

Storing and Retrieving Data

Lecture 1

Introduction and Normalization



Lecturer: Mijail Naranjo-Zolotov
Email: mijail.naranjo@novaims.unl.pt

Overview

Lecturer: Mijail Naranjo-Zolotov

Email: mijail.naranjo@novaims.unl.pt

Office: 138

Office Hours: Thursday from 16:00 to 17:30 (schedule in advance by e-mail)

Overview – Learning units

1. Introduction and Normal forms in relational database.
2. Architecture of a DMBS.
3. Intoduction to SQL. CRUD operations.
4. SQL queries (aggregation, sorting)
5. SQL Joins. SQL views. SQL Triggers
6. Advanced topics: MySQL query optimization.
7. SQL vs NoSQL databases. CAP theorem. Wrap-up;

Regular examination period (1st epoch)

- Group project (50%).
- Final exam (50%).

Resit/improvement examination period (2nd epoch)

- Group project (50%)
- Final exam (50%).

Rules:

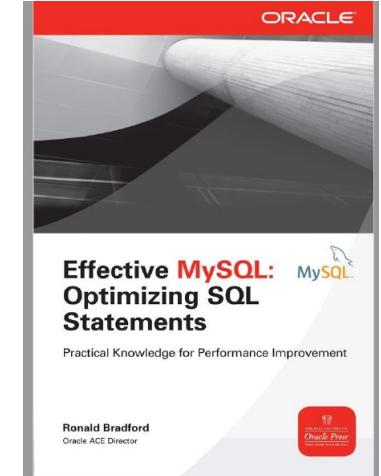
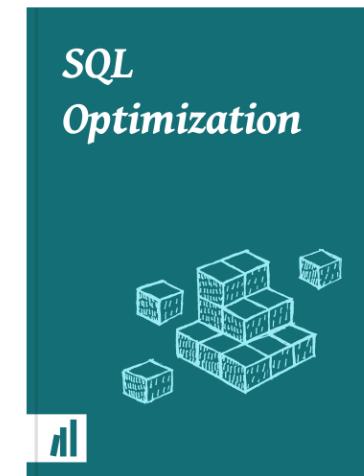
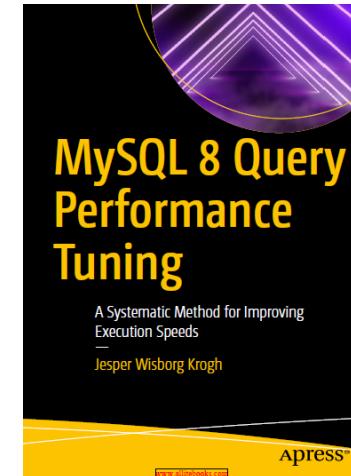
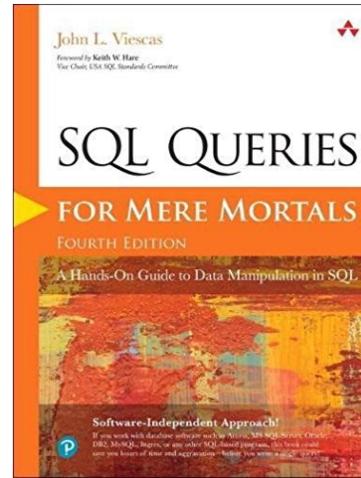
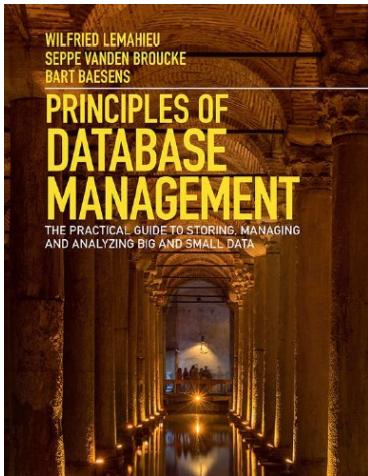
- The score in the exam should at least 9.5 (out of 20).
- The teams are made of 4 students.
- Late deliveries for the project will be penalized with 1 point for each late day up to 5 points.

Choose your team in moodle



Image sources: <https://medium.com/magenta-lifestyle/why-two-large-pizza-team-is-the-best-team-ever-4f19b0f5f719>

Overview - Bibliography



<https://dataschool.com/sql-optimization/>

Lecture 1: Introduction to Databases

What is database?

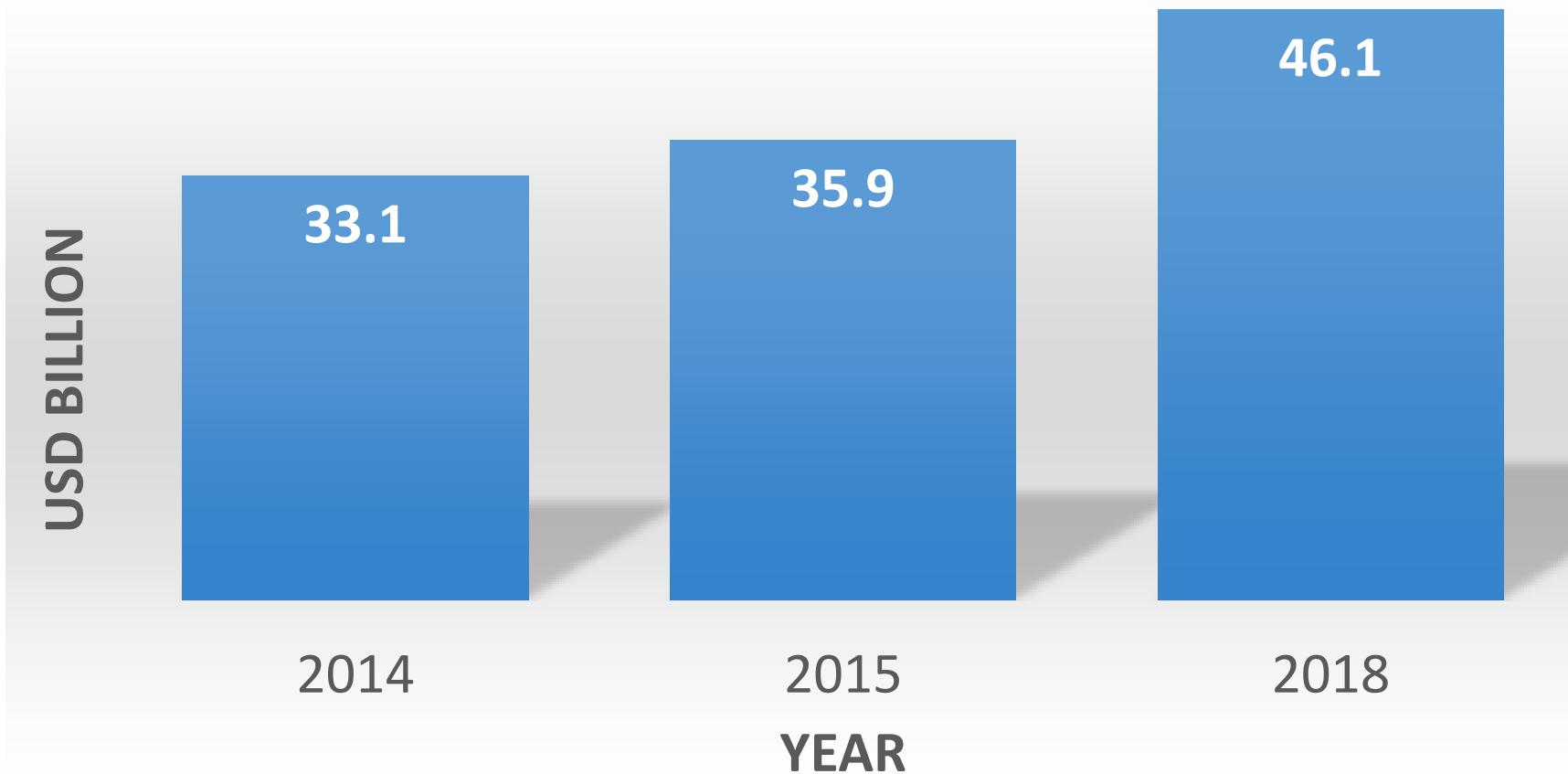


What is database?

- A **database** can be defined as a collection of related data items within a specific business process or problem setting.
- A **database management system (DBMS)** is the software package used to define, create, use, and maintain a database.
- The combination of a DBMS and a database is then often called a **database system**.

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

DBMS market value



Note: \$10.4 billion in 2018 are DBMS cloud services

Source 1: <https://blogs.gartner.com/merv-adrian/2016/04/12/dbms-2015-numberspaint-a-picture-of-slow-but-steady-change/>.

Source 2: <https://blogs.gartner.com/adam-ronthal/2019/06/23/future-database-management-systems-cloud/>

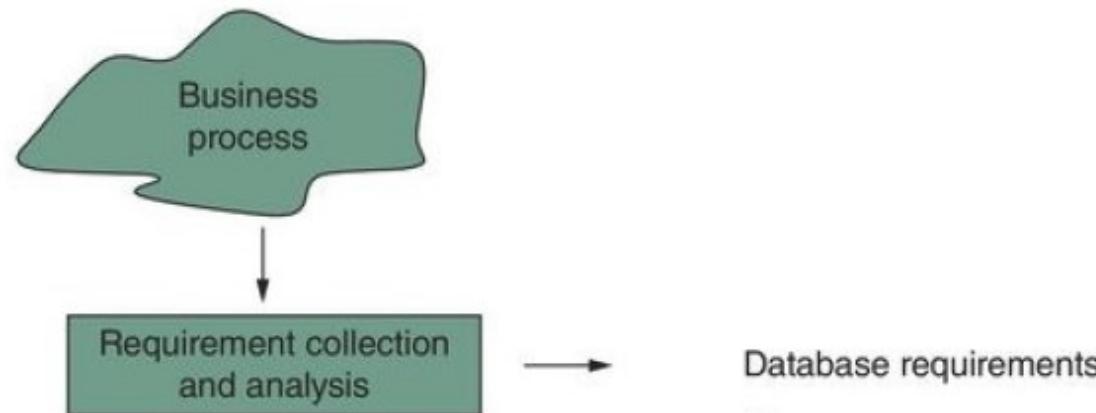
Database conceptual modelling

What is a model?



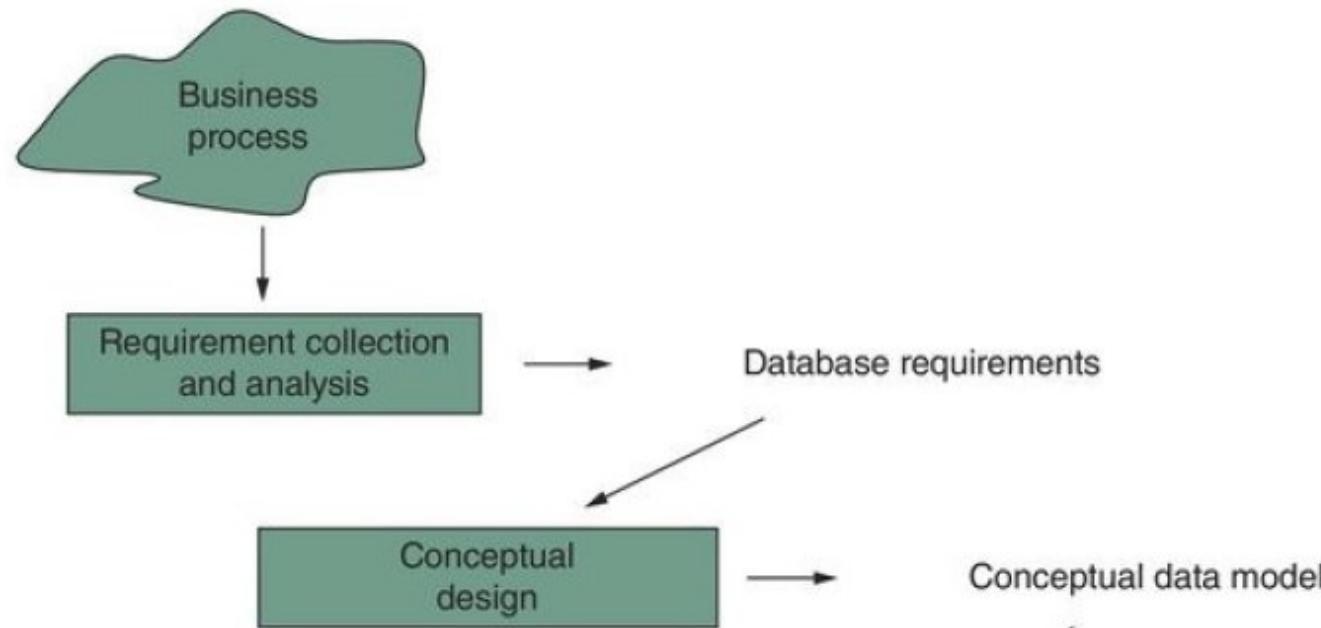
Image Source: https://www.stpetertravel.com/en/viaggi_tour/7072/coliseum-architecture-coliseum-roman-e-imperial-forum-rome-tours.html

The database design process



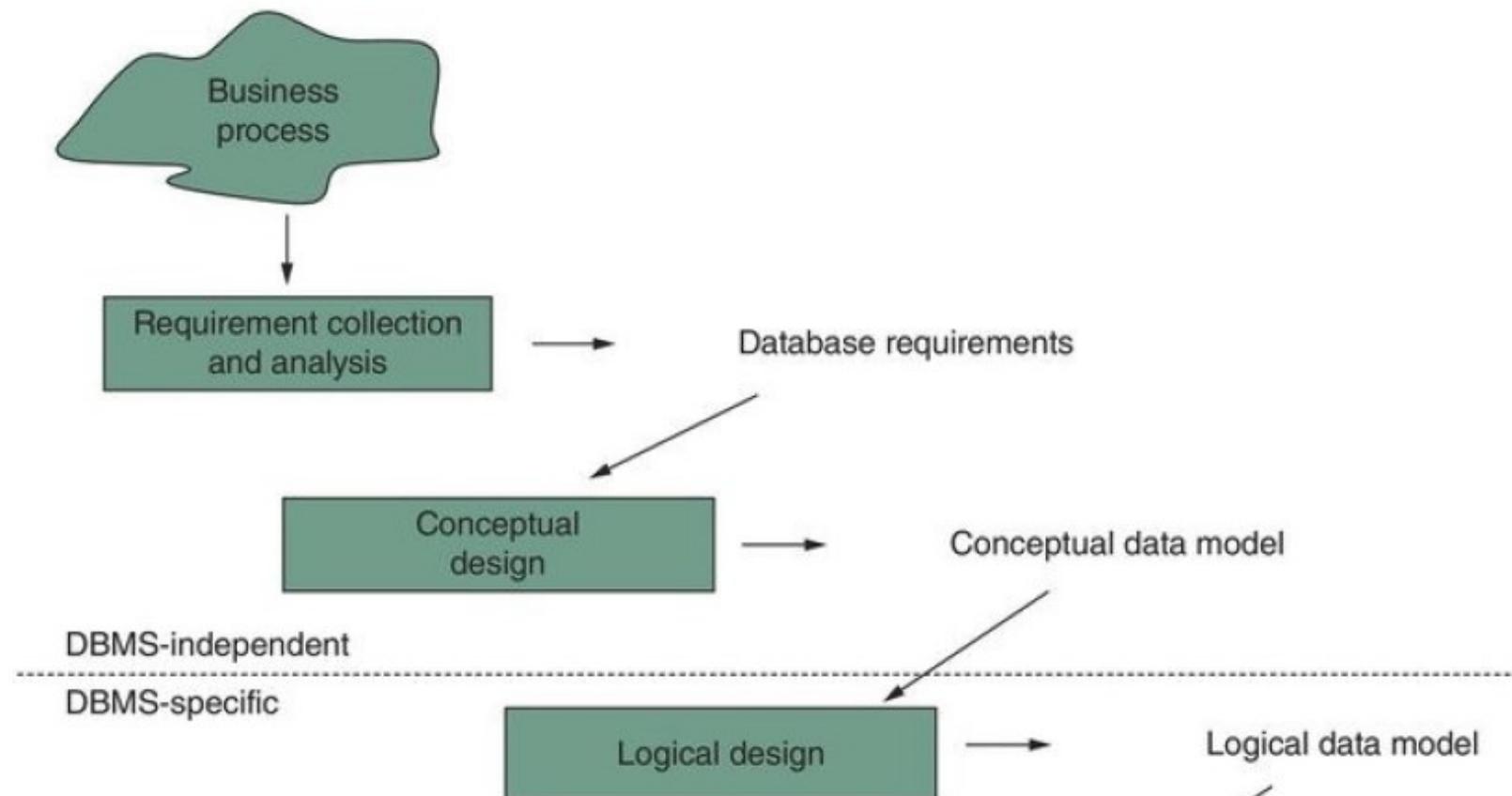
The aim is to understand the different steps and data needs of the process. Techniques: interviews, surveys, inspections of documents, etc

The database design process



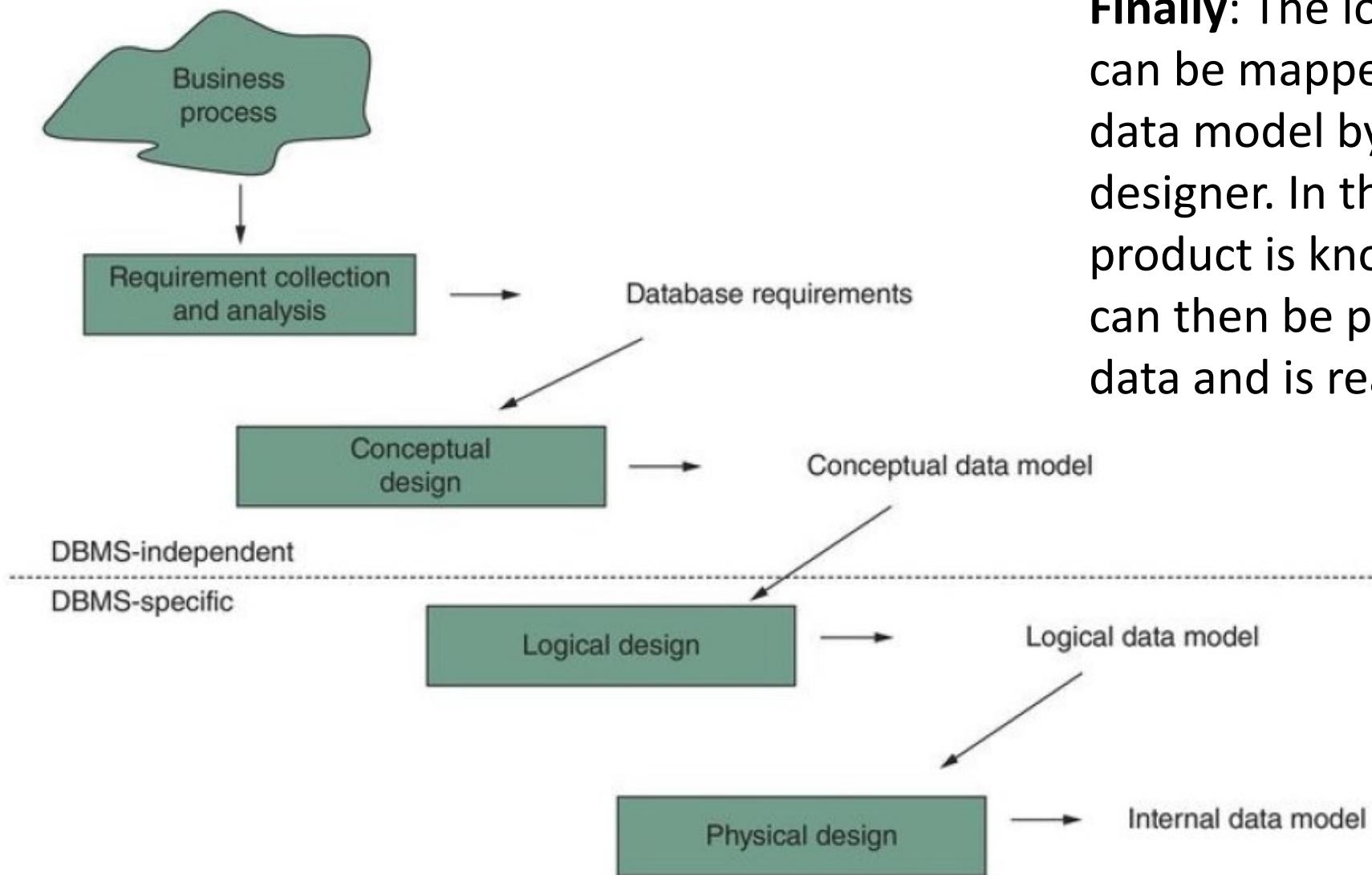
The information architect and the business user formalize the requirements in a **conceptual data model**. This is a high-level model, easy to understand for the business user and formal enough for the database designer who will use it in the next step.

