
BEGIN
    IF v_score < 50 THEN
        dbms_output.put_line('failed in life');
    ELSIF v_score < 60 THEN
        dbms_output.put_line('lazy');
    ELSE
        dbms_output.put_line('promising');
    END IF;
END;
```



CASE Expression

- Similar to switch, but still different
- Two versions: including & omitting the expression clause

Syntax

```
CASE [expression]
  WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  [WHEN condition_n THEN result_n]
  [ELSE result_n+1]
END;
```

CASE Expression – Example

Example

```
DECLARE
    v_perc INTEGER := 82;
    v_grade VARCHAR2(20);
BEGIN
    v_grade := CASE
        WHEN v_perc >= 92.0 THEN 'Sehr Gut'
        WHEN v_perc >= 80.0 THEN 'Gut'
        WHEN v_perc >= 65.0 THEN 'Befriedigend'
        WHEN v_perc >= 50.0 THEN 'Genuegend'
        ELSE 'Nicht Genuegend'
    END;
    dbms_output.put_line(v_grade);
END;
```

CASE Expression – Example with SQL

With Expression

```
SELECT table_name,  
CASE owner  
    WHEN 'SYS' THEN 'The owner  
        is SYS'  
    WHEN 'SYSTEM' THEN 'The  
        owner is SYSTEM'  
    ELSE 'The owner is another value  
        '  
END  
FROM all_tables;
```

Without Expression

```
SELECT table_name,  
CASE  
    WHEN owner='SYS' THEN 'The owner  
        is SYS'  
    WHEN owner='SYSTEM' THEN 'The  
        owner is SYSTEM'  
    ELSE 'The owner is another value'  
END  
FROM all_tables;
```

CASE Statement

- Does not return value, but executes specific action
- Ends with **END CASE**

Example

```
DECLARE
    v_grade VARCHAR2(20) := '3';
BEGIN
    CASE v_grade
        WHEN '1' THEN dbms_output.put_line('Sehr Gut');
        WHEN '2' THEN dbms_output.put_line('Gut');
        WHEN '3' THEN dbms_output.put_line('Befriedigend');
        WHEN '4' THEN dbms_output.put_line('Genuegend');
        ELSE dbms_output.put_line('Nicht Genuegend');
    END CASE;
END;
```

Loops – WHILE

- A typical while loop

Syntax

WHILE condition **LOOP**

```
    statement_1;  
    [statement_n;]
```

END LOOP;

Loops – LOOP

- Similar to a do-while loop
- Runs forever until EXIT (with optional condition) is hit

Syntax

LOOP

```
statement_1;  
[statement_n;]  
EXIT [WHEN condition];
```

END LOOP;

Loops – FOR

- Pretty typical for loop
- Supports range syntax and reverse option

Syntax

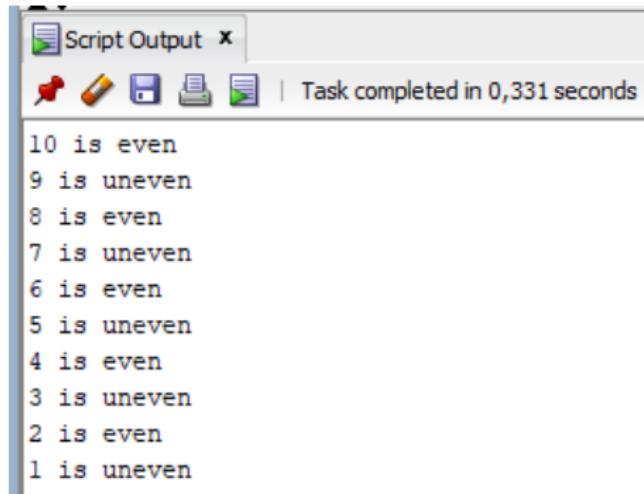
```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
    statement_1;
    [statement_n;]
END LOOP;
```

Loops – FOR – Example

Example

```
DECLARE
    v_even BOOLEAN;
BEGIN
    FOR i IN REVERSE 1..10 LOOP
        v_even := i MOD 2 = 0;
        dbms_output.put_line(i || ' is ' || CASE WHEN v_even =
            TRUE THEN 'even' ELSE 'uneven' END);
    END LOOP;
END;
```

LOOPS – FOR – Example Result



The screenshot shows a window titled "Script Output" with a close button. Below the title bar are several icons: a red heart, a pencil, a blue folder, a printer, and a play button. To the right of these icons, the text "Task completed in 0,331 seconds" is displayed. The main content area of the window contains the following text:
10 is even
9 is uneven
8 is even
7 is uneven
6 is even
5 is uneven
4 is even
3 is uneven
2 is even
1 is uneven

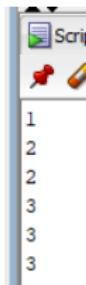
Nested Loops

- Loops can be nested within other Loops
 - Of course the same is true for IFs within IFs and IFs within LOOPS
- Like blocks, they can be labeled

Nested Loops – Example

Example

```
DECLARE
    v_cnt NUMBER := 0;
BEGIN
    <<OUTER_LOOP>>
    LOOP
        v_cnt := v_cnt + 1;
    EXIT WHEN v_cnt > 5;
    <<INNER_LOOP>>
    FOR i IN 1..v_cnt LOOP
        EXIT OUTER_LOOP WHEN v_cnt > 3 AND v_cnt > i;
        dbms_output.put_line(v_cnt);
    END LOOP INNER_LOOP;
END LOOP OUTER_LOOP;
END;
```



Loops – CONTINUE

- Skips the remaining statements of the current iteration
- Syntax: **CONTINUE [label] [WHEN condition]**

Examples

```
IF x < 3 THEN  
    CONTINUE;  
END IF;
```

or

```
CONTINUE WHEN x < 3;
```

Cursor

- A cursor is a pointer to a private memory block assigned by the DBMS
 - Similar to a C Array pointer
- It is used to process result sets (= more than 1 result)
- There are two types:
 - Implicit Cursor: Created automatically when a SQL query is executed
 - Explicit Cursor: Defined manually by the programmer

Implicit Cursor Attributes

- SQL%ROWCOUNT
- SQL%[NOT]FOUND

Example

```
SET SERVEROUTPUT ON
BEGIN
    DELETE FROM dummy WHERE dummy = 3;
    dbms_output.put_line(SQL%ROWCOUNT || ' rows deleted');
END;
```

Explicit Cursor

- Declared and managed by the programmer
 - Contrary to the implicit cursor which is created and managed automatically behind the scenes
- Used for SELECT statements which return multiple rows
- Allow for the row-wise processing of result sets
- Always four steps:
 - 1 Declaring the cursor in the DECLARE section
 - 2 Opening the cursor executes the query and locks the rows
 - 3 Reading the data
 - FETCH each row
 - until %NOTFOUND becomes true indicating no more rows
 - 4 Closing the cursor (CLOSE) releases the rows

Declaring a Cursor

- Syntax: **CURSOR cursor_name IS select_statement**
- The command **INTO** must not be used in the cursor declaration but is used later when **FETCHing**
- Using variables in the **SELECT** is possible

Example

```
v_max_player_no NUMBER := 100;  
CURSOR c_players IS  
    SELECT * FROM players WHERE playerno <=  
        v_max_player_no;
```

Explicit Cursor – Example

Iterating players

```
DECLARE
    v_max_player_no NUMBER := 100;
    CURSOR c_players IS
        SELECT * FROM players WHERE playerno <= v_max_player_no;
    v_player players%ROWTYPE;
BEGIN
    OPEN c_players;
    LOOP
        FETCH c_players INTO v_player;
        EXIT WHEN c_players%NOTFOUND;
        dbms_output.put_line(v_player.name);
    END LOOP;
    CLOSE c_players;
END;
```

Cursor – FOR Loops

- A cursor is basically an iterator
- Using it in a 'foreach' loop simplifies the process
 - OPEN, FETCH, CLOSE and the check for more rows are done automatically
 - Iteration variable record declared implicitly

Syntax

```
FOR record_name IN cursor_name LOOP  
  ...  
END LOOP;
```

Cursor – FOR Loops – Example

Example

```
DECLARE
    v_max_player_no NUMBER := 100;
    CURSOR c_players IS
        SELECT * FROM players WHERE playerno <=
            v_max_player_no;
BEGIN
    FOR v_player IN c_players LOOP
        dbms_output.put_line(v_player.name);
    END LOOP;
END;
```

Cursor Attributes

- Usually used when not iterating via a FOR loop

Attribute	Type	Description
%ISOPEN	BOOLEAN	TRUE if cursor is opened
%NOTFOUND	BOOLEAN	TRUE if last FETCH did not return a row
%FOUND	BOOLEAN	Opposite of %NOTFOUND
%ROWCOUNT	NUMBER	Count of rows returned so far (<i>not total rows in the result set</i>)

Omitting Cursor declaration

- Declaration of the cursor can be omitted

Example

```
DECLARE
    v_max_player_no NUMBER := 100;
BEGIN
    FOR v_player IN (SELECT * FROM players WHERE playerno
                      <= v_max_player_no) LOOP
        dbms_output.put_line(v_player.name);
    END LOOP;
END;
```

Cursor – Parameters

- A cursor can be defined with parameters
- These parameters are supplied when opening the cursor
- This allows the same cursor to be used for several similar queries

Syntax

```
CURSOR cursor_name [(parameter_name datatype,...)] IS  
    select_statement;
```

Cursor – Parameters – Example

Example

```
DECLARE
    CURSOR c_players (p_max_playno NUMBER) IS
        SELECT * FROM players WHERE playerno <=
            p_max_playno;
BEGIN
    FOR v_player IN c_players(100) LOOP
        dbms_output.put_line(v_player.name);
    END LOOP;
END;
```

Cursor – FOR UPDATE

- A DBMS is meant to be used by multiple users at the same time
- Thus it is important to deal with several, different updates occurring at the same time
- Normally, the data set a cursor iterates is a snapshot
 - So changes made while iterating are not visible
- FOR UPDATE locks the affected rows for updates
- This is important if changes are made based on the row values
 - Otherwise the decision may be based on outdated values

Cursor – FOR UPDATE

Syntax

```
SELECT ... FROM ...  
FOR UPDATE [OF column_ref][NOWAIT | WAIT n];
```

- column_ref: one or more columns (to lock)
- NOWAIT: raises an error if the rows are locked by another session already
- WAIT: waits for the specified number of seconds for the lock to release (before raising the error)

Cursor – WHERE CURRENT OF

- Allows to UPDATE or DELETE the current row the cursor points to
- This simplifies the process, because no full primary key condition has to be used
- Usually used together with FOR UPDATE to avoid incorrect changes

Cursor – WHERE CURRENT OF – Example

Example

```
DECLARE
    CURSOR c_players (p_max_playno NUMBER) IS
        SELECT * FROM players WHERE playerno <= p_max_playno FOR
        UPDATE;
BEGIN
    FOR v_player IN c_players(50) LOOP
        IF v_player.town = 'Stratford' THEN
            UPDATE players SET town='Bradford'
            WHERE CURRENT OF c_players;
        END IF;
        dbms_output.put_line(v_player.name || ' ' || v_player.town);
    END LOOP;
END;
```

Record

- Grouping values of different datatypes together
- Similar to a C struct

Syntax

```
TYPE type_name IS RECORD  
(field_declaration [, field_declaration]);
```

-- *Usage*

```
identifier type_name;
```

Record – Example

Example

```
SET SERVEROUTPUT ON;
DECLARE
  TYPE rt_empdata IS RECORD
  (
    emp_no NUMBER(6) NOT NULL := -1,
    salary NUMBER(8,2) NOT NULL := -1,
    comm NUMBER(8,2) NULL
  );
  v_emp rt_empdata;
BEGIN
  v_emp.salary := 2200;
  dbms_output.put_line(v_emp.salary);
END;
```

Record – %ROWTYPE

- Syntax: identifier reference%ROWTYPE
- Declares a record which can hold a whole row of the table
- Can be used for INSERT and UPDATE

Example

```
DECLARE
    v_player players%ROWTYPE;
BEGIN
    SELECT * INTO v_player FROM players WHERE playerno=44;
    dbms_output.put_line(v_player.postcode);
END;
```

Collections

- Can contain multiple values of the same data type
- Comparable to an array
- Three types exist:
 - VARRAY: a pretty standard array
 - Associative array: comparable to a dictionary
 - Nested Table: associative array without INDEX BY definition, extendable and a little like a list

VARRAY

- Syntax: TYPE typename IS VARRAY(**size**)OF datatype;
- Has a fixed size
- Element order is maintained
- Has only one datatype
 - Including the max size (e.g. VARCHAR2(10))
- Array is 1 based, not 0 based as usual!

VARRAY – Example

Example

```
DECLARE
    TYPE at_numbers IS VARRAY(2) OF NUMBER(1);
    v_num_arr at_numbers;
BEGIN
    v_num_arr := at_numbers(2, 4);
    FOR i in 1..v_num_arr.count LOOP
        dbms_output.put_line(v_num_arr(i));
    END LOOP;
END;
```

Associative Array

- Key Value Pairs (like a Dictionary)
- The Key
 - Can be a numeric or string value
 - Instead of NUMBER use BINARY_INTEGER or PLS_INTEGER
 - VARCHAR2 (and DATE)
- The Value
 - Scalar datatype or
 - Record datatype ⇒ nesting is possible

Associative Array

Syntax

```
TYPE typename IS TABLE OF (value_type [NOT NULL] | table%ROWTYPE)
INDEX BY (PLS_INTEGER | BINARY_INTEGER | VARCHAR2(size))
```

Example

```
DECLARE
  TYPE day_of_week IS TABLE OF VARCHAR2(10) INDEX BY PLS_INTEGER;
  days day_of_week;
BEGIN
  days(1) := 'Monday';
  days(2) := 'Tuesday';
  dbms_output.put_line(CASE WHEN days.EXISTS(2) THEN days(2) ELSE 'unknown' END);
END;
```

Associative Array – Functions

- EXIST(n): Returns true if the nth element exists ⇒ looks for the index (not key value per-se) which can be problematic when mixing numeric and alphanumeric values
- COUNT: Returns the number of elements in the array
- FIRST: Returns the smallest index number or NULL if the array is empty
- LAST: Returns the biggest index number or NULL if the array is empty

Associative Array – Functions

- PRIOR(n): Returns the index number preceding the index n in the array
- NEXT(n): Returns the index number succeeding the index n in the array
- DELETE
 - DELETE: Removes all elements from the array
 - DELETE(n): Removes the n^{th} element from the array
 - DELETE(m,n): Removes all elements between the m^{th} and n^{th}

Associative Array – %ROWTYPE Example

Example

```
DECLARE
    TYPE tt_players IS TABLE OF players%ROWTYPE INDEX BY
        PLS_INTEGER;
    v_players tt_players;
BEGIN
    SELECT * INTO v_players(44) FROM players WHERE playerno
        = 44;
    dbms_output.put_line(v_players(44).postcode);
END;
```

Nested Table

- Similar to Associative Table without an INDEX
- Has to be initialized with a constructor
- Has only positive indices
- Could be saved in the database (valid datatype for schema tables)
- Be careful:
 - There is no fixed upper size as with a VARRAY
 - But you still need to extend every time you want to add a row

Nested Table – Example

Example

```
DECLARE
    TYPE tt_dept IS TABLE OF VARCHAR2(20);
    departments tt_dept;
BEGIN
    departments := tt_dept('Informatik', 'Elektronik', 'Medizintechnik');
    departments.extend(1);
    departments(4) := departments(3);
    departments(3) := 'Medientechnik';
    FOR i in 1..departments.COUNT LOOP
        dbms_output.put_line(departments(i));
    END LOOP;
END;
```

Exceptions

- Not compile errors, but runtime exceptions
- Raised either implicitly by the database server or explicitly in the code
- Handling:
 - Either catch and handle or
 - Propagating to the caller ← we rarely want that

Type	Description	To declare	Raised
Predefined Oracle Error	The ~20 most common errors	No	Implicitly
Non-predefined Oracle Errors	All other default errors	Yes	Implicitly
User defined Error		Yes	Explicitly

Exception

Syntax

EXCEPTION

```
WHEN exception1 THEN ...
[WHEN exceptionN THEN ...]
[WHEN OTHERS THEN ...]
```

- **WHEN OTHERS** can be used to catch all unexpected Exceptions
- For a list of predefined Oracle Exceptions see
https://docs.oracle.com/cd/A97630_01/appdev.920/a96624/07_errs.htm#784

Exception – Example

Example

```
DECLARE
    v_name VARCHAR2(20);
BEGIN
    SELECT name INTO v_name FROM players;
EXCEPTION
    WHEN TOO_MANY_ROWS THEN
        dbms_output.put_line('Select with more than 1 result!');
END;
```

Exception – Not Predefined

Declaration

```
DECLARE
    exception EXCEPTION;
    PRAGMA EXCEPTION_INIT (exception, errno);
```

Handling

```
EXCEPTION
    WHEN exception THEN
        ...
```

Exception – Not Predefined – Example

Example

```
DECLARE
    e_insert_null EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_insert_null, -01400);
BEGIN
    INSERT INTO players (playerno) VALUES (NULL);
EXCEPTION
    WHEN e_insert_null THEN
        dbms_output.put_line('No NULL in NOT NULL column');
        dbms_output.put_line(SQLERRM);
END;
```

Exception – Functions

- SQLERRM
 - Returns the message associated with the error code
- SQLCODE
 - Returns the numeric value of the error code
 - Assignable to a NUMBER variable
- If one of the return values is to be used in a SQL statement it has to be assigned to a variable first.

User defined Exceptions

Declaration

```
DECLARE  
    exception EXCEPTION;
```

Raising & Handling

```
-- first raise when appropriate  
RAISE exception;  
  
-- then handle  
EXCEPTION  
    WHEN exception THEN ...
```

User defined Exceptions – Example

Example

```
DECLARE
    e_invalid_cust_no EXCEPTION;
    v_cust_no NUMBER := 15000;
BEGIN
    IF v_cust_no > 9999 THEN
        RAISE e_invalid_cust_no;
    END IF;
EXCEPTION
    WHEN e_invalid_cust_no THEN
        dbms_output.put_line('Invalid customer number');
END;
```

RAISE_APPLICATION_ERROR

Syntax

```
RAISE_APPLICATION_ERROR  
  (error_number, message[, (TRUE | FALSE)]);
```

- Returns user defined errors to the (calling) application
- error_number: Number between -20999 and -20000
- message: User defined error message (max. 2kB)
- TRUE vs. FALSE
 - TRUE: Keeps previous error in stack
 - FALSE: Replaces previous error in stack

Subroutines & Functions

- We differentiate between:
 - Procedures – no return value
 - Functions – return value
- Are basically named PL/SQL blocks
 - Which can accept *parameters*
 - Structure similar, with:
 - Declaration section (optional, *no* DECLARE)
 - Statement section (required)
 - Exception Handling (optional)
- Are compiled once and then stored in the database
 - ⇒ 'Stored procedure'
 - Anonymous blocks are compiled at every execution

Procedures

Syntax

```
CREATE [OR REPLACE] PROCEDURE proc_name
[(
    argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    ...
)] (IS|AS)
proc_body;
```

Procedures – Parameters

- Three different modes:
 - IN (default): regular input parameter
 - OUT: out parameter
 - IN OUT: both input and output
- Data types:
 - The data type of a parameter must not have a specific size defined (CHAR vs. CHAR(5))
 - If tailoring to a specific table %TYPE can be used

Procedures – Example

Example

```
CREATE OR REPLACE PROCEDURE add_employee (
    p_ssn employee.ssn%TYPE,
    p_name employee.name%TYPE,
    p_city employee.city%TYPE)
AS
BEGIN
    INSERT INTO employee(ssn, name, exitdate, city)
    VALUES (p_ssn, p_name, NULL, p_city);
    dbms_output.put_line(SQL%ROWCOUNT || ' row inserted');
END;
```

Procedures

How to call

BEGIN

```
    add_employee('9876080888', 'new emp', 'Sto Lat');
```

END;

Retrieve object info

```
SELECT * from user_objects WHERE object_name = '  
ADD_EMPLOYEE';
```

```
SELECT * from user_source WHERE name = 'ADD_EMPLOYEE';
```

Functions

Syntax

```
CREATE [OR REPLACE] FUNCTION func_name
[(
    argument1 [mode1] datatype1,
    argument2 [mode2] datatype2,
    ...
)]
RETURN datatype
(IS|AS)
func_body;
```

Functions – Parameters

- Only IN parameters are allowed
 - Use RETURN value instead of OUT
 - If returning multiple values a structure is needed
- You have to define which datatype the function returns

Functions – Example

Example

```
CREATE OR REPLACE FUNCTION max_won
    RETURN matches.won%TYPE
AS
    v_max_won matches.won%TYPE;
BEGIN
    SELECT MAX(won) INTO v_max_won FROM matches;
    RETURN v_max_won;
END;
```

Functions

How to call

```
BEGIN  
    dbms_output.put_line(max_won);  
END;
```

Retrieve object info

```
DESCRIBE max_won;
```

Package

- A package is a schema object
- It groups logically dependent Types, Variables, Constants, Exceptions and subroutines
- A package consists of
 - A specification defining the interface (comparable to a C header file)
 - A body containing the actual implementation and 'private' members
- Packages can be used by other programs
- The whole package is loaded into memory if any piece of it is referenced

Package – Benefits

- Enclosure of related constructs
- Improvements for the design by separating specification and body
- Hiding 'private' methods
- Allows to persist variables and cursors for the duration of a session ← yet we do not want to rely on that
- Two edged blade: performance
 - The whole package is loaded into memory with the first usage
 - That speeds up subsequent accesses
 - But it also wastes a lot of memory if only a fraction is actually needed

Package – Specification

Syntax

```
CREATE [OR REPLACE] PACKAGE packageName
(IS | AS)
    <public types and variables>
    <public subroutine prototypes>
END [packageName];
```

- Represents the interface used by other applications

Package – Body

Syntax

```
CREATE [OR REPLACE] PACKAGE BODY packageName  
(IS | AS)  
  <private types and variables>  
  <public and private subroutine implementations>  
[BEGIN <initialization statements>]  
END [packageName];
```

- Contains the implementation of the public subroutines and the supporting private subroutines
- BEGIN Block will be executed when the package is referenced for the first time (cf. constructor)

Package – Example – Specification

Specification

```
CREATE OR REPLACE PACKAGE circle_funcs
AS
    c_pi CONSTANT NUMBER(3,2) := 3.14;

    FUNCTION circumference(p_radius NUMBER) RETURN
        NUMBER;
    FUNCTION area(p_radius NUMBER) RETURN NUMBER;
END circle_funcs;
```

Package – Example – Body

Body

```
CREATE OR REPLACE PACKAGE BODY circle_funcs
AS
    FUNCTION circumference(p_radius NUMBER) RETURN NUMBER AS
        BEGIN
            RETURN p_radius * 2.0 * c_pi;
        END;
    FUNCTION area(p_radius NUMBER) RETURN NUMBER AS
        BEGIN
            RETURN p_radius * p_radius * c_pi;
        END;
    END;
```

Package – Example – Usage

Usage

```
SET SERVEROUTPUT ON;
SELECT circle_funcs.circumference(2) FROM dual;
/
BEGIN
    dbms_output.put_line(circle_funcs.circumference(2));
    dbms_output.put_line(circle_funcs.area(2));
END;
```

Trigger

- A trigger is a code block stored in the database
- Similar to functions and procedures
- Yet it is activated (*triggered*) automatically
- Events that can activate a trigger:
 - DML-Statements (INSERT, UPDATE, DELETE)
 - DDL-Statements (CREATE, ALTER, DROP)
 - Database Events (LOGON, STARTUP, SHUTDOWN,...)

Trigger – Scenarios

- The following scenarios might warrant the use of a trigger:
 - Security
 - Auditing
 - Data Integrity
 - Referential Integrity
 - Table replication
 - Calculating derived data
 - Event protocol

Trigger – Management

- (De)activating a trigger:
 - **ALTER TRIGGER** trigger_name ENABLE | DISABLE
- Deleting a trigger:
 - **DROP TRIGGER** trigger_name
- Querying information about a trigger:
 - **SELECT * FROM USER_TRIGGERS WHERE**
Trigger_name = 'name'

DML-Trigger

Syntax

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF col1[,col2,...]]}
[OR {INSERT | DELETE | UPDATE ...}] ...
ON object_name
[REFERENCES [OLD AS old] [NEW AS new]]
[FOR EACH ROW [WHEN condition]]
<PL/SQL-Block>
```

DML-Trigger – Example

Example

```
CREATE TRIGGER audit_emp_sal
    AFTER UPDATE OF sal ON emp
    FOR EACH ROW
BEGIN
    INSERT INTO emp_audit VALUES ...
END;
```

DML-Trigger – Example

- **UPDATE [OF col1]**

- Trigger can be restricted to check for updates on specific columns

- **FOR EACH ROW [WHEN condition]**

- Trigger runs for each row fulfilling the condition

- The keywords :new & :old can be used to access the original and the updated value

- REFERENCES can be used to overwrite those values

- In the PL/SQL block the triggering event can be determined:

- **IF INSERTING THEN**
 - **IF UPDATING THEN**
 - **IF DELETING THEN**

INSTEAD OF Trigger

- Instead of a specific action the trigger action runs
- Often used when attempting to insert into a read-only view

Example

```
CREATE OR REPLACE TRIGGER emp_view_insert
  INSTEAD OF INSERT on emp_dep_view
  FOR EACH ROW
  BEGIN
    INSERT INTO emp VALUES ...;
    ...
  END;
```

DDL-Trigger

- Can be declared on database and on schema level
- Examples for triggering event: CREATE, DROP, GRANT,...