The database design process



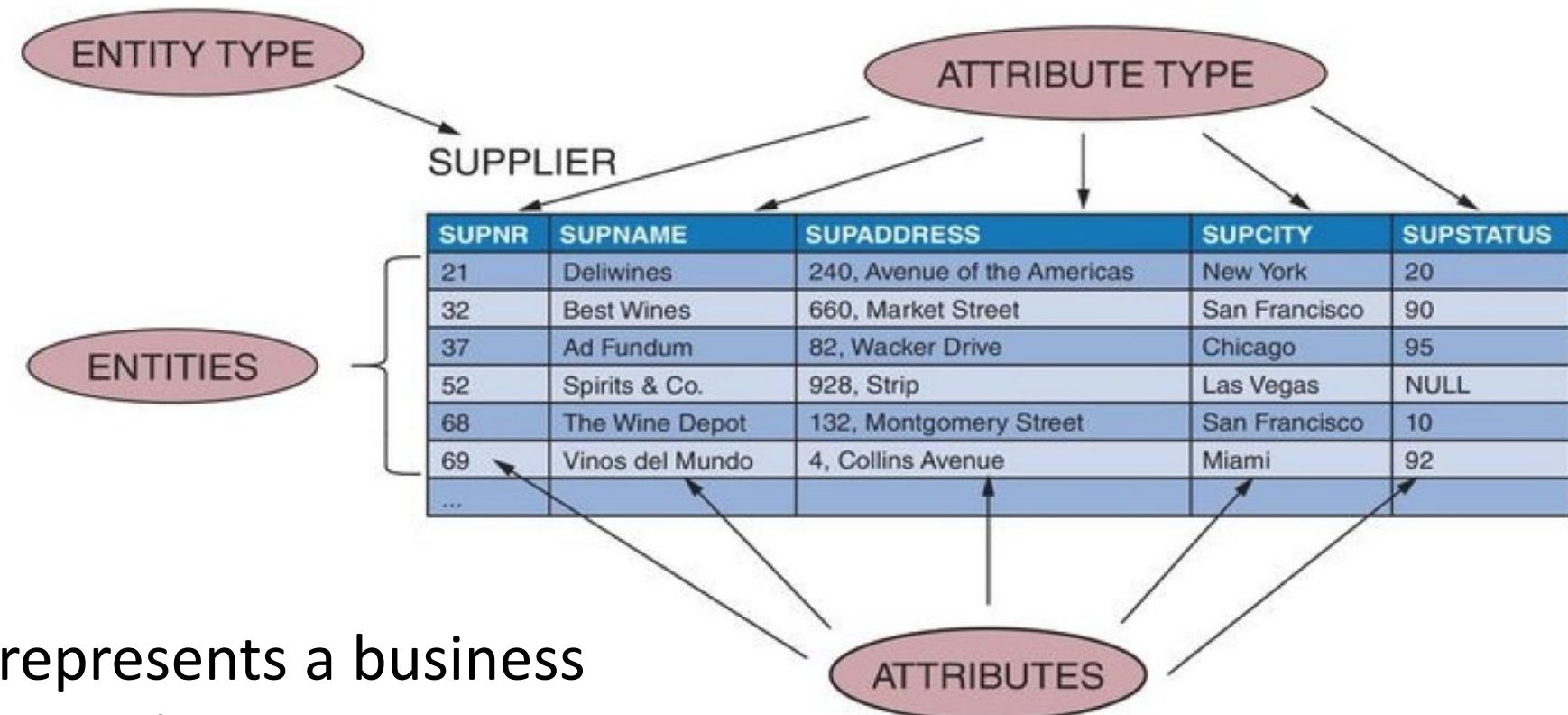
The logical data model is based upon the implementation environment. At this stage it is already known what type of DBMS (e.g., RDBMS, OODBMS, etc.) will be used, the product itself (e.g., Microsoft, IBM, Oracle) has not been decided yet.

The database design process



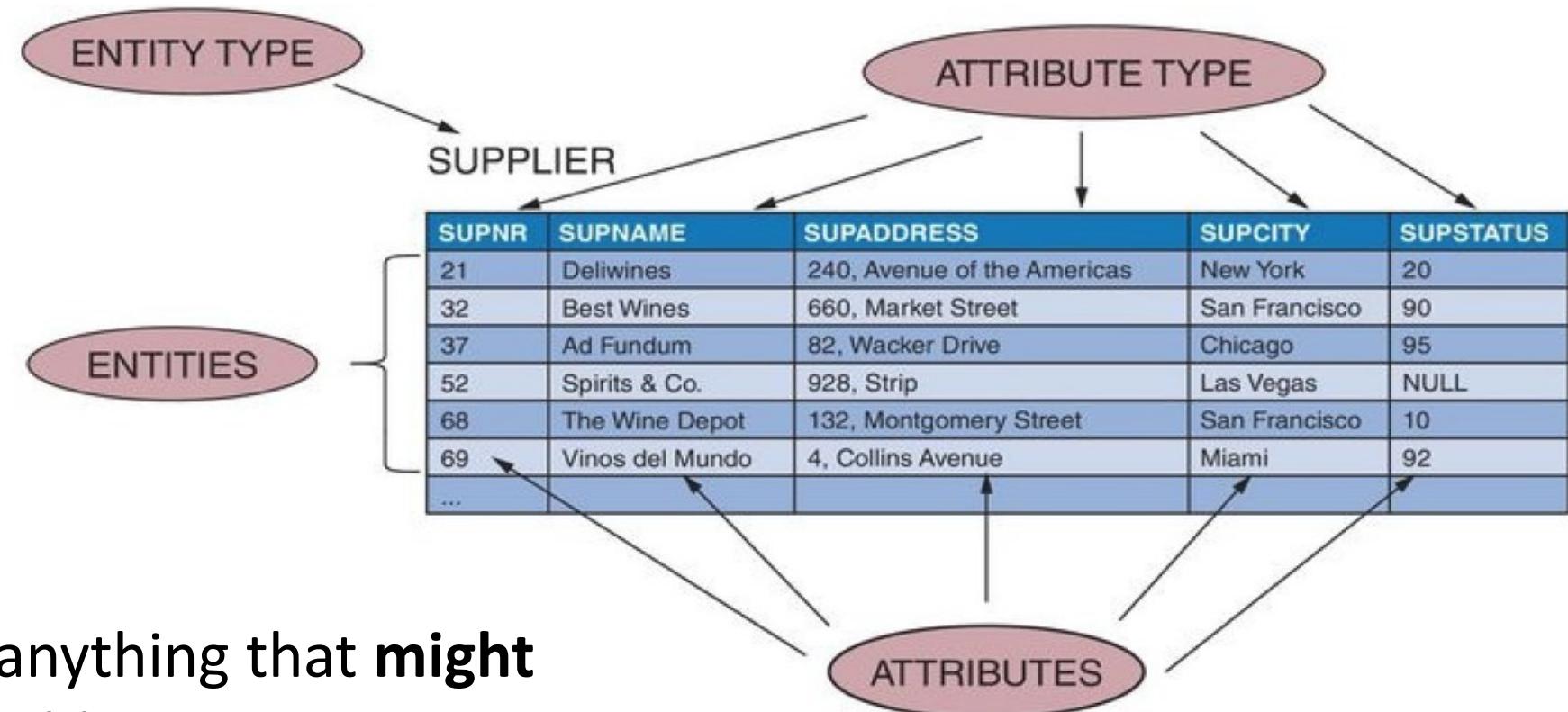
Finally: The logical data model can be mapped to an internal data model by the database designer. In this step, the DBMS product is known. The database can then be populated with data and is ready for use.

The entity relationship model



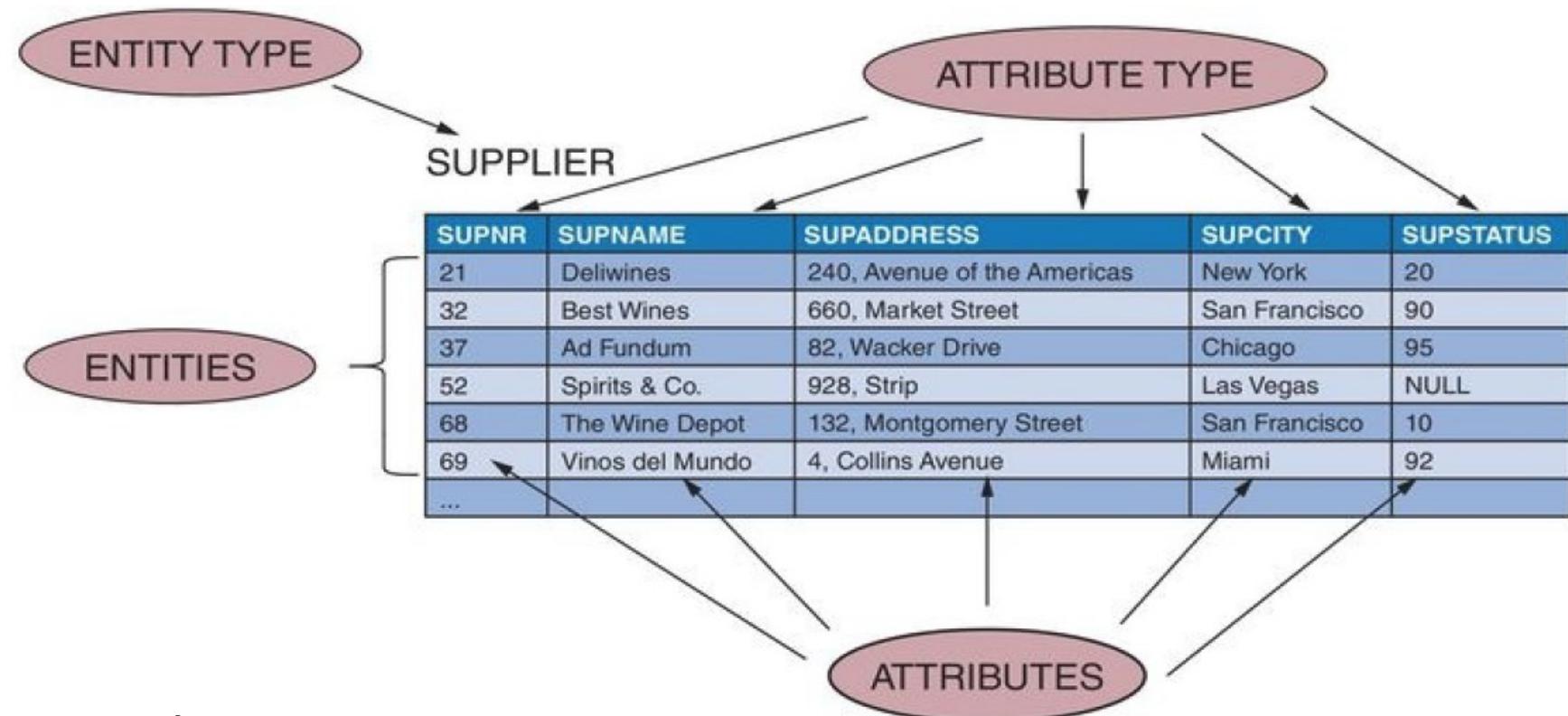
An **ENTITY TYPE** represents a business concept with an unambiguous meaning to a particular set of users.
Examples?

The entity relationship model



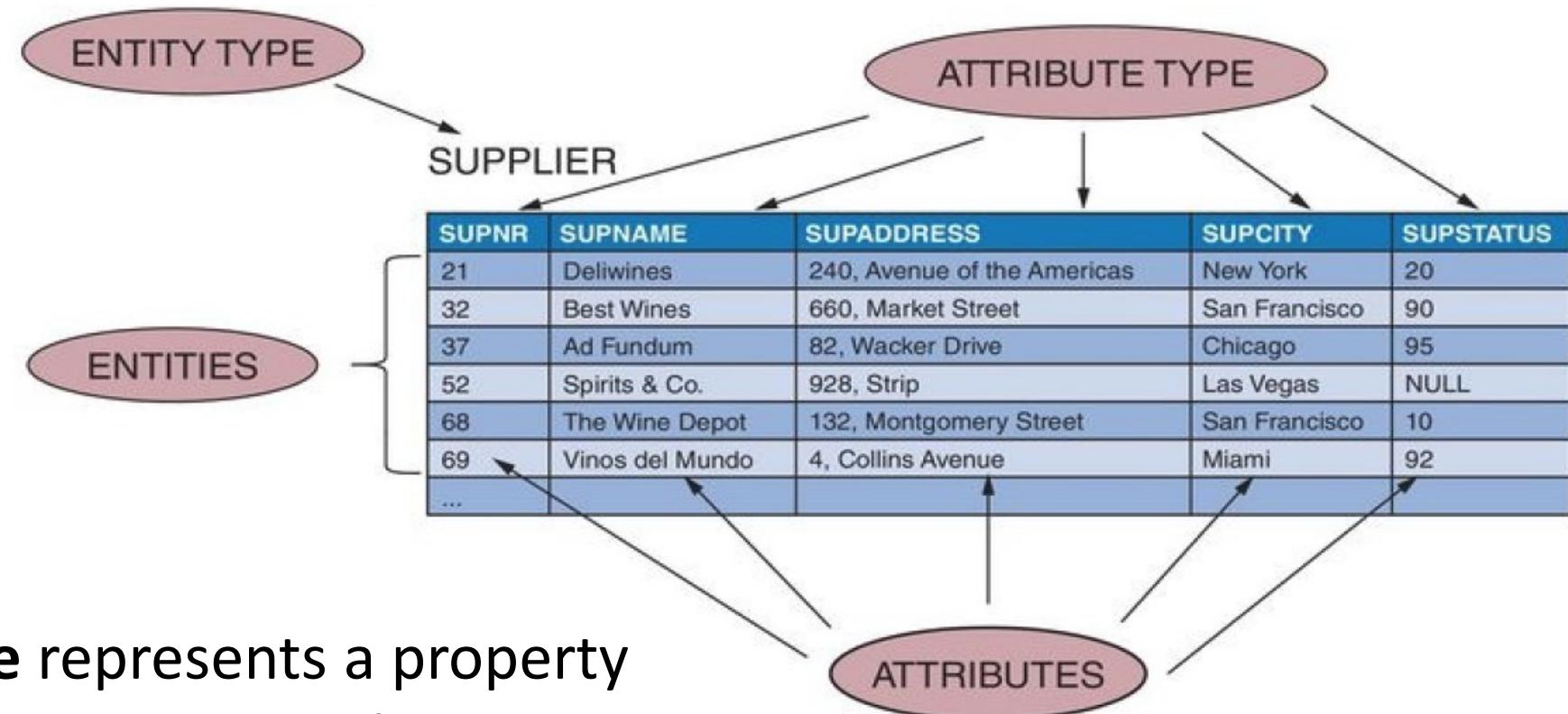
“ENTITY TYPE is anything that **might** deserve its own table in your database model”. (Tekstenuitleg.net)

The entity relationship model



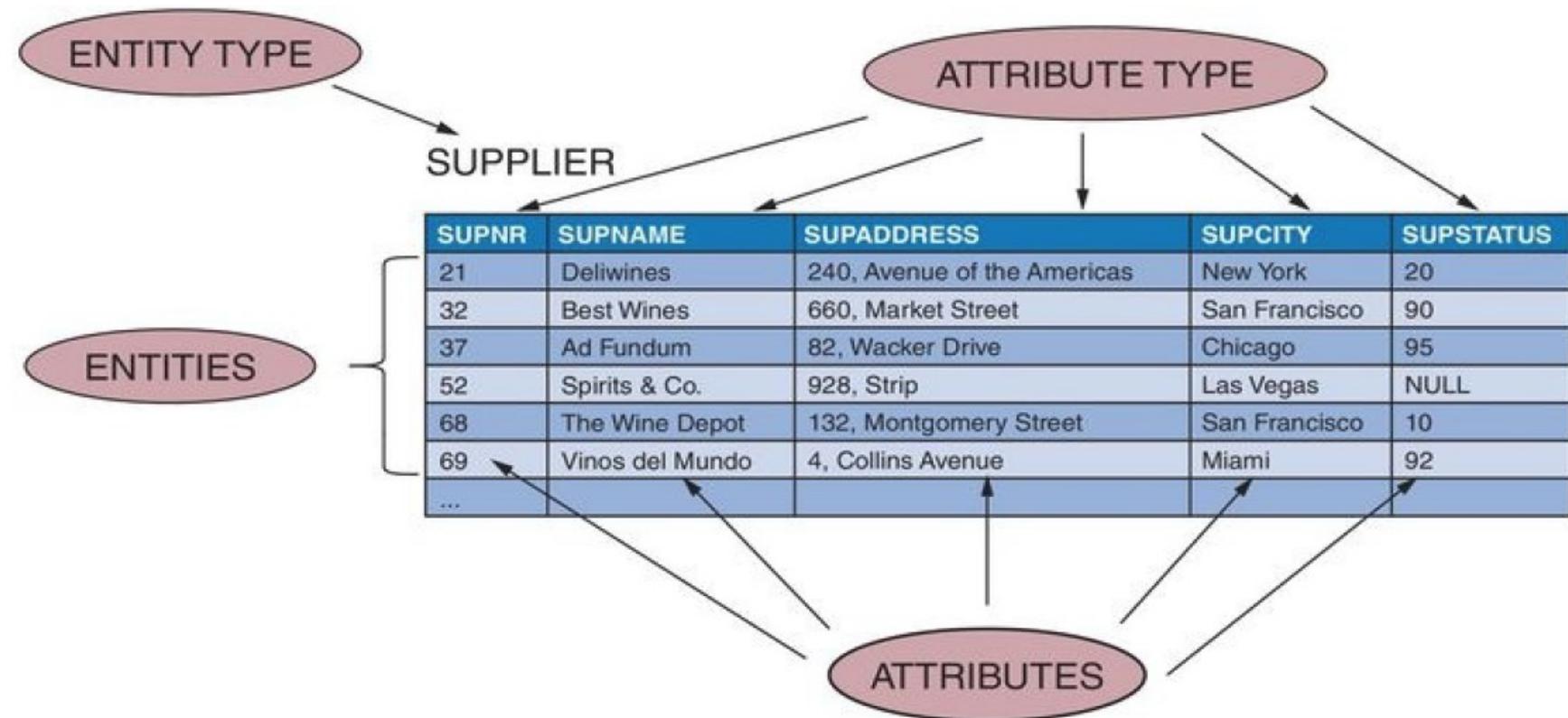
An **entity** is one particular occurrence
or instance of an entity type

The entity relationship model



An **attribute type** represents a property of an entity type. As an example, name and address are attribute types of the entity type supplier.

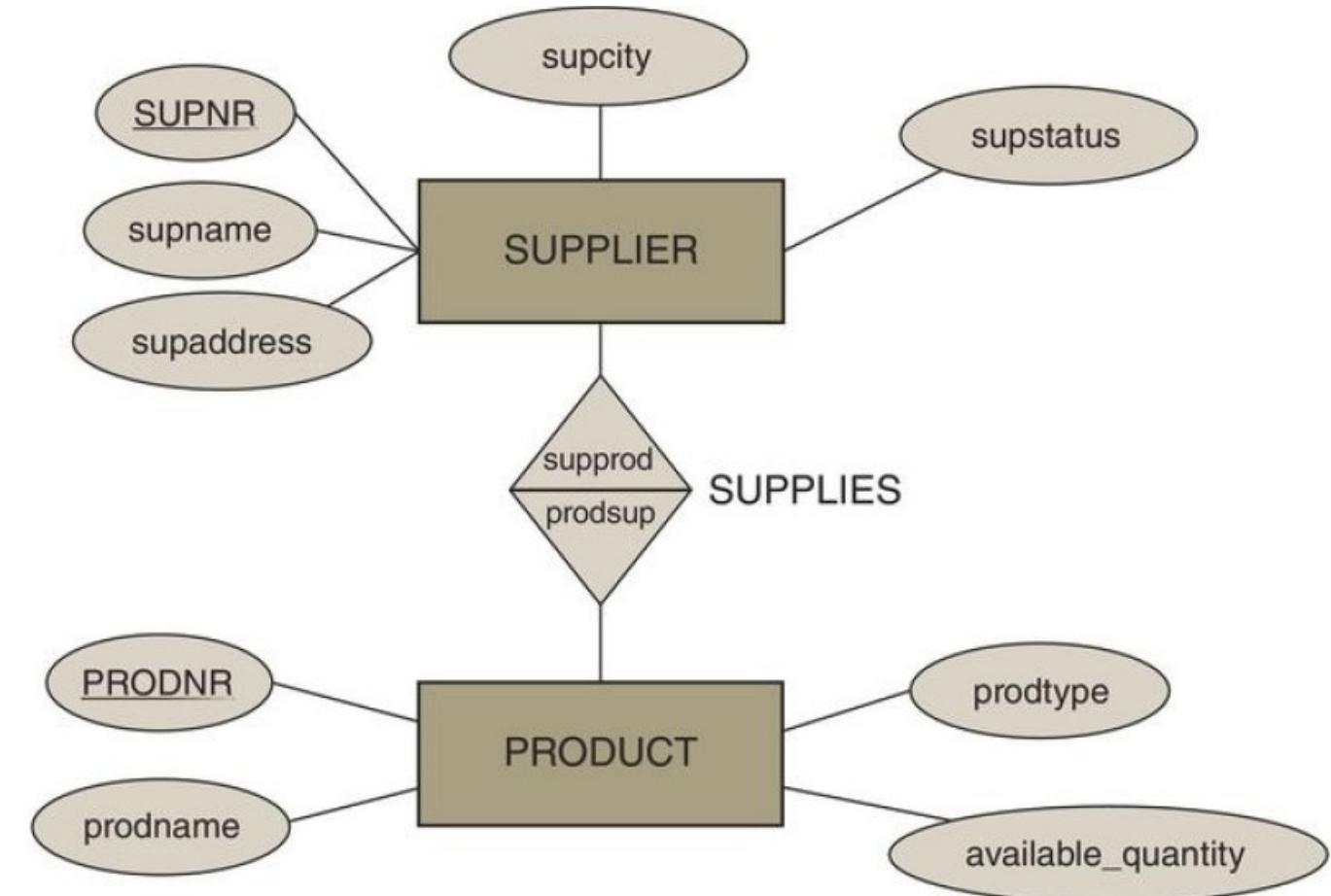
The entity relationship model



An **attribute** is an instance of an attribute type

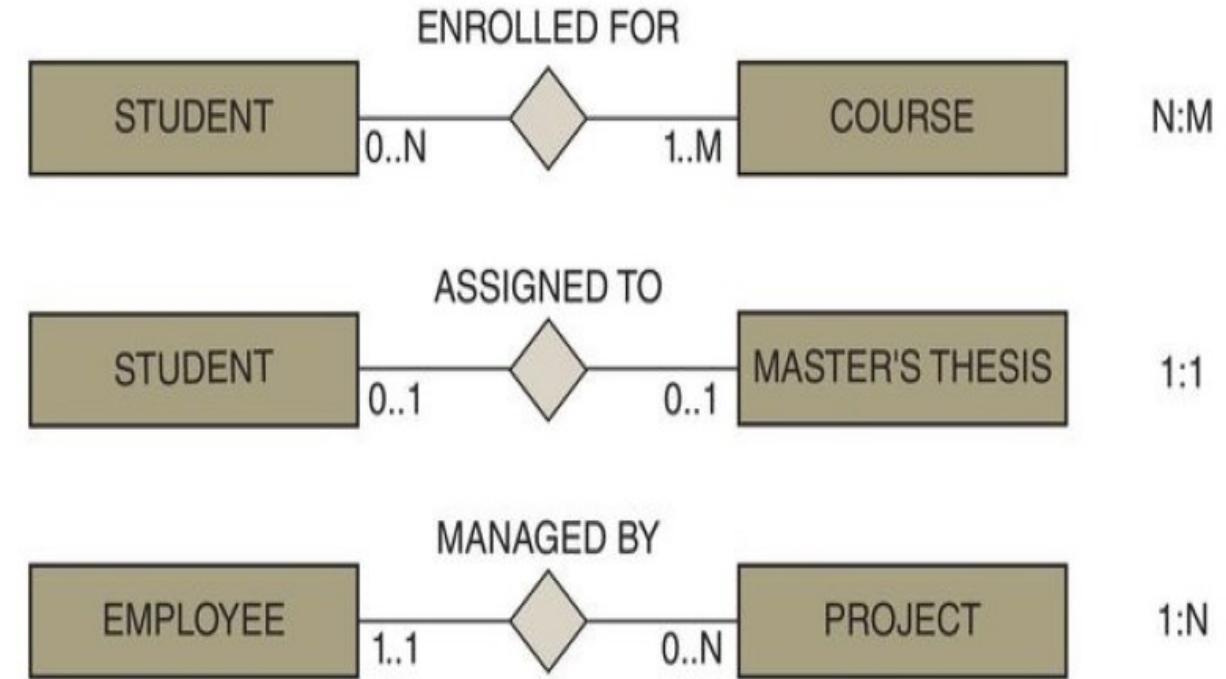
Relationship

A **relationship** represents an association between two or more entities. A **relationship type** then defines a set of relationships among instances of one, two, or more entity types.



Cardinalities

Every relationship type can be characterized in terms of its cardinalities, which specify the minimum or maximum number of relationship instances that an individual entity can participate in.



Normalization

Data Normalization (3-Normal Forms)

Normalization of a relational model is a process of analyzing the given relations to ensure they do not contain any redundant data. The goal of normalization is to ensure that no anomalies can occur during data insertion, deletion, or update. A step-by-step procedure needs to be followed to transform an unnormalized relational model to a normalized relational model.

Relational Databases - Data Normalization

Data Normalization (3-Normal Forms)

Think about the normalization forms as filters. The more filters you apply, the better your DB.

BUT..... The more normal forms, the more complex your database structure.

3 Normal Forms (NF):

1. The **1NF** states that every attribute type must be atomic and single-valued.
Hence, no composite or multi-valued attribute types are tolerated.
2. An entity type is in **2NF** when it is in **1NF** and when all of its non-key attributes are fully functionally dependent on its **primary key**.
3. An entity type is in **3NF** when it is in **2NF** and no non-key attribute is **transitively dependent** on the **primary key**.

Relational Databases - First Normal Form (1NF)

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11

Relational Databases - First Normal Form

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11

Does this table comply with 1NF?

Relational Databases - First Normal Form

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11

Does this table comply with 1NF?

NO

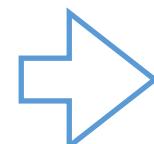
What is the solution?

Relational Databases - First Normal Form

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11

Is this solution enough?



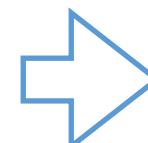
Product ID	Color1	Color2	Price
1	brown	yellow	\$15
2	red	green	\$13
3	blue	orange	\$11

Relational Databases - First Normal Form

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11

Is this solution enough?
NO

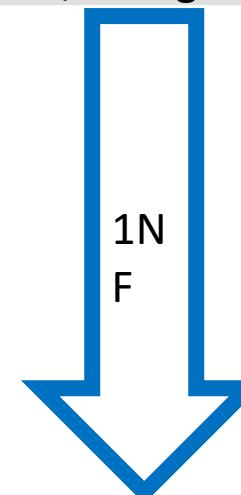


Product ID	Color1	Color2	Price
1	brown	yellow	\$15
2	red	green	\$13
3	blue	orange	\$11

Relational Databases - First Normal Form

An entity type is in 1NF when it contains no repeating groups of data
“each cell in the table can have only one value, never a list of values”

Product ID	Color	Price
1	brown, yellow	\$15
2	red, green	\$13
3	blue, orange	\$11



Product ID	Price
1	\$15
2	\$13
3	\$11

ColorID	ProductID	Color
9001	1	brown
9002	2	red
9003	3	blue
9004	1	yellow
9005	2	green
9006	3	orange

Example – 1NF

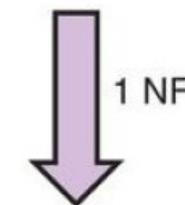
Normalize the following table that contains information about department location:

DNUMBER	DLOCATION	DMGRSSN
15	{New York, San Francisco}	110
20	Chicago	150
30	{Chicago, Boston}	100

Example – 1NF

Solution:

DNUMBER	DLOCATION	DMGRSSN
15	{New York, San Francisco}	110
20	Chicago	150
30	{Chicago, Boston}	100



DEPARTMENT

DNUMBER	DMGRSSN
15	110
20	150
30	100

DEP-LOCATION

DNUMBER	DLOCATION
15	New York
15	San Francisco
20	Chicago
30	Chicago
30	Boston

Relational Databases - Second Normal Form (2NF)

An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.

“features should be fully dependent on the entire primary key”

Primary Keys

OrderNumber	ProductID	ProductName
1	232	Pespsi
2	234	Coca-Cola
3	241	Polar

Does it comply with the 2NF?

Relational Databases - Second Normal Form

An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.

“features should be fully dependent on the entire primary key”

Primary Keys

OrderNumber	ProductID	ProductName
1	232	Pespsi
2	234	Coca-Cola
3	241	Polar

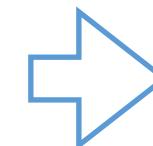
Fails because Product Name Depends on Product ID but not on Order Number.
Solution?

Relational Databases - Second Normal Form

An entity type is in 2NF when it is in 1NF and when all of its non-key attributes are fully dependent on its primary key.

“features should be fully dependent on the entire primary key”

OrderNumber	ProductID	ProductName
1	232	Pespsi
2	234	Coca-Cola
3	241	Polar



OrderNumber	ProductID	ProductName
1	232	
2		
3	232	Pespsi
	234	Coca-Cola
	241	Polar

Example – 2NF

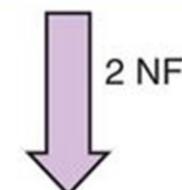
Normalize to the 2NF. The following table contains information about the project and the hours each employee worked on the project:

SSN	PNUMBER	PNAME	HOURS
100	1000	Hadoop	50
220	1200	CRM	200
280	1000	Hadoop	40
300	1500	Java	100
120	1000	Hadoop	120

Example – 2NF

Solution:

SSN	PNUMBER	PNAME	HOURS
100	1000	Hadoop	50
220	1200	CRM	200
280	1000	Hadoop	40
300	1500	Java	100
120	1000	Hadoop	120



PNUMBER	PNAME
1000	Hadoop
1200	CRM
1500	Java

SSN	PNUMBER	HOURS
100	1000	50
220	1200	200
280	1000	40
300	1500	100
120	1000	120

Relational Databases - Third Normal Form

Transitive dependency:

	idBook	name	isbn	edition	status	year	publisher	category	price	author	email_author
▶	1	IT 2, IT 1	1111	1	1	1980	SAGE, O Reilly, Globe	horror, romance	21.00	peter	peter@gmail.com
	2	war and peace; little price	1223	2	1	1850	SAGE, O Reilly, Globe	history, romance	25.00	john	john@gmail.com
*	3	Blindness	2222	1	3	1995	Springer	fiction	35.50	ann	ann@gmail.com
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Author email depends on author, author depends on book_id

Relational Databases - Third Normal Form

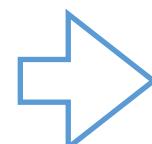
An entity type is in 3NF when it is in 2NF and no non-key attribute is transitively dependent on the primary key.

Tournament	Year	Winner	Winner DoB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Indiana Invitational	1999	Chip Masterson	14 March 1977

Relational Databases - Third Normal Form

An entity type is in 3NF when it is in 2NF and no non-key attribute is transitively dependent on the primary key.

Tournament	Year	Winner	Winner DoB
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Indiana Invitational	1999	Chip Masterson	14 March 1977



Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Indiana Invitational	1999	Chip Masterson

Winner	Winner DoB
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968
Chip Masterson	14 March 1977

Example – 3NF

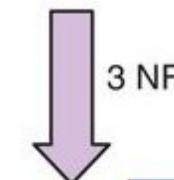
Normalize the following table that contains information about employees and departments:

SSN	NAME	DNUMBER	DNAME	DMGRSSN
10	O'Reilly	10	Marketing	210
22	Donovan	30	Logistics	150
28	Bush	10	Marketing	210
30	Jackson	20	Finance	180
12	Thompson	10	Marketing	210

Example – 3NF

Solution:

SSN	NAME	DNUMBER	DNAME	DMGRSSN
10	O'Reilly	10	Marketing	210
22	Donovan	30	Logistics	150
28	Bush	10	Marketing	210
30	Jackson	20	Finance	180
12	Thompson	10	Marketing	210



SSNR	NAME	DNUMBER
10	O'Reilly	10
22	Donovan	30
28	Bush	10
30	Jackson	20
12	Thompson	10

DNUMBER	DNAME	DMGRSSN
10	Marketing	210
30	Logistics	150
20	Finance	180

How to install MySQL

- **MySQL installation will NOT be done in labs.**
- You can download MySQL community edition from here: <https://dev.mysql.com/downloads/mysql/>

Installation videos MySQL:

- Windows: <https://www.youtube.com/watch?v=u96rVINbAUI>
- Mac: <https://www.youtube.com/watch?v=9sbUsbDWTE8>
- Linux: <https://www.youtube.com/watch?v=ohIn8gMWxYg>

More info about installation:

- The Workbench: <https://dev.mysql.com/doc/workbench/en/wb-installing.html>
- The MySQL server: <https://dev.mysql.com/doc/refman/8.0/en/installing.html>

END OF LECTURE 1

Acreditações e Certificações



UNIGIS



A3ES



Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA

IMS

Information
Management
School

Storing and Retrieving Data

Lecture 2

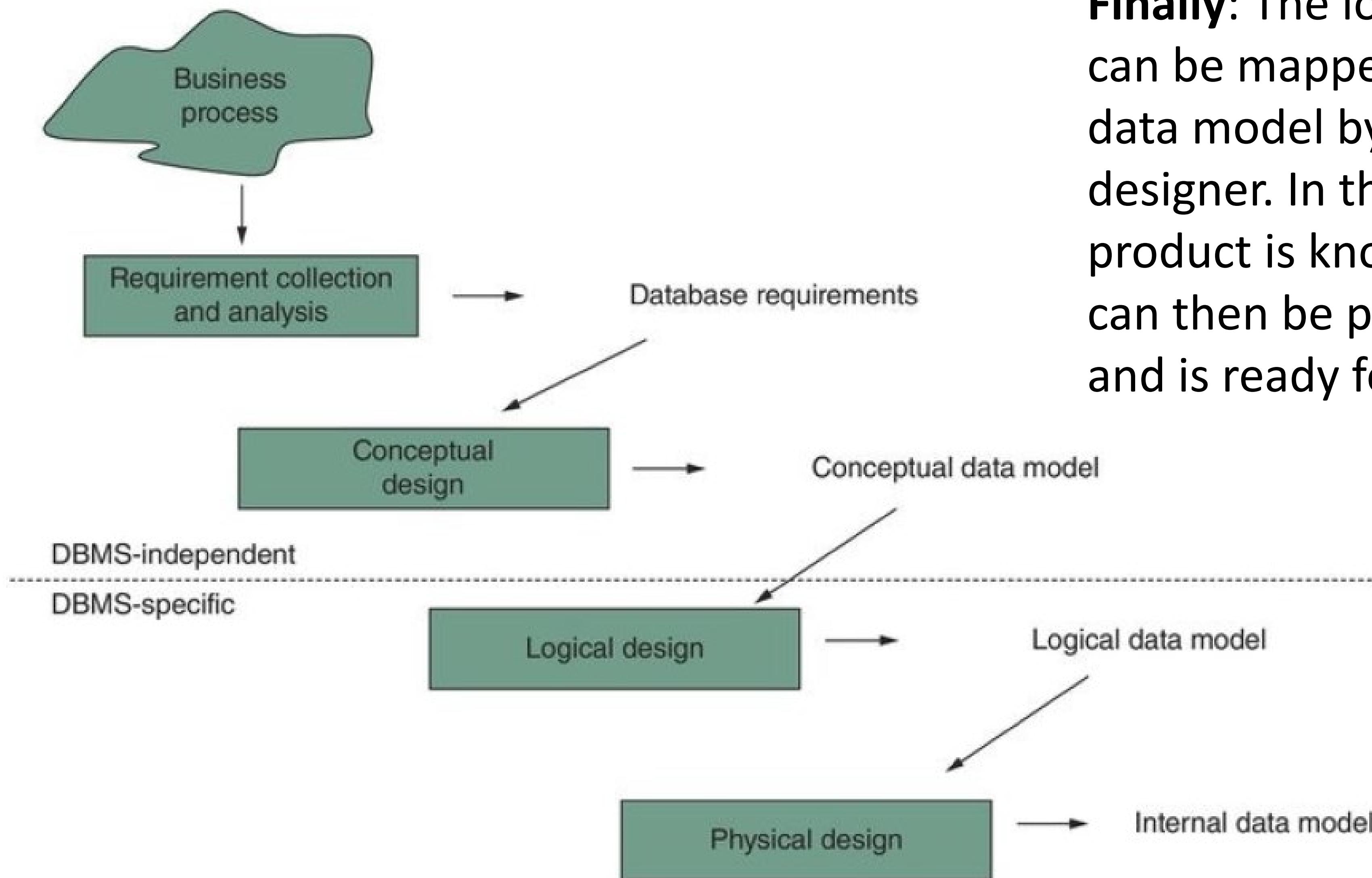
Database modelling

Lecturer: Mijail Naranjo-Zolotov
Email: mijail.naranjo@novaims.unl.pt

Previous class

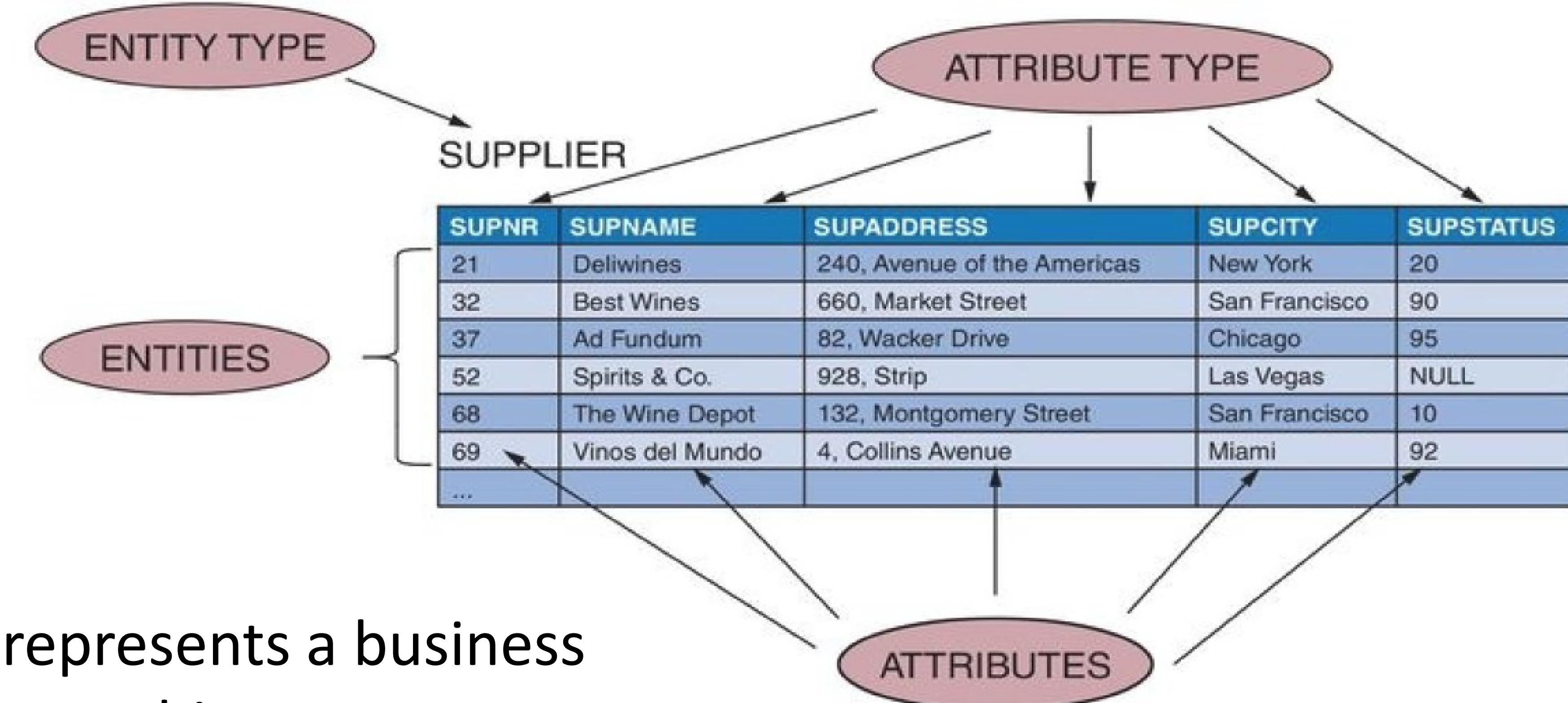


The database design process



Finally: The logical data model can be mapped to an internal data model by the database designer. In this step, the DBMS product is known. The database can then be populated with data and is ready for use.