Example

```
CREATE OR REPLACE TRIGGER drop_trigger
BEFORE DROP ON scott.SCHEMA
BEGIN
    RAISE_APPLICATION_ERROR(-20000, 'Drop not allowed');
END;
```

Query Processing
oooooooo

Logical Optimization
oooooooooooo
oooooooooooo

Physical Optimization
oooooooooooo
oooooooooooooooo

Cost Models
oooooooo
oooooooooooo

Query Processing & Optimization

Markus Haslinger

DBI/INSY

Agenda

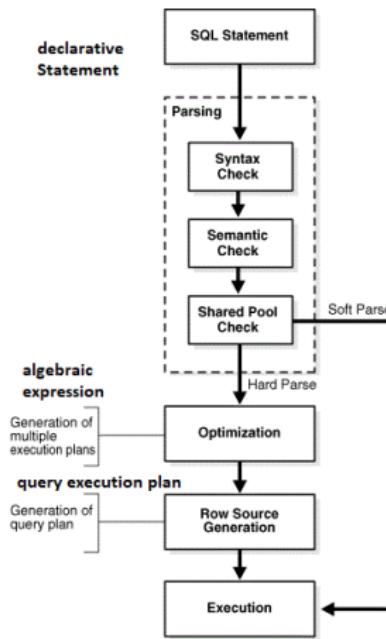
1 Query Processing

2 Logical Optimization

3 Physical Optimization

4 Cost Models

Query Execution Flow Chart



Source:

https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQ176

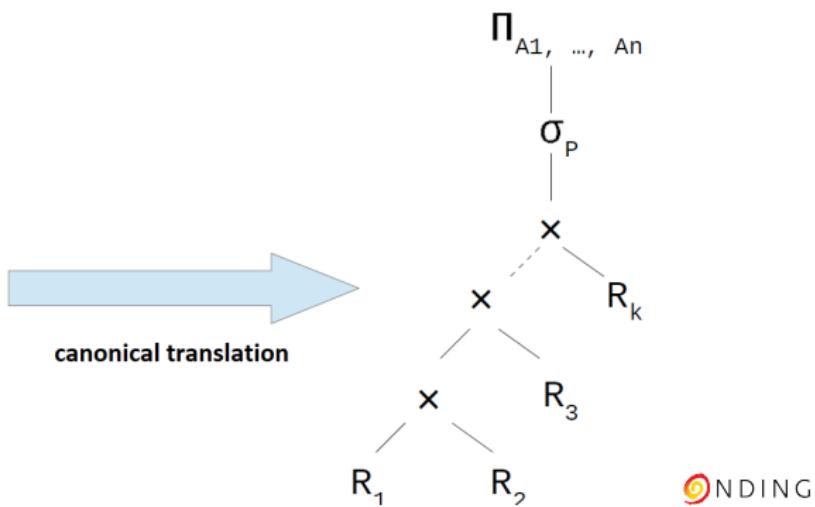
Query Parsing

- 1 Syntax Validation
 - e.g. typos
- 2 Semantic Validation
 - e.g. do the tables exist
- 3 Parser converts query into a relational algebra expression (usually a tree)
- 4 If views are contained in the query those are replaced with the appropriate (sub)query so that only tables remain in the leaves of the expression tree
- 5 Canonical Translation

Canonical Translation

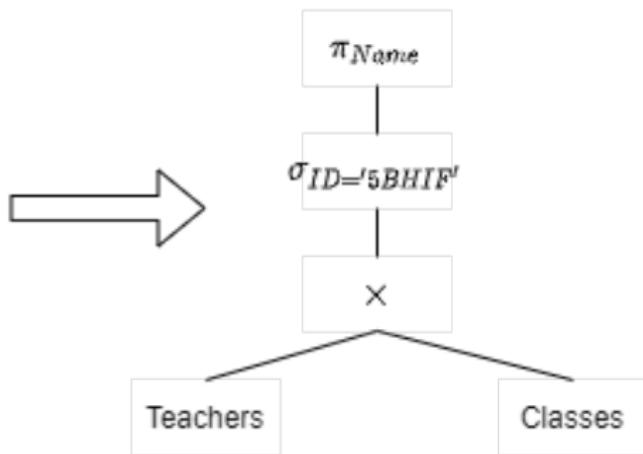
- Starting point is the algebraic normal form
- ⇒ the query is transformed into an algebraic expression with cross products of the base relations

```
select A1, ..., An
from R1, ..., Rk
where P
```



Canonical Translation – Example

```
SELECT t.Name  
FROM Teachers t, Classes c  
WHERE c.ID = '5BHIF'
```



- $\pi_{Name}(\sigma_{ID='5BHIF'}(Teachers \times Classes))$

Query Optimization

- It is usually possible to get the result via several paths
- Thus, it is important to differentiate between good and bad execution plans
- An exact determination of the best execution plan is often not possible
 - We create alternatives and evaluate them with a cost model
 - Utilizing schema information and statistics
- The goal is to find a *sufficiently* good variante

Query Optimization – Guidelines

- We are dealing with a high abstraction level
 - SQL is set based
 - SQL is declarative and not procedural
 - We declare what we want to find
 - We do not define how to find it
- For one 'what' there can be several 'how's
- The 'how' is defined by logical and physical optimization
 - Which operations are performed in which order and how?
- Usually we do not look for the optimal execution plan (and also will not find it), but we settle for a sufficient variant
 - *Avoiding the worst case*

Optimization

■ Logical Optimization

- Often we can find *equivalent expressions* for an expression of the relational algebra
- Order of the relational operators can influence how many tuples are passed on to the next step

■ Physical Optimization

- Methods for table access
- Implementation of the relational operators

Logical Optimization – Basic Idea

Lectures			
Lec#	Title	ECTS	Lecturer#
5001	Grundzüge	6	2137
5041	Ethik	6	2125
5043	Erkenntnistheorie	4	2126
5049	Mäeutik	3	2125
4052	Logik	6	2125
5052	Wissenschaftstheorie	4	2126
5216	Bioethik	3	2126
5259	Der Wiener Kreis	3	2133
5022	Glaube und Wissen	3	2134
4630	Die 3 Kriterien	6	2137

Lecturers			
Pers#	Name	Rank	Room
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	8

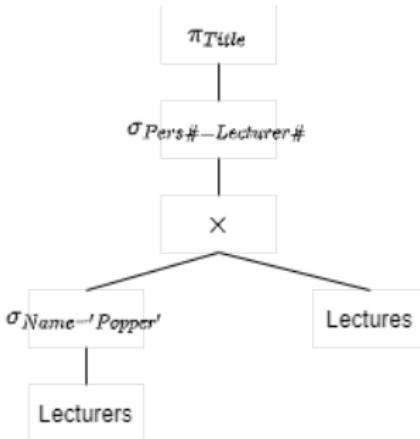
```
SELECT Title FROM Lecturers, Lectures
WHERE Name='Popper'
AND Pers# = Lecturer#
```

- $\pi_{Title}(\sigma_{Name='Popper'} \wedge \text{Lecturer\#}(Lecturers \times Lectures))$

Logical Optimization – Basic Idea

- How many steps are necessary?
 - $7 \text{ Lecturers} * 10 \text{ Lectures} = 70 \text{ Tuple}$
 - But only one fulfills the selection criteria
 - \Rightarrow we wasted resources
- Improvement 1 would be to first find the matching lecturer and then create the cross product
 - $7 + 10 = 17$ steps
- Improvement 2 would be to merge selection and cross product by using a join
 - $\bowtie_{\text{Pers}\# = \text{Lecturer}\#}$

Logical Optimization – Basic Idea



■ $\pi_{Title}(\sigma_{Pers\#=Lecturer\#}(\sigma_{Name='Popper'}(Lecturers) \times Lectures))$

Equivalence in Relational Algebra

1 Join, Union, Intersection and Cross Product are commutative

- $R_1 \bowtie R_2 = R_2 \bowtie R_1$
- $R_1 \cup R_2 = R_2 \cup R_1$
- $R_1 \cap R_2 = R_2 \cap R_1$
- $R_1 \times R_2 = R_2 \times R_1$

2 Selection is interchangeable

- $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

3 Join, Union, Intersection and Cross Product are associative

- $R_1 \bowtie (R_2 \bowtie R_3) = (R_1 \bowtie R_2) \bowtie R_3$
- $R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3$
- $R_1 \cap (R_2 \cap R_3) = (R_1 \cap R_2) \cap R_3$
- $R_1 \times (R_2 \times R_3) = (R_1 \times R_2) \times R_3$

Equivalence in Relational Algebra

- 4 Conjunctions within one selection expression can be split or sequentially executed selections be merged
 - $\sigma_{p_1 \wedge p_2 \wedge \dots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\dots(\sigma_{p_n}(R))\dots))$
- 5 Nested projections can be eliminated:
 - $\pi_{I_1}(\pi_{I_2}(\dots(\pi_{I_n}(R))\dots)) = \pi_{I_1}(R)$
 - If $I_1 \subseteq I_2 \subseteq \dots \subseteq I_n \subseteq R$ is true

Example Rule 5

$$\begin{aligned}\pi_{Rank}(\pi_{Name, Rank}(\pi_{Pers\#, Name, Rank}(Lecturers))) &\equiv \\ \pi_{Rank}(\pi_{Name, Rank}(Lecturers)) &\equiv \\ \pi_{Rank}(Lecturers)\end{aligned}$$

DING

Equivalence in Relational Algebra

- 6 A selection can happen before or after a projection¹
- If the projection does not remove attributes from the selection condition
 - $\pi_I(\sigma_p(R)) = \sigma_p(\pi_I(R))$ if $attr(p) \subseteq I$

Example Rule 6

$$\begin{aligned}\sigma_{Term > 10}(\pi_{Name, Term}(Students)) \equiv \\ \pi_{Name, Term}(\sigma_{Term > 10}(Students))\end{aligned}$$

Students		
Mat#	Name	Term
24002	Xenokrates	18
25403	Jonas	12

¹Please remember: SELECT = projection, WHERE = selection

Equivalence in Relational Algebra

- 7 A selection can happen before or after a join (or cross product)
- If the projection only uses attributes from *one* of the two join arguments
 - $\sigma_p(R_1 \bowtie R_2) = \sigma_p(R_1) \bowtie R_2$ if $\forall attr(p) \in R_1$
 - $\sigma_p(R_1 \times R_2) = \sigma_p(R_1) \times R_2$ if $\forall attr(p) \in R_1$

Example Rule 7

$$\begin{aligned}\sigma_{Rank='C4' \wedge ECTS \geq 4}(\text{Lecturers} \bowtie_{Pers\# = \text{Lecturer}\#} \text{Lectures}) \equiv \\ \sigma_{Rank='C4'}(\text{Lecturers}) \bowtie_{Pers\# = \text{Lecturer}\#} \sigma_{ECTS \geq 4}(\text{Lectures})\end{aligned}$$

Equivalence in Relational Algebra

- 8 It is possible to switch projection and join
- If the join arguments are kept until the join is performed
 - $\pi_I(R_1 \bowtie_p R_2) = \pi_I(\pi_{I_1}(R_1) \bowtie_p \pi_{I_2}(R_2))$

Example Rule 8

$$\begin{aligned}\pi_{Name, Title}(Lecturers \bowtie_{Pers\#=Lect.\#} Lectures) &\equiv \\ \pi_{Name, Title}(\pi_{Name, Pers\#}(Lecturers) \bowtie_{Pers\#=Lect.\#} \pi_{Title, Lect.\#}(Lectures))\end{aligned}$$

- 9 Selections can be replaced with set operations
- $\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$ (Union)
 - $\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$ (Intersection)
 - $\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S)$ (Difference)

Equivalence in Relational Algebra

- 10 The projection operator can be split when using a union
 - $\pi_I(R_1 \cup R_2) = \pi_I(R_1) \cup \pi_I(R_2)$
 - Not possible with Intersection and Difference
- 11 A selection and a cross product can be merged to one join²
 - If the selection condition is a join condition
 - $\sigma_{R_1.A_1=R_2.A_2}(R_1 \times R_2) = R_1 \bowtie_{R_1.A_1=R_2.A_2} R_2$

Example Rule 11

$$\begin{aligned}\sigma_{\textit{Pers}\#=\textit{Lecturer}\#}(\textit{Lecturers} \times \textit{Lectures}) \equiv \\ \textit{Lecturers} \bowtie_{\textit{Pers}\#=\textit{Lecturer}\#} \textit{Lectures}\end{aligned}$$

²cf. Equijoin & Inner Join

Equivalence in Relational Algebra

- 12 Join or selection predicates can be transformed using De Morgan's laws³
- Thus we can move a negation from the outer to the inner scope
 - $\neg(p_1 \vee p_2) = \neg p_1 \wedge \neg p_2$
 - $\neg(p_1 \wedge p_2) = \neg p_1 \vee \neg p_2$

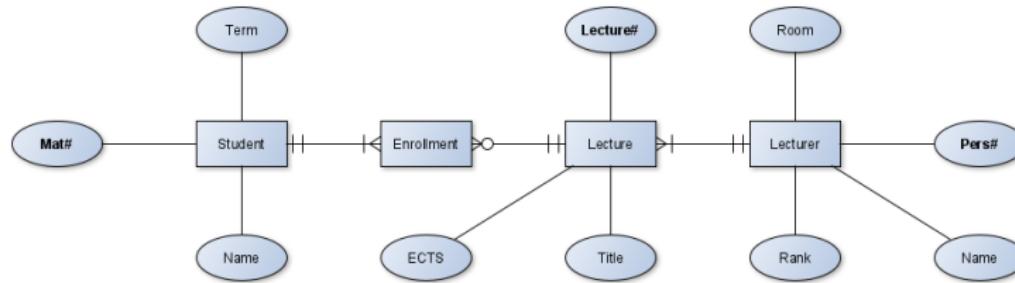
Example Rule 12

$$\begin{aligned}\neg(Lecturer\# = 2125 \wedge ECTS \geq 4) &\equiv \\ (\neg Lecturer\# = 2125) \vee (\neg ECTS \geq 4) &\equiv \\ (Lecturer\# \neq 2125) \vee (ECTS < 4)\end{aligned}$$

³https://en.wikipedia.org/wiki/De_Morgan%27s_laws

Application of Transformation Rules

Transformation Rules – Example



- Student(Mat#, Name, Term)
- Enrollment(Mat#, Lecture#)
- Lecture(Lecture#, Title, ECTS, Lecturer#)
- Lecturer(Pers#, Name, Rank, Room)

Application of Transformation Rules

Transformation Rules – Example – Query

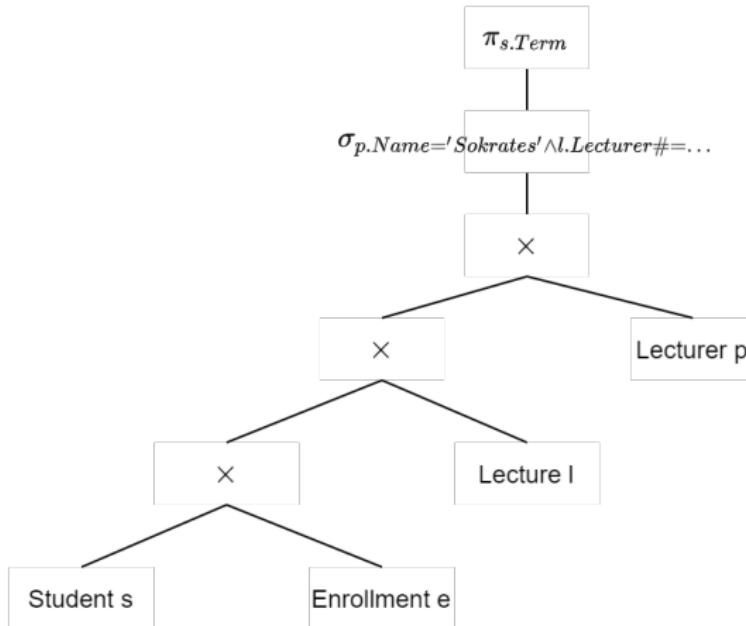
- Question: In which terms are students who are enrolled in a lecture given by Sokrates?

Query

```
SELECT DISTINCT s.Term
FROM Student s, Enrollment e, Lecture l, Lecturer p
WHERE p.Name = 'Sokrates' AND
    l.Lecturer# = p.Pers# AND
    e.Lecture# = l.Lecture# AND e.Mat# = s.Mat#
```

Application of Transformation Rules

Transformation Rules – Example – Tree

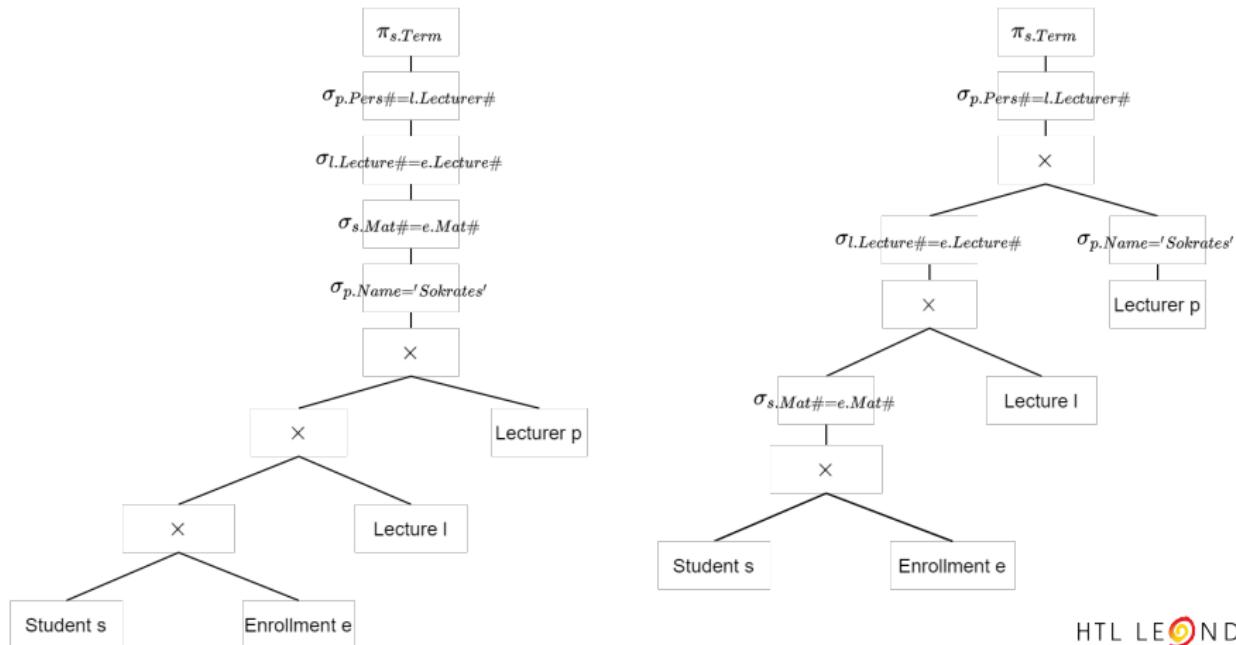


Transformation Rules – Example – Splitting Selection

- We want to perform the selection *as soon as possible* to exclude a big part of all tuples from further processing right away
- According to rule 4 conjunctive parts of a selection can be split
- According to rules 2, 6, 7 & 9 we can move those within an expression

Application of Transformation Rules

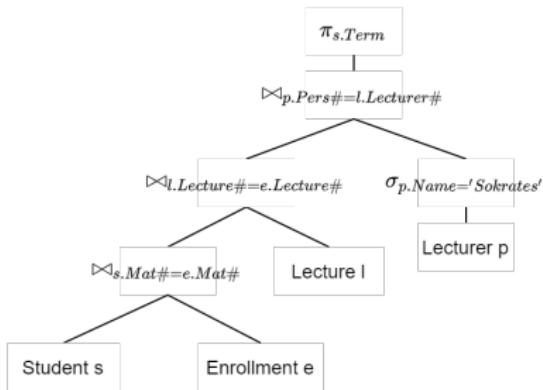
Transformation Rules – Example – Splitting Selection



Application of Transformation Rules

Transformation Rules – Example – Using Joins

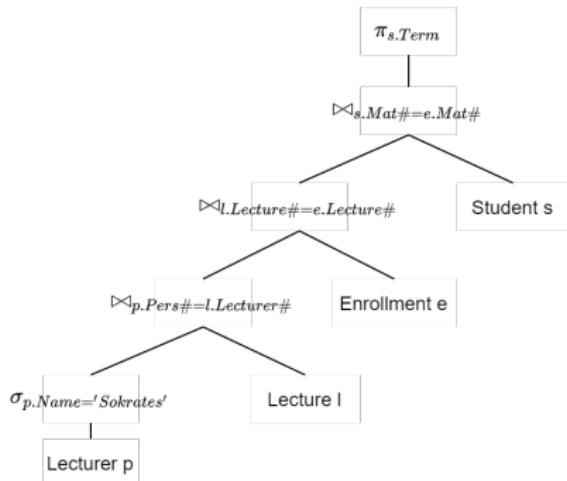
- Joins are more performant than cross products
- According to rule 11 we can use joins instead of cross products
 - \Rightarrow That is our goal
- Tuples:
 - Join Student – Enrollment: 13
 - Join Lecture: 26 (+13)
 - Join Lecturer: 30 (+4)



Application of Transformation Rules

Transformation Rules – Example – Join Order

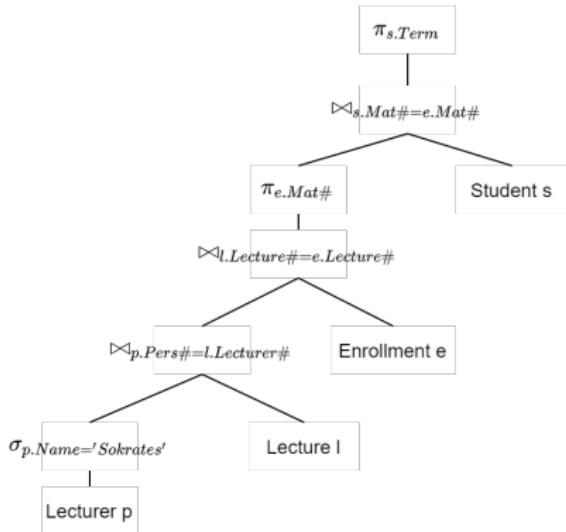
- Changing the order in which joins are performed can reduce the amount of tuples further
 - \Rightarrow We want to find the optimal order
- Tuples:
 - Join Lecturer – Lecture: 3
 - Join Enrollment: 7 (+4)
 - Join Student: 11 (+4)



Application of Transformation Rules

Transformation Rules – Example – Adding Projections

- According to rules 5, 6, 8 & 10 we can move or even add projections
- Two goals:
 - Eliminate duplicates
 - Reduce tuple size
- The process is laborious so don't do it without a good reason
 - e.g. huge attributes like a binary (PDF, image,...)
- ⇒ We removed 1 Tuple



Optimization Heuristics for Canonical Normal Form

- 1 Split selections
- 2 Move selections as far down the tree as possible
- 3 Merge selections and cross products to joins
- 4 Improve order of the joins
- 5 Add additional projections
- 6 Move projections as far down the tree as possible

Physical Optimization

- Physical algebra operators are the implementation of the logical operators
- For one logical operator there can be several different physical operators⁴
- Those operators use the physical structure of the database
 - Indices, Sorting,...

⁴Think: interface & implementations

Iterator

- Allows to put together the separate parts of the execution plan
- Is an abstract data type / interface with the following methods:
 - 1 open: Open Input Stream, Initializations
 - 2 next: Returns the next tuple of the result set
 - 3 close: Close Input Stream, release resources⁵
 - 4 cost: Information about predicted (resource) costs
 - 5 size: Size of the result set
- Iterators are represented as a tree⁶
 - Then combined to one execution plan

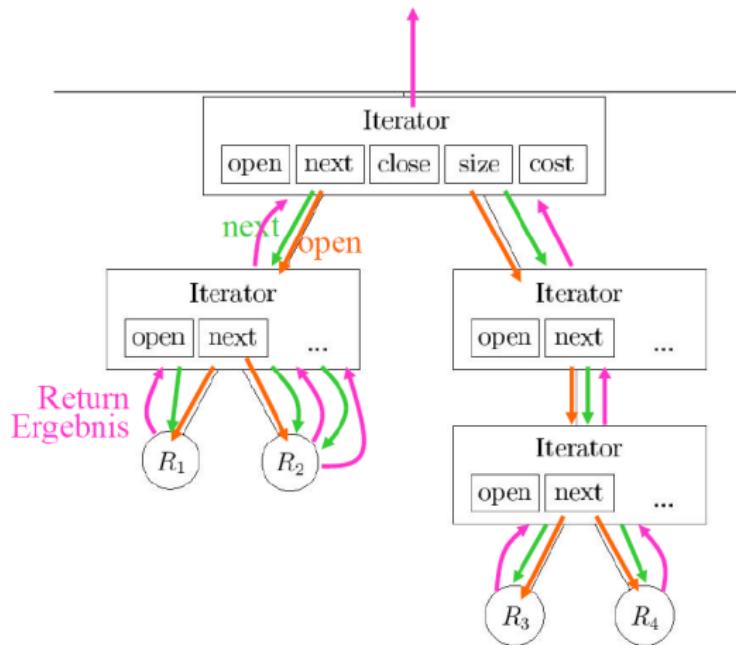
⁵Up to this point the iterator behaves pretty much like a cursor.

⁶Just as the relational operators are.

Iterator

- Traversing:
 - 1 Start at the root iterator
 - 2 Process one tuple after the other (with the `next` command)
until no more data is available
 - 3 The tuples are calculated via child-iterators
 - Recursively down to the leaves
- Intermediate results don't have to be necessarily saved (not even temporarily) for example when doing a projection or selection
 - ⇒ Pipelining
 - Sort & Hash would always need a temporary save
 - Join, Distinct, Group By & Set Operations may need a temporary save depending on the implementation

Iterator Tree – Pull-based Query Evaluation



Selection

- 'Brute-Force' Mode
- **iterator** Select_p
- **open**
 - Open Input Stream
- **next**
 - Get tuple until one fulfills condition p, end if none
 - Return this tuple
- **close**
 - Close Input Stream

- Index Access Mode
- **iterator** IndexSelect_p
- **open**
 - Search in the index for the position of a tuple matching the condition
- **next**
 - Return next tuple while condition fulfilled
- **close**
 - Close Input Stream

Nested Loop Join

- Two loops one nested within the other
- Comparing every tuple of one set with all tuples in the other set
- Example $R \bowtie_{R.A=S.B} S$

Nested Loop Join Example

```
for each  $r \in R$ 
  for each  $s \in S$ 
    if  $r.A = s.B$  then
       $\text{res} := \text{res} \cup (r \times s)$ 
```

Nested Loop Join – with Iterator

- **iterator** NestedLoop_p
- **open**
 - Open left Input Stream
- **next**
 - Is the right Input Stream closed? If yes: open
 - Get tuple from the right stream until condition is met
 - Is right Input Stream exhausted? If yes:
 - Close right Input Stream
 - Request next tuple from left Input Stream
 - Go to **next** start
 - Return the joined set of current left and right tuples
- **close**
 - Close both Input Streams

Nested Loop Join – Page Aware

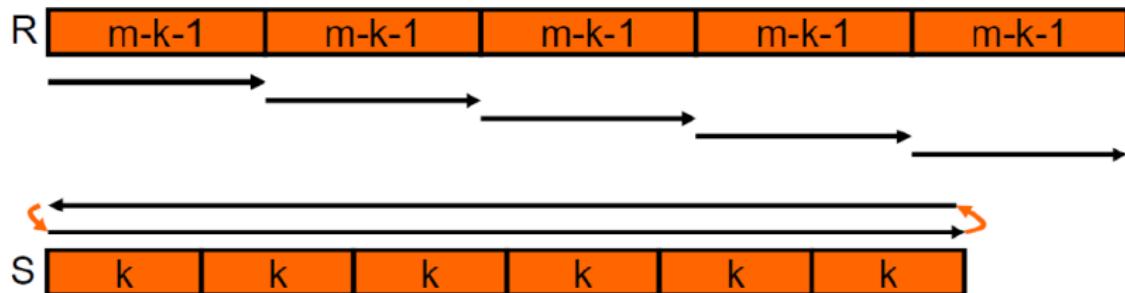
- Acknowledges the page structure when performing joins
- For the join we have m pages available
 - k for the inner relation
 - $m - k$ for the outer relation
- We test all combinations of tuples $r \in R$ & $s \in S$ **currently in the buffer**

Page Aware Nested Loop Join Example

```
for each  $p_R$  of  $R$ 
  for each  $p_S$  of  $S$ 
    for each tuple  $r \in p_R$  and  $p_S \in S$ 
      if  $r.A = s.B$  then
        res := res  $\cup$  ( $r \times s$ )
```

Nested Loop Join – Page Aware – Optimization

- We can optimize the process further by moving through S in a zick-zack order
 - Would save 1 I/O operation per loop iteration of S



Merge Join

- If both Input Streams are sorted by the join attributes we can work more efficiently
 - It might be a viable option to sort ourselves before joining

- Parallel execution top to bottom
- We start with a pointer at the first tuple
- Smallest value is 0 and on the other side 5
 - ⇒ no match, move z_r to 7
- Now 5 is the smallest number, move z_s
 - ⇒ match at 7 (join partner)
- z_r is moved on ⇒ again a match (join) with 7
- Once a join partner is found for the first time it has to be marked so that we are able to reset one side if there are multiple equivalent values on both sides (see join with 8)

R	A	S
...	0	B
	7	5
	7	6
	8	7
	8	8
	10	8
		11

Merge Join – with Iterator

- **iterator** MergeJoin_p

- **open**

- Open both Input Streams
 - Set *current* to the left input
 - Mark the right input

- **next**

- While condition p is not true
 - Set *current* to input with the lower value
 - Call **next** for *current*
 - Mark the other input
 - Return the joined set of current left and right tuples
 - ...

Merge Join – with Iterator

■ **next** (*continued*)

- ...
- Move other input to the next element
- If condition is no longer true or other input is exhausted
 - Call **next** for *current*
 - Did the value of the join attribute in *current* change? If not reset other input to the mark otherwise mark the other input.

■ **close**

- Close both Input Streams

Index Join

- An Index Join utilizes an index on one of the join attributes
- Thus for every tuple in R we can fetch the matching tuples in S from the index on S

- **iterator** IndexJoin_p

- **open**

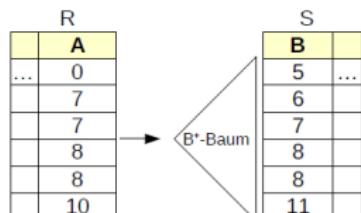
- Open left Input Stream
- Take first tuple
- Search for join attribute value in index

- **next**

- Perform join if index produces one (or more) matching tuples
- If not move left Input Stream forward and search for attribute value in index

- **close**

- Close both Input Streams

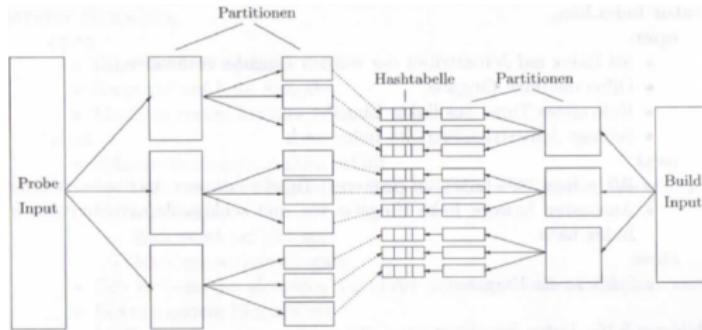


Hash Join

- We want to partition the Input Streams in such a way that the use of a hash table in primary memory (RAM) is possible
- The *smaller relation* becomes the *build input*
 - We partition again and again until the partitions fit into the RAM
 - If m pages are available we use $m - 1$ pages for output and 1 for input
 - The distribution happens recursively via hash function
- The *bigger relation* becomes the *probe input*
 - It also gets partitioned with the hash function
 - But those partitions don't necessarily have to fit into the RAM

Hash Join

- In the next step (= Probe or Matching-Phase) the partitions are compared
 - Load 1 bucket of the Build-Input and create a hash table
 - Load from the respective bucket in the Probe-Input one page after the other and test each tuple



Grouping & Duplicate Elimination

- Three options:
 - Brute-Force with nested loops
 - If a sort order is already established only one set iteration with duplicate removal is required
 - A secondary index can be used – e.g. with a B+ tree the leaves contain either tuples or pointers in sorted order
- For duplicate elimination often hashing is used
 - Build-Phase: Hash function for combination of all attributes
 - For every bucket an in-memory hash table is used discarding duplicates

Projection & Union

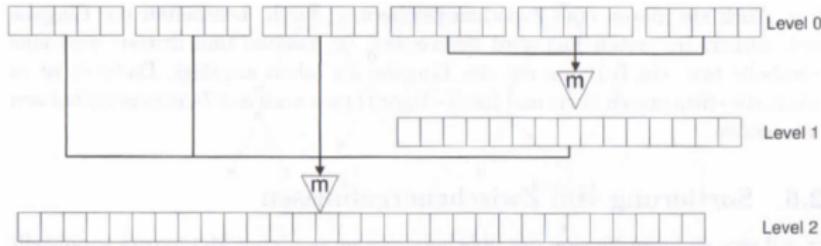
- For projection and union no automatic duplicate elimination is present in physical algebra.
- If it is desired it has to be explicitly stated
- So the default implementation is pretty simple:
 - Projection: Tuple of the input stream is reduced to the requested attributes.
 - Often combined with another step, e.g. a join
 - Union: Into the output stream we concatenate all tuples of the left and right input streams

Sorting

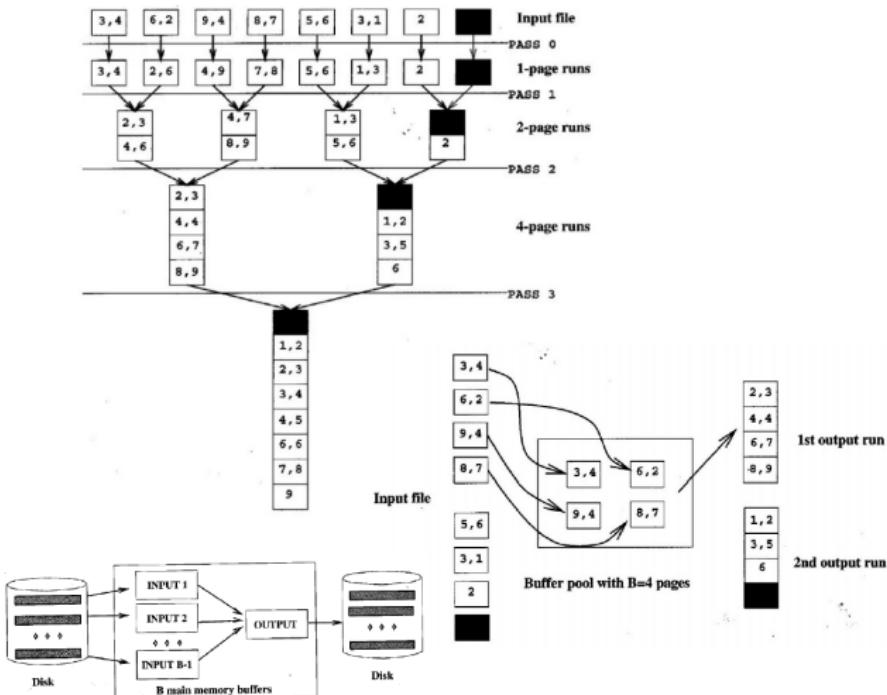
- Use Cases:
 - Sorting is *not* a relational operation but required for some implementations (e.g. Merge Join)
 - Order By clause
 - One option for duplicate elimination
- A problem is sorting huge amounts of data which is primarily persisted in secondary storage
 - No in-memory sorting (quick sort,...) possible
 - ⇒ Merge Sort: piecewise sorting and then combining

Merge Sort

- Initialization; Level-0 Runs
 - Every page is read, then sorted in primary memory and the result is written to a temporary relation
- Merge
 - If m pages of primary memory are available we merge $m - 1$ runs
 - The free page can be used for output
- In intermediate steps as few runs as possible should be merged



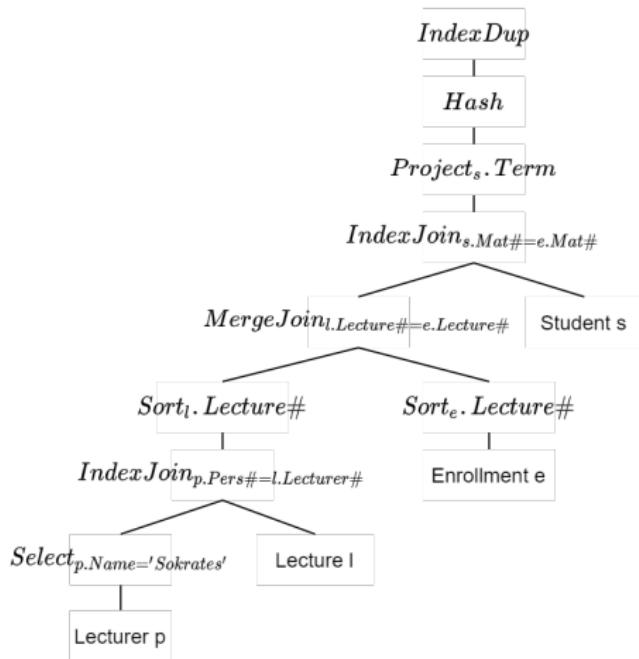
Merge Sort



Logical Algebra Translation



Logical Algebra Translation – Example



Cost Model

- In a perfect world we would find the optimal execution plan
- To do this all possible plans would have to be generated and tested ⇒ illusional
- Heuristic optimizations are supposed to find acceptable results in a short time – *for most cases*
- Cost models help when comparing execution plans, because they estimate the effort of the various operators
 - Using parameters like indices, cluster, cardinalities, distribution,...

Query Processing
oooooooo

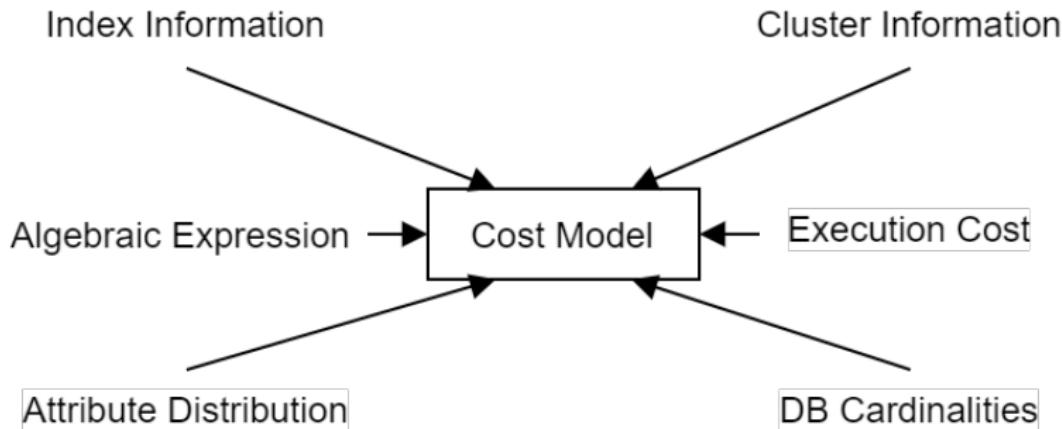
Logical Optimization
oooooooooooo
oooooooooooo

Physical Optimization
oooooooooooo
oooooooooooo

Cost Models
○●oooooooo
oooooooooooo

Selectivity

Cost Model



Selectivity

- Selectivity defines the relative amount of tuple which fulfill the selection criteria p
- Selectivity with condition p
 - $sel_p := \frac{|\sigma_p(R)|}{|R|}$
- Selectivity with a join of R and S
 - $sel_{RS} := \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| * |S|}$

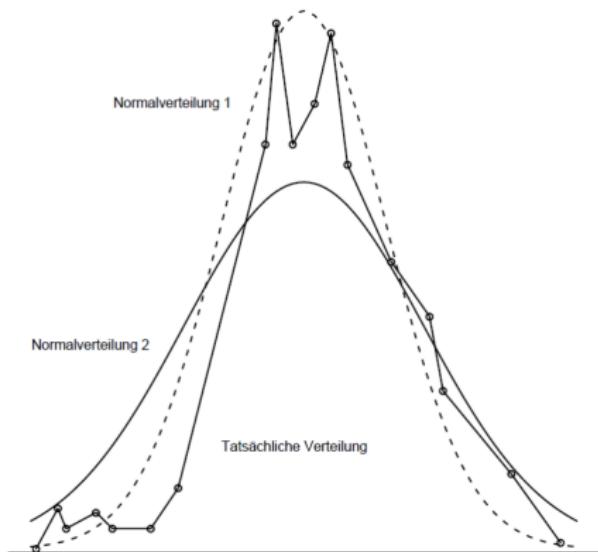
Selectivity Prediction

- Selectivity for a key ($\sigma_{R.A=c}$)
 - $sel_{R.A=c} := \frac{1}{|R|}$
- Selectivity for a normal distribution of attribute ($R.A$) values to i different values
 - $sel_{R.A=c} := \frac{1}{i}$
- Selectivity for an equijoin with $R.A$ as key and $S.B$ as foreign key
 - $sel_{R\bowtie_{R.A=S.B} S} := \frac{1}{|R|}$
 - Ex: R=Cust(CustNo,...), S=Order(OrderNo,...,CustNo)
 - The result set will be equal to $|Order|$, because every order has one customer assigned
 - $sel_{RS} := \frac{|R\bowtie S|}{|R \times S|} = \frac{|R\bowtie S|}{|R| * |S|} = \frac{|S|}{|R| * |S|} = \frac{1}{|R|}$

Selectivity

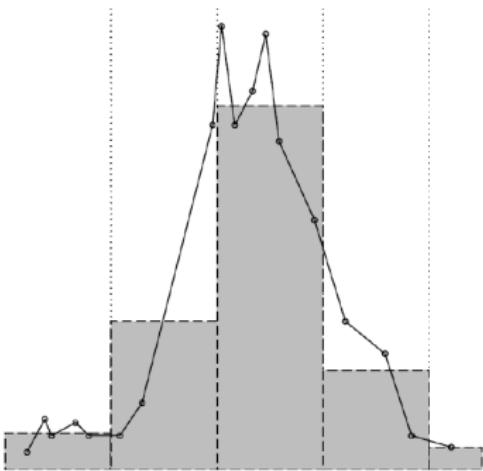
Selectivity Prediction – Parametric Distribution

- We try to find parameters for a function so that it closes in on the actual distribution
- Calling this function returns an estimate for the number of tuples



Selectivity Prediction – Histogram

- The value range of an attribute is split into sub-ranges
- Then we determine the relative probability of those sub-ranges
- We can flexibly close in on the actual distribution
- Two approaches:
 - Equi-Width: intervals have the same size
 - Equi-Depth: intervals have a different width but all contain the same number of elements



Selectivity Prediction – Random Sample

- Very easy to implement
- A random set of tuples of a relation are analyzed
- Their distribution is assumed to be true for the whole relation
- But we need expensive secondary storage access

Cost Estimation

- Biggest cost factor is secondary storage access
- CPU cycles are comparatively cheap
- Notation:
 - m : Number of pages in the DB buffer
 - b_R (b_S): Number of pages (in secondary storage) for relation R (S)
 - M_R (M_S): Number of tuples of relation R (S)
 - p_R (p_S): Number of tuples per page

Estimation

Cost Estimation – Selection

- Selection with a relation in secondary storage as input \Rightarrow read all pages
 - Cost: b_R
- Selection with an input from another operator \Rightarrow filtering only, no additional reads
 - Cost: 0 (we ignore CPU cycles)
- Selection with an index:
 - B+ tree with a height of max. 4, root and parts of the first level in primary memory – Cost: $\leq 4(\sim 2)$
 - Hash Index
 - Static hashing (no overflow) – Cost 1
 - Extendable Hashing – Cost +1
 - If index only contains TID – Cost: +1

Cost Estimation – Sorting

- Creation of level-0-runs: read every page, sort and then write it
 - Cost: $2b_R$
- Length level-0-runs: m pages
 - Number of level-0-runs: $i = [b_R/m]$
- For every pass: $m - 1$ runs merged into one
 - Number of required passes: $l = [\log_{m-1}(i)]$
- For every pass: all pages read and written again
 - Cost per pass: $2b_R$
- Total Cost:
$$2b_R + l * 2b_R = 2b_R * (1 + l) = 2b_R * \left(1 + [\log_{m-1}(\lceil \frac{b_R}{m} \rceil)]\right)$$

Cost Estimation – Joins – Nested Loop Join

- Every page of R is read once – Cost: b_R
- For every page of R every page of S is read once – Cost: $b_R * b_S$
- Total Cost: $b_R + b_R * b_S$

Example			Total Cost: $1000 + 1000 * 500 = 501000 \text{ I/Ops}$ (10ms per I/O): ~1.4 hours
No. of pages	$b_R = 1000$	$b_S = 500$	
No. of tuples	$M_R = 100000$	$M_S = 50000$	
Tuples/page	$p_R = 100$	$p_S = 100$	
DB buffer	$M = 100$		

Cost Estimation – Joins – Block Nested Loop Join

- Every page of R is read once – Cost: b_R
- For every block of $m - k - 1$ pages of R every page of S has to be read once
 - From the second run of S the first k pages are already in the buffer
 - First run of S : b_S
 - Further runs of S : $(b_S - k)$
 - Total runs: $\frac{b_R}{m-k-1}$
- Total Cost: $b_R + k + \frac{b_R}{m-k-1} * (b_S - k)$

Cost Estimation – Joins – Block Nested Loop Join

- Should R or S be the bigger relation?
- How big should you choose k ?

Examples

$b_R = 1000, b_S = 500$

$k = 32 \quad 8017$ reads

$k = 16 \quad 6847$ reads

$k = 1 \quad 6092$ reads

$b_R = 500, b_S = 1000$

$k = 32 \quad 7755$ reads

$k = 16 \quad 6443$ reads

$k = 1 \quad 5597$ reads

Cost Estimation – Joins – Index Nested Loop Join

- Every page of R is read once – Cost: b_R
- For every tuple of R we access a tuple of S
 - Depending on the index type – Cost: $c = 1 - 5$ I/O ops
- Total Cost:
 - If at most 1 tuple in S (B key in S): $b_R + c * M_R$
 - If multiple hits in S possible:
 - Clustered index: $b_R + M_R * (c + b_S * sel_{RS})$
 - Non-clustered index: $b_R + M_R * (c + M_S * sel_{RS})$

Cost Estimation – Joins – Sort Merge Join

- Sorting R – Cost: $2b_R * (1 + l_R)$
- Sorting S – Cost: $2b_S * (1 + l_S)$
- Cost of merge join if:
 - A is key in R or B is key in S
 - 1 run each for R and S
 - Cost: $b_R + b_S$
 - For every R there can be multiple tuples in S
 - Worst case *nested loop* if almost all values of $R.A$ and $S.B$ are equal

Cost Estimation – Joins – Sort Merge Join – Example

- Cost for sorting:

- Consider:

- $i_R = \lceil \frac{b_R}{m} \rceil = \lceil \frac{1000}{100} \rceil = 10$ $I_R = \lceil \log_9(10) \rceil = 1$

- $i_S = \lceil \frac{b_S}{m} \rceil = \lceil \frac{500}{100} \rceil = 5$ $I_S = \lceil \log_9(5) \rceil = 1$

- Cost for sorting R : $2 * 1000 * (1 + 1) = 4000$

- Cost for sorting S : $2 * 500 * (1 + 1) = 2000$

- Cost for merge join:

- B is key in S

- Cost: $1000 + 500$

- Total Cost: $4000 + 2000 + 1000 + 500 = 7500$

Cost Estimation – Joins – Hash Join

- Build-Phase (read and write once) – Cost: $2(b_R + b_S)$
- Probe-Phase (each page of R and S once) – Cost: $(b_R + b_S)$
- Total Cost: $3 * (b_R + b_S)$

Example			
No. of pages	$b_R = 1000$	$b_S = 500$	
No. of tuples	$M_R = 100000$	$M_S = 50000$	
Tuples/page	$p_R = 100$	$p_S = 100$	
DB buffer	$M = 100$		Total Cost: $3 * (1000 + 500) = 4500I/Oops$ (10ms per I/O): ~45 secs

Introduction

oooooooo
ooo

Collections

oooooooooooooo
oooooooooooooooooooo

Data Frames

oooooooooooooo
ooooooo

Connections

oooooo
oooo

Advanced

oooooo
ooooo

R Programming Language

Markus Haslinger

DBI

Agenda

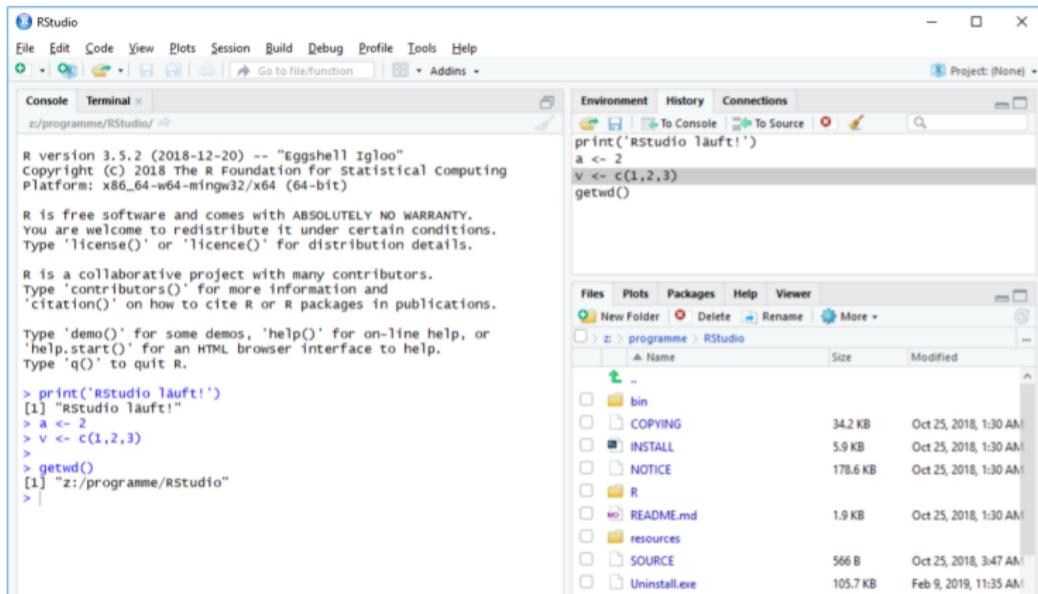
- 1** Introduction
 - Basics
 - Programming
 - 2** Collections
 - Vectors & Lists
 - Matrices
 - 3** Data Frames
 - Data Frames
 - Data Frame Usage
 - 4** Connections
 - Files
 - Database & Web
 - 5** Advanced
 - R Features
 - Miscellaneous

What is R?

- Free programming language
- Meant for statistical calculations and plots
- Dynamic type system
- Functional
- Interpreted
- Extensible via packages

Basics

IDE RStudio



R Files

- You can use the REPL to try out things
- Variables will be available in the current R instance scope
- For longer programs you can write R files

- Comments can be added using #

Data Types

- Everything is an object
 - ⇒ no primitives
 - Five types:
 - character: "a", "abc" (equals char & string, double quotes)
 - numeric: 2, 2.1 (equals double)
 - integer: 2L (the 'L' literals forces an integer)
 - logical: TRUE & FALSE (the boolean type)
 - Be careful to use capital letters!
 - complex: 1+4i (complex numbers, real and imaginary parts)
 - Show value ranges by using .Machine in the R console

Variables

- Dynamic typing, no declaration necessary
- You can only use what has been assigned before
- Assignment with <- & ->
 - '=' will work in most but not all cases so just stick to the notation above
- Read access by simply using the variable name
- `class(<var>)` returns the type of a variable
- Comparisons work as usual:
 - <, <=, >, >=, !=, ==
- You do not have to worry about memory – a GC is in place and our basic scenarios will not be memory constrained

Arithmetics

- Arithmetic operations mostly work as usual

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Power	^
Modulo	%%
Integer Division	%/%

Logical Operators & Conditionals

■ Logical Operators:

- `&`, `l`, `!` ⇒ and, or, not
- `&&`, `||` ⇒ compares first element of a vector

Conditional

```
if (cond) { }
else if (cond2) { }
else { }
```

Loops

- while loop as usual
- for loop similar to foreach (iterating over collection)

Example

```
i <- 0
v <- c(1,2,3)
while(i < 2){
  for (j in v){
    print(j)
  }
  i <- i+1
}
```

Functions

- Functions are basically lambdas assigned to variables
 - R ⇒ functional programming language
- Return types are not specified, just use `return(<val>)` if you want to return something (default NULL)
- Parameters can have default values (thus becoming optional)

Example

```
fAdd <- function (x, y, z=0) { return (x+y+z) }  
print(fAdd(1,2))  
print(fAdd(1,2,3))
```

Vectors

- A vector is a one-dimensional array, containing any objects
 - If different types are used R converts all elements to one
 - Syntax: `c(elem1, ..., elemN)`

Example

```
v1 <- c(1, TRUE, "test") # c = combine function  
class(v1) # same as typeof  
print(v1)
```

```
# Output  
[1] "character"  
[1] "1" "TRUE" "test"
```

Vectors – Assigning Names

- Use the `names` function to assign names to values of a vector

Vector Names

```
noOfDays <- c(31,28,31,30,31,30,31,31,30,31,30,31)
names(noOfDays) <- c('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                     'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
print(noOfDays)
```

```
# Output  
# Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec  
# 31 28 31 30 31 30 31 31 30 31 30 31
```

Vectors – Operations

- Two vectors can be used with the `x`, `-`, `*`, `/` operator
 - Operating on values at the same position
 - Requires vectors to have same length
- There are also some operations which can be applied to one (or multiple) vectors:
 - `sum`
 - `min / max`
 - `var` (variance)
 - `prod` (product)
 - `sd` (standard deviation)
- Use vectors with values of an appropriate data type

Vectors – Operations

Vector Operation Examples

```
nums1 <- c(1,2,3)
nums2 <- c(4,5,6)
print(nums2 / nums1)
prod(nums1)
sum(nums1, nums2)
```

```
# Output  
# 4.0 2.5 2.0  
# 6  
# 21
```

Vectors – Comparison

- You can compare a value with a vector
- This operation will return a result in which every value of the vector has been compared to the one value

Example

```
v <- c(1,2,3,4,5)
v >= 3

# Output
# FALSE FALSE TRUE TRUE TRUE
```

Vectors – Indexing

- You can use an indexer access to get values of a vector
- Index starts at 1!
- You can also create a *slice*

Example

```
v <- c(1,2,3,4,5)
```

```
v[2]
```

```
v[c(2,4)]
```

```
v[2:4]
```

```
# Output
```

```
# 2
```

```
# 2 4
```

```
# 2 3 4
```

DING

Vectors – Indexing by Name

- Remember how to assign names to vector values
- If names are assigned those can be used to get the related value

Example

```
v <- c(1,2,3,4,5)
names(v) <- c("a", "b", "c", "d", "e")
v["d"]
```

```
# Output
# d
# 4
```

Vectors – Filter

- You can use a comparison operator in the indexer to filter vector values based on some criteria

Example

```
v <- c(1,2,3,4,5)  
v[v>3]
```

```
# Output  
# 4 5
```

Vectors – Filter Assignment

- A new vector can be created based on an existing one by applying a filter

Example

```
v <- c(1,2,3,4,5)
v2 <- v[v>3]
v2

# Output
# 4 5
```

Vectors – Create by Range

- A new vector with consecutive numbers can be created by using the range operator

Example

```
v <- 1:10  
v  
  
# Output  
# 1 2 3 4 5 6 7 8 9 10
```

Lists

- Similar to a Vector, but can hold different data types
- Values can have labels and can be accessed by those
 - Lists support indexer notation
- Lists are used to organize different objects (matrices, data frames,...) which will be used in actual operations

Lists – Examples

List Examples

```
myList <- list(vec=c(1,2,3), foo="bar")
myList["vec"]
myList[2]
class(myList)
```

```
# Output
# $vec
# [1] 1 2 3
# $foo
# [1] "bar"
# "list"
```

Lists – Access actual item

List Examples

```
myList <- list(vec=c(1,2,3), foo="bar")
myList$vec # $ notation
myList[[2]] # double-bracket notation
```

```
# Output
# [1] 1 2 3
# [1] "bar"
```

Lists – Structure Information

- To get information about the content of a list you can use the `str` function

List Structure

```
myList <- list(c(1,2,3), foo="bar")
str(myList)
```

```
# Output
# List of 2
# $ : num [1:3] 1 2 3
# $ foo: chr "bar"
```

What are Matrices?

- A matrix is
 - a rectangular array of
 - numbers
 - symbols
 - expressions
 - arranged in rows and columns
- An element is an individual item in a matrix
- A row vector is a matrix with a single row
- A column vector is a matrix with a single column
- A square matrix is a matrix with the same number of rows and columns

$$A = \begin{pmatrix} 2 & 8 & \frac{1}{5} \\ \frac{5}{2} & 3 & 4 \end{pmatrix}$$

Creating a Matrix

- A matrix is a two dimensional data structure
- Create it using the `matrix` function
- Syntax: `matrix(data = <data>, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)`
 - `nrow/ncol`: desired number of rows and columns
 - `byrow`: flag to indicate if the matrix should be filled by rows (TRUE) or columns (FALSE)
- Assigning names:
 - `colnames` function
 - `rownames` function

Creating a Matrix

Example

```

gradeNames <- c('LF1','LF2','Quiz','MA')
studentNames <- c('Sepp','Horst')
seppGrades <- c(1,4,4,5)
horstGrades <- c(2,5,4,3)
allGrades <- c(seppGrades, horstGrades) # *not* vector of vectors — concats!
gradesTable <- matrix(allGrades, byrow=TRUE, nrow=2)
colnames(gradesTable) <- gradeNames
rownames(gradesTable) <- studentNames
gradesTable

# Output
#      LF1 LF2 Quiz MA
# Sepp   1   4   4   5
# Horst  2   5   4   3

```

Creating a Matrix – byrow

byrow Difference

```
m1 <- matrix(1:10, byrow=FALSE, nrow=2) #default
m2 <- matrix(1:10, byrow=TRUE, nrow=2)

# Output m1
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    1    3    5    7    9
# [2,]    2    4    6    8   10

# Output m2
#      [,1] [,2] [,3] [,4] [,5]
# [1,]    1    2    3    4    5
# [2,]    6    7    8    9   10
```

Repetition (from AM): Matrix Arithmetics

- Addition: add each element in the first matrix to the corresponding one in the second.
- Subtraction: subtract each element in the second matrix from the corresponding one in the first.
- This requires both matrices to have the same size!
- Scalar multiplication increases magnitude without changing direction, e.g. $2 * \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$
- When multiplying matrices, the elements of the rows in the first matrix are multiplied with corresponding columns in the second matrix.

Matrix Arithmetics

- Similar to vectors arithmetic and logical operations can be applied
- Also like with a vector those are performed per element

Matrix Power Operation

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
m1 ^ 2
```

```
# Output
#      [,1] [,2] [,3]
# [1,]    1    4    9
# [2,]   16   25   36
# [3,]   49   64   81
```

Matrix Arithmetics

- You can also check if elements fulfill a condition

Matrix Element Condition Check

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
m1 > 5

# Output
#   [,1]  [,2]  [,3]
# [1,] FALSE FALSE FALSE
# [2,] FALSE FALSE  TRUE
# [3,]  TRUE  TRUE  TRUE
```

Matrix Arithmetics

- It is also possible to calculate with multiple matrices

Matrix Multiplication

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
m1 * m1
```

```
# Output
#      [,1] [,2] [,3]
# [1,]    1    4    9
# [2,]   16   25   36
# [3,]   49   64   81
```

Matrix Arithmetics

- Be careful: *actual* matrix multiplication requires `%*%` operator
- And the number of columns in the first matrix has to equal the number of rows in the second

Matrix Multiplication

```
m1 <- matrix(1:6, byrow=TRUE, ncol=2)
m2 <- matrix(1:6, byrow=TRUE, nrow=2)
m1 %*% m2
```

```
# Output
#      [,1] [,2] [,3]
# [1,]    9   12   15
# [2,]   19   26   33
# [3,]   29   40   51
```

Matrix – Filter

- Similar to a vector you can apply a filter to a matrix
- The result is a vector of elements matching the criteria

Matrix Filtering

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
m1[m1>5]
```

```
# Output – note unordered!
# 7 8 6 9
```

Matrix – Operations – Row Sum

- Use the `rowSums` function to calculate the sum for each row

Matrix RowSum

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
rowSums(m1)
```

```
# Output
# 6 15 24
```

Matrix – Operations – Column Sum

- Use the `colSums` function to calculate the sum for each column

Matrix ColSum

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
colSums(m1)
```

```
# Output
# 12 15 18
```

Matrix – Operations – Mean

- In a similar fashion the `rowMeans` and `colMeans` functions can be used to calculate the respective mean

Matrix Means

```
m1 <- matrix(1:9, byrow=TRUE, ncol=3)
colMeans(m1)
rowMeans(m1)
```

```
# Output
# 4 5 6
# 2 5 8
```

Matrix – Operations – Add Row

- A matrix is immutable
- But you can use the `rbind` function to create a new matrix with the data from the old and an additional row

Matrix Add Row

```
m1 <- matrix(1:6, byrow=TRUE, ncol=3)
v1 <- c(-1, -2, -3)
m2 <- rbind(m1, v1)
m2

# Output
#   [,1] [,2] [,3]
# 1    1    2    3
# 4    5    6
# v1 -1   -2   -3  # note how the variable name was used as row identifier
```

DING

Matrix – Operations – Add Column

- Use the `cbind` function to create a new matrix with the data from the old and an additional column

Matrix Add Row

```
m1 <- matrix(1:6, byrow=TRUE, nrow=3)
v1 <- c(-1, -2, -3)
m2 <- cbind(m1, v1)
m2

# Output
#      v1    # note how the variable name was used as column header
# [1,] 1 2 -1
# [2,] 3 4 -2
# [3,] 5 6 -3
```

Matrix – Indexing

- Like a vector a matrix can also be indexed
- Keep in mind that we have two dimensions in a matrix
- So indexing also requires two dimensions
 - Similar to a two dimensional array
 - But with distinct differences
 - Separate by using a comma (,)
- One dimension can be omitted if all values of it are relevant

Matrix – Indexing

Matrix Indexing Example

```
m1 <- matrix(1:9, byrow=TRUE, nrow=3)
m1[3,1]
m1[2,]
m1[,2]

# Output
# 7
# 4 5 6
# 2 5 8
```

Matrix – Slicing

- Slicing operations can also be performed
- Like indexing, slicing can be applied to
 - both directions or
 - only one (ommitting the other)
- Can be combined with indexing (e.g. `matrix[2:5,8]`)

Matrix – Slicing

Matrix Slicing Example

```
m1 <- matrix(1:16, byrow=TRUE, nrow=4)
m1[1:2,3:4]
# 3 4
# 7 8
m1[2:3,]
# 5 6 7 8
# 9 10 11 12
m1[,2:3]
# 2 3
# 6 7
# 10 11
# 14 15
m1[1:3,2]
# 2 6 10
```

DING

Categorial Matrices

- A *factor* in R describes a nominal attribute of data
- For example the gender of study participants
- It can be determined with the **factor** method
 - Which will basically calculate the distinct entries to be used further
- The parameter **ordered** allows to define if the order is relevant
 - For example the order between 'm' & 'f' is not important
 - But the order between 'cold', 'temperate' and 'hot' is

Categorial Matrices

Categorial Matrices Example

```
gender <- factor(c('m','m','f','f','m','f'))  
gradeLevels <- c('NGD','BEF','SGT')  
grades <- c('NGD','BEF','SGT','BEF','SGT','NGD')  
factGrades <- factor(grades, order=TRUE, levels=gradeLevels)  
  
gender  
# m m f f m f  
# Levels: f m  
factGrades  
# NGD BEF SGT BEF SGT NGD  
# Levels: NGD < BEF < SGT  
summary(factGrades)  
# NGD BEF SGT  
# 2 2 2
```

What are Data Frames?