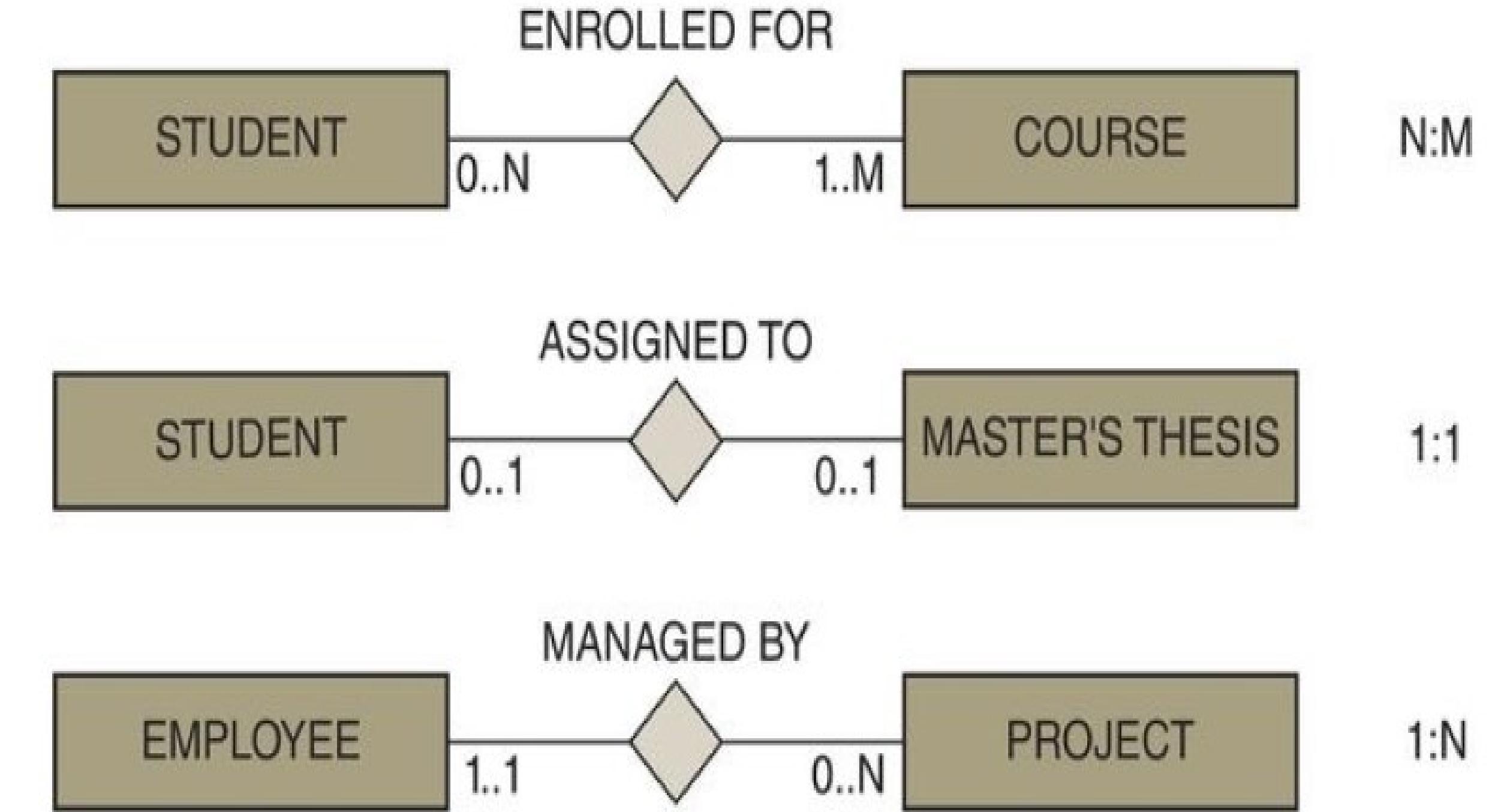
The entity relationship model



An **ENTITY TYPE** represents a business concept with an unambiguous meaning to a particular set of users.
Examples?

Cardinalities

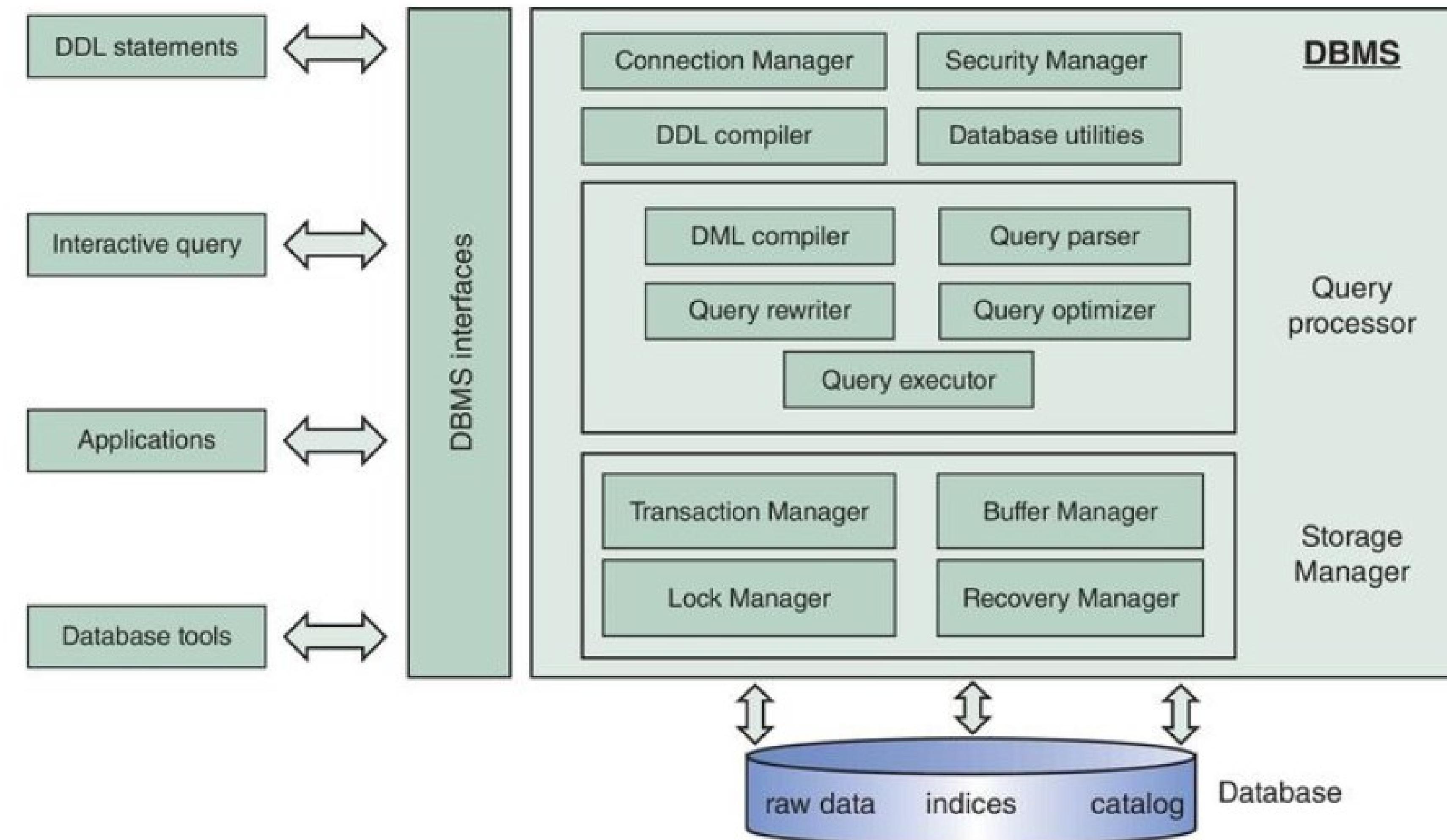
Every relationship type can be characterized in terms of its cardinalities, which specify the minimum or maximum number of relationship instances that an individual entity can participate in.



- DBMS Architecture
- Entity Relationship Diagram (ERD)
- Crow's foot notation

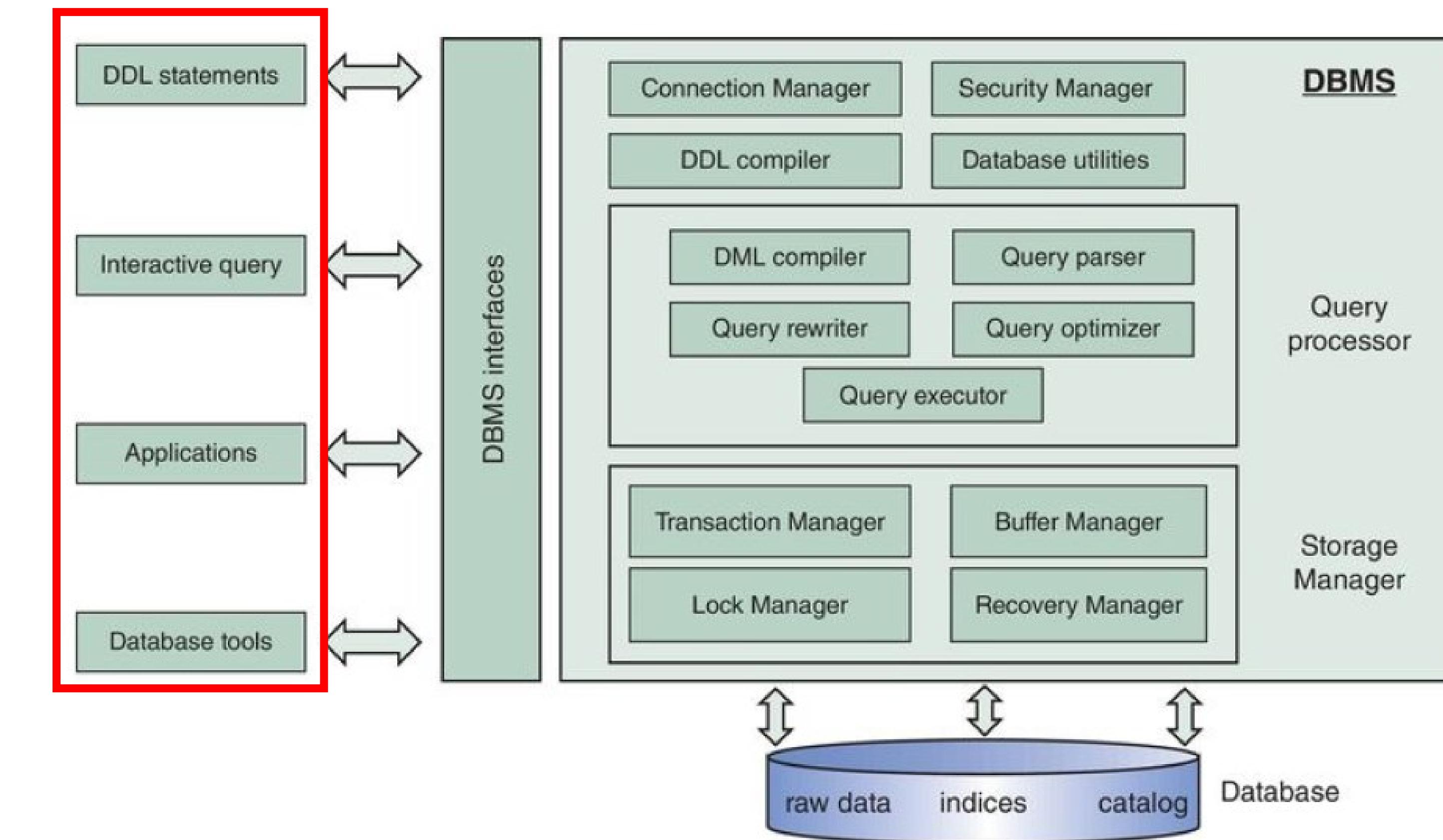
DBMS Architecture

Architecture of a DBMS



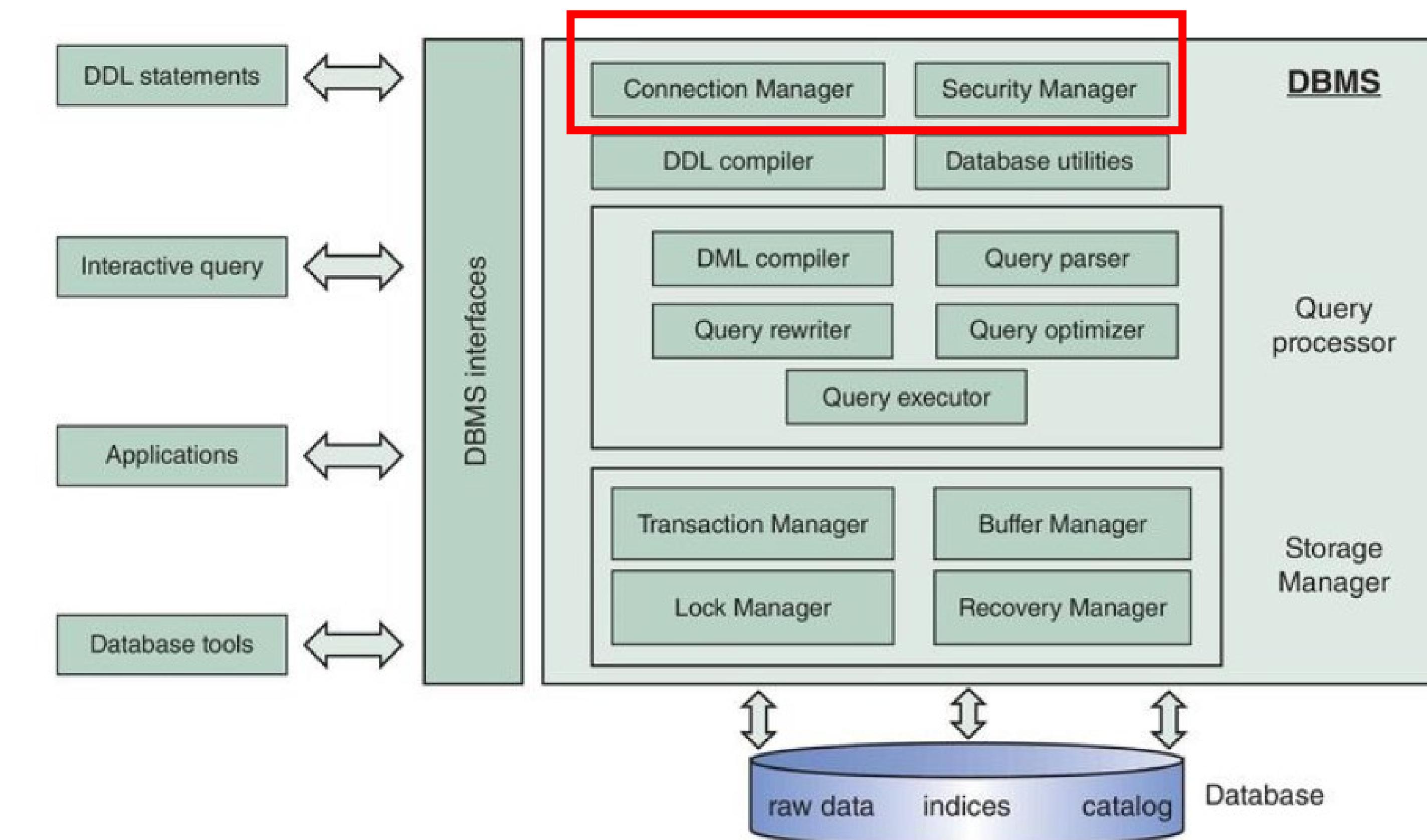
Architecture of a DBMS

Various ways of interacting with the DBMS. Command line interface, form-base interface, tools to maintain of fine-tune the DBMS.



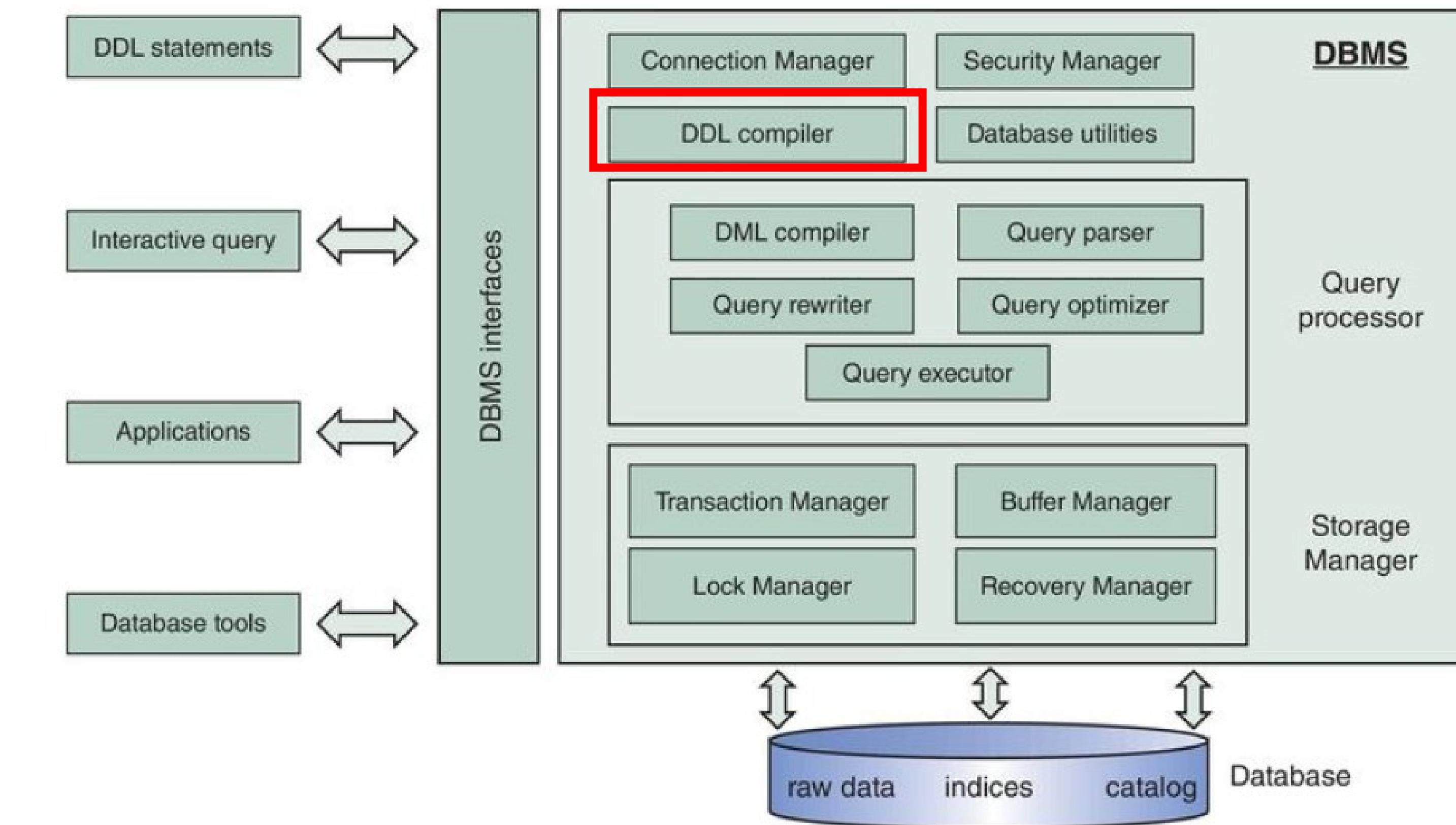
Architecture of a DBMS

The **connection manager** provides facilities to set-up a database connection. It can be set-up locally or through a network. The **security manager** verifies whether a user has the right privileges to execute the database actions required.



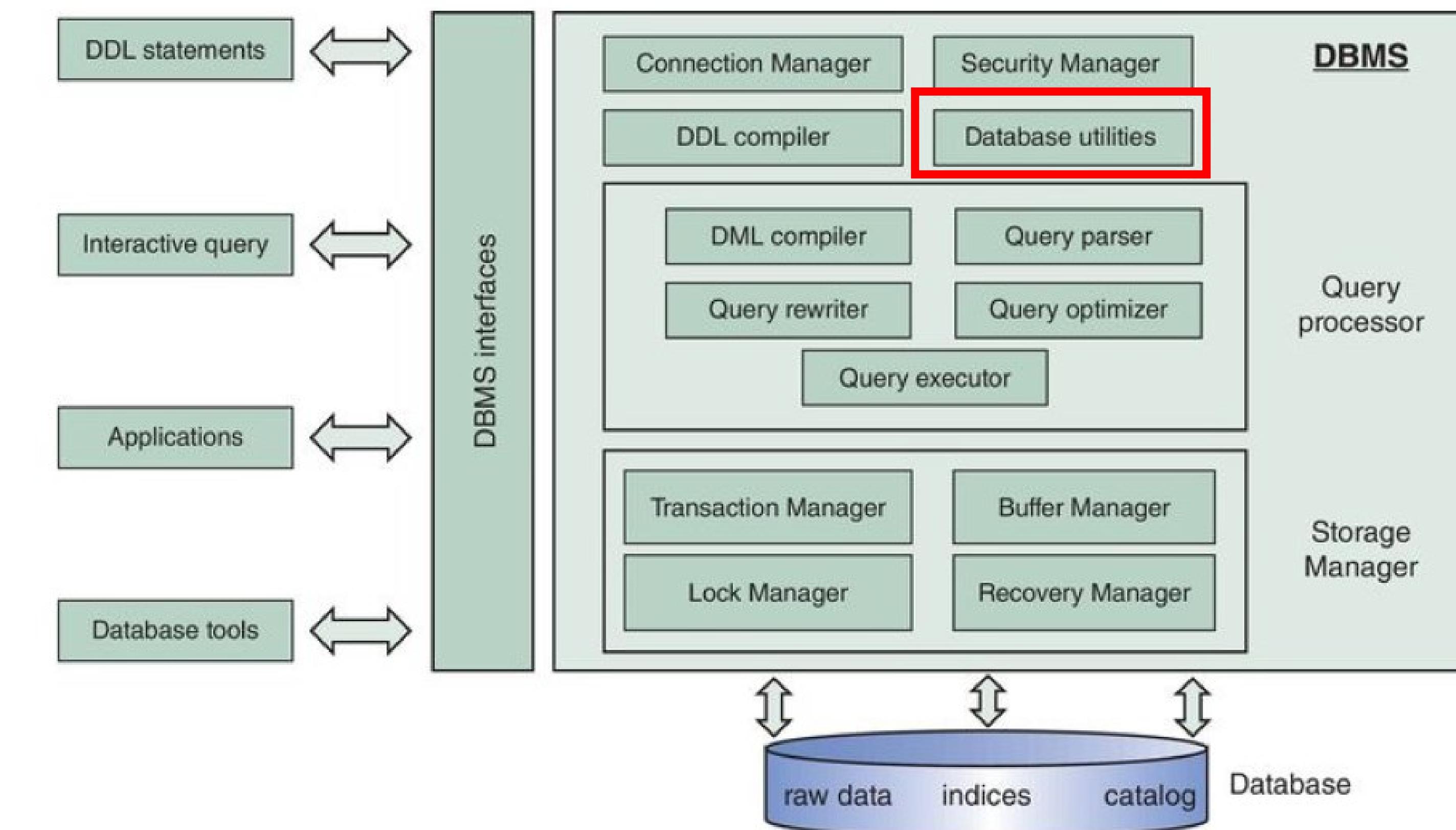
Architecture of a DBMS

The **Data Definition Language (DDL) compiler** compiles the data definitions specified in DDL. Most relational databases use SQL as their DDL.



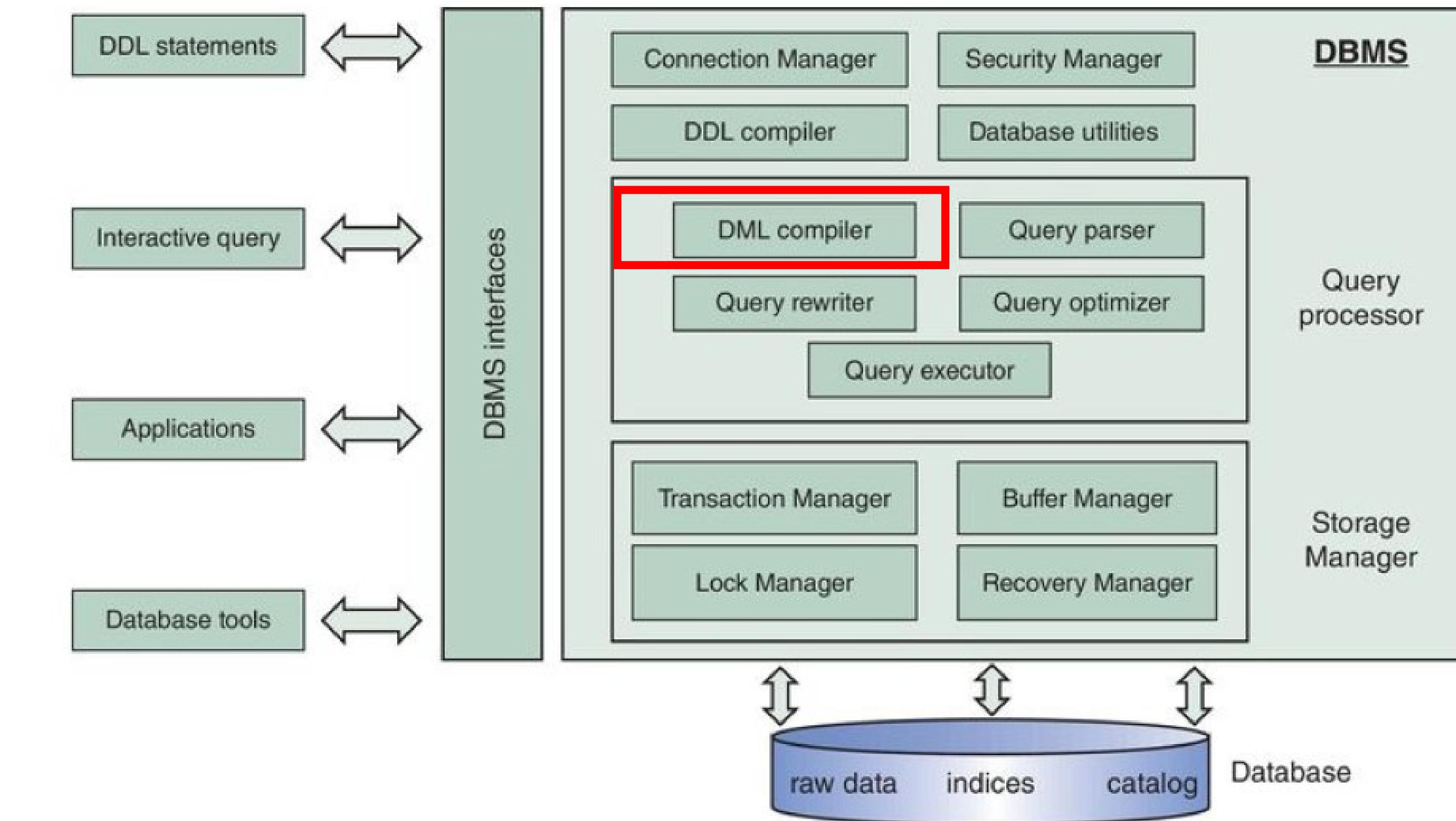
Architecture of a DBMS

Various utilities, example: A loading utility (load data from a variety of sources), reorganization utility (reorganizes the data), user management utilities (support the creation of user groups or accounts), etc.



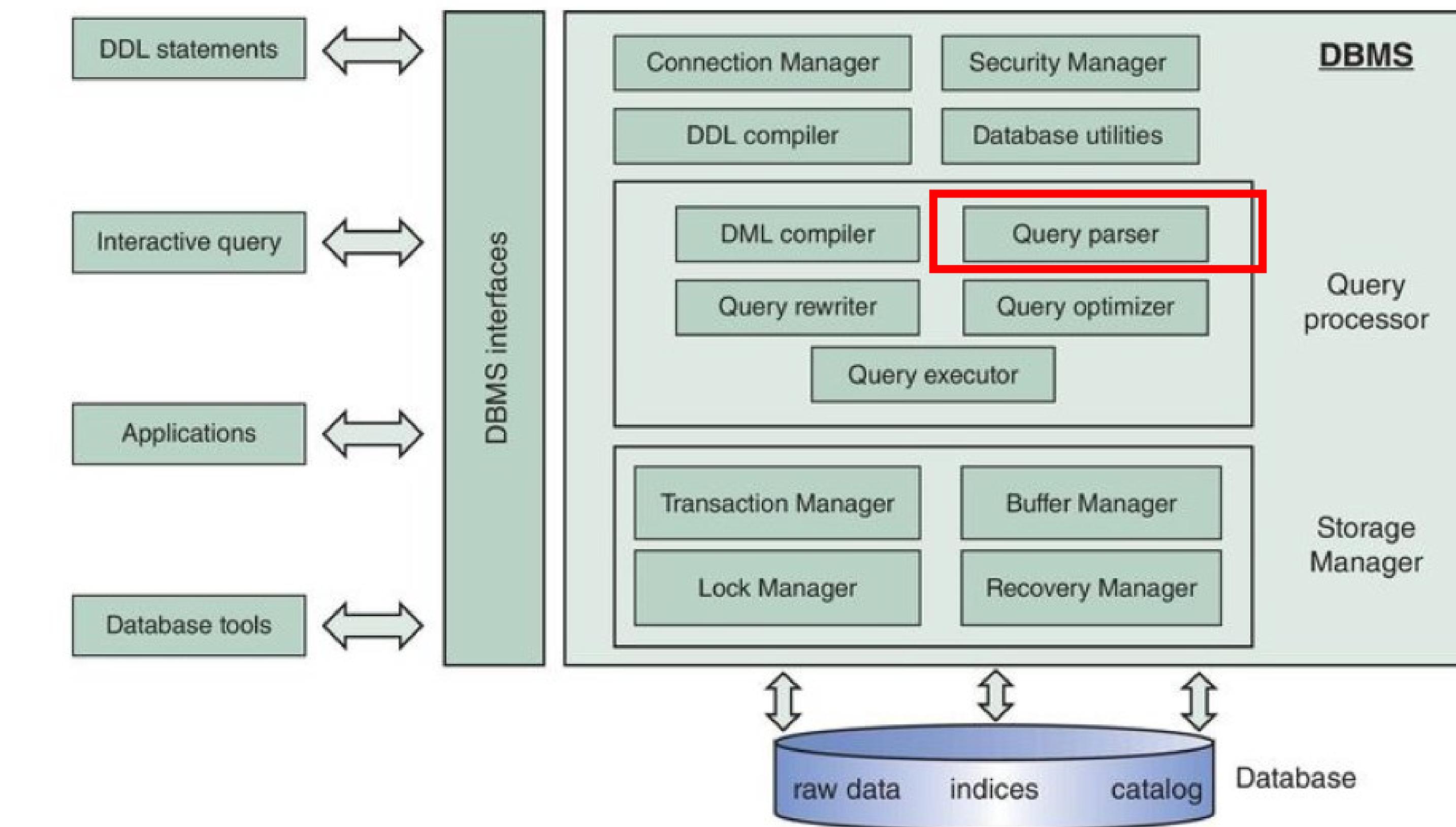
Architecture of a DBMS

The Data Manipulation Language (DML) compiler compiles the data manipulation statements specified in DML.



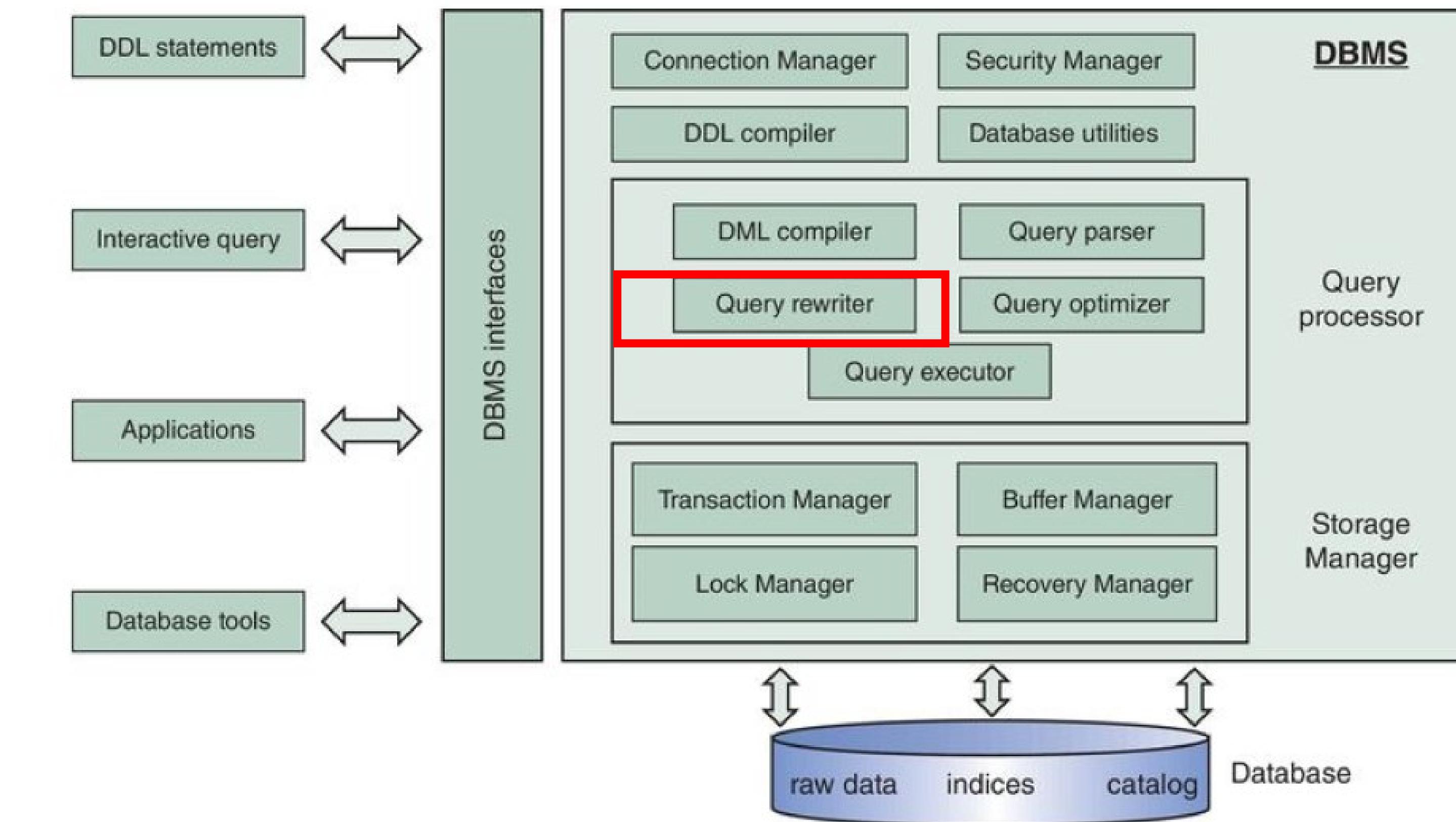
Architecture of a DBMS

The **query parser** parses the query into an *internal representation format* that can then be further evaluated by the system. It checks the query for syntactical and semantical correctness.



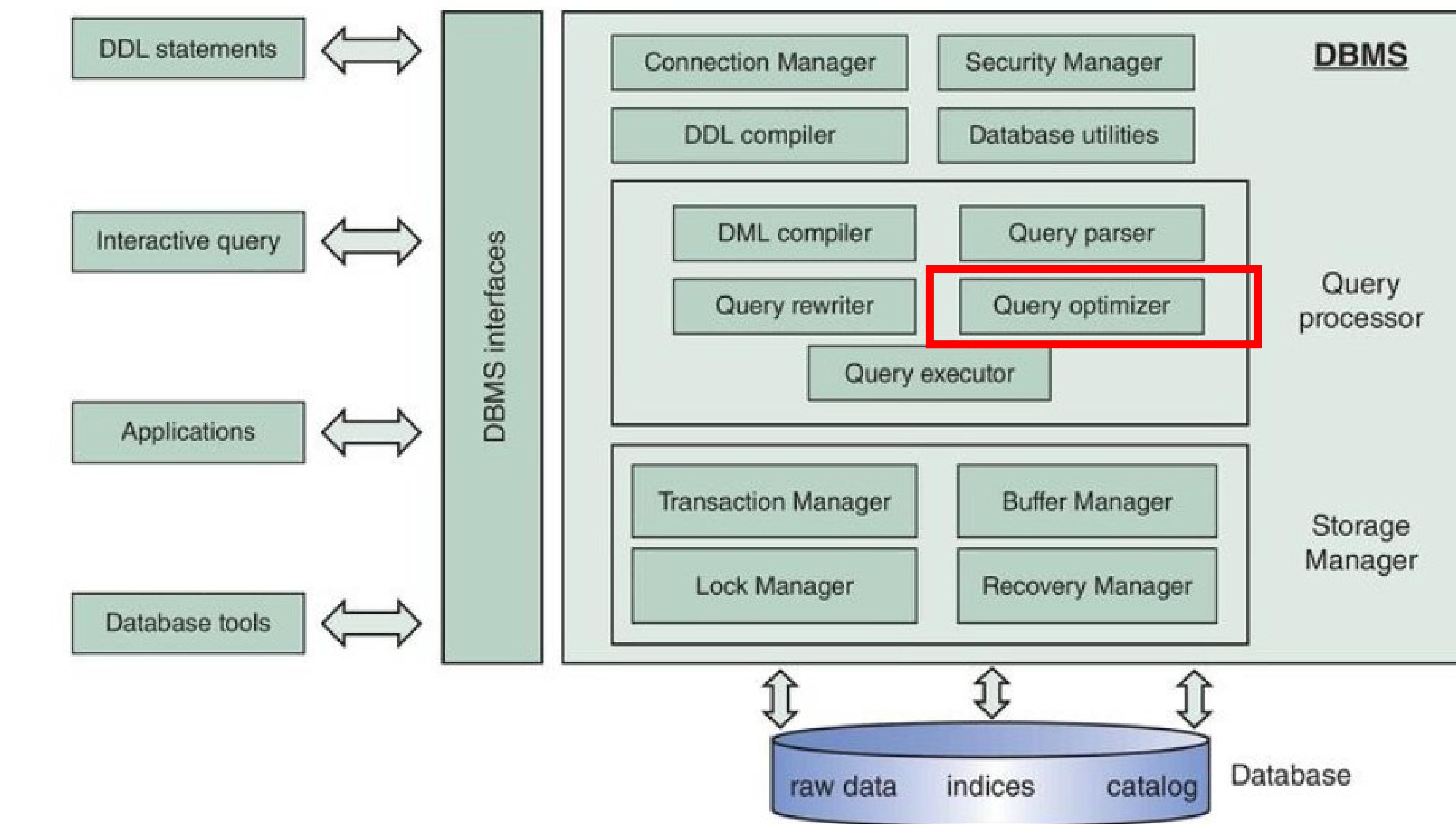
Architecture of a DBMS

The **query rewriter** optimizes the query, independently of the current database state. It simplifies it using a set of predefined rules and heuristics that are DBMS-specific.