- Data Frames are the 'workhorse' of R data structures
 - Thus they are quite important
- Most collections so far were constrained to one data type for all elements (vector, matrix) or did not provide a lot of functionality (lists)
- Data Frames allow us to have named rows and columns
- Think of a Data Frame as something like an Excel worksheet
- R has basically been built to process Data Frames
 - Despite being theoretically general purpose

Built-in Data Frames

- A little unusual for a programming language R includes some sample data
- This shows the huge focus of this language towards data analysis (vs. general purpose)
- Use the `data` function to show them



A screenshot of the RStudio interface. The top menu bar shows 'File', 'Edit', 'View', 'Project', 'Session', 'Help'. Below the menu is a toolbar with icons for file operations. The main area is divided into three panes: 'Console' (active), 'Terminal', and 'Jobs'. In the 'Console' pane, the user has typed '`> data()`' and is pressing the Enter key. A red arrow points from the bottom left towards the Enter key. To the right of the console, a list of datasets from the 'datasets' package is displayed, each with a brief description. The 'data()' function is also listed at the bottom of the list.

```
Data sets in package 'datasets':
AirPassengers      Monthly Airline Passenger Numbers 1949-1960
BJsales            Sales Data with Leading Indicator
BJsales.lead (BJsales) Sales Data with Leading Indicator
BOD                Biochemical Oxygen Demand
CO2                Carbon Dioxide Uptake in Grass Plants
ChickWeight        Weight versus age of chicks on different diets
DNase              Elisa assay of DNase
EuroStockMarkets  Daily Closing Prices of Major European Stock Indices,
1991-1998          Determination of Formaldehyde
Formaldehyde       Head and Brain Content of Formaldehyde
```

Console Terminal Jobs

> data()

Data Frames – Functions

- There are a couple of useful functions for Data Frames:
 - head returns the first 1..6 rows
 - tail returns the last 1..6 rows
 - str returns information about the structure of a Data Frame
 - summary returns a summary of the contained data
 - The function `data.frame` allows to create a new data frame
 - Syntax: `data.frame(<vec1>, [vec2, ... vecN])`
 - Each vector represents a column

Data Frames – Creation

Data Frame Creation Example

```
days <- c('Mon','Tue','Wed','Thu','Fri','Sat','Sun')
temps <- c(22.2,21,23,24.3,25,21.8,26.1)
rained <- c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE)
weather <- data.frame(days, temps, rained)
weather

# Output
# days temps rained
# 1 Mon 22.2 TRUE
# 2 Tue 21.0 FALSE
# 3 Wed 23.0 TRUE
# 4 Thu 24.3 TRUE
# 5 Fri 25.0 FALSE
# 6 Sat 21.8 TRUE
# 7 Sun 26.1 FALSE
```

DING

Data Frames – Creation

- It is also possible to create an empty Data Frame (to be populated later) using the `data.frame` function without any arguments(`()`)
 - When populating a Data Frame from vectors the columns can be names
 - e.g. `df <- data.frame(col1 = <v1>, [col2 = <v2>[,...]])`

Data Frames – Indexing

- Similar to a matrix by defining row(s) and/or column(s)

Data Frame Indexing Example

```
weather[2,]  
#   days temps rained  
# 2  Tue  21.0 FALSE  
  
weather[,2]  
# 22.2 21.0 23.0 24.3 25.0 21.8 26.1  
weather[2,2]  
# 21
```

Data Frames – Indexing

- It is also possible to use the column name(s)

Data Frame Indexing Example

```
weather[2:4,'temp']  
# 21.0 23.0 24.3  
weather[,c('days','rained')]  
#   days rained  
# 1 Mon  TRUE  
# 2 Tue FALSE  
# 3 Wed  TRUE  
# 4 Thu  TRUE  
# 5 Fri FALSE  
# 6 Sat  TRUE  
# 7 Sun FALSE
```

Data Frames – Indexing

- Columns can be accessed via '\$' notation as well

Data Frame Indexing Example

```
weather$rained
# TRUE FALSE TRUE TRUE FALSE TRUE FALSE
weather['rained']
#   rained
# 1  TRUE
# 2 FALSE
# 3  TRUE
# 4  TRUE
# 5 FALSE
# 6  TRUE
# 7 FALSE
```

Data Frames – Subset

- Retrieve the subset of a Data Frame fulfilling a specific condition with the `subset` function

Data Frame Indexing Example

```
subset(weather, subset = rained == TRUE)
#   days temps rained
# 1 Mon  22.2  TRUE
# 3 Wed  23.0  TRUE
# 4 Thu  24.3  TRUE
# 6 Sat  21.8  TRUE
```

Data Frames – Ordering

- The `order` function can be used to sort the entries of a Data Frame
 - The function expects a Data Frame and a column (name) to sort by
 - Returned is a vector containing *not* the values but the index numbers in sorted order
 - This vector can then be passed to the indexer to retrieve the Data Frame in sorted order

Data Frames – Ordering

Data Frame Ordering Example

```
sortedWeather <- order(weather['temps'])  
sortedWeather  
# 2 6 1 3 4 5 7  
weather[sortedWeather,] # mind the comma!  
#   days temps rained  
# 2  Tue  21.0 FALSE  
# 6  Sat  21.8 TRUE  
# 1  Mon  22.2 TRUE  
# 3  Wed  23.0 TRUE  
# 4  Thu  24.3 TRUE  
# 5  Fri  25.0 FALSE  
# 7  Sun  26.1 FALSE
```

Data Frames – Ordering

- To achieve descending order use a '-'

Data Frame Descending Order Example

```
sortedWeather <- order(-weather['temp']) # mind the '-'  
sortedWeather  
# 7 5 4 3 1 6 2  
weather[sortedWeather,2:3]  
#   temps rained  
# 7  26.1 FALSE  
# 5  25.0 FALSE  
# 4  24.3 TRUE  
# 3  23.0 TRUE  
# 1  22.2 TRUE  
# 6  21.8 TRUE  
# 2  21.0 FALSE
```

DING

Data Frame – Operations

- `nrow`: returns number of rows
 - `ncol`: returns number of columns
 - `rownames`: returns names of rows
 - `colnames`: returns names of columns

Data Frames – CSV Data Source

- Use the `read.csv` function (passing the file path) to read data from a CSV file to a Data Frame
- The `write.csv` function (passing Data Frame and file path) can be used to create a CSV file from a Data Frame
 - This function will also export the index as a column (1,2,3,...)
 - So if importing after exporting you'll get an additional column

Data Frame from CSV Example

```
df <- read.csv('path/to/file.csv')
```

Data Frame – Single Element

- To access a single element use the double-square-bracket notation → returns element instead of new Data Frame

Data Frame Single Element Example

```
weather[[2,3]]  
# FALSE  
weather[[3,2]]  
# 23  
weather[[3,2]] <- 99.9 # re-assignment!  
  
weather[[3,'temp']] # using column name  
# 99.9
```

Data Frame Usage

Data Frame – Adding Row

- 1 Create a new Data Frame of the same format
- 2 Use the rbind function to combine the two

Data Frame Add Row Example

```
weatherAdd <- data.frame(days='Mon2', temps = 24.3, rained=FALSE)
weather2 <- rbind(weather, weatherAdd)
weather2[5:8,]

# Output
# days temps rained
# 5 Fri 25.0 FALSE
# 6 Sat 21.8 TRUE
# 7 Sun 26.1 FALSE
# 8 Mon2 24.3 FALSE
```

DING

Data Frame Usage

Data Frame – Adding Column

- Use the '\$' notation to add a new column (new name)
- You'll need to provide row values for it
 - For every existing row!

Data Frame Add Column Example

```
weather$sunshine <- !weather$rained # using negated values of rained vector
weather

#   days temps rained sunshine
# 1 Mon  22.2  TRUE  FALSE
# 2 Tue  21.0 FALSE  TRUE
# 3 Wed  23.0  TRUE  FALSE
# 4 Thu  24.3  TRUE  FALSE
# 5 Fri  25.0 FALSE  TRUE
# 6 Sat  21.8  TRUE  FALSE
# 7 Sun  26.1 FALSE  TRUE
```

DING

Data Frame Usage

Data Frame – Rename Column

- Use the 'colnames' function together with an indexer and an assignment

Data Frame Rename Column Example

```
colnames(weather)[4] <- 'notRained'  
head(weather)  
  
#   days temps rained notRained  
# 1 Mon  22.2  TRUE  FALSE  
# 2 Tue  21.0 FALSE  TRUE  
# 3 Wed  23.0  TRUE  FALSE  
# 4 Thu  24.3  TRUE  FALSE  
# 5 Fri  25.0 FALSE  TRUE  
# 6 Sat  21.8  TRUE  FALSE
```

DING

Data Frame Usage

Data Frame – Selecting specific rows

- With the slicing operator together with the indexer a certain set of rows can be selected
- Alternatively pass a number of rows to the `head` function

Data Frame – select specific rows Example

```
weather[2:4]  
  
# Output  
#   days temps rained  
# 2 Tue 21.0 FALSE  
# 3 Wed 23.0 TRUE  
# 4 Thu 24.3 TRUE
```

Data Frame – Selecting specific rows

- A (complex) condition can also be used to filter for rows

Data Frame – select filtered rows Example

```
weather[(weather$temp >= 23) & (!weather$rained),] # mind the comma and the  
braces!
```

```
# Output  
# days temps rained  
# 5 Fri 25.0 FALSE  
# 7 Sun 26.1 FALSE
```

Data Frame Usage

Data Frame – Selecting specific columns

- Either the slicing operator or a list of column names can be supplied to constrain the columns in the result
- The same can be achieved with the `subset` function

Data Frame – filtered columns Example

```
weather[weather$temp < 23,c('days','rained')]  
subset(weather, temp < 23, c('days','rained')) # yields same result (rows & cols!)  
  
#   days rained  
# 1 Mon FALSE  
# 2 Tue FALSE  
# 6 Sat FALSE
```

Data Frame – Handling missing Data

- The `is.na` function determines if there is missing data in a Data Frame
 - It will return a result in which each cell contains either `TRUE` or `FALSE`
 - `FALSE` is good in this case as it means no missing data!
 - Combine that with the `any` function to quickly check for missing data

Data Frame – Check for missing data Example

```
any(is.na(weather))  
# [1] FALSE
```

Data Frame – Handling missing Data

- To fix missing data combine the `is.na` function with
 - A filter for a specific column
 - An assignment of a default value
 - As default value you can either
 - Use any literal (e.g. 0)
 - Or base the default on the other (existing) values
 - ⇒ whatever makes more sense

Data Frame – Fix missing data Example

```
weather$temp[is.na(weather$temp)] <- mean(weather$temp)
```

Data Frame Usage

Data Frame – merge

- The `merge` function can be used to perform *joins*
- The `all` parameter defines the join type

Data Frame – Join Example

```
customers <- data.frame(customerId = c(1,2,3), custName = c('Maier', 'Huber', 'Hauer'))  
orders <- data.frame(orderId=c(1,2,3), customerId=c(1,2,4))  
# Full Outer Join  
merge(customers, orders, by="customerId", all=T)  
# Left Outer Join  
merge(x=customers, y=orders, by="customerId", all.x = T)  
# Right Outer Join  
merge(customers, orders, by="customerId", all.y = T)  
# Cross Join  
merge(customers, orders, by=NULL)
```

DING

CSV Import

- Importing (and exporting) from (to) CSV files is very common when working with R
 - Use the `read.csv` function and pass the full file path
 - If the file is in your working directory the name is enough
 - Assign the return value (data frame!) to a variable

CSV Import Example

```
ex <- read.csv('example.csv')
```

CSV Export

- Use the `write.csv` function
- Supply a data frame and a file name
 - If you supply a matrix it will try to convert it to a data frame

CSV Export Example

```
write.csv(mtcars, file='example.csv')
```

Excel Files

- Since the business world still runs on Excel, R supports those files as well
 - But extra packages are required
 - Preparations:
 - Install the packages 'readxl' & 'xlsx'
`(install.packages(c('readxl','xlsx')))`
 - Include the packages:
 - `library(readxl)`
 - `library(xlsx)`
 - Hint: you just learnt how to install packages from console

Excel Import

- Use the `excel_sheets` function to show the sheets in the file
 - Pass in a file name (or path if not in working directory)
 - Then use the `read_excel` function
 - Pass the file name (or path) and the sheet name
 - Assign return value (data frame) to a variable

Excel Import Example

```
excel_sheets('Sample-Sales-Data.xlsx')  
# [1] "Sheet1"  
df <- read_excel('Sample-Sales-Data.xlsx', sheet='Sheet1')
```

Excel Import

- Of course we can only import data (no diagrams or macros)
- To import an entire workbook (instead of single sheets) use the `lapply` function
- The result is a *list* of data frames

Excel Workbook Import Example

```
entireWorkbook <- lapply(excel_sheets('Sample-Sales-Data.xlsx'), read_excel,  
path='Sample-Sales-Data.xlsx')
```

Excel Export

- Use the `write.xlsx` function
- Supply a data frame and a file name (or path)
- Optionally a sheet name can be specified

Excel Export Example

```
write.xlsx(mtcars,'cars.xlsx',sheetName='MySheet')
```

Connecting to a Database

- It is possible to connect to a database from a R program
 - Then SQL queries can be issued and the results processed further
 - The exact details differ for each database
 - The following presents a general approach (using 'RODBC')
 - There is a specialized package (like 'ROracle') for most databases
 - Google is your friend!
 - It is also possible to write to a database (e.g. dbWriteTable) but we'll only consume and process data from our R programs

R + Postgres

- As an example we will connect to a postgres database:
 - 1 Install and include the 'RPostgreSQL' package
 - 2 Get the driver with the dbDriver function
 - 3 Create a connection with the dbConnect function
 - 4 Test the connection, e.g. check for table with dbExistsTable

Postgres Connection Example

```
driver <- dbDriver('PostgreSQL')
conn <- dbConnect(driver, dbname='<egpostgres>', host='<IPorDomain>', port
    =5432, user='<egpostgres>', password='<strongpw>')
# You can optionally include options="-c search_path=myschema" in the
# dbConnect call to set a specific schema
dbExistsTable(conn, 'products') # based on Northwind DB
# [1] TRUE
```

R + Postgres

- With an established connection we can send a query:
 - 1 Use the dbGetQuery function
 - 2 Pass it the connection and a SQL query
 - 3 Store the result (data frame) in a variable

Postgres Query Example

```
queryRes <- dbGetQuery(conn, 'SELECT "ProductName", "UnitPrice", "  
    UnitsInStock" FROM products WHERE "UnitPrice" > 20;')  
head(queryRes,2)  
#       ProductName      UnitPrice  UnitsInStock  
# 1 Chef Anton's Cajun Seasoning   22.00      53  
# 2 Chef Anton's Gumbo Mix     21.35       0
```

Scraping Web Data

- Sometimes you might need to get data from websites
- To do this you can use the 'rvest' package
 - You might want to try `demo(package='rvest')`
- As an example we getting data for a movie from IMDB

Web Scrape Example

```
legoMovie <- read_html('http://www.imdb.com/title/tt1490017') # this gives us  
# raw html (split into head and body)  
legoMovie %>% html_node("strong span") %>% html_text() %>% as.numeric() #  
# we retrieve the rating – !!notice the pipe operator!!  
# [1] 7.8
```

Built-in Functions

- Let's look at a couple more built-in functions:
 - `seq`: creates a sequence
 - `sort`: sorts a vector
 - `rev`: reverses order of elements
 - `append`: merges lists and vectors
 - `is.<datatype>`: checks if an object is of a specific type
 - e.g. `is.data.frame`
 - `as.<datatype>`: attempts to cast object to specific type
 - e.g. `as.data.frame`

Built-in Functions

Built-in Functions Examples

```
seq(0,10,by=2) # start, end, stepwidth
# [1] 0 2 4 6 8 10
v1 <- c(1,4,7,2,11,3)
sort(v1, decreasing = TRUE) # decreasing default FALSE => asc
# [1] 11 7 4 3 2 1
rev(1:10) # pass vector
# [1] 10 9 8 7 6 5 4 3 2 1
v1 <- 1:3
v2 <- 4:6
append(v1,v2) # pass the two vectors or lists
# [1] 1 2 3 4 5 6
```

Sample Function

- The function `sample` takes the given number of elements randomly from the given set

Sample Function Example

```
print(sample(1:10,2))
# [1] 1 7
print(sample(1:10,2))
# [1] 9 6
```

Lapply Function

- The function `lapply` allows to apply a certain function to each element of a vector
- It returns the results as a *list*

Lapply Function Example

```
v1 <- c(1,2,3,4,5)
addRand <- function(x) {
  rand <- sample(1:100,1)
  return(x+rand)
}
lapply(v1, addRand)
# [[1]]
# [1] 79

# [[2]]
# [1] 36
# ...
```

DING

Sapply Function

- The function `sapply` works similar to `lapply`, but
- It returns the results as a *vector*

Apply Function Example

```
v1 <- c(1,2,3,4,5)
addRand <- function(x) {
  rand <- sample(1:100,1)
  return(x+rand)
}
sapply(v1, addRand)
# [1] 50 94  5 99  9
```

Apply with multiple parameters

- Functions you want to *apply* may have more than one argument
 - You have to supply those additional arguments
 - Either a constant or
 - Another vector \Rightarrow results in a matrix

Apply multiple parameters Example

```
v1 <- 1:3
simpleAdd <- function(x,y){return(x+y)}
print(sapply(v1,simpleAdd,y=2))
# [1] 3 4 5
print(sapply(v1,simpleAdd,v1))
#      [,1] [,2] [,3]
# [1,]    2    3    4
# [2,]    3    4    5
# [3,]    4    5    6
```

Anonymous Functions

- Instead of defining a named (global function) you'll often want something more lightweight
- In other languages this construct is called a *lambda*

Anonymous Function Example

```
v1 <- seq(0,30,by=3)
sapply(v1,function(n){n*n})
# [1]  0  9 36 81 144 225 324 441 576 729 900
```

Math Functions

- Two additional, interesting math functions:
 - **abs**: returns the absolute value
 - **round**: rounds value to specified number of digits

Math Functions Example

```
n1 <- -4.12345
abs(n1)
# [1] 4.12345
round(n1,2)
# [1] -4.12
round(abs(n1),2) # two places after comma
# [1] 4.12
```

Regex

- Two main functions:
 - grep: will return index (where found)
 - grepl: will return logical value (if found)

Regex Example

```
text <- '<style type="text/css">code{white-space: pre;}</style>'  
grepl('^.*>(.+)<.*', text) # first pattern, then text  
# [1] TRUE  
grepl('b',c('a','b','c')) # works with vectors  
# [1] FALSE TRUE FALSE  
grep('b',c('a','b','c','b')) # 'grep' to get the indices  
# [1] 2 4
```

Dates

- The function `Sys.Date` returns the current date
- Format is '`yyyy-mm-dd`' ⇒ ISO, good and sortable
- The object is of the *Date* type (not a string)
- To create a data object use `as.Date`
 - Either pass a properly formatted string or
 - Pass an additional format string
 - e.g. `as.Date('Nov-03-90', format='%b-%d-%y')`
 - Look here for format codes:
<https://www.r-bloggers.com/date-formats-in-r/>¹

¹last accessed on 2020-03-09

Timestamps

- R uses *POSIX* timestamps (aka Epoch)
- Use `as.POSIXct` to convert a string to a time object
 - e.g. `as.POSIXct('14:09:22',format='%H:%M:%S')`
 - The data type is '`POSIXct`' and '`POSIXt`'
- This will also add date information (default: current date)
- So it is more like a `DateTime` object than a `Timestamp`
- Often simpler to use: `strptime`
 - e.g. `strptime('14:09:22',format='%H:%M:%S')`
 - Use `help(strptime)` for the format codes

dplyr

oooooooooooo

tidyr

oooo

prophet

ooo

ggplot2

oooooooooooooooooooooooooooo

plotly

ooo

R Data Processing

Markus Haslinger

DBI

Agenda

1 dplyr

2 tidyr

3 prophet

4 ggplot2

5 plotly

What is dplyr?

- dplyr provides functions for data manipulation
- It is an additional package, install with
`install.packages('dplyr')`
- Include as usual with `library(dplyr)`
- Function calls may have to be prefixed with `dplyr::!`
- The following examples use the `nycflights13` package¹

¹Containing flight information from the New York City airport.

dplyr – filter

- The filter function can be used to filter data
 - The syntax is a little easier than using indexers
 - Conditions are chained with 'and' – if you need 'or' use an indexer

Filter Example

```
head(dplyr::filter(flights, month==11, day==3, carrier=='AA'))
```

dplyr – slice

- The slice function can be used to select specific rows
 - There is no real benefit compared to using an indexer

Slice Example

```
dplyr::slice(flights, 9:10)
```

dplyr – arrange

- The `arrange` function can be used to sort a data frame
 - The sorting order is applied like with SQL 'order by', one column after the other, later ones only used when previous one is equal
 - The `desc` function can be used in addition to get descending order

Arrange Example

```
dplyr::arrange(flights,year,month,desc(day))
```

dplyr – rename

- The `rename` function can be used to rename columns
- Be careful: the old and new name are put in the reverse order (than expected)
- This also does not affect the data frame, a new one is returned

Rename Example

```
dplyr::rename(flights,dayOfMonth=day) # renames 'day' to 'dayOfMonth'
```

dplyr – select

- The `select` function can be used to select columns
- It does exactly what one would assume it does
- It can be used together with the `distinct` function

Select Example

```
dplyr::select(flights,carrier,air_time)  
dplyr::distinct(dplyr::select(flights,carrier))
```

dplyr – mutate

- The `mutate` function can be used to add a new column based on other column(s)
- This is also possible with indexers
- Does not affect the data frame, a new one is returned

Mutate Example

```
dplyr::mutate(flights, newCol = arr_delay - dep_delay)
```

dplyr – summarise

- The `summarise` function can be used to reason over groups
- Similar to SQL 'group by'
- Requires a data frame and a 'calculation' function as input
 - Can break if not all rows have a value
 - Some functions have a 'na.rm' for 'remove if NA' switch

Summarise Example

```
dplyr::summarise(flights,avgAirTime=mean(air_time,na.rm=T))
```

dplyr – sampleX

- The `sample_n` function can be used to retrieve n random rows
- The `sample_frac` function can be used to retrieve n% random rows

SampleX Example

```
dplyr::sample_n(flights, 10)  
dplyr::sample_frac(flights, 0.001)
```

Pipe Operator & dplyr

- Some function call chains using `dplyr` can get long and hard to read
- Using the pipe operator (`%>%`) can (slightly) help to improve readability

Pipe Operator Example

```
arrange(sample_n(filter(flights,air_time<200),5),desc(air_time)) # equivalent  
flights %>% filter(air_time<200) %>% sample_n(5) %>% arrange(desc(air_time))
```

tidyR

- Used to clean up data (basically helping us reach 'NF for R')
 - Each row represents one observation
 - Every column is one variable
- Install using `install.packages('tidyR')`
- Include using `library(tidyR)` & `library(data.table)`
 - data tables are a more performant version of data frames

Sample Data Frame

```
stocks <- data.frame(time=as.Date('2020-03-16') + 0:9, stock1 = rnorm(10,0,1),  
                      stock2 = rnorm(10,0,1), stock3 = rnorm(10,0,1))
```

tidy় – gather

- The `gather` function takes multiple columns and rearranges them into key-value pairs

gather Example

```
gathered <- stocks %>% tidyr::gather(stock, priceChange, stock1:stock3)
# Output
#      time   stock priceChange
# 1 2020-03-16 stock1  1.04773288
# 2 2020-03-17 stock1 -0.12213104
# 3 2020-03-18 stock1  0.01958765
# 4 2020-03-19 stock1  0.19523999
# ...
# 11 2020-03-16 stock2  0.26203320
# 12 2020-03-17 stock2 -0.50349851
# ...
```

tidyr – spread

- The spread function takes key-value pairs and splits them into multiple columns

spread Example

```
gathered %>% tidyr::spread(stock, priceChange)
# Output
#       time      stock1      stock2      stock3
# 1 2020-03-16  1.04773288  0.2620332  0.09230775
# 2 2020-03-17 -0.12213104 -0.5034985  0.92818651
# 3 2020-03-18  0.01958765  0.9977473 -0.59905604
# 4 2020-03-19  0.19523999  1.0124920 -0.80294775
# 5 2020-03-20  0.41304733 -0.9475768  1.43448327
# 6 2020-03-21 -0.05218093  0.9645940 -2.08093934
# 7 2020-03-22 -2.00180246  0.8967869 -0.22564568
# 8 2020-03-23  0.80132968  0.9692264 -1.78054525
# 9 2020-03-24  0.50877842  2.2511848 -0.09526893
# 10 2020-03-25  0.69527242 -1.0196440  1.56249558
```

tidy় – separate

- The `separate` function attempts to split non atomic values in one column into two
- Similar to a string split, trying to achieve something like NF₁
- The reverse is the `unite` function

separate Example

```
df <- data.frame(coords=c("12x4","7x1","8x19","2x6"))
tidy়::separate(df,col=coords,into=c('x','y'),sep='x') # sep is regex
#   x   y
# 1 12  4
# 2  7  1
# 3  8 19
# 4  2  6
```

prophet

- The prophet package is maintained by Facebook²
- It is used to predict trends based on existing data
 - How will a situation probably develop based on data from the past
- Install using `install.packages('prophet')`
- prophet works mostly automatic
 - So it is easy to use
 - But it also only allows for very specific input:
 - The base data frame only has two columns
 - One called 'ds' contains a date or timestamp (YYYY-MM-DD [HH:MM:SS])
 - One called 'y' containing the value

²see <https://cran.r-project.org/web/packages/prophet/index.html>, 2020-03-19

prophet

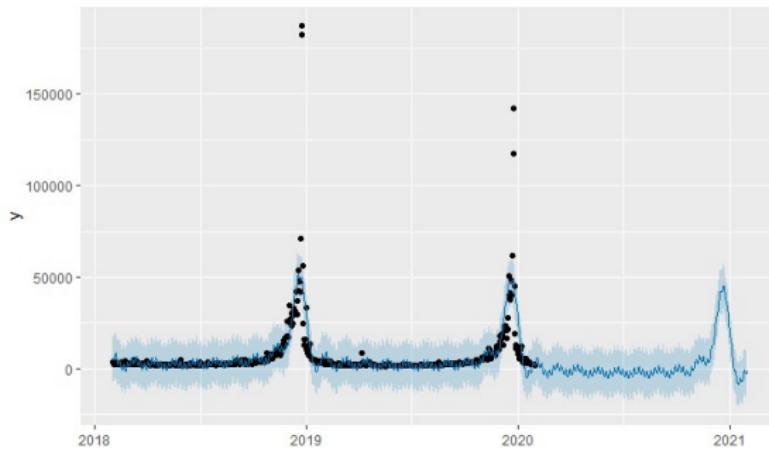
- Once you have a data frame in the required format use the prophet function to create a model
- Then use the make_future_dataframe function to prepare the data
- Create a prediction using the predict function
- (Finally plot the result using prophet_plot_components)

prophet Example

```
model <- prophet(df)
future <- make_future_dataframe(model, periods = 365) # create future data (for
            365 days) extrapolated from the model
forecast <- predict(model, future) # make a forecast
plot(model, forecast) # plot
```

prophet – Example

- A prediction for page views of the 'Christmas' Wikipedia page for the next year based on the last two years



Data Visualization

- An important part of data processing is visualization
 - ⇒ representation of data or information in a graph, chart, or other visual format
- Raw numbers are hard to grasp for (most) people
- Graphics allow trends and patterns to be more easily seen
- So to convey your message visual aids are of great help
 - After all the purpose of data analysis is to gain insights
- We already know how to process data with R, now it is time to visualize it
- **Plots can be exported as images and used also outside of the R environment (e.g. thesis,...)**

ggplot2

- We already used some 'out-of-the-box' plots
- Now we are going to explore the 'ggplot2' package which improves upon them
 - The most often used R package for data visualization
- ggplot2 is based on the 'The Grammar of Graphics³'
- Install using `install.packages('ggplot2')`
 - Additionally requires the 'data.table' package to function
 - We will use the 'ggplot2movies' data package for samples

³Wilkinson et. al, 2005, Springer

ggplot2 – Layers

- `ggplot2` layers based on the 'grammar of graphics'



ggplot2 – Foundations

- The first three layers are the foundation for all plots
 - Data
 - Aesthetics
 - Geometries
- Once geometry is added the plot is rendered

ggplot2 Foundations Example

```
pl <- ggplot(data=mtcars) # defines data (layer) – nothing shown yet  
pl <- ggplot(data=mtcars,aes(x=mpg,y=hp)) # aes in this case stands for '  
      aesthetics' (layer) defining x & y axes  
pl + geom_point() # now adds geometry (layer) – points representing data
```

dplyr

10 of 10

ggplot2

tidyverse

10

prophet

000

ggplot2

plotly

100

ggplot2 – Foundations

dplyr
oooooooooooo

tidyR
oooo

prophet
ooo

ggplot2
oooooooooooooooooooo

plotly
ooo

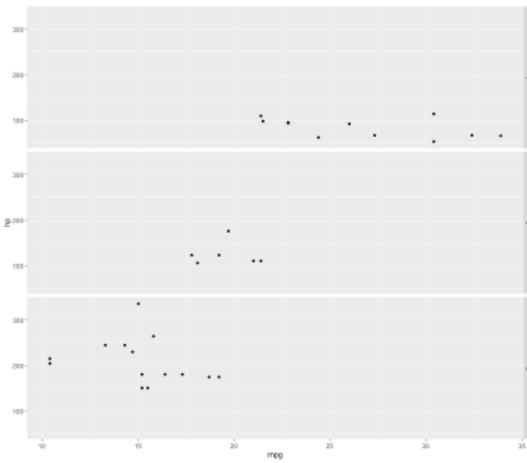
ggplot2

ggplot2 – Facets

- Facets allow for 'grouping' within the plot

ggplot2 Facet Example

```
pl + geom_point() + facet_grid(cyl ~.)  
# mind the second '+' we are  
adding one layer atop another
```



ggplot2 – Statistics

- We can add an additional 'statistics' layer
 - E.g. stat_smooth adds a 'trend' line and an 'error' area

ggplot2 Statistic Example

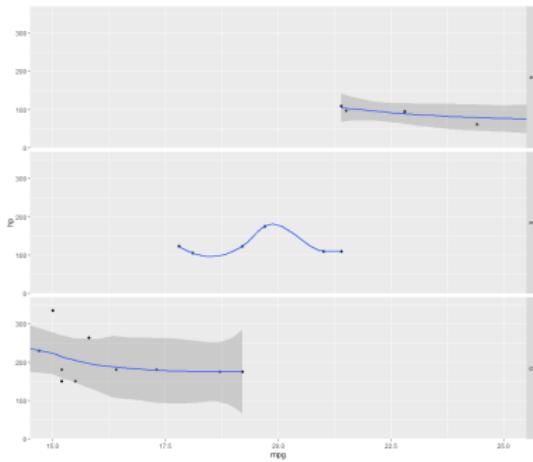
pl
+ geom_point()
+ facet_grid(cyl ~ .)
+ stat_smooth()

ggplot2 – Coordinates

- Coordinates basically allow us to constrain the axes

ggplot2 Coordinates Example

```
pl  
  + geom_point()  
  + facet_grid(cyl ~ .)  
  + stat_smooth()  
  + coord_cartesian(xlim=c(15,25)) #  
    limiting x axis to the range 15–25
```

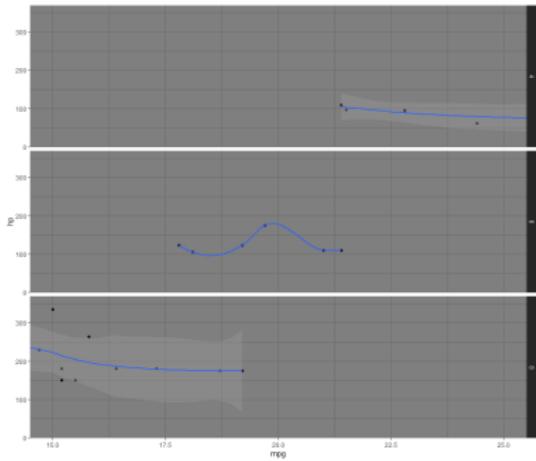


ggplot2 – Theme

- The theme defines the overall look of the plot

ggplot2 Theme Example

```
pl  
+ geom_point()  
+ facet_grid(cyl ~.)  
+ stat_smooth()  
+ coord_cartesian(xlim=c(15,25))  
+ theme_dark()
```



ggplot2 – Histogram

- A Histogram can be used to display distribution of one value over a sample

ggplot2 Histogram Example

```
df <- movies[sample(nrow(movies), 1000),] # selecting 100 random movies –  
# optional!  
pl <- ggplot(df,aes(x=rating)) # define foundations: data and aesthetics  
pl <- pl + geom_histogram(binwidth=0.1,color='blue',fill='lightblue') # plot  
# histogram  
pl + xlab('Movie star rating') + ylab('Movie Count') + ggtitle('Movie Ratings') #  
# add (optional) axis labels and title
```

dplyr
oooooooooooo

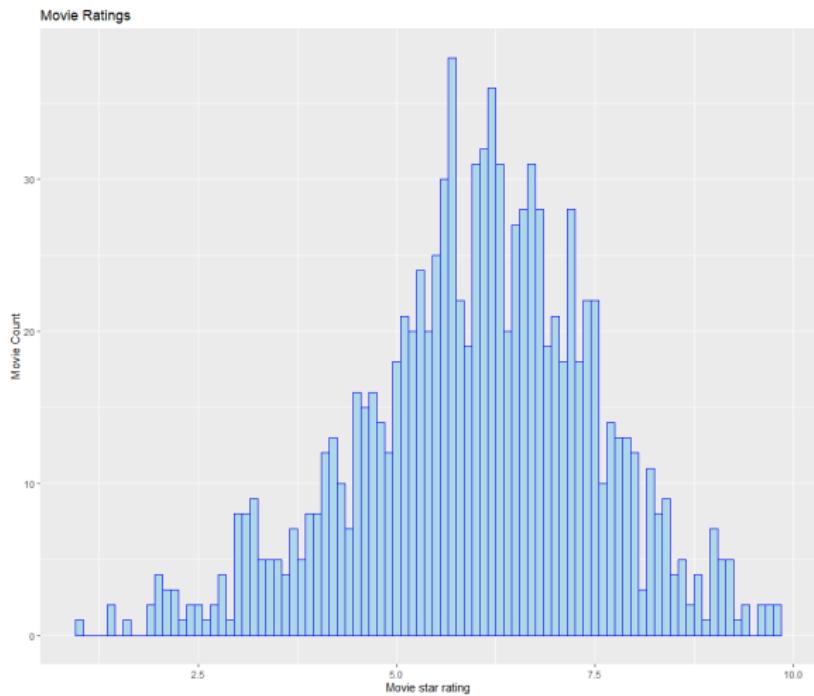
tidyR
oooo

prophet
ooo

ggplot2

plotly
ooo

ggplot2 – Histogram – Sample Plot



ggplot2 – Scatterplot

- A Scatterplot visualizes data points in two axes thus allowing to show correlations
- You can also utilize size or point shape – reference the cheat sheet for options

ggplot2 Scatterplot Example

```
pl <- ggplot(df, aes(x=year,y=length)) # display year and movie length  
pl <- pl + geom_point(aes(color=rating)) # (optional) use color for rating  
pl + scale_color_gradient(low='red',high='green') # (optional) define color gradient
```

dplyr
○○○○○○○○○○

tidyr
○○○○

prophet
○○○

ggplot2
○○○○○○○○○○○○○○●○○○○○○○○○○○○

plotly
○○○

ggplot2

ggplot2 – Scatterplot – Sample Plot



ggplot2 – Barplot

- While a histogram requires continuous data for the x axis a barplot is used for categorial data

ggplot2 Barplot Example

```
pl <- ggplot(mpg, aes(x=class))
pl + geom_bar(aes(fill=drv)) # (optional) aes def.: stack different drive train counts
```

dplyr
○○○○○○○○○○

tidyR
○○○○

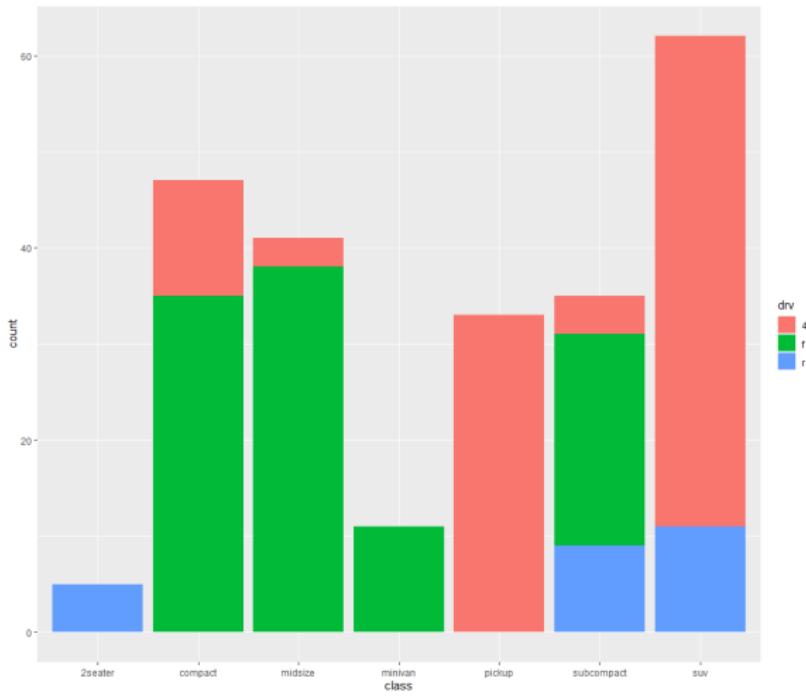
prophet
○○○

ggplot2
○○○○○○○○○○○○○○●○○○○○○○○

plotly
○○○

ggplot2

ggplot2 – Barplot – Sample Plot



ggplot2 – Boxplot

- Boxplots are a powerful tool to show and compare value distribution for different categories
- But they are also a little harder to understand than the 'basic' plot types

ggplot2 Boxplot Example

```
pl <- ggplot(mtcars,aes(factor(cyl),mpg))
pl + geom_boxplot(aes(fill=factor(cyl))) # remember the 'factor' function
pl <- pl + geom_boxplot(aes(fill=factor(cyl))) # creating boxplot
pl + coord_flip() # (optional) flipping by 90 degrees for better display
```

dplyr
○○○○○○○○○○

tidyr
○○○○

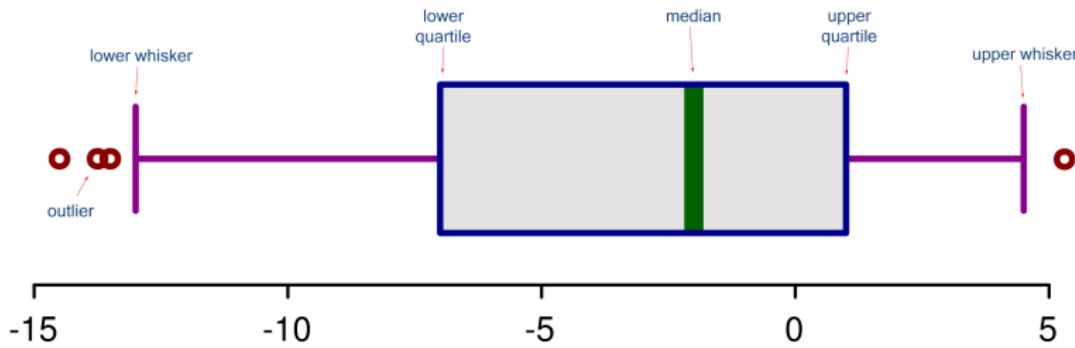
prophet
○○○

ggplot2
○○○○○○○○○○○○○○○○●○○○○○○

plotly
○○○

ggplot2

ggplot2 – Boxplot – Structure



Source: adapted from 'RobSeb' for Wikipedia, 2011

HTL LEONDING

dplyr

○○○○○○○○○○

ggplot2

tidyr

○○○○

prophet

○○○

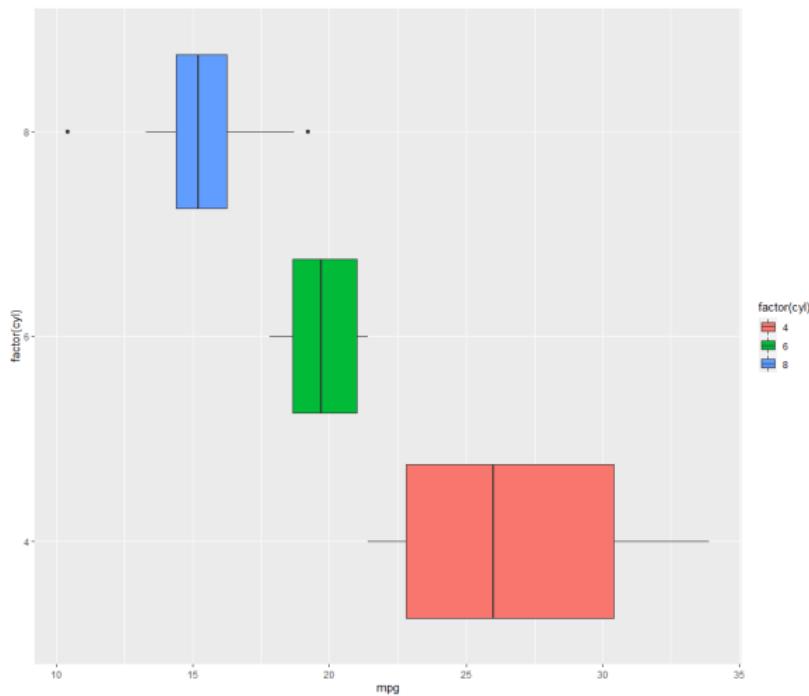
ggplot2

○○○○○○○○○○○○○○○○○○○○○○●○○○○○○

plotly

○○○

ggplot2 – Boxplot – Sample Plot



ggplot2 – Density Plot

- Based on three metrics density plots can show value distribution along two axes

ggplot2 Density Plot Example

```
ggplot(movies[sample(nrow(movies), 1000),],aes(x=year,y=rating)) + geom_bin2d()  
# simple, use lower sample size to get clearer result  
pl <- ggplot(movies, aes(x=year, y=rating))  
pl + geom_density_2d() # heatmap  
pl + stat_density_2d(aes(fill = after_stat(level)), geom = "polygon") # heatmap
```

dplyr
○○○○○○○○○○

tidyR
○○○○

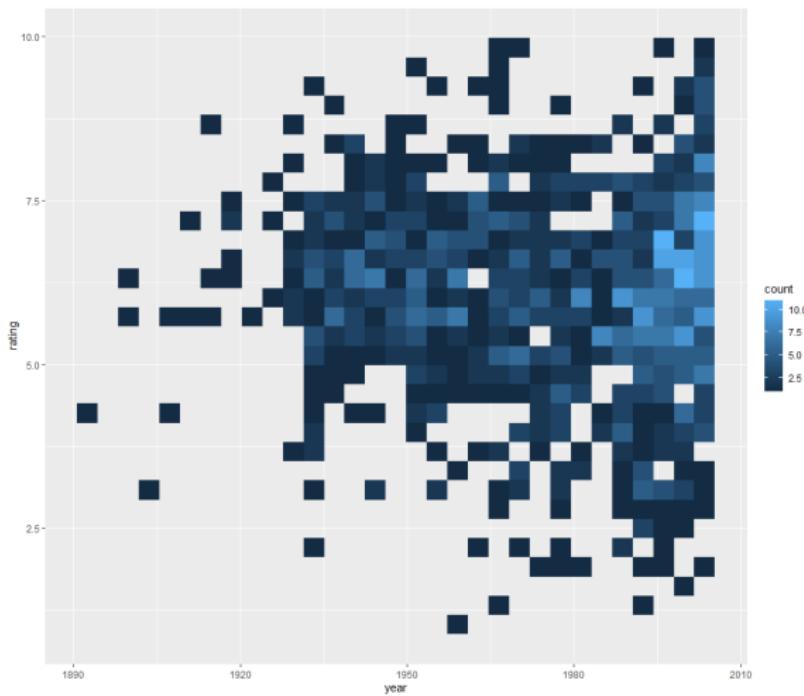
prophet
○○○

ggplot2
○○○○○○○○○○○○○○○○○○○○○○○●○○○○

plotly
○○○

ggplot2

ggplot2 – Density Plot – Simple Sample



HTL LEONDING

dplyr
○○○○○○○○○○

ggplot2

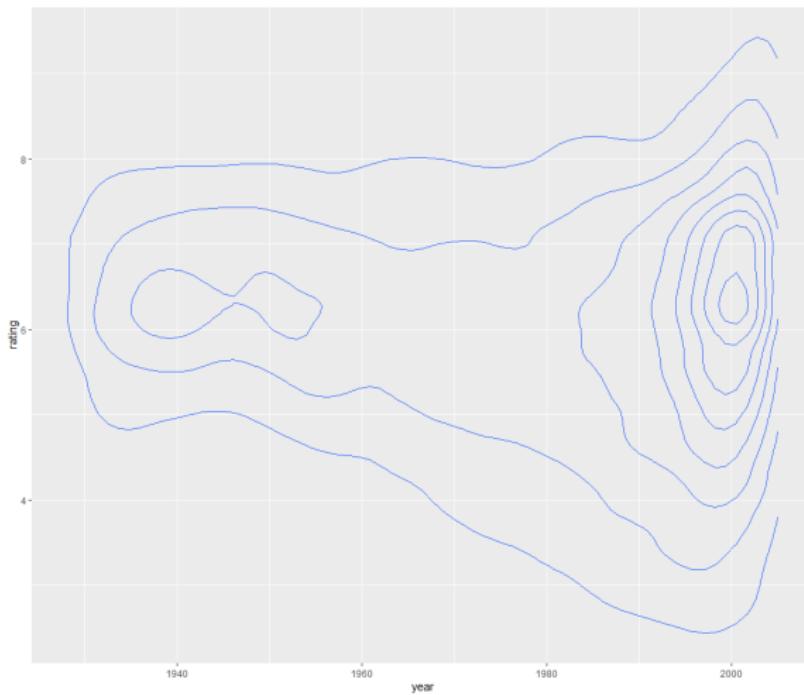
tidyr
○○○○

prophet
○○○

ggplot2
○○○○○○○○○○○○○○○○○○○○○○○○○○○○●○○○

plotly
○○○

ggplot2 – Density Plot – Heightmap Sample



dplyr
○○○○○○○○○○

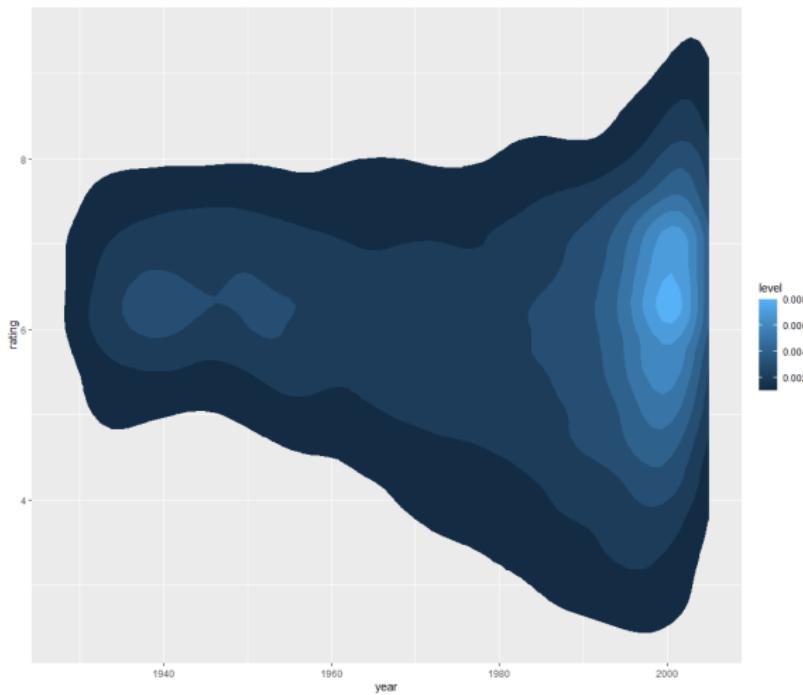
tidyr
○○○○

prophet
○○○

ggplot2
○○○○○○○○○○○○○○○○○○○○○○○○○●○○

plotly
○○○

ggplot2 – Density Plot – Heatmap Sample



ggplot2 – Coordinate Zoom

- It is possible to 'zoom in' by limiting the plot to a certain range of coordinates

ggplot2 Coordinate Zoom Example

```
pl <- ggplot(movies[sample(nrow(movies), 1000),],aes(x=year,y=rating))
pl + geom_hex() # a Civ style hex plot – use install.packages('hexbin')
pl + geom_hex() + coord_cartesian(xlim=c(1975,2010),ylim=c(5.0,7.5)) # 'zoom'
into a year and rating range
```

dplyr
oooooooooooo

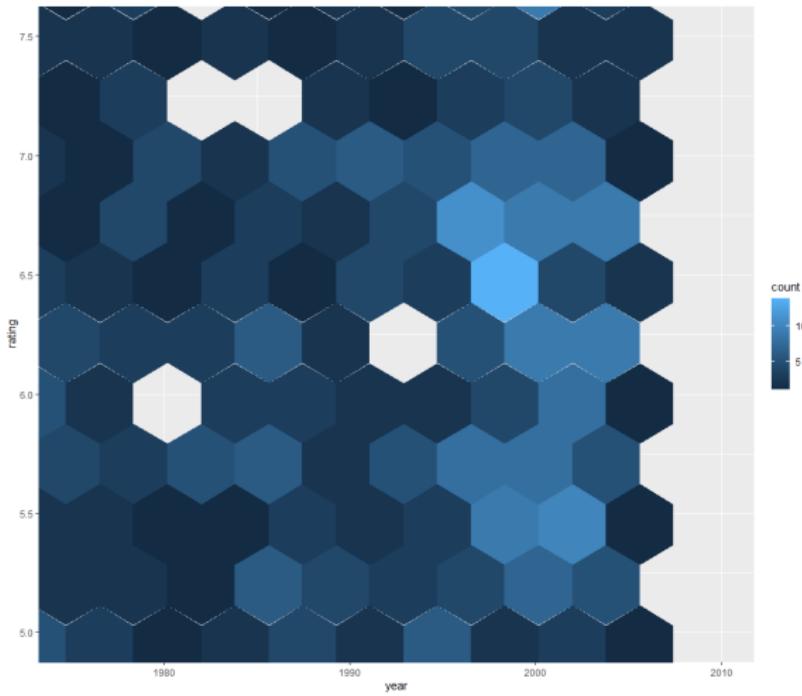
tidyverse
0.0.0

prophet
ooo

ggplot2

plotly
ooo

ggplot2 – Coordinate Zoom – Sample Plot



plotly

- The `plotly` package allows to create interactive diagrams
- For example hovering with the cursor at a data point shows a tooltip with its value
- These features obviously do not translate to exported pictures
⇒ use within R only
 - But you *can* export as HTML file!
- For details visit <https://plotly.com/>

plotly – Example

- In general just apply the `ggplotly` function to an existing `ggplot` plot

ggplot2 Coordinate Zoom Example

```
pl <- ggplot(mtcars, aes(mpg,wt)) + geom_point() # create a plot and save it into  
# a variable  
ggplotly(pl) # apply ggplotly
```

dplyr
oooooooooooo

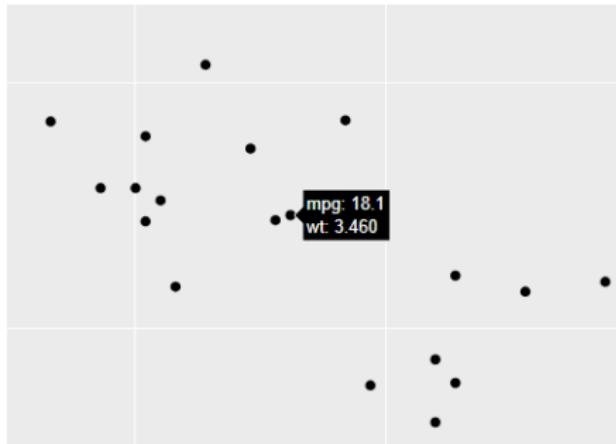
tidyR
ooooo

prophet
ooo

ggplot2

plotly
oo•

plotly – Example



Machine Learning with R

Markus Haslinger

DBI

Agenda

1 Introduction

- ML Introduction
- Statistical Learning

2 Regression

- Linear Regression
- Logistic Regression

3 KNN

Scope

- We will roughly define machine learning
- We will look at a few different approaches using the R programming language
 - Commonly used for this purpose
 - Another very popular option would be Python
- We will work through little exercises
- We will *not* go into details¹

¹Extracurricular classes exist for those with a deeper interest.

Machine Learning

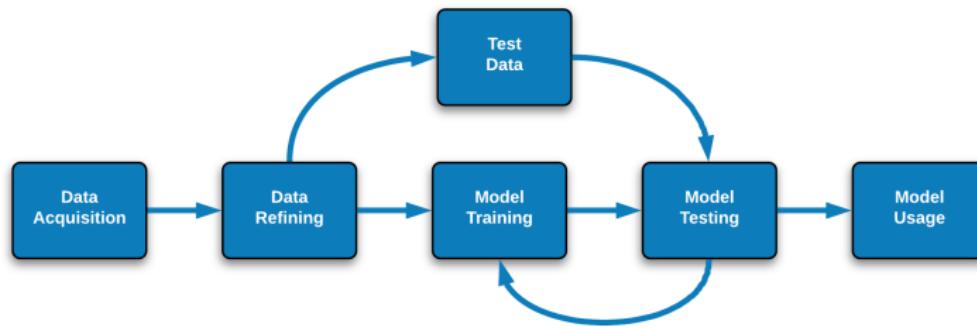
- A method for data analysis
- Automates the creation of analytical models
- Uses:
 - Fraud detection
 - Realtime advertising
 - Image recognition
 - Trend prediction
 - Financial models
 - Recommendation systems
 - ...

Machine Learning

Definition

By means of algorithms, which are adapting in an iterative fashion from data, machine learning enables a computer to draw conclusions without previous knowledge of the domain or metrics to look for.

ML Cycle



ML Algorithm Types – Supervised Learning

- The goal is to assign data to certain classes or groups defined by the user
- Two types:
 - Classification: output is a category, e.g. 'red' or 'blue'
 - Regression: output is a value, e.g. 'weight'
- Example: mail spam filter

ML Algorithm Types – Unsupervised Learning

- Knowledge about the groups of data is not initially known but the desired outcome
- Two types:
 - Clustering: goal is to discover the inherent groupings
 - Association: goal is to find relation rules
- Example: customers who buy product A are likely to buy product B as well

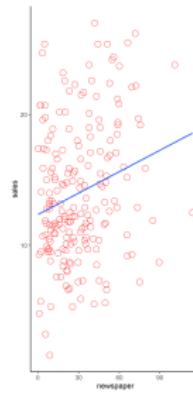
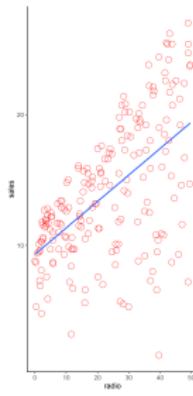
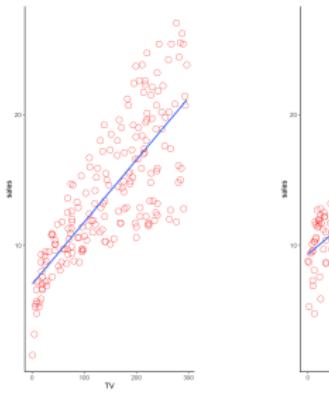
ML Algorithm Types – Reinforcement Learning

- A kind of conditioning with the system being rewarded for doing good and penalized for doing wrong²
- Based on trial and error
- The feedback might not be instantaneous but delayed
- The goal is to find a model which maximizes the reward for the agent
 - ⇒ which is usually what we then want to use ourselves
- Example: automatically manage an investment portfolio

²Kind of what we are doing to students at school :)

Sample Data

- We will use example³ data from an advertising campaign
- We have information about spending for TV, radio and the newspaper and related sales figures
- The goal is to try and find a correlation so that sales can be maximized



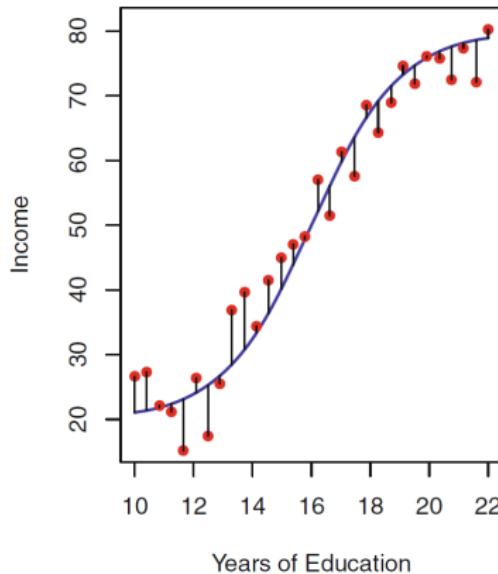
³cf. James et al., 'An Introduction to Statistical Learning, with applications in R', Springer 2013

Sample Data

- The spending is the input $X(X_1, X_2, X_3)$
- The sales are output Y
- We assume a relation between X & $Y \Rightarrow Y = f(X) + \epsilon$
 - ϵ is the 'error term' which is not influenced by observations X
- The function f is unknown so we use an approximation based on the given data points

Error Term

- The vertical lines show the error term – the deviation of actual values from the calculated ones



Why determine f ?

■ Prediction

- The error term should become 0 on average, so:
- $\hat{Y} = \hat{f}(x)$ (\hat{f} ... predicted function, \hat{Y} ... predicted value)

■ Inference

- We want to know how Y is influenced by X_1, \dots, X_n :
- Which values X actually have an influence on Y ?
- Which relation is there between X & Y (pos. / neg.)?
- Can the relation be put in simple terms?