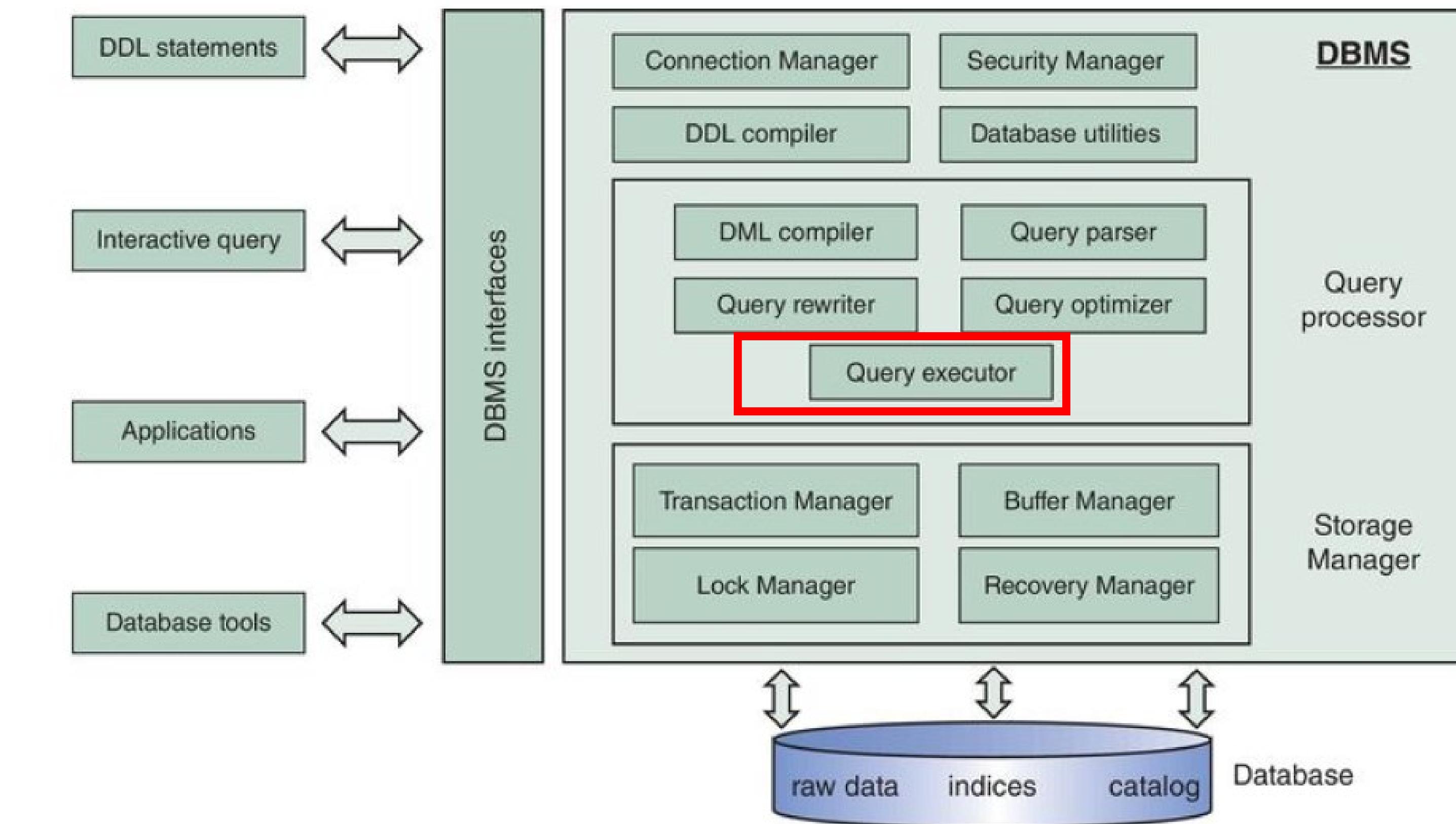
Architecture of a DBMS

The **query optimizer** optimizes the query based upon the current database state. It can make use of predefined indexes that are part of the internal data model and provide quick access to the data.



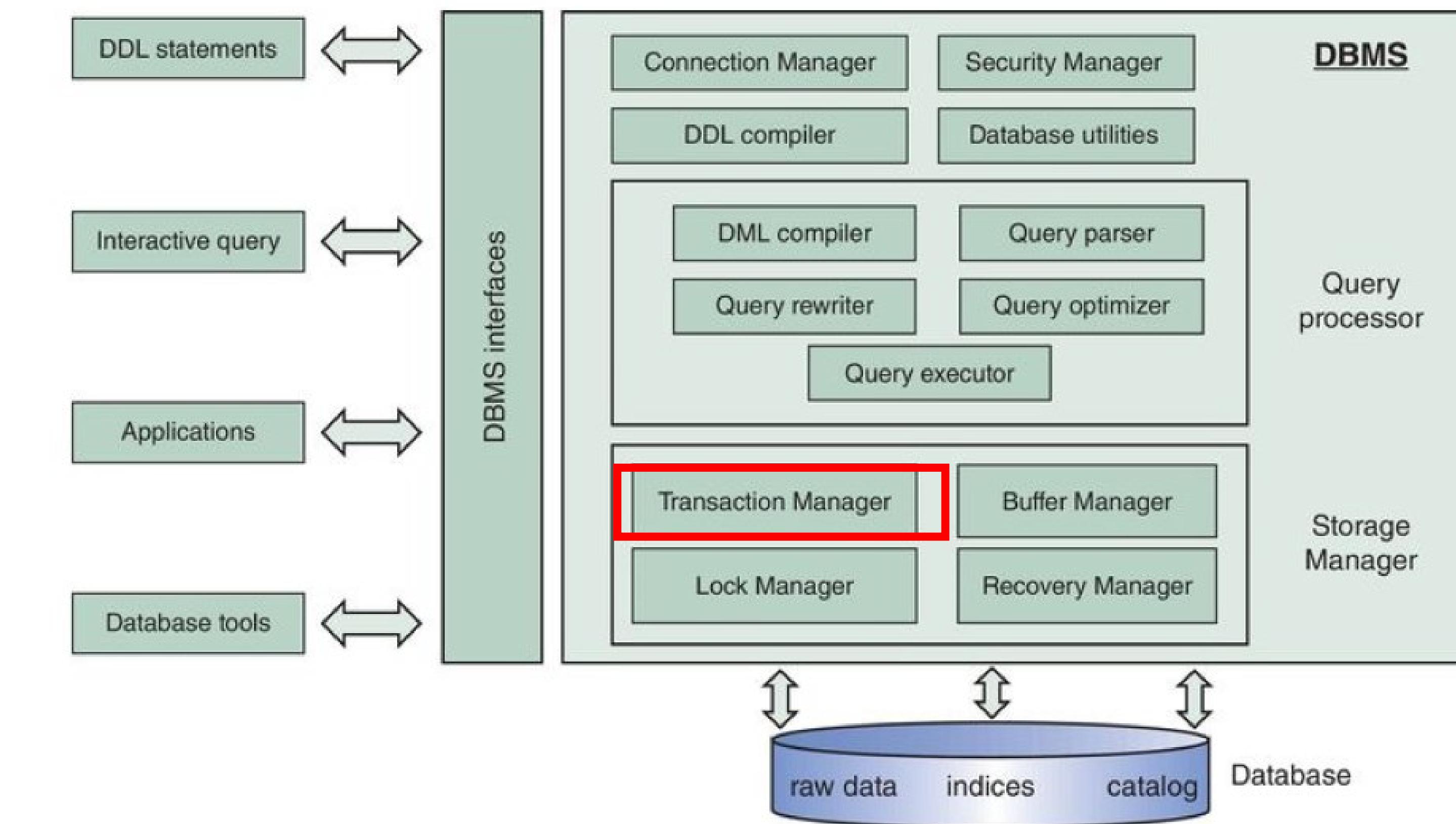
Architecture of a DBMS

The **query executor** takes care of the actual execution by calling on the storage manager to retrieve the data requested



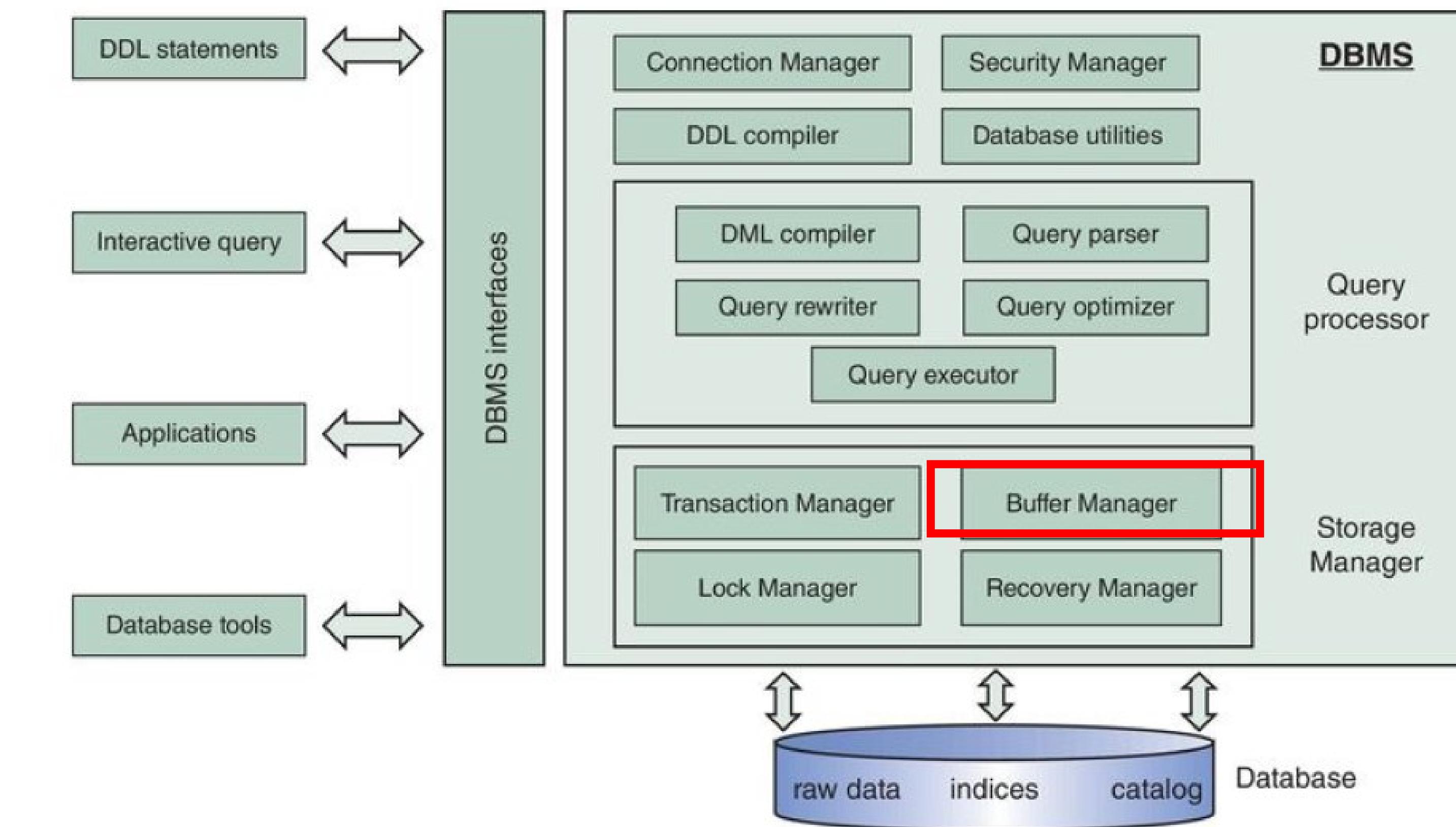
Architecture of a DBMS

The **transaction manager** supervises the execution of database transactions. Remember, a database transaction is a sequence of read/write operations considered to be an atomic unit.



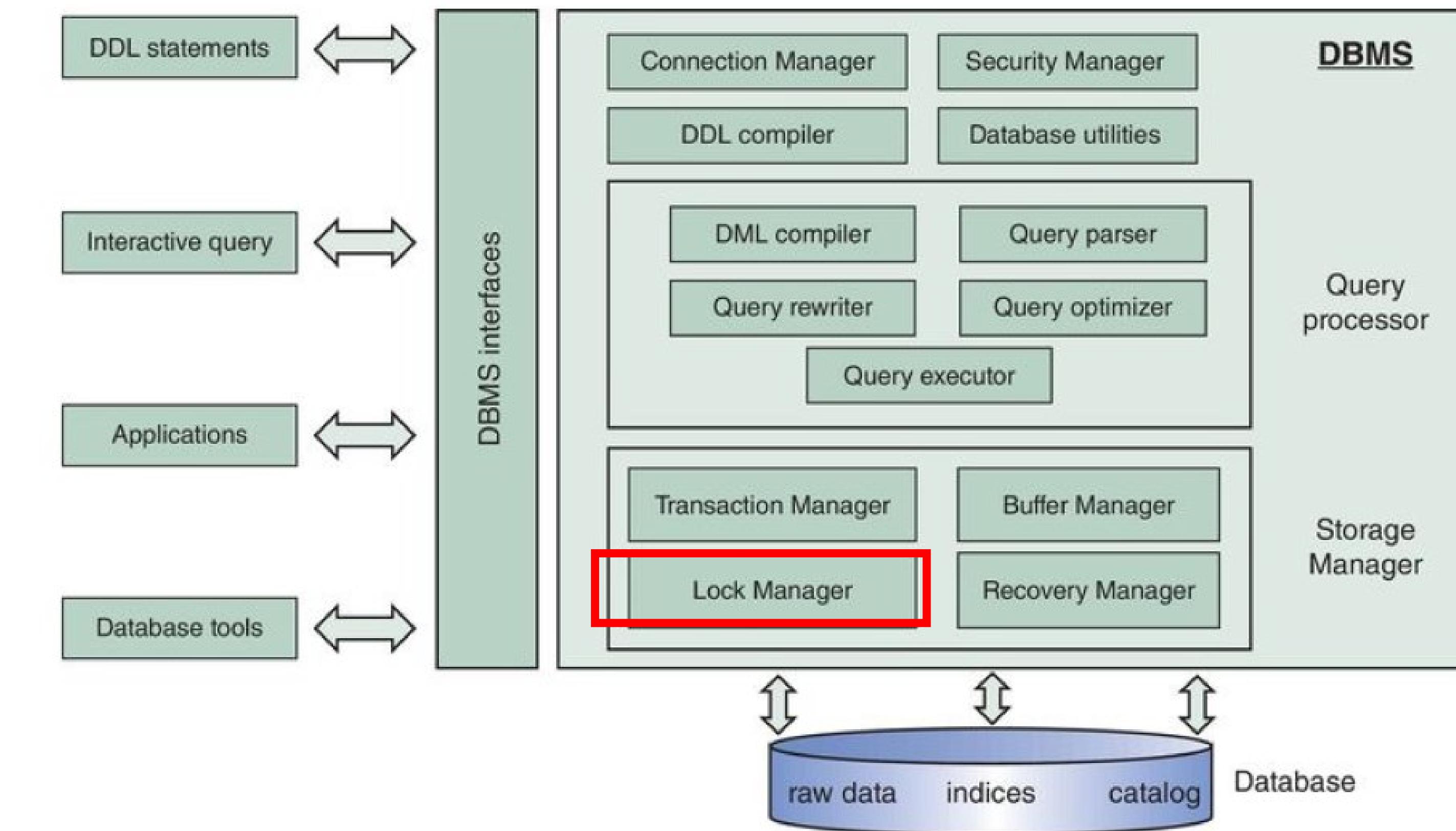
Architecture of a DBMS

The **buffer manager** is responsible for managing the buffer memory of the DBMS. The DBMS checks first the memory when data need to be retrieved. Retrieving data from the buffer is significantly faster than retrieving them from external disk-based storage.



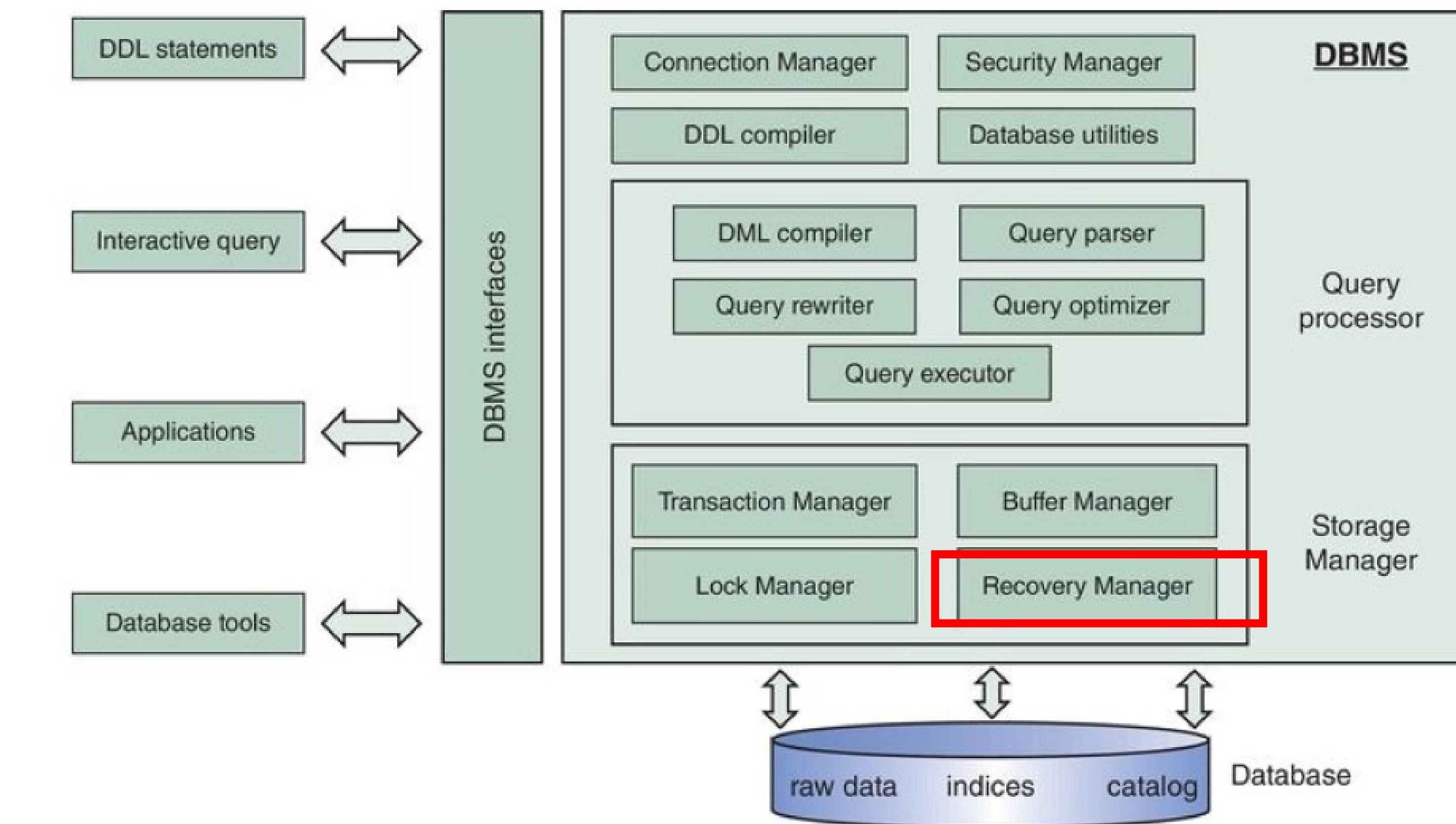
Architecture of a DBMS

The **lock manager** is an essential component for providing concurrency control, which ensures data integrity at all times.



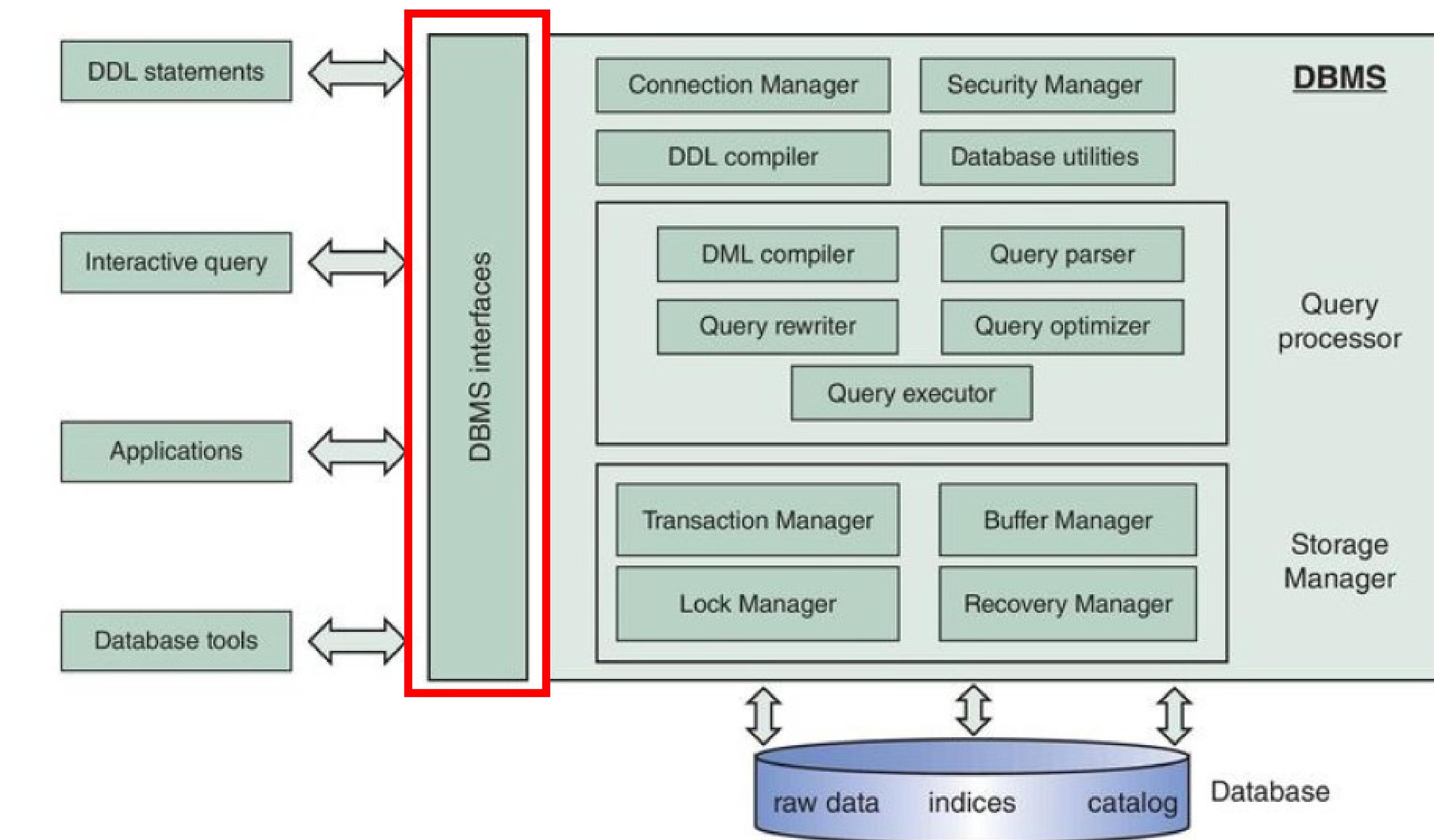
Architecture of a DBMS

The **recovery manager** supervises the correct execution of database transactions. It keeps track of all database operations in a **logfile** and will be called upon to undo actions of aborted transactions or during crash recovery.