How to determine f – parametric

- Two steps:

- 1 Assumption of the functional representation of f
 - e.g. a linear function $f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$
 - 2 Finding a matching model by comparison with training data
 - $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$

- To do this a common approach is the *least squares* method⁴

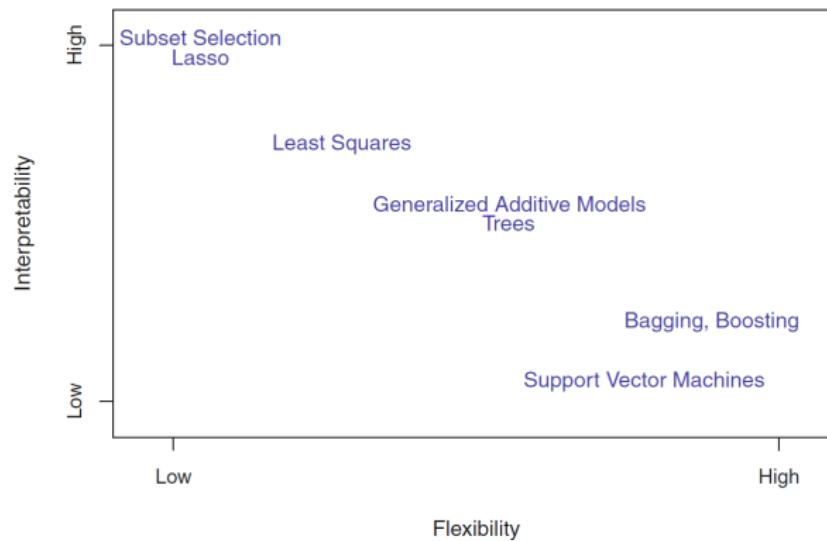
⁴see https://mathepedia.de/Methode_der_kleinsten_Quadrate.html, 2020-04-09

How to determine f – non-parametric

- A second approach is to not make assumptions about the functional representation
 - So we do not constrain it to a linear or exponential or ... function
- Instead any kind of function can be used [in each iteration]
- pro:
 - no constraints for the function
 - often enables a better fit
- con:
 - due to the missing limit on tweakable parameters more observations are required
 - can easily lead to overfitting

Flexibility vs. Interpretability

- As the flexibility of a method increases its interpretability decreases



Model Quality

- It is important to determine the quality of the model
- A low quality can be an indication that the wrong method was used or that the training data does not allow for a solid prediction
- To check the quality the model is applied to the test data and the result compared with the actual one
- In regression models most commonly used is the *Mean Squared Error* (MSE):
 - $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$
- For classification models it looks like this:
 - $\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$
 - $I(y_i \neq \hat{y}_i)$ is an *indicator variable*
 - it equals 1 if $y_i \neq \hat{y}_i$ and 0 if $y_i = \hat{y}_i$

Linear Regression

- The regression line should be as close as possible to all data points
 - Assume $y = a + bx$
 - $b = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$
 - $a = \bar{y} - b\bar{x}$
 - Where \bar{x} is the average value of x
- For the least squares method we are only interested in the vertical error (pos. or neg.)
 - The *residual sum of squares (RSS)*: $RSS = e_1^2 + e_2^2 + \dots + e_n^2$

Linear Regression – Model Accuracy

- To determine the accuracy of the model you can use:
- Residual Standard Error (RSE)
 - Represents the standard deviation of $e \Rightarrow$ the average difference between predicted and actual value
 - $RSE = \sqrt{\frac{1}{n-2} RSS} = \sqrt{\frac{1}{n-2} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$
- R^2 Statistic
 - A proportional value between 0 (bad) and 1 (perfect)
 - $R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$
 - *total sum of squares (TSS)*
 - $TSS = \sum (y_i - (\frac{1}{n} \sum_{i=1}^n y_i))^2$

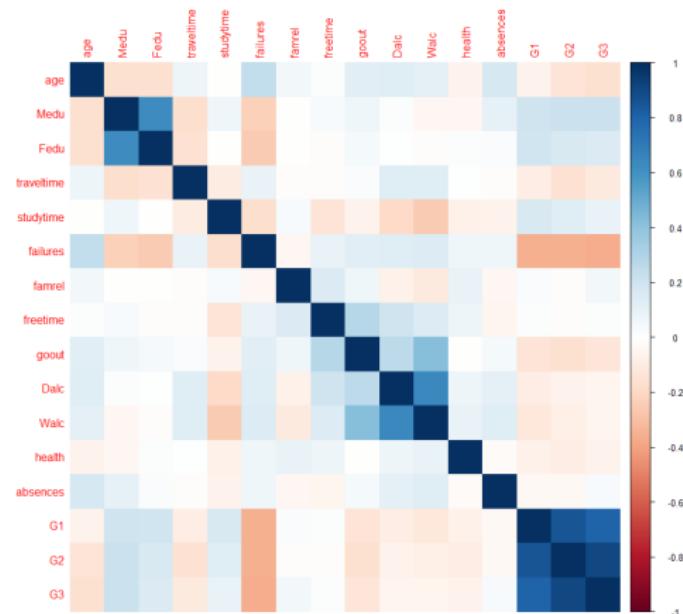
Linear Regression with R

- In a first step we'll use R to find correlations in data

Finding Correlations

```
# additional packages: library(corrgram), library(corrplot), library(viridisLite)
df <- read.csv('student-mat.csv',sep=',') # read CSV file
numCols <- sapply(df,is.numeric) # only numeric columns
correlationData <- cor(df[,numCols]) # compares all columns with each other and
# shows pos. or neg. correlation
corrplot(correlationData, method='color') # easier to find patterns
```

Linear Regression with R



Training the Model

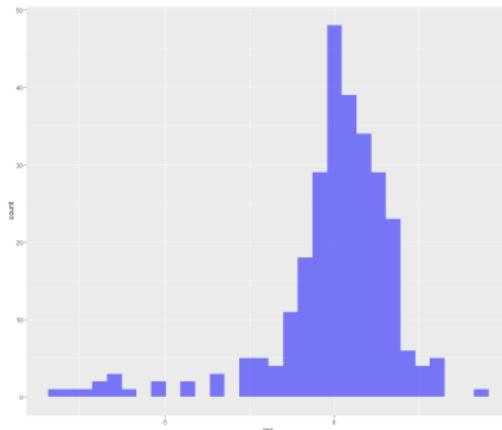
Model Creation

```
# additional package library(caTools)
sample <- sample.split(df$age, SplitRatio = 0.7)
train = subset(df, sample==T)
test = subset(df, sample==F)
model <- lm(G3 ~ ., train) # residuals will now be read
summary(model)
res <- as.data.frame(residuals(model)) # convert to data frame
res <- dplyr::rename(res, res='residuals(model)')
ggplot(res, aes(res)) + geom_histogram(fill="blue", alpha=0.5) # plot the residuals
```

Linear Regression

Residuals

- Residuals are deviations between predicted and actual values
- These should follow a normal distribution
- If that is not the case linear regression might be the wrong choice for the data set



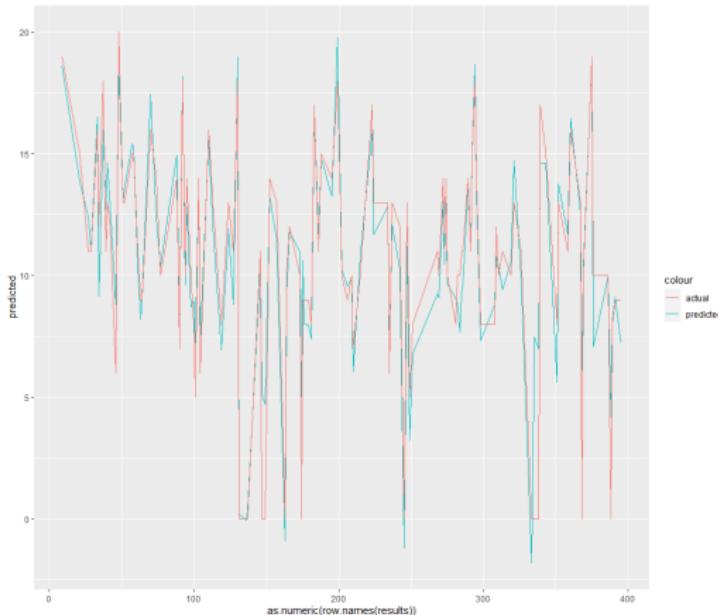
Linear Regression – Prediction

- Using the predict function you can create a prediction based on the model (for test data)

Prediction Creation

```
predictionsG3 <- predict(model, test)
results <- cbind(predictionsG3, test$G3)
colnames(results) <- c('predicted', 'actual')
results <- as.data.frame(results)
ggplot(results, aes(x = as.numeric(row.names(results)))) + geom_line(aes(y=
predicted, colour='predicted')) + geom_line(aes(y=actual, colour='actual'))
```

Linear Regression – Prediction



Multiple Predictors

- If multiple predictors are to be used for a prediction the function changes
 - $y = a + b_1x_1 + b_2x_2 + \dots + b_nx_n$
- The function for the advertising example to take into account all media would thus be
 - $\text{sales} = a + b_1 \text{TV} + b_2 \text{Radio} + b_3 \text{Newspaper} + e$

Model with multiple predictors

```
model <- lm(sales~TV+radio+newspaper, df)
summary(model)
# Coefficients:
#                 Estimate Std. Error t value Pr(>|t|)
# (Intercept) 2.938889  0.311908  9.422  <2e-16 ***
# TV          0.045765  0.001395 32.809  <2e-16 ***
# Radio       0.188530  0.008611 21.893  <2e-16 ***
# Newspaper   -0.001037  0.005871 -0.177    0.86
```

DING

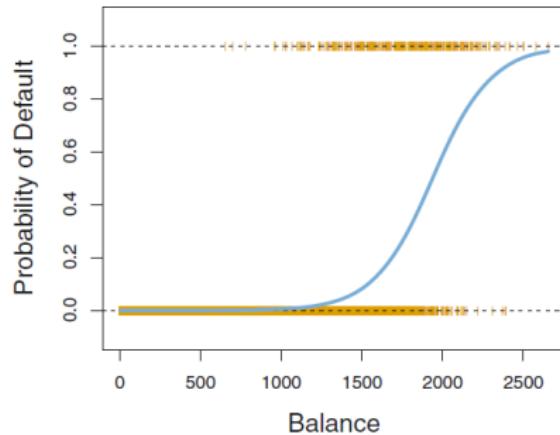
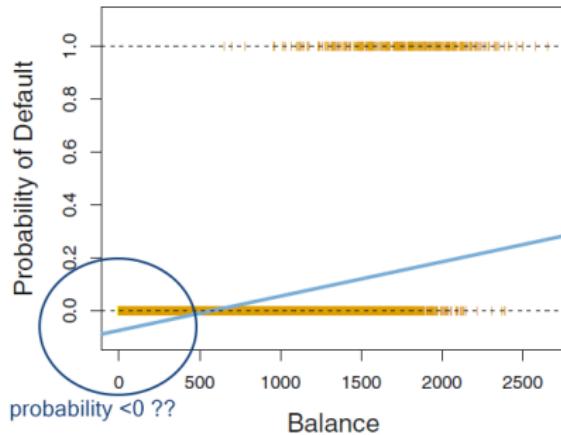
Relation between Prediction & Predictors

- The relation between the prediction and the predictor variables can be determined using the *F-test* (aka F-statistic)
- $$F = \frac{\frac{TSS - RSS}{p}}{\frac{RSS}{n-p-1}}$$
- If there is no correlation the F value approaches 1
- If there *is* a correlation the F value is much bigger than 1

Logistic Regression

- Logistic regression is used for classification problems
- Especially for a binary decision (yes/no)
 - Spam mail
 - Loan default
 - Illness diagnosis
- The goal is to determine a probability if a certain classification applies or not

Why not use Linear Regression?



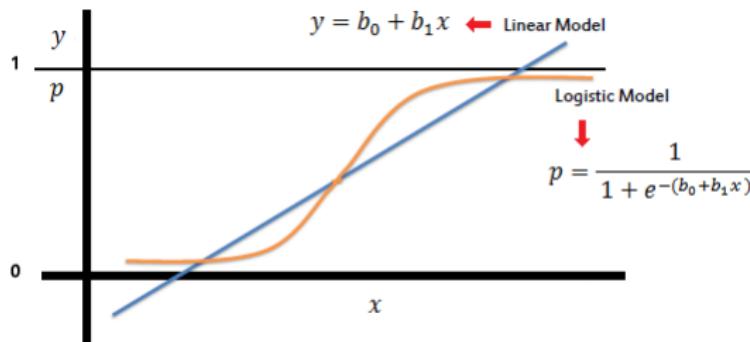
Creating a Logistic Regression Curve

- Can be created using a *sigmoid* function:

- $p = \frac{1}{1+e^{-z}}$

- For z a linear function can be used:

- $z = a + bx \Rightarrow p = \frac{1}{1+e^{-(a+bx)}}$ $(b_0 = a \ b_1 = b)$



Source: <http://juangabrielgomila.com/en/logistic-regression-derivation>,
2020-04-09

Model Quality

- Data is split into training and test data (as usual)
- After training the model is applied to the test set and the results are put into a confusion matrix

$n=165$	Prediction: F	Prediction: T
Actual: F	TN = 50	FP = 10
Actual: T	FN = 5	TP = 100

TN = True Negative

TP = True Positive

FN = False Negative

FP = False Positive

- Precision can than be determined with the following formula(s):

$$\text{■ Precision} = \frac{TP+TN}{n} = \frac{150}{165} = 0.91$$

$$\text{■ Imprecision} = \frac{FP+FN}{2} = \frac{15}{165} = 0.09$$

Logistic Regression with R

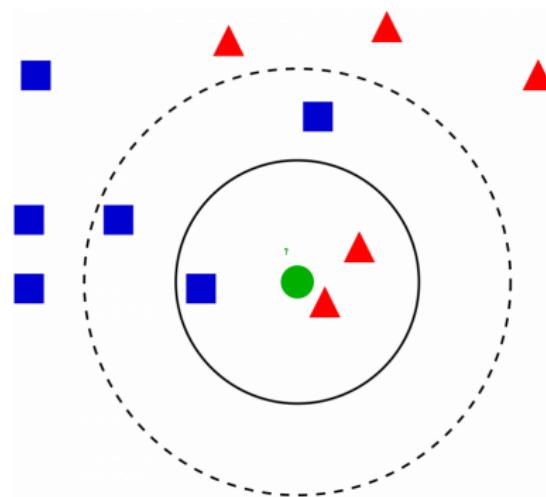
- Model creation: `glm(formula=<ResultCol> ~ ., family=binomial(link='logit'), data=<trainData>)`
- Calculating probabilities:
`predict(model, newdata=data, type='response')`
- Define threshold:
`fittedRes <- ifelse(probs > 0.5, 1, 0)`
- Classification error: `mean(fittedRes != df$Result)`
- Confusion Matrix: `table(df$Result, probs > 0.5)`

K-Nearest Neighbor

- KNN is a very simple classification method
- The training consists simply of storing all (already classified) data points
- For the prediction there are a few simple steps:
 - 1 Calculate the 'distance' (imagine a cartesian coordinates) from a new data point to all others
 - 2 Sort the existing data points by distance
 - 3 Take the k nearest
 - 4 The new data point is predicted to have the same class as the *majority* of the k nearest neighbors
- To avoid a draw:
 - Either use an odd k
 - Or simply flip a coin

K-Nearest Neighbor

- The result can depend heavily on the chosen k value



Source: <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>, 2020-04-09

KNN – Pros & Cons

■ Pro:

- Very easy to understand and implement
- Trivial training logic and time
- Works with any number of classes
- Simple to add new data
- Few Parameters (only k and the distance)

■ Con:

- High prediction cost (inefficient for large data sets)
- Not well suited for data points with many relevant dimensions
- Distinct properties of categories are difficult to include

Queries



Joins



Sets



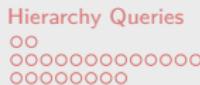
Hierarchy Queries



Structured Query Language

Markus Haslinger

DBI/INSY 3rd year



Agenda

- 1** Queries
 - Basics
 - Conditions
 - Ordering & Grouping

 - 2** Joins
 - Sample Database
 - Oracle Joins
 - Standard Joins

- 3 Sets
 - Set Operations
 - Set Examples
 - 4 Hierarchy Queries
 - Introduction
 - Oracle CONNECT BY
 - WITH clause

Queries



Joins



Sets



Hierarchy Queries



Basics

SQL

Definition

Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data.¹

- Despite the name → more than just queries
- Designed with the relational model in mind
- 'Lingua Franca' for RDBMS
 - But with variations in most implementations

¹<https://www.techopedia.com/definition/1245/structured-query-language-sql>, 2018-08-16

Queries



Basics

Joins



Sets



Hierarchy Queries



Syntax

```
SELECT [DISTINCT] expression1 [alias_1] [, expression2 [alias_2], ....]
FROM table_name1 [,table_name2, ....]
[WHERE condition]
[GROUP BY grouping_expr1 [, grouping_expr2, ....]]
[HAVING having_condition]
[ORDER BY order_expr1 [, order_expr2, ....]]
```

- Each SELECT statement has at least two clauses (SELECT & FROM)
- The order in which clauses (may) appear is fixed
- A HAVING clause can only exist together with a GROUP BY clause

Queries



Joins



Sets



Hierarchy Queries



Basics

Literals

- Numeric
 - Integer (WHERE Quantity = 3)
 - Floating point (WHERE Price > 99.9)
- Alphanumeric
 - WHERE LastName = 'Knuth'
- Dates
 - WHERE DateOfBirth > '1982-04-01'
- Usually available in every RDBMS

Queries



Joins



Sets



Hierarchy Queries



Basics

System variables

- Similar but with different syntax and availability depending on the RDBMS
- Examples:
 - USER
 - SYSDATE
 - ROWNUM
 - LEVEL
- `SELECT user, sysdate, rownum FROM dual;` [Oracle]

Queries



Joins



Sets



Hierarchy Queries



Basics

Numeric expressions

- Operators (+, -, *, /, mod)

- NULL
- Function NVL [Oracle]
 - SELECT NVL(leagueno, 'no league') FROM Players;
 - Replaces all NULL values with the provided value

Queries



Joins



Sets



Hierarchy Queries



Basics

Date expressions [Oracle]

- Date differences, e.g. `SELECT SYSDATE - PenaltyDate
FROM Penalties;`
- Date Input and Output:
 - Requires format string
 - `TO_DATE('2018-08-16', 'YYYY-MM-DD')`
 - `TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI')`
 - Format string is case sensitive
 - See https://docs.oracle.com/cd/B19306_01/server.102/b14200/sql_elements004.htm

Queries

○○○○○●○○○○
○○○○○○○○○○○○○○
○○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○○○○○○○○○○○○

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○○○
○○○○○○○○

Basics

Date expressions [Oracle]

`TO_CHAR(PENALTYDATE,'Dy,DD-MONTH-YYYY')`

Mo, 08-Dezember -1980
Di, 05-Mai -1981
Sa, 10-September-1983
Sa, 08-Dezember -1984
Mo, 08-Dezember -1980
Mo, 08-Dezember -1980
Do, 30-Dezember -1982
Mo, 12-November -1984

`TO_CHAR(PENALTYDATE,'DY,DD-MONTH-YYYY')`

MO, 08-Dezember -1980
DI, 05-Mai -1981
SA, 10-September-1983
SA, 08-Dezember -1984
MO, 08-Dezember -1980
MO, 08-Dezember -1980
DO, 30-Dezember -1982
MO, 12-November -1984

Queries



Basics

Joins



Sets



Hierarchy Queries



Basic query

- **FROM** defines the source (tables)
- **SELECT** defines the projection on the result
 - without condition the source is already the result
- **SELECT Name, City **FROM** Players;**

	NAME	CITY
1	Parmenter	Stratford
2	Baker	Inglewood
3	Hope	Stratford
4	Everett	Stratford
5	Collins	Eltham
6	Moorman	Eltham
7	Wise	Stratford

Queries



Joins



Sets



Hierarchy Queries



Basics

Schemas and Pseudonyms

- Tables can reside in a schema, e.g. **mycorp.employees**
- Oracle calls this user- or workspace

- Pseudonyms for tables can be used
 - **SELECT p.Name, p.City **from** Players p**
 - Especially useful for joins
- Aliases for columns can be used as well
 - **SELECT Name "LastName", City **from** Players**

- **DISTINCT** removes duplicate rows from the result set

Queries



Joins



Sets



Hierarchy Queries



Basics

Statistical functions

- Calculated over the query result ⇒ single row returned
 - Exception: GROUP BY
 - Affected by DISTINCT
- Available functions:
 - COUNT
 - MIN
 - MAX
 - SUM
 - AVG
 - STDDEV
 - VARIANCE



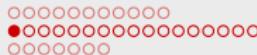
Statistical functions

- **SELECT COUNT(Amount)**
from Penalties;

- **SELECT COUNT(
DISTINCT Amount)from**
Penalties;

PENALTYNNO	PLAYERNNO	PENALTYDATE	AMOUNT
1	1	6 08.12.80	100
2	2	44 05.05.81	75
3	3	27 10.09.83	100
4	4	104 08.12.84	50
5	5	44 08.12.80	25
6	6	8 08.12.80	25
7	7	44 30.12.82	30
8	8	27 12.11.84	75

Queries



Joins



Sets



Hierarchy Queries



Conditions

WHERE clause

- Filters/restricts the query result
- Only those rows are returned which fulfill the conditions
- Comparison operators are used
 - $=$, $<$, $>$, \leq , \geq , \neq (or ' $!=$ ')
- Comparable are chars, numbers and dates [Oracle]
- Alphanumeric comparison based on ANSI value
- Multiple values can be compared using $=$ and \neq [Oracle]
 - **WHERE** ($c1, c2$) $=$ ($v1, v2$)
- Possible results: TRUE, FALSE, UNKNOWN [Oracle]

Queries

○○○○○○○○○○○○
○●○○○○○○○○○○○○
○○○○○○○○

Joins

○○○
○○○○○○○○
○○○○○○○○○○○○○○

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○

Conditions

WHERE clause [Oracle]

and	T	F	?
T	T	F	?
F	F	F	F
?	?	F	?

or	T	F	?
T	T	T	T
F	T	F	?
?	T	?	?

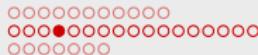
not	T	F	?
	F	T	?



BETWEEN operator

- Syntax: $\text{expr1 } [\text{NOT}] \text{ BETWEEN } \text{expr2 AND expr3}$
- Returns TRUE if
 - $\text{value} \geq \text{lower bound}$
 - $\text{value} \leq \text{upper bound}$
- If any of the expressions (expr1, expr2, expr3) is NULL the result is UNKNOWN [Oracle]
- Example: **SELECT * from Employees WHERE EmployeeNo BETWEEN 120 AND 200;**

Queries



Joins



Sets



Hierarchy Queries



Conditions

LIKE operator

- Syntax: expr1 [NOT] **LIKE** expr2
- Used for wildcard string comparison
 - '%' ⇒ any 0, 1 or n chars (cf. '*' Regex)
 - '_' ⇒ any exactly one character
- Example:
SELECT * from Employees WHERE Name LIKE 'K%';
- Be careful with (6 byte) blank padding of char (vs. varchar2)
[Oracle]

Queries



Joins



Sets



Hierarchy Queries



Conditions

IN operator

- Syntax: `expr1 [NOT] IN expr2`
- Returns TRUE if value (element) is contained in a set
- Example:
`SELECT * from Employees WHERE EmployeeNo IN (123,321);`

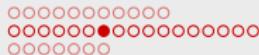
Queries**Joins****Sets****Hierarchy Queries****Conditions**

Subqueries with the IN operator

- Syntax:

WHERE colName1 IN (SELECT colName2 FROM table2)

- Avoids copy/pasting value lists for the IN clause
- Updated with every execution
- Especially useful for large lists
- If the subquery does not return a result (0 rows)
 - IN (subquery) evaluates to FALSE
 - NOT IN (subquery) evaluates to TRUE

Queries**Conditions****Joins****Sets****Hierarchy Queries**

Subqueries with the IN operator

Example

- Tables:
 - Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth)
 - Order(OrderNo, CustNo, Amount, Date)
- Select all orders from customers who have a last name starting with 'Ma'
- **SELECT * FROM Order WHERE CustNo IN (SELECT CustNo FROM Customer WHERE LastName LIKE 'Ma%')**

Queries

○○○○○○○○○○○○
○○○○○●○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○○○○○○○○○○○○

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○

Conditions

Comparisons with subqueries

- Syntax: expr operator (subquery))
- Possible operators: $<$, $>$, $=$, \geq , \leq , \neq
- When using a comparison operator the subquery must only return a single value

Queries**Conditions****Joins****Sets****Hierarchy Queries**

Comparisons with subqueries

Example

- Select all orders from customers who match these criteria:
LastName=Maier, FirstName=Franz, ZIP=4020, City=Linz,
DateOfBirth=1980-05-04

```
SELECT * FROM Order WHERE CustNo =  
(SELECT CustNo FROM Customer WHERE LastName='Maier'  
AND FirstName='Franz'  
AND ZIP=4020 AND City='Linz'  
AND DateOfBirth='1980-05-04');
```

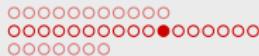
Queries**Joins****Sets****Hierarchy Queries**

Conditions

Subqueries with ALL/ANY operator

- Syntax: expr comparison_operator (**ALL|ANY**)(subquery))
- ALL evaluates to TRUE if all rows returned by the subquery fulfill the condition
- ANY evaluates to TRUE if at least one row returned by the subquery fulfills the condition

- Consider expressing the same conditions using the MIN() and MAX() functions

Queries**Conditions****Joins****Sets****Hierarchy Queries**

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers from Wels who have a higher revenue than any customer from Linz

```
SELECT * FROM Customer  
WHERE City='Wels' AND Revenue >= ANY  
(SELECT Revenue FROM Customer WHERE City='Linz');
```

Queries

Conditions

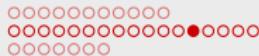
Joins**Sets****Hierarchy Queries**

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers from Wels who have a higher revenue than all customers from Linz

```
SELECT * FROM Customer  
WHERE City='Wels' AND Revenue >= ALL  
(SELECT Revenue FROM Customer WHERE City='Linz');
```

Queries**Joins****Sets****Hierarchy Queries****Conditions**

Subqueries with ALL/ANY operator

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select all customers who's birthday is at the same day as the one of the oldest customer

```
SELECT * FROM Customer  
WHERE DateOfBirth <= ALL  
(SELECT DateOfBirth FROM Customer);
```

Queries**Joins****Sets****Hierarchy Queries****Conditions**

Subqueries with EXISTS operator

- Syntax: **[NOT] EXISTS (subquery)**
- Evaluates to TRUE if at least one row is returned by the subquery, otherwise FALSE
- Never returns UNKNOWN [Oracle]

Queries**Joins****Sets****Hierarchy Queries****Conditions**

Subqueries with EXISTS operator

Example

- Select all customers who placed at least one order

```
SELECT * FROM Customer c
WHERE EXISTS
(SELECT * FROM Order o
WHERE o.CustNo = c.CustNo);
```

Queries



Joins



Sets



Hierarchy Queries



Conditions

NULL comparison

- Syntax: `expr1 IS [NOT] NULL`
- When comparing with `NULL` neither `=` nor `<>` can be used
- Explicitly evaluating `IS NULL` or `IS NOT NULL` prevents getting `UNKNOWN` values [Oracle]
 - **WHERE** `Name <> 'Knuth'` vs.
 - **WHERE** `Name IS NOT NULL AND Name <> 'Knuth'`

Queries

Conditions

Joins**Sets****Hierarchy Queries**

NULL comparison

Value a	Condition	Evaluates to
10	a IS NULL	FALSE
10	a IS NOT NULL	TRUE
NULL	a IS NULL	TRUE
NULL	a IS NOT NULL	FALSE
10	a = NULL	UNKNOWN
10	a != NULL	UNKNOWN
NULL	a = NULL	UNKNOWN
NULL	a != NULL	UNKNOWN
NULL	a = 10	UNKNOWN
NULL	a != 10	UNKNOWN



ORDER BY clause

- Syntax: **ORDER BY** expr [**ASC|DESC**] [, expr [**ASC|DESC**], ...]
- Order of rows in a result set is usually not guaranteed (sets vs. (ordered) lists)
- ASC order is the default ⇒ **ORDER BY** Quantity =
ORDER BY Quantity **ASC**



ORDER BY clause

- If multiple ORDER BY expressions are present ordering is applied in order of appearance in the query
- ASC|DESC can differ for each expression
- NULL values are placed at the end when ordering ascending and at the beginning when ordering descending
- Column name, alias and column index can be used to identify the column



Ordering & Grouping

ORDER BY clause

Example

- Table Customer(CustNo, LastName, FirstName, ZIP, City, Street, StreetNo, DateOfBirth, Revenue)
- Select LastName, FirstName, Revenue. Order by Revenue descending, then by LastName ascending, then by FirstName ascending

```
SELECT LastName, FirstName fn, Revenue r FROM Customer  
ORDER BY r DESC, LastName ASC, fn;
```

Queries



Joins



Sets



Hierarchy Queries



Ordering & Grouping

GROUP BY clause

- Syntax: **GROUP BY** col1 [, col2, ...]
- Groups rows in the result set based on similar properties
- Usually used in conjunction with statistical functions ⇒ those are evaluated for each group independently
- Only those columns used in the GROUP BY clause may appear in the SELECT clause
 - With the exception of using COUNT(*)
 - Example: **SELECT LastName, COUNT(*) FROM Customer GROUP BY LastName**

Queries

○○○○○○○○○○
○○○○○○○○○○○○○○
○○○●○○

Joins

○○○
○○○○○○○○
○○○○○○○○○○○○○○

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○

Ordering & Grouping

GROUP BY clause

Example

- Table Order(OrderNo, CustNo, Amount, Date)
- Select the number of orders, the total revenue and the average order amount (rounded to two decimal places) per calendar year. Sort the result descending based on the calendar year.

```
SELECT TO_CHAR(Date, 'YYYY') Year, COUNT(*) Count,
SUM(Amount) "Total Revenue",
ROUND(AVG(Amount),2) "Avg. Revenue" FROM Order
GROUP BY TO_CHAR(Date, 'YYYY') ORDER BY Year DESC;
```



HAVING clause

- Syntax: **HAVING** condition[s]
- Defines conditions for the filtered and grouped result set ⇒ 'second level WHERE'
 - Contrary to WHERE the conditions can contain statistical function
- Only those columns used in the GROUP BY clause or a statistical function may appear in the HAVING clause
- Example: **SELECT LastName, COUNT(*) FROM Customer GROUP BY LastName HAVING COUNT(*) > 10**

Queries**Joins****Sets****Hierarchy Queries**

Ordering & Grouping

HAVING clause

Example

- Table Order(OrderNo, CustNo, Amount, Date)
- Select the CustNo and number of orders (for the year 2014) of all customers who had at least two orders in 2014. Sort the result descending based on the order count.

```
SELECT CustNo, COUNT(*) "Order Count",
FROM Order WHERE TO_CHAR(Date, 'YYYY') = '2014'
GROUP BY CustNo HAVING COUNT(*) >= 2
ORDER BY COUNT(*) DESC;
```

Queries



Sample Database

Joins



Sets



Hierarchy Queries



The Tennis Club

- The club has amateur and professional players as members
- Professional players play in teams against other clubs
- Every professional player has a unique league id
- The club has several teams which participate in the cup
- Every team has a captain
- Members of a team change. So for each match it is necessary to log which player started for which team and which score was reached
- A player can get a penalty from the league for unfair behavior

Queries



Joins



Sets

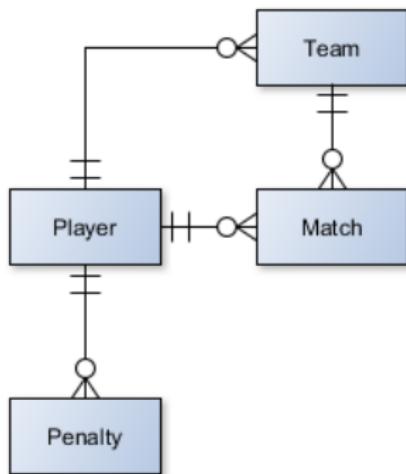


Hierarchy Queries



Sample Database

The Tennis Club



Queries



Joins



Sets



Hierarchy Queries



Sample Database

The Tennis Club

■ Tables:

- Players (PlayerNo, Name, LeagueNo, YearOfBirth, ...)
- Teams (TeamNo, Name, PlayerNo, Division)
- Matches (MatchNo, TeamNo, PlayerNo, SetsWon, SetsLost)
- Penalties (PenaltyNo, PlayerNo, PenaltyDate, Amount)

Queries

A grid of 15 red circles arranged in three rows of five. The circles are evenly spaced and aligned vertically.

Joins

Sets

10

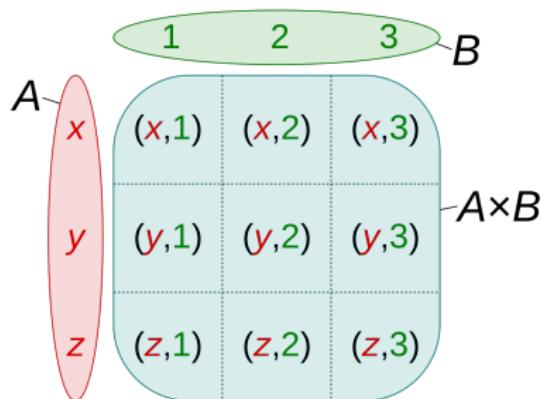
Hierarchy Queries

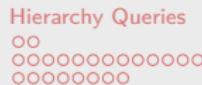
A decorative border consisting of a grid of small red circles. The circles are arranged in three rows: a top row with two circles, a middle row with ten circles, and a bottom row with six circles.

Oracle Joins

Basic Join

- FROM clause contains (at least) two tables
 - **SELECT * FROM A, B;**
 - The result is a cartesian product

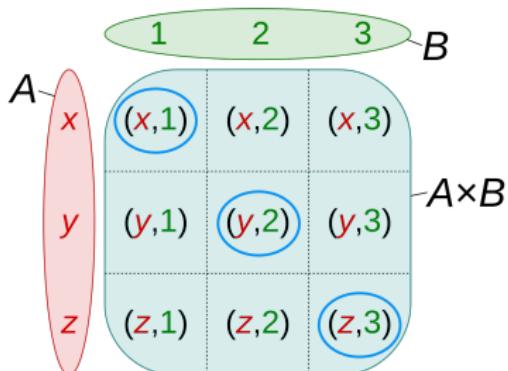




Oracle Joins

Inner Equijoin

- Adding Join Columns creates what is usually called an 'inner join'
 - **SELECT * FROM A, B WHERE A.FK = B.PK;**
 - This is the most common join for tables with foreign key relationships



Queries



Joins



Sets



Hierarchy Queries



Oracle Joins

Inner Equijoin

Example

Players	
PlayerNo	Name
2	Everett
6	Parmenter
44	Baker

Penalties			
PenaltyNo	PlayerNo	Date	Amount
1	6	2000	100
2	44	2001	75
5	44	2003	25

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Inner Equijoin

Example

```
SELECT pl.PlayerNo, pl.Name, pe.Amount  
FROM Players pl, Penalties pe  
WHERE pl.PlayerNo = pe.PlayerNo;
```

PlayerNo	Name	Amount
6	Parmenter	100
44	Baker	25
44	Baker	75

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Equijoin

- Two types:
 - inner equijoin
 - outer equijoin
- Previous example was an inner equijoin \Rightarrow only those players with at least one penalty are in the result set
- An outer equijoin is requested by adding '(+)' to the condition

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Outer Equijoin

Example

```
SELECT pl.PlayerNo, pl.Name, pe.Amount FROM Players pl,  
Penalties pe WHERE pl.PlayerNo = pe.PlayerNo (+);
```

PlayerNo	Name	Amount
6	Parmenter	100
44	Baker	25
44	Baker	75
2	Everett	

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Equijoin

Example

- Tables:
 - Players (PlayerNo, Name, Initials, ...)
 - Teams (TeamNo, PlayerNo, Division)
- Select for each team the TeamNo, the Name of the captain and the Division

```
SELECT t.TeamNo, p.Name, t.Division  
FROM Players p, Teams t  
WHERE p.PlayerNo = t.PlayerNo;
```

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Equijoin

Example

- Tables:
 - Players (PlayerNo, Name, Initials, ...)
 - Matches (MatchNo, PlayerNo, SetsWon, SetsLost)
- Select for each match the MatchNo, the name of the player and the won and lost sets. Sort the result by player name.

```
SELECT m.MatchNo, p.Name, m.SetsWon Won, m.SetsLost Lost  
FROM Players p, Matches m WHERE p.PlayerNo = m.PlayerNo  
ORDER BY p.Name;
```

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries



Equijoin

Example

- Table Penalties (PenaltyNo, PlayerNo, PenaltyDate, Amount)
- Select all PlayerNo and Names who ever got a penalty together with their average penalty (round to two decimal places).

```
SELECT pl.PlayerNo, pl.Name, ROUND(AVG(pe.Amount),2)  
FROM Players pl, Penalties pe WHERE pl.PlayerNo = pe.PlayerNo  
GROUP BY pl.PlayerNo, pl.Name;
```

Queries



Oracle Joins

Joins



Sets



Hierarchy Queries

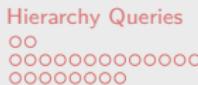


Equijoin

Example

- Select all PlayerNo and Names together with their average penalty (round to two decimal places). If a player hasn't ever got a penalty use 0 for the amount.

```
SELECT pl.PlayerNo, pl.Name, ROUND(AVG(NVL(pe.Amount,0)),2)  
FROM Players pl, Penalties pe  
WHERE pl.PlayerNo = pe.PlayerNo (+)  
GROUP BY pl.PlayerNo, pl.Name;
```



SQL Standard Joins

- Starting with version 9i Oracle supports joins according to the SQL:1999 Standard:
 - Cross Join
 - Natural Join
 - Equijoins with USING clause
 - Outer Joins (full, left, right)
 - Those have an easier syntax with a separate join condition

Queries



Standard Joins

Joins



Sets



Hierarchy Queries



Cross Join

- Syntax: **FROM** table1 **CROSS JOIN** table2
- Returns the cartesian product (cf. Oracle Equijoin without join columns)
- Example: **SELECT** pl.PlayerNo, pl.Name, pe.Amount **FROM** Players pl **CROSS JOIN** Penalties pe;

Queries



Standard Joins

Joins



Sets

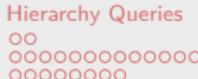


Hierarchy Queries



Natural Join

- Syntax: **FROM** table1 **NATURAL JOIN** table2
- Based on columns with the same name
 - Join columns (= those with the same name) have to have the same data types
 - Shared (join) columns are returned only once in the result set
⇒ Aliases cannot be used
- Example: **SELECT** PlayerNo, Name, Amount **FROM** Players **NATURAL JOIN** Penalties;



Standard Joins

Natural Join

Example

- Tables:
 - Players (PlayerNo, Name, Initials, ...)
 - Matches (MatchNo, PlayerNo, SetsWon, SetsLost)
 - Select for each match the MatchNo, the player's Name and the won and lost sets. Sort the result descending based on the difference between won and lost sets.

```
SELECT PlayerNo, pl.Name, SetsWon, SetsLost FROM Players pl  
NATURAL JOIN Matches ORDER BY (SetsWon—SetsLost) DESC;
```

Queries

○○○○○○○○○○
○○○○○○○○○○○○○○
○○○○○○

Standard Joins

Joins

○○○
○○○○○○○○○○
○○○●○○○○○○○○

Sets

○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○○○○○

Equijoin with USING clause

- Syntax: **FROM** table1 **JOIN** table2 **USING** (col)
- Based on columns with the same name
- Similar to NATURAL JOIN but the join columns can be chosen
- Example: **SELECT** PlayerNo, Name, Amount **FROM** Players **JOIN** Penalties **USING**(PlayerNo);

Queries



Standard Joins

Joins



Sets



Hierarchy Queries



Equijoin with USING clause

Example

- Tables:
 - Players (PlayerNo, Name, Initials, ...)
 - Teams (TeamNo, PlayerNo, Division)
- Select for each team the TeamNo, the captain's Name and the Division.

```
SELECT TeamNo, pl.Name, Division  
FROM Teams JOIN Players pl USING(PlayerNo);
```

Queries

○○○○○○○○○○○○
○○○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○●○○○○○○○

Standard Joins

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○

Join with ON clause

- Syntax: **FROM** table1 **JOIN** table2 **ON** (condition)
- Flexible and easy to control:
 - Aliases for columns can be used
 - Subqueries can be used
 - Logical operators can be used
- Example: **SELECT** pl.PlayerNo, pl.Name, pe.Amount **FROM** Players pl **JOIN** Penalties pe **ON**(pl.PlayerNo=pe.PlayerNo)
WHERE pl.PlayerNo=44;

Queries

○○○○○○○○○○
○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○
○○○○○●○○○○○

Standard Joins

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○○○○○○○○○
○○○○○○○

Join with ON clause

- It is even possible to join multiple tables:

```
SELECT d.DepartmentName, l.City, c.CountryName  
FROM Departments d  
JOIN Locations l ON (d.LocationId=l.LocationId)  
JOIN Countries c ON (l.CountryId=c.CountryId)  
WHERE c.RegionId=1;
```

Queries**Joins****Sets****Hierarchy Queries****Standard Joins**

Join with ON clause

Example

- Tables: Players (PlayerNo, Name, Initials, ...), Teams (TeamNo, PlayerNo, Division), Matches (MatchNo, TeamNo, SetsWon, SetsLost)
- Select for each match the MatchNo, the team's Division, the player's Name and the won and lost sets.

```
SELECT m.MatchNo, t.Division, p.Name, m.SetsWon, m.SetsLost  
FROM Matches m JOIN Teams t ON (m.TeamNo=t.TeamNo)  
JOIN Players p ON (p.PlayerNo=t.PlayerNo);
```

Queries



Standard Joins

Joins



Sets



Hierarchy Queries



Outer Join

- Syntax: **FROM** table1 (**LEFT|RIGHT|FULL**)**OUTER JOIN** table2 **ON** (condition)
- Comparable to the '(+)' Syntax in Oracle, but allows for better control
- Three Variants:
 - LEFT: Include rows from the left hand side table even if no match
 - RIGHT: Include rows from the right hand side table even if no match
 - FULL: Include rows from both tables even if no match

Queries



Joins



Sets



Hierarchy Queries



Standard Joins

Left Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM  
Players p LEFT OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		Teams	
PlayerNo	Name	PlayerNo	TeamNo
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

Queries



Joins



Sets



Hierarchy Queries



Standard Joins

Right Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM
Players p RIGHT OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		Teams	
PlayerNo	Name	PlayerNo	TeamNo
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

Queries



Standard Joins

Joins



Sets



Hierarchy Queries



Full Outer Join

```
SELECT p.PlayerNo, p.Name, t.PlayerNo, t.TeamNo FROM  
Players p FULL OUTER JOIN Teams t ON (p.PlayerNo=t.PlayerNo)
```

Players		Teams	
PlayerNo	Name	PlayerNo	TeamNo
2	Everett		
6	Parmenter	6	1
27	Collins	27	2
44	Baker		
		200	3

Queries

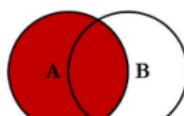


Standard Joins

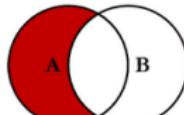
Joins



Overview Joins



```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```



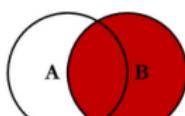
```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key  
WHERE B.Key IS NULL,
```

```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key
```

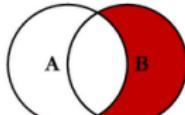
SQL JOINS



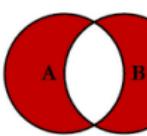
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```



```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL,
```



```
SELECT <select_list>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.Key = B.Key  
WHERE A.Key IS NULL  
OR B.Key IS NULL
```

© C.L. Meffert, 2008

Hierarchy Queries



Queries



Set Operations

Joins



Sets



Hierarchy Queries



Set Theory

- SQL queries return sets \Rightarrow sets can be merged
- Conditions:
 - Number of columns in the result has to be equal
 - Data types of the columns in the result have to be equal (keep column order in mind)
- Three variants:
 - UNION
 - INTERSECT
 - MINUS

Queries



Set Operations

Joins



Sets

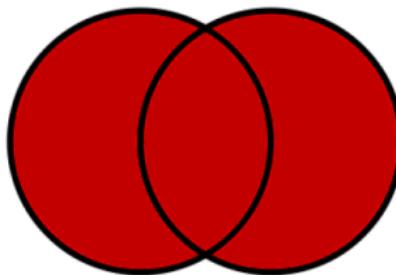


Hierarchy Queries



UNION

- Syntax: stmt1 **UNION [ALL]** stmt2
- The result set contains all rows of both statements
 - If the modifier ALL is added duplicate (=identical) rows are not removed



Queries

○○○○○○○○○○
○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○
○○○○○○○○○○○○○○

Sets

○○●○
○○○○○

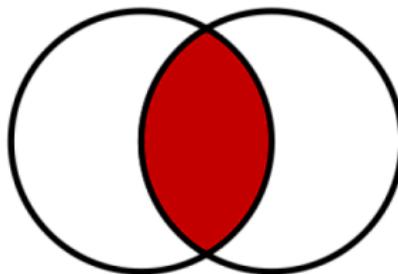
Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○

Set Operations

INTERSECT

- Syntax: stmt1 **INTERSECT** stmt2
- The result set contains the intersection of both statements
- Alternatives: EXISTS, IN



Queries



Set Operations

Joins



Sets

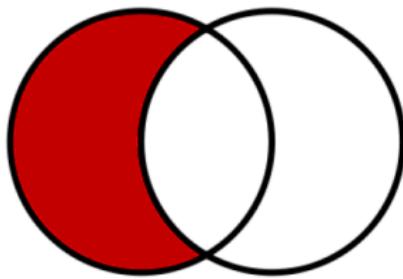


Hierarchy Queries



MINUS

- Syntax: stmt1 MINUS stmt2
- The result set contains the difference of both statements (rows are contained in the first but not in the second set)
- Alternatives: NOT EXISTS, NOT IN



Queries



Joins



Sets



Hierarchy Queries



Set Examples

UNION

Example

- Select all Players and their penalties.

```
SELECT Name, Initials, Amount FROM Players  
NATURAL JOIN Penalties
```

UNION

```
SELECT Name, Initials, 0 FROM Players
```

WHERE NOT EXISTS

```
(SELECT * FROM Penalties WHERE PlayerNo=Players.PlayerNo)  
ORDER BY Amount DESC;
```

Queries



Set Examples

Joins



Sets



Hierarchy Queries



UNION

Example Result

	NAME	INITIALS	AMOUNT
1	Collins	DD	100
2	Parmenter	R	100
3	Baker	E	75
4	Collins	DD	75
5	Moorman	D	50
6	Baker	E	30
7	Baker	E	25
8	Newcastle	B	25
9	Bailey	IP	0
10	Bishop	D	0
11	Brown	M	0
12	Collins	C	0
13	Everett	R	0
14	Hope	PK	0
15	Miller	P	0
16	Parmenter	P	0
17	Wise	GWS	0

Queries

○○○○○○○○○○○○
○○○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○○○○○○○○○○○○

Sets

○○○○
○○●○○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○○

Set Examples

INTERSECT

Example

- Select all PlayerNos of captains.

```
SELECT PlayerNo FROM Players
INTERSECT
SELECT PlayerNo FROM Teams;
```

Queries

○○○○○○○○○○○○
○○○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○○○○○○○○○○○○

Sets

○○○○
○○○●○○

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○

Set Examples

INTERSECT

Example Result

	PLAYERNO
1	6
2	27

Queries



Set Examples

Joins



Sets



Hierarchy Queries



MINUS

Example

- Select all PlayerNos of players who have never competed in a league match.

```
SELECT PlayerNo FROM Players  
MINUS  
SELECT PlayerNo FROM Matches;
```

Queries

○○○○○○○○○○○○
○○○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○○○
○○○○○○○○○○○○○○○○

Sets

○○○○
○○○○●

Hierarchy Queries

○○
○○○○○○○○○○○○
○○○○○○○

Set Examples

MINUS

Example Result

	PLAY...	Y
1		7
2		28
3		39
4		95
5		100

Queries



Joins



Sets



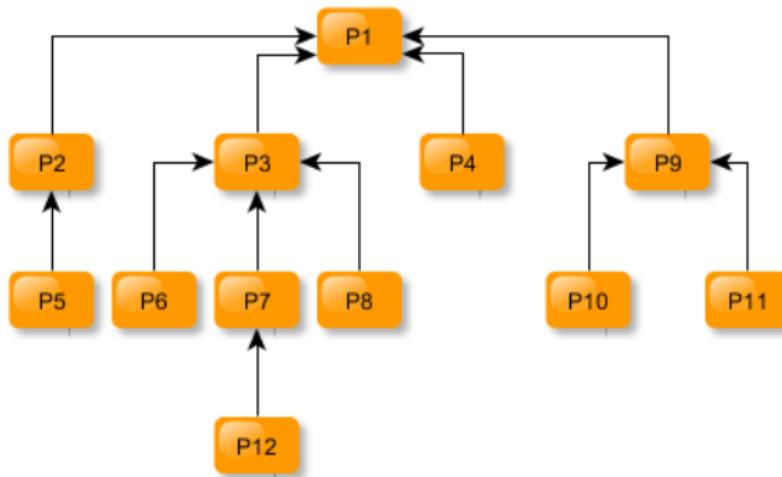
Hierarchy Queries



Introduction

Sample Data

- Parts list of a production company



Queries



Joins



Sets



Hierarchy Queries



Introduction

Sample Data

```
CREATE TABLE Parts(  
    Sub varchar2(3) NOT NULL,  
    Super varchar2(3),  
    Price number(7,2) DEFAULT 0  
);
```

Sub	Super	Price
P1		130
P2	P1	15
P3	P1	65
P4	P1	20
P9	P1	45
P5	P2	10
P6	P3	10
P7	P3	20
P8	P3	25
P12	P7	10
P10	P9	12
P11	P9	21

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY

- Syntax: **CONNECT BY [PRIOR]** condition [**START WITH** condition] [**ORDER SIBLINGS BY column**]
- Exists only in Oracle (SQL Standard: **WITH clause**)
- Clauses:
 - PRIOR: placed left or right of the equality sign denotes the subordinated column
 - START WITH: defines the starting point of the recursive query
 - ORDER BY SIBLINGS: enforces ordering on the tuple of the same level
 - LEVEL: pseudo column indicating the hierarchy level

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY

Example

- Select all parts (down to the lowest level) component P3 consists of.

```
SELECT Sub FROM Parts  
CONNECT BY PRIOR Sub = Super  
START WITH Super = 'P3'
```

Queries

○○○○○○○○○○
○○○○○○○○○○○○○○
○○○○○○

Joins

○○○
○○○○○○○○
○○○○○○○○○○○○○○

Sets

○○○○
○○○○○

Hierarchy Queries

○○
○○●○○○○○○○○○○
○○○○○○○

Oracle CONNECT BY

CONNECT BY – Condition Placement

```
SELECT Sub, Super FROM Parts  
WHERE sub != 'P7'  
CONNECT BY PRIOR Sub=Super  
START WITH Sub='P3';
```

Sub	Super
P3	P1
P6	P3
P12	P7
P8	P3

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – Condition Placement

```
SELECT Sub, Super FROM Parts  
CONNECT BY PRIOR Sub=Super  
    AND sub != 'P7'  
START WITH Sub='P3';
```

Sub	Super
P3	P1
P6	P3
P8	P3

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – LPAD

- Syntax: LPAD(str1, paddedLength [, padStr])
- Can be used to display hierarchy levels with indentation

```

SELECT LEVEL, LPAD(' ', 2*LEVEL-1)
    || Sub
FROM Parts
CONNECT BY PRIOR Sub=Super
START WITH Sub='P3';
    
```

LEVEL	Part
1	P3
2	P6
2	P7
3	P12
2	P8



Oracle CONNECT BY

CONNECT BY – CONNECT_BY_ROOT

- Modifier for resolving the root element (pay attention to the starting point)

```
SELECT Sub, CONNECT_BY_ROOT
      Sub "Root"
FROM Parts START WITH Super='P1'
CONNECT BY PRIOR Sub=Super;
```

P2	P2
P5	P2
P3	P3
P6	P3
P7	P3
P12	P3
P8	P3
P4	P4
P9	P9
P10	P9
P11	P9

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – NOCYCLE

- Hierarchies are basically trees
- Recursive cycles in trees are not allowed ⇒ leads to a DB error
- The NOCYCLE clause can be used to suppress the recursion and prevent the issue

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – NOCYCLE

- Precondition: P12 is set as super element of P1

```
SELECT LPAD(' ', 2*LEVEL-1) || Sub,
       Super
  FROM Parts START WITH Sub='P3'
 CONNECT BY NOCYCLE PRIOR Sub=
        Super;
```

<code>LPAD(' ', 2*LEVEL-1) SUB</code>	<code>SUPER</code>
P3	P1
P6	P3
P7	P3
P12	P7
P1	P12
P2	P1
P5	P2
P4	P1
P9	P1
P10	P9
P11	P9
P8	P3

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – CONNECT_BY_ISCYCLE

- Shows if and where a cycle exists

```
SELECT Sub, Super,
      CONNECT_BY_ISCYCLE
FROM Parts START WITH Sub='P1'
CONNECT BY NOCYCLE PRIOR Sub=
      Super;
```

	Sub	Super	CONNECT_BY_ISCYCLE
P1	P12		0
P2	P1		0
P5	P2		0
P3	P1		0
P6	P3		0
P7	P3		0
P12	P7		1
P8	P3		0
P4	P1		0
P9	P1		0
P10	P9		0
P11	P9		0

Queries



Joins



Sets



Hierarchy Queries



Oracle CONNECT BY

CONNECT BY – CONNECT_BY_ISLEAF

- Shows if an element is a leaf node in the tree

```

SELECT Sub, Super,
       CONNECT_BY_ISLEAF
  FROM Parts START WITH Sub='P1'
 CONNECT BY NOCYCLE PRIOR Sub=
           Super;
  
```

Sub	Super	Leaf
P1	P12	0
P2	P1	0
P5	P2	1
P3	P1	0
P6	P3	1
P7	P3	0
P12	P7	1
P8	P3	1
P4	P1	1
P9	P1	0
P10	P9	1
P11	P9	1

Queries**Joins****Sets****Hierarchy Queries**

Oracle CONNECT BY

CONNECT BY – SYS_CONNECT_BY_PATH

- Shows the path from the root to the sub node
- Precondition: P12 is no longer super for P1

```
SELECT SYS_CONNECT_BY_PATH(  
    Sub, ' > ') "Path"  
FROM Parts START WITH Super IS  
    NULL  
CONNECT BY PRIOR Sub=Super;
```

Path
> P1
> P1 > P2
> P1 > P2 > P5
> P1 > P3
> P1 > P3 > P6
> P1 > P3 > P7
> P1 > P3 > P7 > P12
> P1 > P3 > P8
> P1 > P4
> P1 > P9
> P1 > P9 > P10
> P1 > P9 > P11

Queries



Joins



Sets



Hierarchy Queries

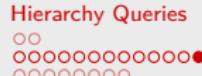


Oracle CONNECT BY

CONNECT BY – ORDER SIBLINGS BY

```
SELECT LPAD(' ', LEVEL)||Sub "Part",
       Super
  FROM Parts CONNECT BY PRIOR Sub
           =Super
 START WITH Super IS NULL
 ORDER BY Sub DESC;
```

Part	SUPER
P9	P1
P8	P3
P7	P3
P6	P3
P5	P2
P4	P1
P3	P1
P2	P1
P12	P7
P11	P9
P10	P9
P1	(null)



Oracle CONNECT BY

CONNECT BY – ORDER SIBLINGS BY

```
SELECT LPAD(' ', LEVEL)||Sub "Part",
        Super
FROM Parts CONNECT BY PRIOR Sub
        =Super
START WITH Super IS NULL
ORDER SIBLINGS BY Sub DESC;
```

Part	SUPER
P1	(null)
P9	P1
P11	P9
P10	P9
P4	P1
P3	P1
P8	P3
P7	P3
P12	P7
P6	P3
P2	P1
P5	P2

Queries



WITH clause

Joins



Sets



Hierarchy Queries



WITH clause

- SQL Standard way of performing hierarchy queries
- Supported in Oracle since version 11g
- Utilizes recursive calls of inline views

Queries



Joins



Sets



Hierarchy Queries



WITH clause

WITH clause

```
WITH PartsRec(Sub, Super) AS
(
  SELECT Sub, Super FROM Parts
  WHERE Super IS NULL
  UNION ALL
    (
      SELECT p.Sub, p.Super
      FROM PartsRec pr, Parts p
      WHERE pr.Sub = p.Super
    )
)
SELECT Sub, Super FROM PartsRec;
```

SUB	SUPER
P1	(null)
P2	P1
P3	P1
P4	P1
P9	P1
P5	P2
P6	P3
P7	P3
P8	P3
P10	P9
P11	P9
P12	P7

Queries



Joins



Sets



Hierarchy Queries



WITH clause

WITH clause – Ordering

- By default sorts on the same hierarchy level first (BREADTH)
- Can be overwritten to sort down (DEPTH) first

Queries



Joins



Sets



Hierarchy Queries



WITH clause

WITH clause – Ordering

```
WITH PartsRec(Sub, Super) AS
(
  SELECT Sub, Super FROM Parts
  WHERE Super IS NULL
  UNION ALL
    (
      SELECT p.Sub, p.Super
      FROM PartsRec pr, Parts p
      WHERE pr.Sub = p.Super
    )
)
SEARCH DEPTH FIRST BY Sub SET sort1
SELECT Sub, Super FROM PartsRec
ORDER BY sort1;
```

Sub	SUPER
P1	(null)
P2	P1
P5	P2
P3	P1
P6	P3
P7	P3
P12	P7
P8	P3
P4	P1
P9	P1
P10	P9
P11	P9

Queries



Joins



Sets



Hierarchy Queries



WITH clause

WITH clause – Advanced features

- Several features provided by the Oracle approach are not directly available using the WITH clause
 - LEVEL
 - SYS_CONNECT_BY_PATH
 - CONNECT_BY_ISCYCLE
- Can be substituted by other constructs



WITH clause

WITH clause – 'LEVEL'

- Using a counter (pseudo column LEVEL not available)

```
WITH PartsRec(Lvl, Sub, Super) AS
(
  SELECT 1 "Lvl", Sub, Super FROM Parts
  WHERE Super IS NULL
  UNION ALL
    (
      SELECT Lvl+1 "Lvl", p.Sub, p.Super
      FROM PartsRec pr, Parts p
      WHERE pr.Sub = p.Super
    )
)
SELECT Lvl, Sub, Super FROM PartsRec;
```

LVL	SUB	SUPER
1	P1	(null)
2	P2	P1
2	P3	P1
2	P4	P1
2	P9	P1
3	P5	P2
3	P6	P3
3	P7	P3
3	P8	P3
3	P10	P9
3	P11	P9
4	P12	P7

Queries



Joins



Sets



Hierarchy Queries



WITH clause

WITH clause – 'SYS_CONNECT_BY_PATH'