Architecture of a DBMS

A DBMS needs to interact with various parties, such as a database designer, a database administrator, an application, or even an end-user. Ex. MySQL interface.



Review question

Fill in the gaps in the following sentences:

When, during crash recovery, aborted transactions need to be undone, that is a task of the ...A...

The part of the storage manager that guarantees the ACID properties is the ...B...

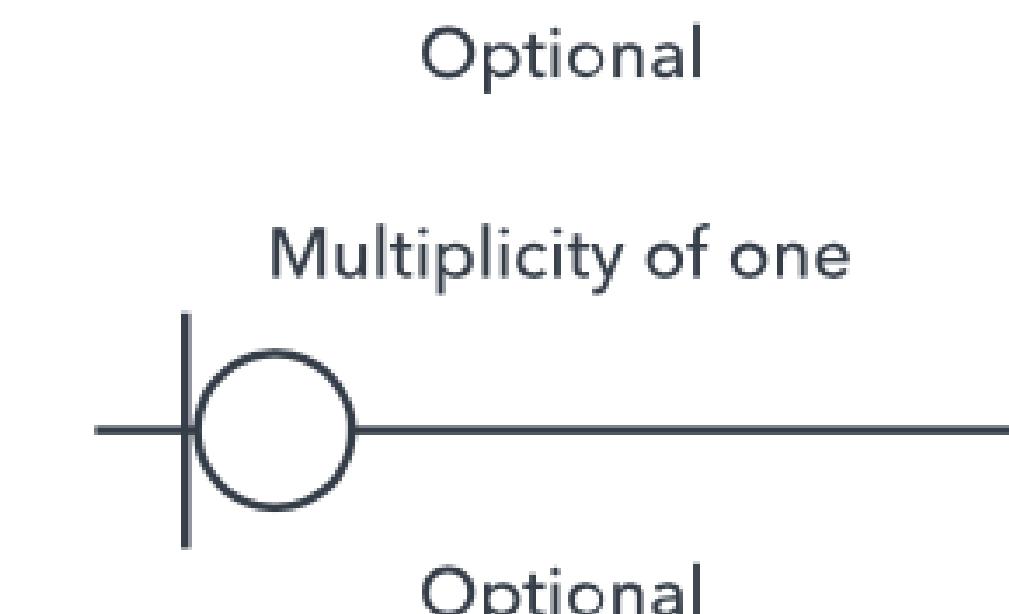
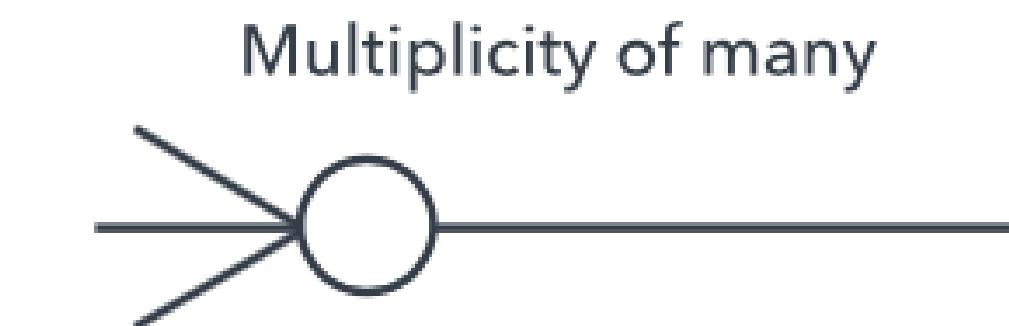
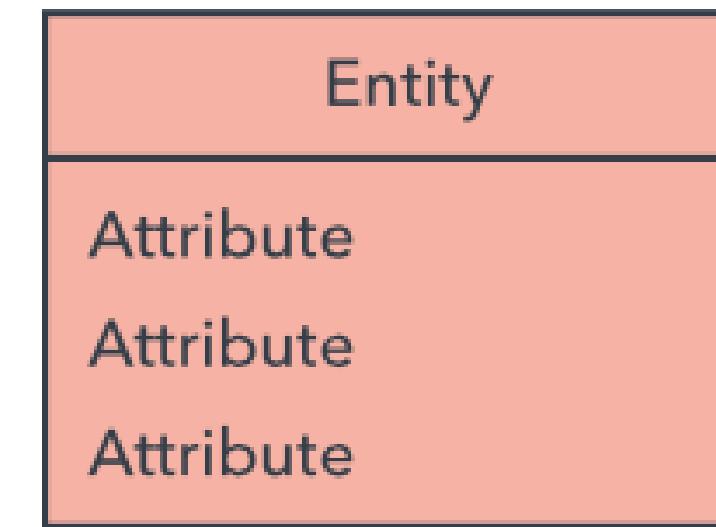
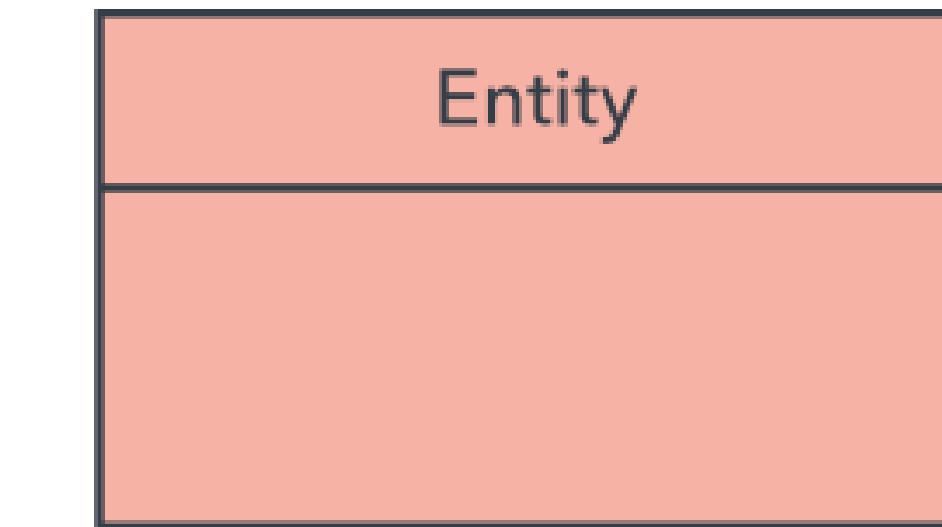
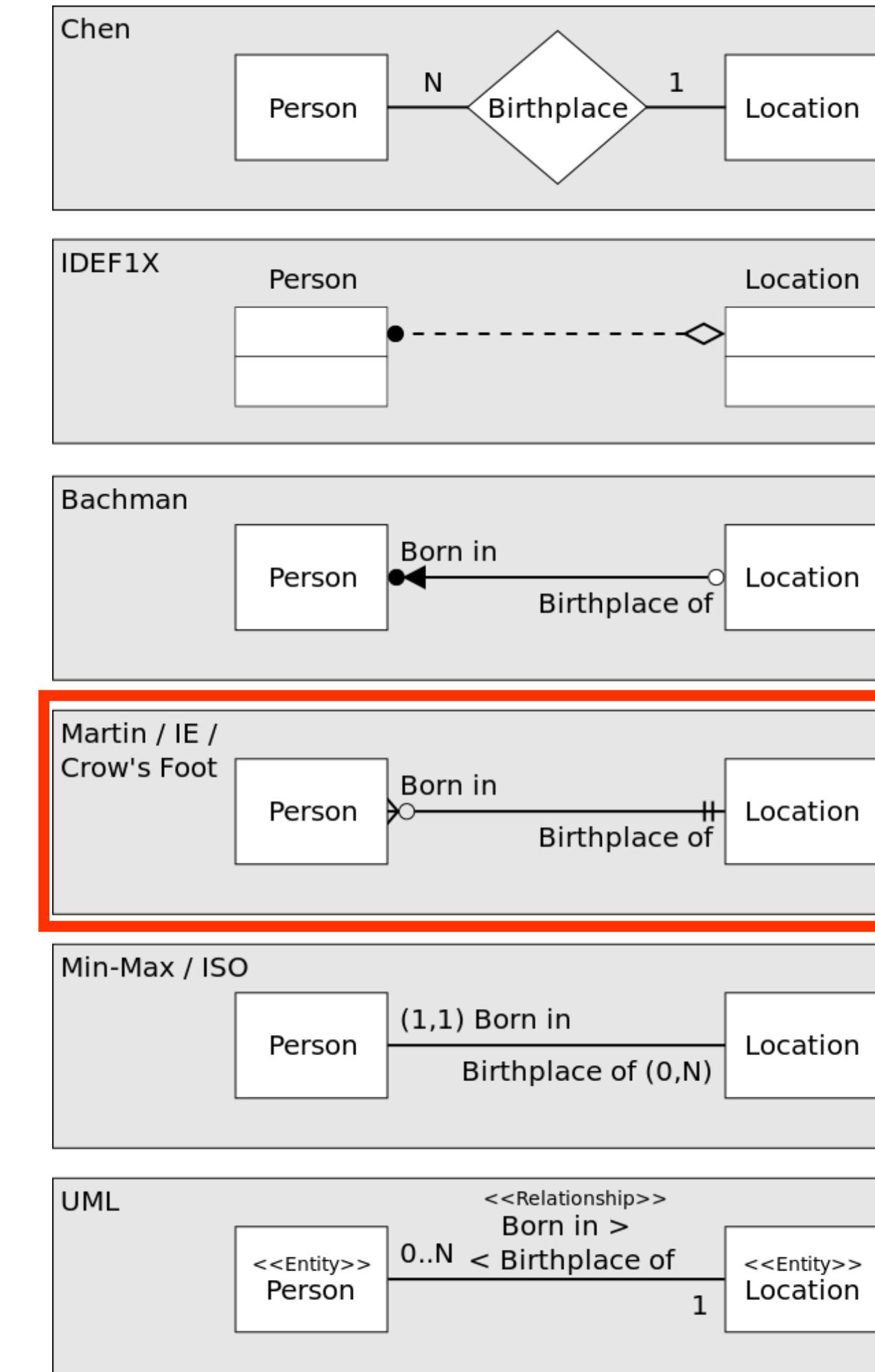
- a.** A: lock manager, B: recovery manager.
- b.** A: lock manager, B: lock manager.
- c.** A: recovery manager, B: buffer manager.
- d.** A: recovery manager, B: transaction manager.

Relational Modelling

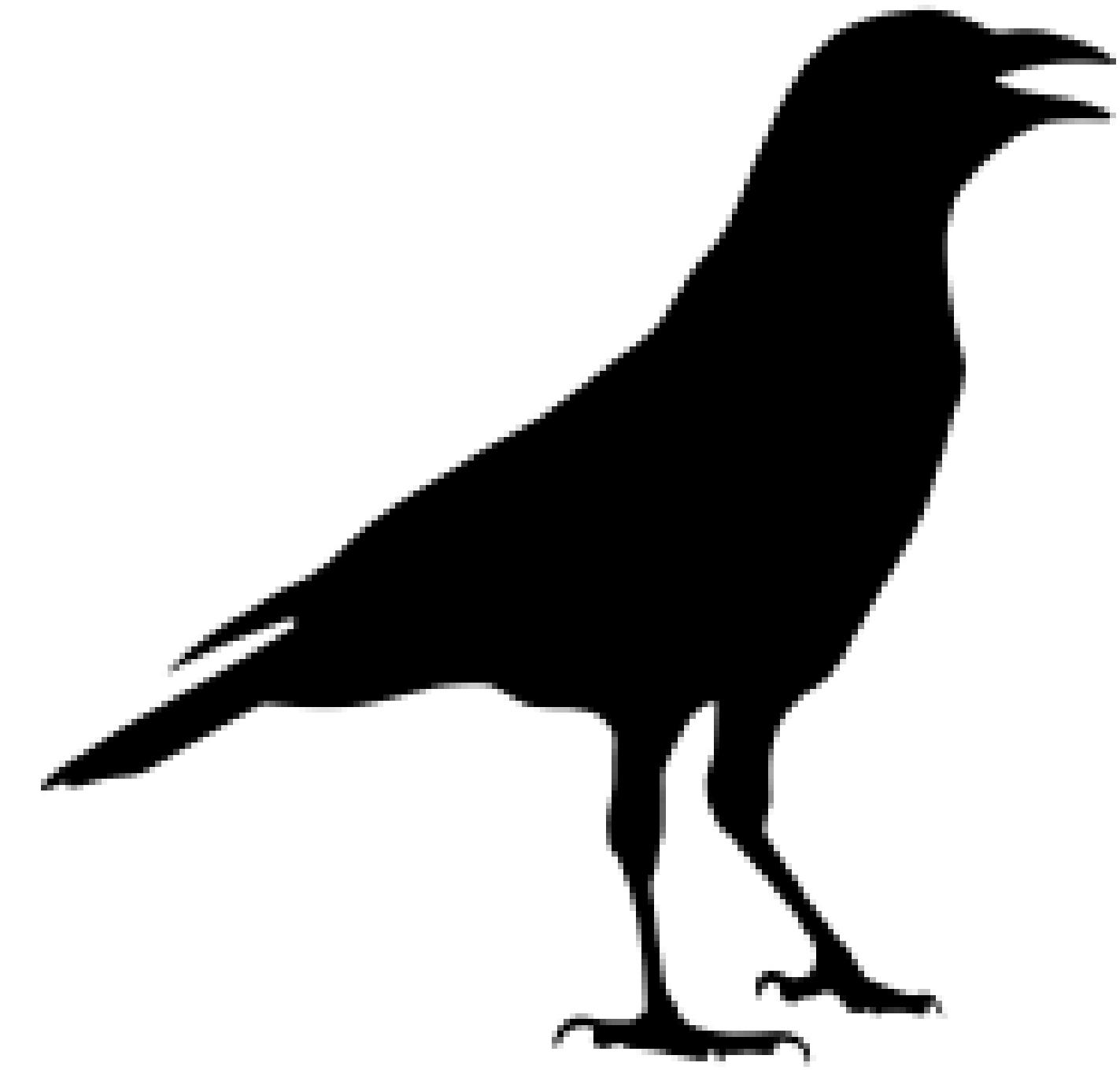
Entity Relationship Diagram (ERD)



Different types of Notations for the ERD



CROW's foot notation



Gordon Everest proposed the Crow's Foot notation in a 1976 paper. Several development methodologies enhanced the notation in the 1980s. Most CASE tools support some variation of the Crow's Foot notation.

Source: Mannino, M. V. (2019). *Database design, application development, and administration*. 7th edition. Chicago Business Press

CROW's foot notation - Illustrating Basic Symbols

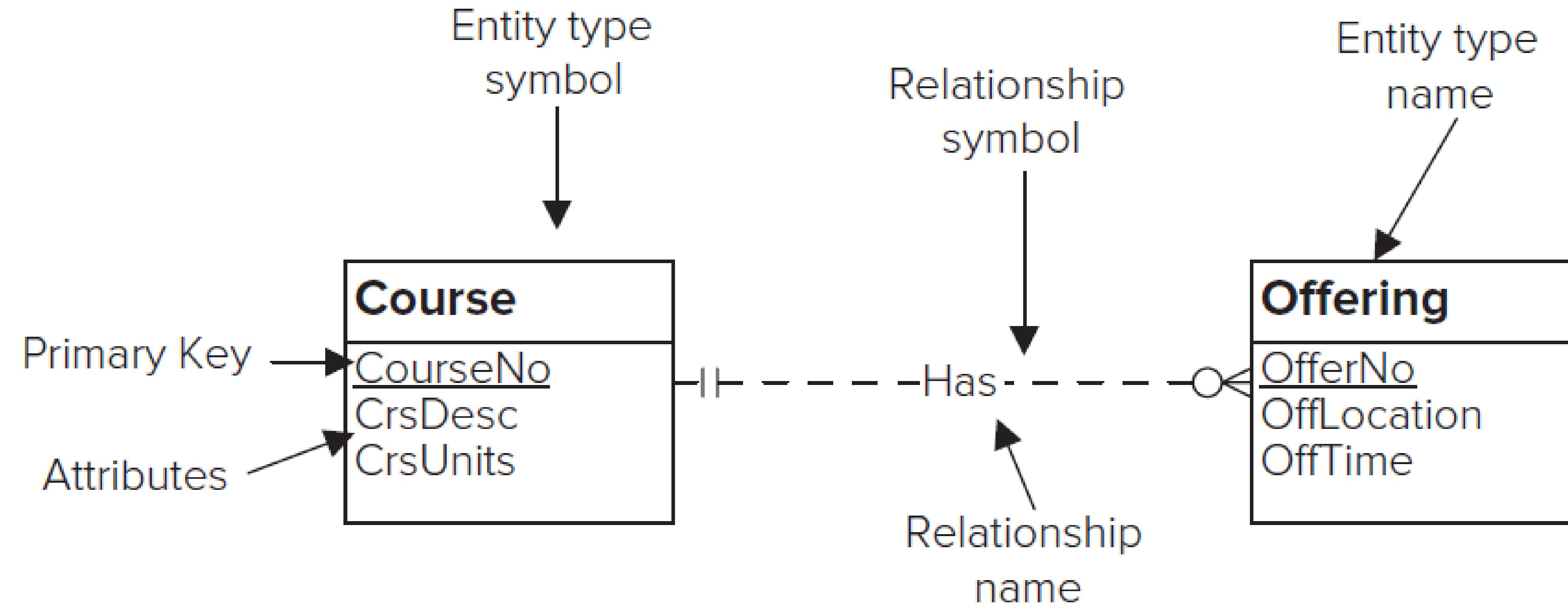
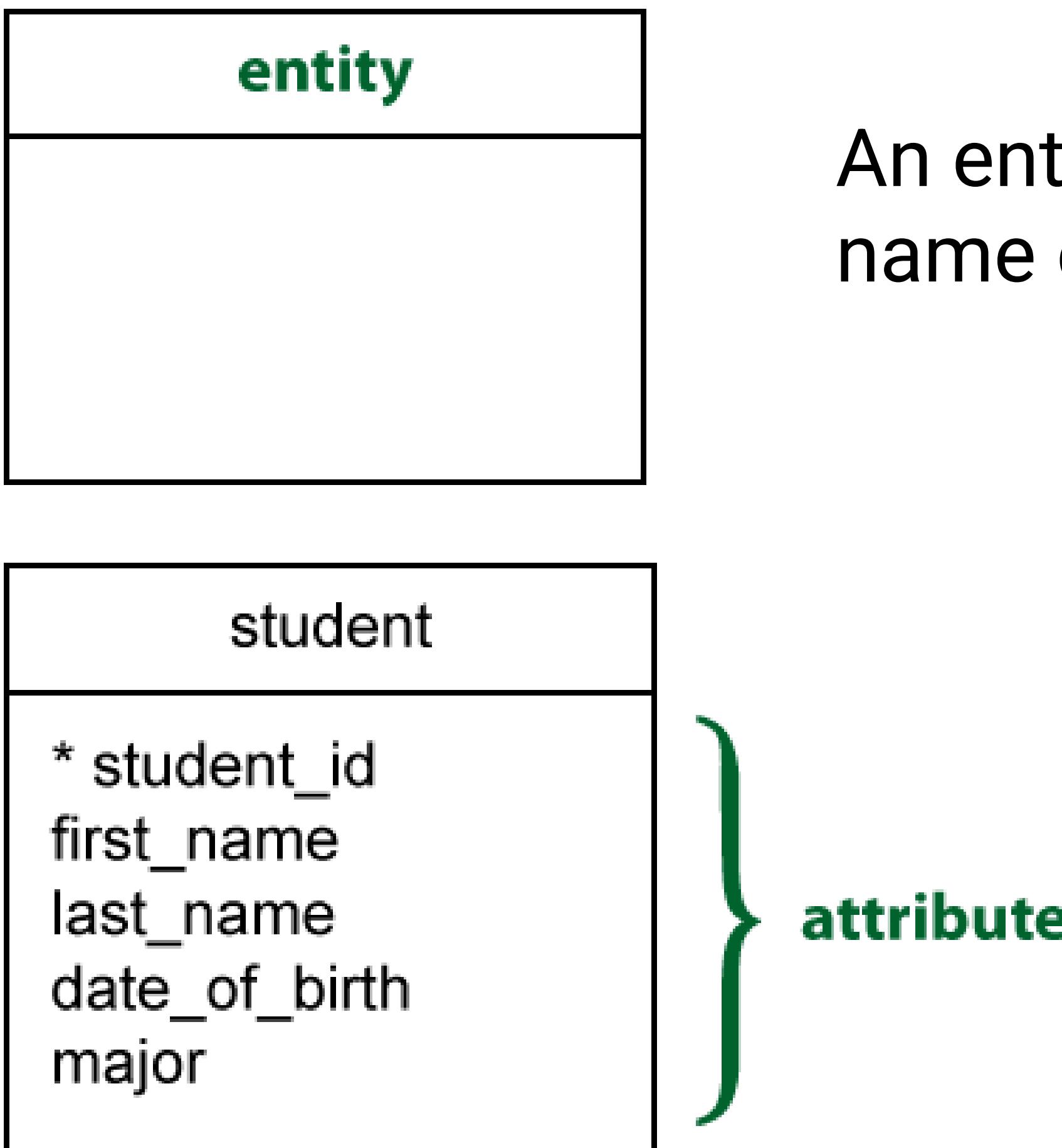


Image Source: Mannino, M. V. (2005). *Database design, application development, and administration*. 7th edition. Chicago Business Press

CROW's foot notation - Notation



An entity is represented by a rectangle, with its name on the top. The name is singular.