```

WITH PartsRec(Lvl, Path, Sub, Super) AS
(
SELECT 1 "Lvl", '/'||Sub "Path", Sub, Super
  FROM Parts
 WHERE Super IS NULL
UNION ALL
  (
    SELECT Lvl+1 "Lvl", SUBSTR(Path||'/'||
      p.Sub,0.100) "Path", p.Sub, p.Super
      FROM PartsRec pr, Parts p
     WHERE pr.Sub = p.Super
  )
)
SELECT Lvl, Path, Sub, Super FROM PartsRec;
  
```

LVL	PATH	SUB	SUPER
1 /P1	P1	(null)	
2 /P1/P2	P2	P1	
2 /P1/P3	P3	P1	
2 /P1/P4	P4	P1	
2 /P1/P9	P9	P1	
3 /P1/P2/P5	P5	P2	
3 /P1/P3/P6	P6	P3	
3 /P1/P3/P7	P7	P3	
3 /P1/P3/P8	P8	P3	
3 /P1/P9/P10	P10	P9	
3 /P1/P9/P11	P11	P9	
4 /P1/P3/P7/P12	P12	P7	

Queries

A grid of 18 circles arranged in three rows of six circles each.

Joins

Sets

10

Hierarchy Queries

WITH clause

WITH clause – 'SYS_CONNECT_BY_ISCYCLE'

```

WITH PartsRec(Lvl, Sub, Super) AS
(
    SELECT 1 "Lvl", Sub, Super FROM Parts
    WHERE Super = 'P12'
    UNION ALL
        (
            SELECT Lvl+1 "Lvl", p.Sub, p.Super
            FROM PartsRec pr, Parts p
            WHERE pr.Sub = p.Super
        )
)
CYCLE Sub SET IsCycle TO 1 DEFAULT 0
SELECT Lvl, Sub, Super, IsCycle
FROM PartsRec;

```

LVL		SUPER	ISCYCLE
1	P1	P12	0
2	P2	P1	0
2	P3	P1	0
2	P4	P1	0
2	P9	P1	0
3	P5	P2	0
3	P6	P3	0
3	P7	P3	0
3	P8	P3	0
3	P10	P9	0
3	P11	P9	0
4	P12	P7	0
5	P1	P12	1

Database Tuning

Markus Haslinger

DBI/INSY

Agenda

- 1 DB Buffer Cache**
 - DB Buffer Cache
- 2 Optimizer**
 - Optimizer
- 3 SQL Tuning**
 - SQL Tuning
- 4 Hints & Tips**
 - Hints & Tips

DB Performance Optimization

- If a database shows performance problems there are various possible bottlenecks and mitigation strategies
 - Configuration of the DB buffer (cache)
 - Optimizer
 - Statistics and data analytics
 - Search for costly SQL queries
 - Optimizing SQL queries using execution plans
 - Hints
- As with every optimization: don't do it 'just because'
 - ⇒ only if actual measurements show a bottleneck, specific (manual) steps should be taken
 - And every optimization attempt has to be *tested* to ensure that it doesn't actually makes things worse

DB Buffer Cache

- Write- and Read-Cache for user data¹ of the database
- Blocks in the buffer correspond to file blocks
- Data is *always* read from the cache
 - If the required data is not already in the cache it is first loaded into the cache, then read from there
- The cache hit rate represents the percentage of blocks which could be read from the cache directly (compared to the total amount of data access operations)
 - A hit rate of $\geq 95\%$ is the target
 - A hit rate of $< 90\%$ is already an indication of performance problems
 - This means that the buffer size has to be big enough \Rightarrow DB servers need & use lots of RAM

¹That means user (application) data instead of system data, not data *about* users (roles).

DB Buffer Cache

- The cache size is defined by the parameter DB_BLOCK_BUFFERS (number of blocks)
- The actual size is DB_BLOCK_BUFFERS * DB_BLOCK_SIZE
- Alternatively the size can be controlled via DB_CACHE_SIZE or SGA_TARGET² (XE Edition)
- The parameter DB_CACHE_ADVICE provides an estimate about how a change of the buffer cache size would influence the amount of physical data access operation

²Remember: SGA = Shared Global Area

CMD: Read & Set Parameters

```
SQL> show parameter db_cache
```

NAME	TYPE	VALUE
db_cache_advice	string	ON
db_cache_size	big integer	52M

```
SQL> show parameter db_block
```

NAME	TYPE	VALUE
db_block_buffers	integer	0
db_block_checking	string	FALSE
db_block_checksum	string	TYPICAL
db_block_size	integer	8192

```
SQL> alter system set db_cache_advice=ON;
SQL> alter system set db_cache_advice=OFF;
SQL> alter system set DB_CACHE_SIZE=52M;
```

Determine Cache Hit Ratio

- The view v\$sysstat contains various statistical values
- Here relevant are:
 - physical reads: Number of data blocks which have been read from a file³ into the cache⁴
 - db block gets: Number of data requests which have been served directly from the segments⁵
 - db consistent gets: Number of data requests which have been served – for read consistency reasons – from rollback- or undo-segments

³A file on the disk

⁴Since the last (re)start of the instance – stopping it clears the cache

⁵Excluding rollback- and undo-segments

DB Buffer Cache

CMD: Determine Cache Hit Ratio

```
SQL> select name, value from v$sysstat

SQL> select round(100*(1-(p.value / (b.value + c.value))),2) "Hit Ratio"
  2  from v$sysstat p, v$sysstat b, v$sysstat c
  3  where p.name = 'physical reads'
  4  and b.name = 'db block gets'
  5  and c.name = 'consistent gets';

      Hit Ratio
-----  
97,47
```

CMD: Use Cache Advice

```
SQL> select size_for_estimate "BC_Size", size_factor,
      estd_physical_read_factor "PHYS_Factor",
      estd_physical_reads "PHYS_Reads"
    from v$db_cache_advice;
```

BC_Size	SIZE_FACTOR	PHYS_Factor	PHYS_Reads
4	,0769	1,5648	43586
8	,1538	1,342	37380
12	,2308	1,2182	33932
16	,3077	1,161	32339
20	,3846	1,1304	31487
24	,4615	1,0974	30566
28	,5385	1,0745	29928
32	,6154	1,052	29303
36	,6923	1,0356	28845
40	,7692	1,0243	28532
44	,8462	1,0148	28265
48	,9231	1,0081	28080
52	1	1	27854
56	1,0769	,9942	27692
60	1,1538	,9875	27506
64	1,2308	,9861	27466
68	1,3077	,9861	27466
72	1,3846	,9651	26881
76	1,4615	,9286	25866
80	1,5385	,8099	22558

Optimizer

- With the optimizer we attempt to find an optimal execution plan for a statement
- Different goals can be defined:
 - RULE: Rule based, determined priority definition
 - ALL_ROWS: Cost based, goal is to achieve the highest data rate overall
 - FIRST_ROWS_n: Cost based, goal is to achieve fast response for the first n^6 data rows

⁶ $n=1, n=10, n=100, n=1000$

Optimizer

- There are several options to set the optimizer:
 - For the instance in the <SID>.ora file:
`OPTIMIZER_MODE=<mode>`
 - For the session: `ALTER SESSION SET OPTIMIZER_MODE=<mode>`
 - For a statement using Hints⁷: `SELECT /*+<mode>/ ...`
 - For modes `FIRST_ROWS(n)` & `ALL_ROWS`

⁷More on that later

Statistics

- To create good results the cost based optimizer requires statistical data about DB objects
- This data is stored in the Data Dictionary
 - Tables: DBA_TAB_STATISTICS, DBA_TAB_COL_STATISTICS, DBA_TAB_HISTROGRAMS, DBA_IND_STATISTICS,...

Tables	Columns	Indexes	System
Row Count	Number of distinct values	Number of leaves	I/O Performance
Number of blocks	Number of NULL values	Levels	CPU Performance
Avg. row length	Value distribution (Histogram)	Clustering-Factor	
	Extended Statistics		

Statistics

- Statistics are usually automatically updated by the Oracle DBS (during maintenance windows)
- However, it is also possible to update the statistics manually using the following procedures⁸:
 - GATHER_INDEX_STATS
 - GATHER_TABLE_STATS
 - GATHER_SCHEMA_STATS
 - GATHER_DICTIONARY_STATS
 - GATHER_DATABASE_STATS

⁸Package DBMS_STATS

Statistics

- Sampling can be used to prevent full table scans
 - ⇒ especially important with huge tables

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('OE',
                                         DBMS_STATS.AUTO_SAMPLE_SIZE);

select num_rows, blocks, avg_space, sample_size
from user_tab_statistics
where table_name='TEST1';
```

Statistics – Histograms

- Histograms can also be recreated on demand

```
DBMS_STATS.GATHER_TABLE_STATS (OWNNAME => 'OE',
                                TABNAME => 'INVENTORIES',
                                METHOD_OPT => 'FOR COLUMNS SIZE 20 warehouse_id');

SELECT column_name, num_distinct, num_buckets, histogram
FROM   USER_TAB_COL_STATISTICS
WHERE  table_name = 'INVENTORIES'
AND    column_name = 'WAREHOUSE_ID';

SELECT endpoint_number, endpoint_value
FROM   USER_HISTOGRAMS
WHERE  table_name = 'INVENTORIES'
AND    column_name = 'WAREHOUSE_ID'
ORDER BY endpoint_number;
```

Statistics – Generate/Delete

■ Check for existing statistics data

```
SELECT owner, table_name, last_analyzed
FROM   dba_tables
WHERE  table_name='PRODUCTS'
AND    owner='HTL';
```

■ Rebuild index using statistics

```
ALTER INDEX htl.idx_products
REBUILD PARALLEL NOLOGGING COMPUTE STATISTICS;
```

■ Generate statistics for all indexes of a table

```
Execute DBMS_STATS.GATHER_TABLE_STATS (ownname => 'HTL',
                                         TABNAME      => 'PRODUCTS',
                                         CASCADE      => TRUE,
                                         METHOD_OPT   => 'FOR ALL INDEXED COLUMNS',
                                         ESTIMATE_PERCENT => DBMS_STATS.AUTO_SAMPLE_SIZE);
```

■ Delete statistics data

```
EXECUTE DBMS_STATS.DELETE_TABLE_STATS (
  ownname => 'HTL',
  Tabname => 'PRODUCTS');
```

SQL Cache

- The view v\$sql provides details about cached SQL statements
 - Number of calls
 - Number of block accesses
 - Buffer hit ratio
- This is helpful in identifying SQL statements which require optimization

```
select to_char(executions, '999G999G990') "executions",
       to_char(buffer_gets, '999G999G990') "gets",
       to_char(buffer_gets/greatest(nvl(executions,1),1),
              '999G999G990') "gets je exec",
       to_char (round(100*(1-(disk_reads/greatest(nvl(buffer_gets,1),1))),2),
              '990D00') Trefferquote,
       sql_text
  from v$sql
 where buffer_gets > 1000
 order by buffer_gets desc;
```

SQL Tuning / Explain Plan

- The optimizer creates an execution plan for every query
- The creation of this plan is influenced by many factors
 - e.g. optimizer mode, statistical data, hints,...
- The plan can be retrieved and inspected⁹

```
delete plan_table;

explain plan
set statement_id = 'MySelect'
for
Select ...; /* Hier das Select Statement angeben */

select lpad(' ',2*level) || OPERATION || ' ' ||
       OPTIONS || ' ' || OBJECT_NAME QUERY_PLAN
from PLAN_TABLE WHERE STATEMENT_ID = 'MySelect'
connect by prior id = PARENT_ID AND STATEMENT_ID = 'MySelect'
start with id = 1;
```

⁹In SQL Developer a graphical view is available as well

Indices

- With the use of indices (some) costly full table scans can be averted
- Several types of index are available in Oracle:
 - B-Tree Index (default)
 - Bitmap Index
 - Function-based Index
 - Partitioned Index
- Due to an index being cost intensive itself (creation, update, storage) they are *not* a cure-all but usage has to be carefully deliberated ⇒ cost vs. benefit

B-Tree Index

- The default index
- Well suited for primary key and highly-selective¹⁰ indices
- Good for providing sorted data (by indexed columns)

```
Select *  
From employees  
Where lastname='King'  
And firstname='Robert';  
  
CREATE INDEX emp_name_Idx ON Employees(lastname, firstname);
```

¹⁰ Meant for a specific scenario, not 'general purpose'

Bitmap Index

- Good for data with low cardinality
- Columns can be effectively joined using and & or
- count is fast as well
- *Not available in Oracle XE Edition*

ROWID	male	female	active	inactive
12345	1	0	1	0
12346	1	0	0	1
12347	0	1	1	0
12348	0	1	0	1

```
CREATE BITMAP INDEX cust_bmx ON Customers(gender, status);
```

Function Based Index

- A B-Tree index is created using the results of a function
- This can be useful, e.g. if a name search should be *not* case sensitive

```
Select *  
From employees  
Where upper(lastname)='KING'  
And upper(firstname)='ROBERT';  
  
CREATE INDEX emp_uppername_Idx ON  
Employees(upper(lastname), upper(firstname));
```

Index Usage

- An existing index cannot be used by every query
- Following are examples what to avoid to allow index usage

Index not used	Index used
<pre>select * from Employees where substr(lastname,1,3)='Kin';</pre>	<pre>select * from Employees where lastname like 'Kin%';</pre>
<pre>select * from Products where categoryid != 0;</pre>	<pre>select * from Products where categoryid >0;</pre>

Index Usage

Index <i>not</i> used	Index used
select * from Employees where lastname firstname = 'KingRobert';	select * from Employees where lastname='King' and firstname='Robert';
select * from Products where unitprice*1.2 >120;	select * from Products where unitprice >100;

Hints

- Hints allow the user to make decisions which are usually made by the optimizer
- <hint> is a hint keyword, while [text] represents (optional) parameters for the hint

Syntax

```
<statement> /*+ <hint> [text] [<hint>[text]] ... */  
<statement> --+ <hint> [text] [<hint>[text]] ...
```

Hints – Optimize

- The following hints allow to set the optimizer mode
 - ALL_ROWS
 - FIRST_ROWS(n)
 - RULE

```
SELECT /*+ ALL_ROWS */ kundennr, nachname
FROM kunde
WHERE kundennr=7566;
```

```
SELECT /*+ RULE */ kundennr, nachname
FROM kunde
WHERE kundennr=7566;
```

Hints – Access Path

Either	Or	Notes
FULL		
CLUSTER	HASH	Only for cluster objects
INDEX	NO_INDEX	
INDEX_ASC	INDEX_DESC	
INDEX_COMBINE	INDEX_JOIN	
INDEX_FFS	NO_INDEX_FFS	FFS=Fast Full Index Scan
INDEX_SS	NO_INDEX_SS	SS=Index Skip Scan
INDEX_SS_ASC	INDEX_SS_DESC	

- More details and more hints can be found [here](#)

Hints – Access Path

```
SELECT /*+ NO_INDEX (kunde kunde_nachname_idx) */ *
FROM kunde
WHERE nachname='Gehrke' ;
```

Hints – Join Method

- The following hints allow to control the join method
 - USE_NL: Nested Loop-Join is used
 - Good for OLAP system which should return the first result rows as quickly as possible.
 - The result set should be < 10% of total rows
 - USE_HASH: Hash Join is used
 - The whole result is determined, then rows are returned to the user
 - USE_MERGE: Sort Merge Join is used
 - Performant if all columns of the join clause are pre-sorted by an index

Tuning Tips – Database

- Block size at least 8KB
- Ensure sufficiently large buffer cache
 - Monitor cache hit ratio
- Redo log files of sufficient size
 - Depending on the number of transactions 10 to 200MB
- Number of rollback segments big enough
 - This limits the number of *concurrent* transactions!
- Avoid unnecessary checkpoints
 - Parameter `log_checkpoint_interval=1000000`
 - Parameter `log_checkpoint_timeout=0`

Tuning Tips – Database

- Indices on tables which are frequently updated need to be reorganized regularly
 - `alter index <index> rebuild`
 - Choose a time frame with low user activity
 - Make sure there is enough TEMP space available
- Use proper storage parameters to limit the number of extents
 - A few hundred at most

Tuning Tips – SQL Statements

- Use cost model based optimizer with up to date analysis data
- Create histograms
- When using RULE be careful which table is the driving table¹¹
- When using or conditions consider activating RULE or use UNION
- Often used, small tables can be marked cache to not be removed from the buffer after a full table scan
- Don't 'forget' now unused indices
 - Which still consume system resources
- Avoid generalized, non-selective indices

¹¹Last in join clause

Tuning Tips – SQL Statements

- When performing performance measurements mind the cache effect
 - The first query execution might load data from a file or over the network (I/O)
 - The second and consecutive requests are possibly served from the buffer cache
- Indices are not used if the query contains one of the following operators:
 - `is not null`
 - `<>`
 - `like '%abc...'`
- Do not use `exists` subqueries, but `in` subqueries instead

XML

Markus Haslinger

DBI

Agenda

1 Introduction

- Semi-structured Data
- XML document structure

2 Schema Definition

- XSD

3 XPath

- XPath
- XQuery

Introduction

- Concept of semi-structured data:
 - Structured data
 - Does not adhere to a formal structure (like a relational database)
 - Contains markers to separate semantics from data (e.g. tags)
 - Also known as self-describing structure
- Examples:
 - XML
 - JSON

Properties

- Can describe entities
- Entities can have attributes
- Entities belonging to the same class do not have to have the same attributes
- Order of attributes is not important
- Allows for nested data
- Allows for list representations
- Be careful: can quickly get messy



XML

- Extensible Markup Language
- Markup Language:
 - Allows to annotate a document
 - The markup is syntactically distinguishable from the text
- Extensible: allows to create new tags
- Often used for:
 - Transferring data between applications (APIs)
 - Storing data (without a database, e.g. config file)
- Widespread use, despite rise of JSON

XML

- XML documents use tags to add semantic information
 - Either start (`<foo>`) and end (`</foo>`) tag
 - Or no-content-tag: `<bar/>`
- Tags can have attributes (`<foo bar="baz">foobar</foo>`)
- Array type is supported
- Usually Unicode
- Goal:
 - Machine readable *and*
 - Human readable

Working with XML

- Rarely (never?) write XML by hand
- All languages have libraries available for:
 - Serialization
 - Deserialization
 - ⇒ very common (and good?) interface file type
- Despite XML being human readable you rarely should have to
- The structure of an XML document allows for easy representation of an object graph

XML – Example

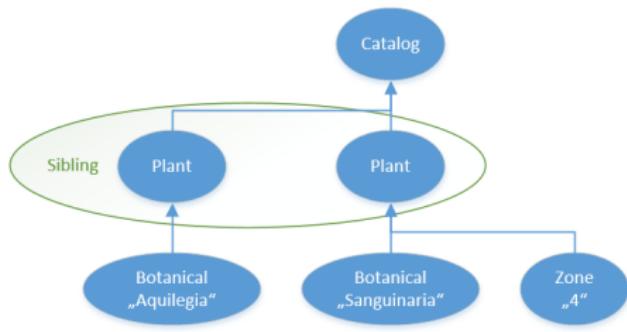
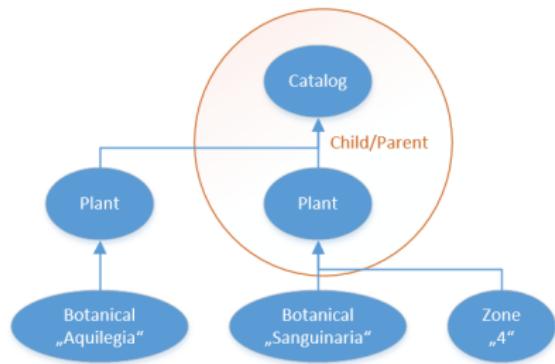
Example

```
<CATALOG>
  <PLANT>
    <COMMON>Bloodroot</COMMON>
    <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
    <ZONE>4</ZONE>
    <PRICE>$2.44</PRICE>
    <AVAILABILITY>031599</AVAILABILITY>
  </PLANT>
  <PLANT>
    <COMMON>Columbine</COMMON>
    <BOTANICAL>Aquilegia canadensis</BOTANICAL>
    <ZONE>3</ZONE>
    <PRICE>$9.37</PRICE>
    <AVAILABILITY>030699</AVAILABILITY>
  </PLANT>
</CATALOG>
```

XML – Example – Observations

- Elements nested in other elements
- No data types specified
- Everything appears to be text
 - Including the numeric values
 - Different formats (leading 0, currency)
- Lots of structure information compared to the actual data

Tree structure



- Always one root node
- Always only one parent
- 0..n siblings
- 0..n children

Attributes vs. Child-Elements

Attribute

```
<Product Id="3">  
...  
</Product>
```

Child Element

```
<Product>  
  <Id>3</Id>  
  ...  
</Product>
```

- Represents the same information
- Attributes cannot contain child elements
 - Use for simple, directly dependent values
- Both can be optional

Collections

- No special syntax (but often array type attribute)
- Several children of the same type

Collection

```
<Products type="array">
    <Product>...</Product>
    <Product>...</Product>
    <Product>...</Product>
</Products>
```

XML DTD

- DTD stands for Document Type Definition
- Describes for an XML document:
 - The structure
 - The allowed elements
 - The allowed attributes
- A XML document with correct syntax is called 'well formed'
- If additional conforming to a DTD it is called 'valid'

XML DTD – Example

DTD Example¹

```
<!DOCTYPE note
[
<!ELEMENT note (to,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

- !DOCTYPE note: root element of the document is note
- !ELEMENT note: note element must contain the elements:
to, body
- !ELEMENT to [body]: element to be of type #PCDATA
 - PCDATA = parseable character data

¹https://www.w3schools.com/xml/xml_dtd.asp, 2019-07-25

XML Schema Definition – XSD

- Alternative/extension to DTD
- Much more common
- Uses XML syntax
 - More flexible
 - More verbose
- Major differences:
 - XSD allows for datatype definitions
 - XSD allows for namespaces
 - XSD allows to define a number and order of child items

XSD – Example

XSD – Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name = "uniType">
        <xsd:sequence>
            <xsd:element ref="student" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="stuType">
        <xsd:sequence>
            <xsd:element ref="name"/>
            <xsd:element ref="year"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="university" type="uniType"/>
    <xsd:element name="student" type="stuType"/>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="year" type="xsd:integer"/>
</xsd:schema>
```

DING

XSD – SimpleType

SimpleType Example – Enumeration

```
<xs:simpleType name="color" final="restriction">
  >
    <xs:restriction base="xs:string">
      <xs:enumeration value="green" />
      <xs:enumeration value="red" />
      <xs:enumeration value="blue" />
    </xs:restriction>
</xs:simpleType>
```

SimpleType Example – Range Constraints

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Allows to put additional constraints on a 'primitive' type
- Can be used to for an enumeration as well

XSD – Analysis

- 'Importing' schemas which define data types
 - In addition to public schemas you can also reference your own
 - Be careful to always provide the namespace ('xsd:')!
- Definition of composite (complex) data types
 - Defining sequences (vs. single occurrence elements)
 - minOccurs & maxOccurs
- Referencing elements
 - to avoid multiple declaration of same type
- Definition of elements with data type

XSD – Validation

- Libraries and several tools² allow validation of a XML document against a XSD
- This can help to ensure that e.g. an API understands the input
- Checks not only for correct data types, but also number of items (e.g. in a collection)
- When using XML it is almost always a good idea to create a XSD as well
- Referenced schemas have to be provided to potential API users as well

²e.g. <https://www.freeformatter.com/xml-validator-xsd.html>, 2019-08-19

XSD – Validation

Valid

```
<?xml version="1.0" encoding="UTF-8"?>
<university>
  <student>
    <name>
      John Nielson
    </name>
    <year>
      2000
    </year>
  </student>
</university>
```

Invalid

```
<?xml version="1.0" encoding="UTF-8"?>
<university>
  <student>
    <name>
      John Nielson
    </name>
    <year>
      2000ad
    </year>
  </student>
</university>
```

- Cvc-datatype-valid.1.2.1: '2000ad' Is Not A Valid Value For 'integer'., Line '8', Column '16'.

XPath

- Allows to select nodes (elements/attributes) in a XML document
- Not really a query language (cf. XQuery)
- Very useful when extracting data from XML without converting it to a (programming language) object model first
- Structure:
 - Defining a 'path' through the document from root to the relevant node(s) with '/'
 - Also allows for simple conditionals
 - Some functions for getting first/last element or the count
- There are XPath tools and every major language has a library available

Sample Data

XML Sample Data

```
<books>
  <book id="1" category="cs">
    <title>The Art of Computer Programming</title>
    <author>D. Knuth</author>
    <price>40</price>
    <year>1968</year>
  </book>
  <book id="2" category="science">
    <title>What If</title>
    <author>R. Munroe</author>
    <price>9</price>
    <year>2014</year>
  </book>
  <book id="3" category="science">
    <title>We have no idea</title>
    <author>J. Cham</author>
    <price>12</price>
    <year>2017</year>
  </book>
</books>
```

DING

XPath Example

- Selecting all authors
- XPath = `//author`
 - or: `/books/book/author`
 - `//` is a shorthand for 'whatever comes before'

Result

```
Element='<author>D. Knuth</author>'  
Element='<author>R. Munroe</author>'  
Element='<author>J. Cham</author>'
```

XPath Syntax Overview

<foo baz="5"><bar>25</bar></foo>	
Command	Meaning
/foo/bar	Select all bar's of foo (bar's with foo as parent)
//bar	Select all bar's (in the whole document)
/foo[bar>20]	Sub-Element condition for foo filtering
/foo[@baz<10]	Attribute condition for foo filtering

- Combinations possible, e.g. /foo[@baz<10]/bar

XPath Example

- Selecting all books from the science genre
- XPath = //book[@category='science']

Result

```
Element='<book category="science" id="2">
<title>What If</title>
<author>R. Munroe</author>
<price>9</price>
<year>2014</year>
</book>'
Element='<book category="science" id="3">
<title>We have no idea</title>
<author>J. Cham</author>
<price>12</price>
<year>2017</year>
</book>'
```

XPath Example

- Selecting all books with a price higher than 20
- XPath = //book[price>20]

Result

```
Element='<book category="cs" id="1">
<title>The Art of Computer Programming</title>
<author>D. Knuth</author>
<price>40</price>
<year>1968</year>
</book>'
```

XPath Example

- Selecting second to last book in books
- XPath = //book[last()-1]

Result

```
Element='<book category="science" id="2">
<title>What If</title>
<author>R. Munroe</author>
<price>9</price>
<year>2014</year>
</book>'
```

XQuery

- Superset of XPath (based on XPath expressions)
- Is for XML what SQL is for databases ⇒ query language
- Very flexible and powerful
- Supported by all major databases (think XML column)

Example

```
for $x in doc("books.xml")/books/book  
where $x/price>30  
order by $x/title  
return $x/title
```

XQuery – FLWOR

- While most XPath expressions are also valid XQuery there is one much more powerful feature
- A FLWOR expression (pronounced 'flower')
 - For
 - Let
 - Where
 - Order
 - Return
- Very similar to SQL (see previous example)
- To open a data source you can use the doc(foo.xml) command
 - e.g. doc(foo.xml)//bar

XQuery Syntax Overview

Clause	Meaning
for \$x in [doc() XPath expr.]	Define data source and 'row' variable
let \$n := <a value>	Assigns a value (values) to a variable
where \$x <condition>	Filters 'rows' fulfilling the condition
order by \$x [descending]	Orders the remaining 'rows'
return \$x[/foo /@bar]	Returns the 'row' or a subset of it

XQuery – Tool

- There are fewer good online tools for XQuery than for XPath
- Thus I recommend using BaseX
 - Free & Open Source
 - Download at <http://baseX.org/download/> (2019-08-25)
- BaseX Hints:
 - Attention: you have to create a database from the XML before you can start to execute queries!
 - When creating the database select 'strip namespaces' in the 'Parsing' tab
 - Once this is done you *don't* have to use the `doc(foo.xml)` command

XQuery – Tool

- If BaseX is setup correctly, the XML file is loaded and XQuery works it should look like this

The screenshot shows the BaseX 9.2.4 XQuery tool interface. The top menu bar includes Database, Editor, View, Visualization, Options, and Help. The main window has tabs for XQuery and xQuery... The left sidebar shows a file tree with files like recipes.xml, solution.txt, and xml_exercise_03.pdf. The central XQuery editor pane contains the following code:

```

File* [03_XPath] - BaseX 9.2.4
Database Editor View Visualization Options Help
XQuery xQuery...
File* 
1 for $x in //title
2 where starts-with($x, 'Beef')
3 return $x

```

The right pane displays the results of the query execution. It shows a table-like structure with columns for recipe, filling, dough, etc., and rows for various food items like baked chicken, sauteed marinated chicken, etc.

recipe	filling	dough	etc.	pastry
o.. g.. ol..	nic.. w.. fl.. wh..	et.. m..	c.. fl.. s.. i..	preparat..
be..	ea.. b.. s..	e..	filling	
g.. Ita.. s.. s..	ez..	S.. s..	baked chicken	sauteed ..
ang.. pr.. c..	v.. s.. p.. e.. v.. p..	Al.. f..	marinated ch..	w.. b.. m m
min.. s.. s.. n..	m.. prepara.. nutr..	le..	s.. d o.. p..	s..
but.. s.. s..	prepa..	stock	H.. m..	dry..
t.. sauce	s.. s..	sauce	c.. b.. w.. s..	c.. s.. bu..
L.. o.. mi.. cr..	pr..	o.. c.. b.. w.. s..	m..	m.. pr..
dried thyme	s.. s.. g.. s..	s.. s..	s.. pre..	
dr.. c.. wh..	nu..	c.. nutri..	pa.. e..	preparat.. nutrit..

The bottom Result pane shows the output of the query: <title>Beef Parmesan with Garlic Angel Hair Pasta</title>

XQuery – Example

Example

```
for $b in //book
where $b/@id = 2 or $b/year > 2015
return $b/title
```

-- Result

```
<title>What If</title>
<title>We have no idea</title>
```

- Complex conditionals possible
- The basis is still a XPath expression

XQuery – Example

Example

```
for $b in //book
where fn:tokenize($b/[author])[2] = 'Knuth'
return ($b/price, $b/year)
```

-- Result

```
<price>40</price>
<year>1968</year>
```

- Huge function libraries available
- Multiple values can be returned