The attribute(s) that uniquely distinguishes an instance of the entity is the identifier.

Image Source: <https://www.vertabelo.com/blog/crow-s-foot-notation/>

CROW's foot notation - Notation

Relationships have two indicators.

The **first one** refers to the maximum number of times that an instance of one entity can be associated with instances in the related entity.

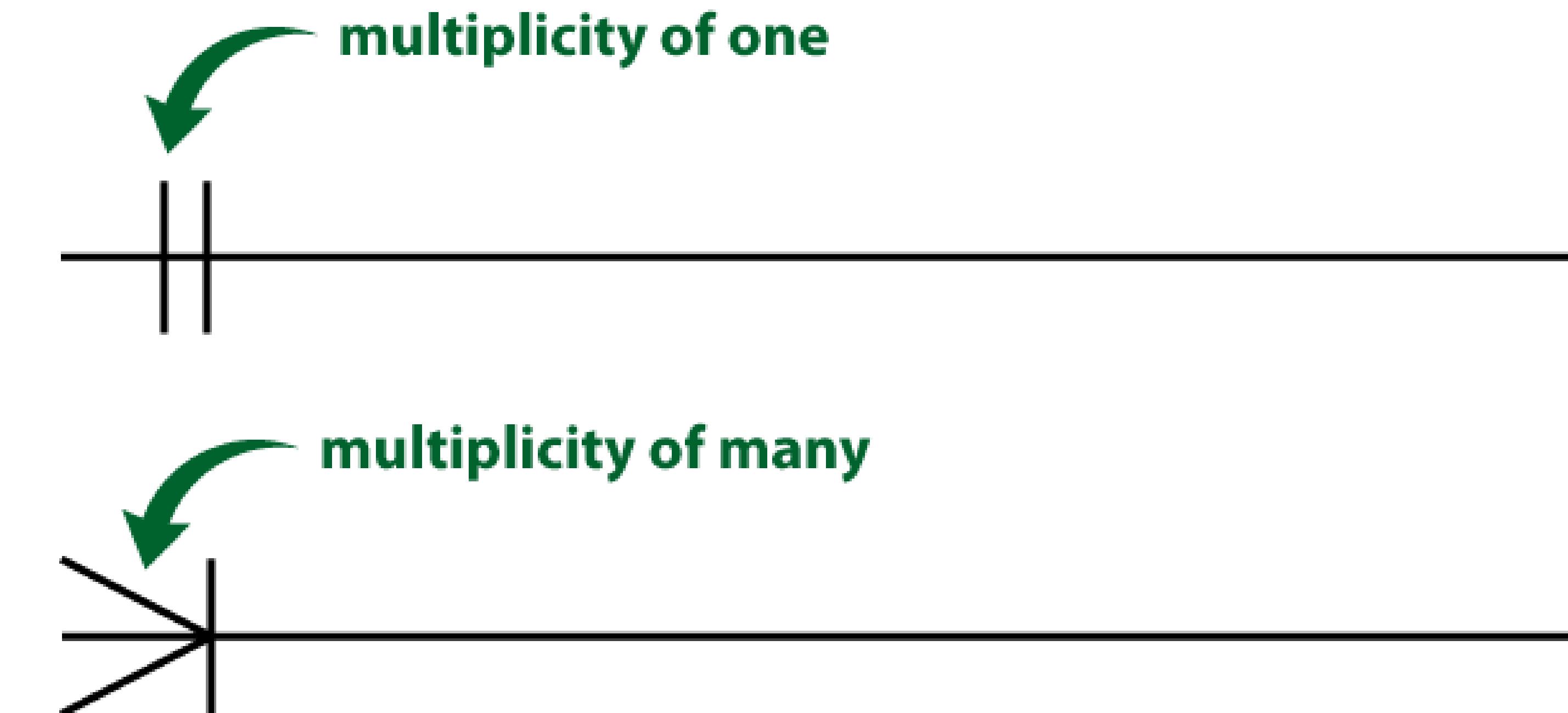


Image Source: <https://www.vertabelo.com/blog/crow-s-foot-notation/>

CROW's foot notation - Notation

The **second one** describes the minimum number of times one instance can be related to others. It can be zero or one, and optional or mandatory.

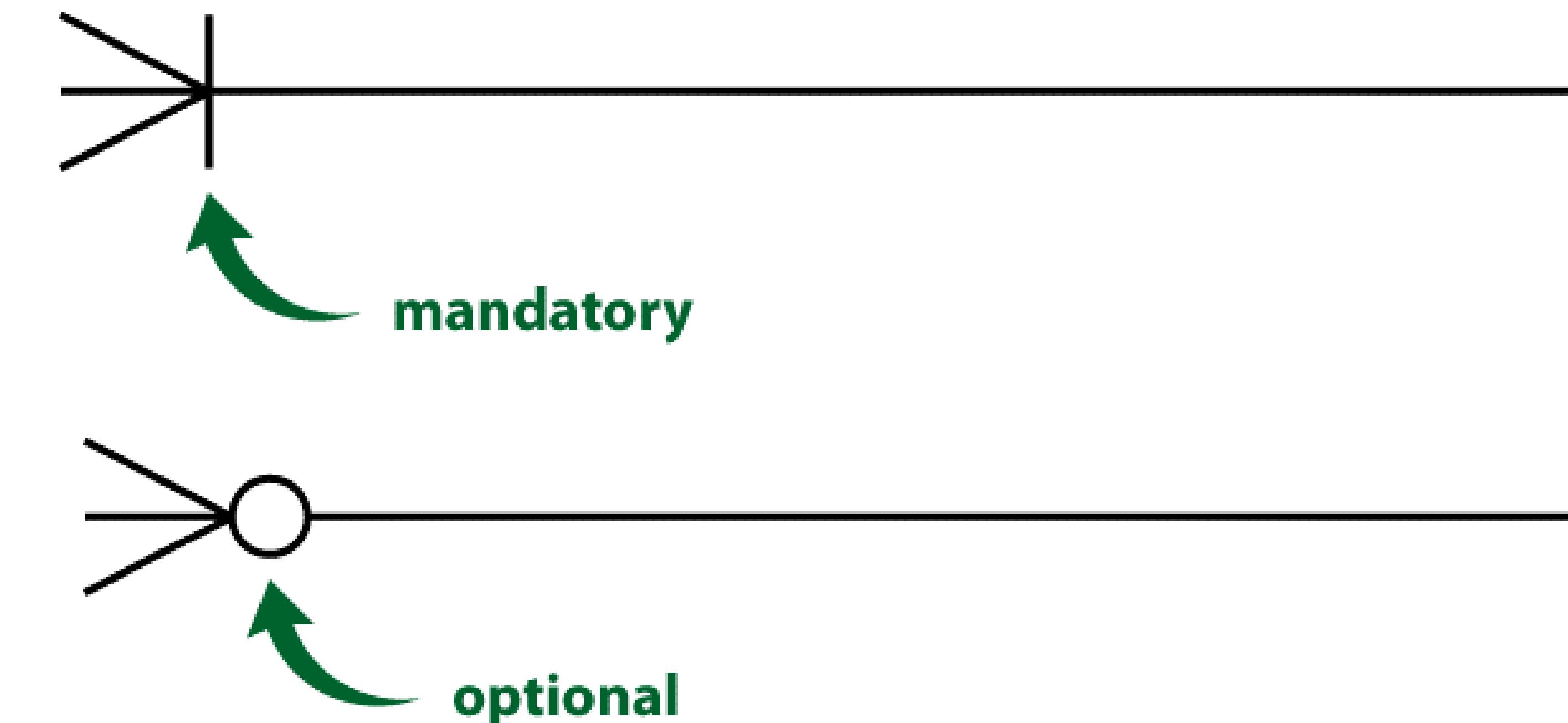
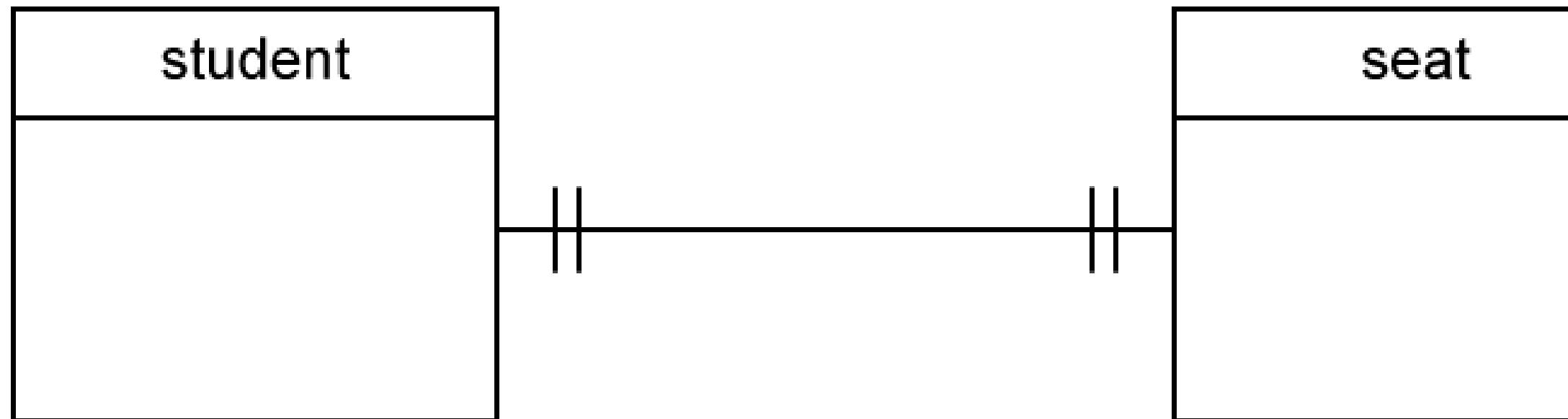
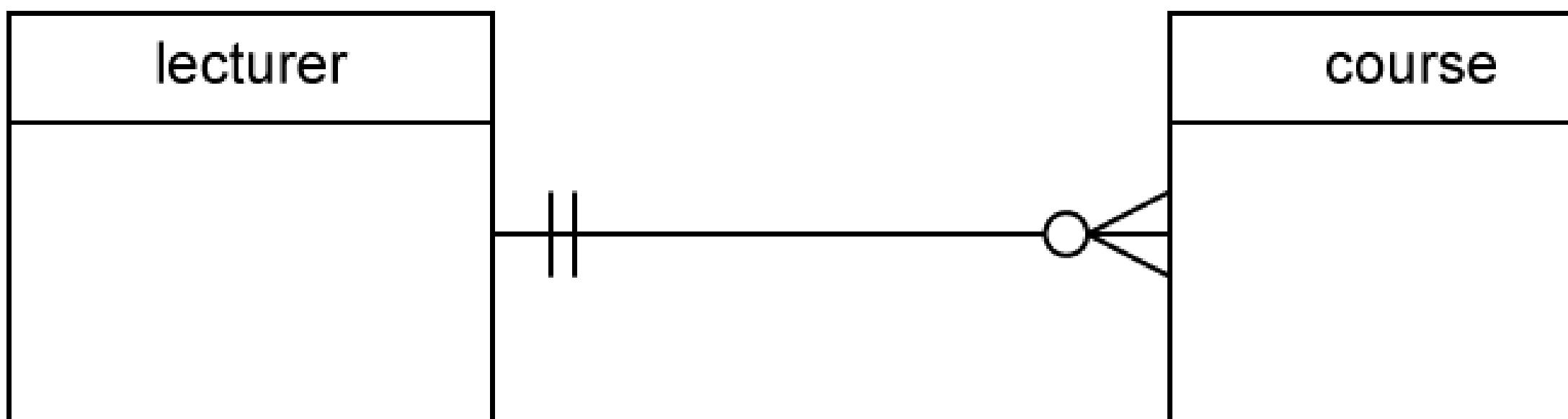


Image Source: <https://www.vertabelo.com/blog/crow-s-foot-notation/>

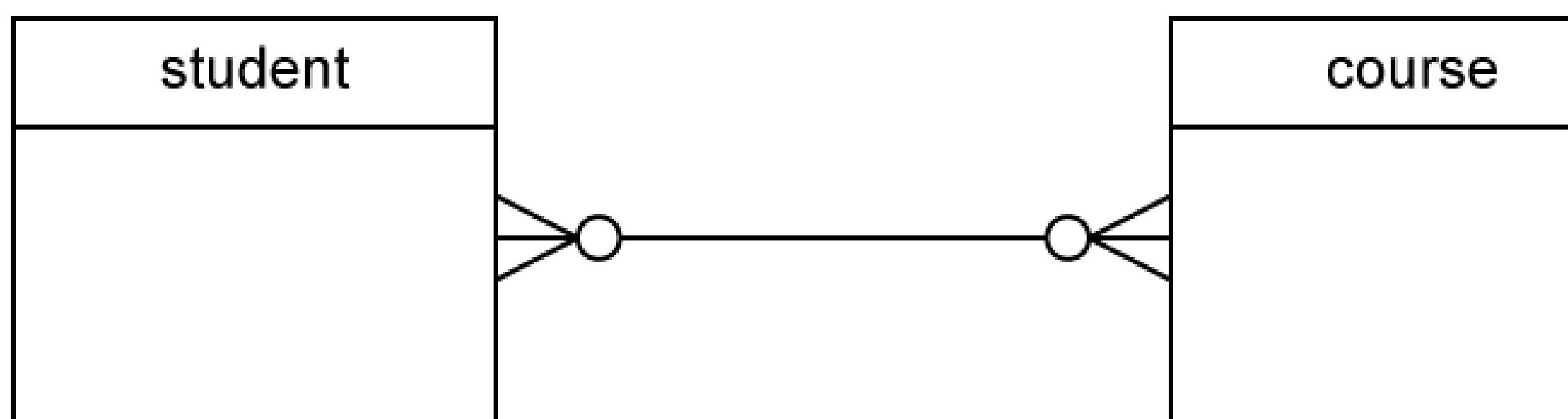
CROW's foot notation - Examples



One-to-one



One-to-many



Many-to-many

Image Source: <https://www.vertabelo.com/blog/crow-s-foot-notation/>

CROW's foot notation - Examples

Consider the following entity types for a patient appointment system:

- Doctor
- Appointment
- Bill
- Payment
- Patient
- Insurance_company

Draw an entity relationship diagram (ERD) with crow's foot notation.
Several solutions can be valid

<http://www2.cs.uregina.ca/~bernatja/crowsfoot.html>

From ERD to Tables

Relational Databases - Tables

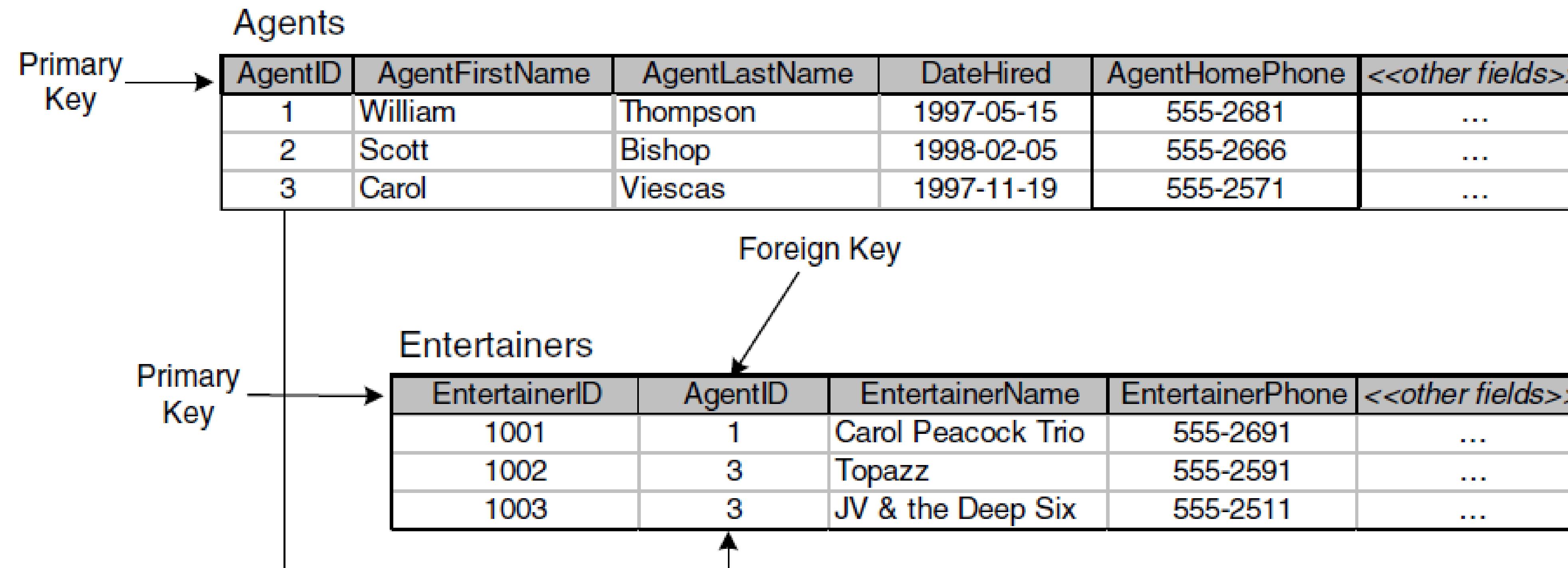
Patient Visit

PatientID	VisitDate	VisitTime	Physician	BloodPressure	Temperature
92001	2006-05-01	10:30	Ehrlich	120 / 80	98.8
97002	2006-05-01	13:00	Hallmark	112 / 74	97.5
99014	2006-05-02	9:30	Fournier	120 / 80	98.8
96105	2006-05-02	11:00	Hallmark	160 / 90	99.1
96203	2006-05-02	14:00	Hallmark	110 / 75	99.3
98003	2006-05-02	9:30	Fournier	120 / 82	98.6

Records / Rows

Fields / Columns / Features

Primary Keys - Unique Identifier of Records in each Table



Foreign Keys - Establish relationships between tables

Table A

One to One

Table B

Example:

Agents					
AgentID	AgentFirstName	AgentLastName	DateOfHire	AgentHomePhone	<<other fields>>
1	William	Thompson	1997-05-15	555-2681	...
2	Scott	Bishop	1998-02-05	555-2666	...
3	Carol	Viescas	1997-11-19	555-2571	...

Compensation		
Salary	CommissionRate	<<other fields>>
\$35,000.00	4.00%	...
\$27,000.00	4.00%	...
\$30,000.00	5.00%	...

to each Agent is linked one and only one Row in Compensation Table

Relational Databases - Relationships

Table A

One to Many

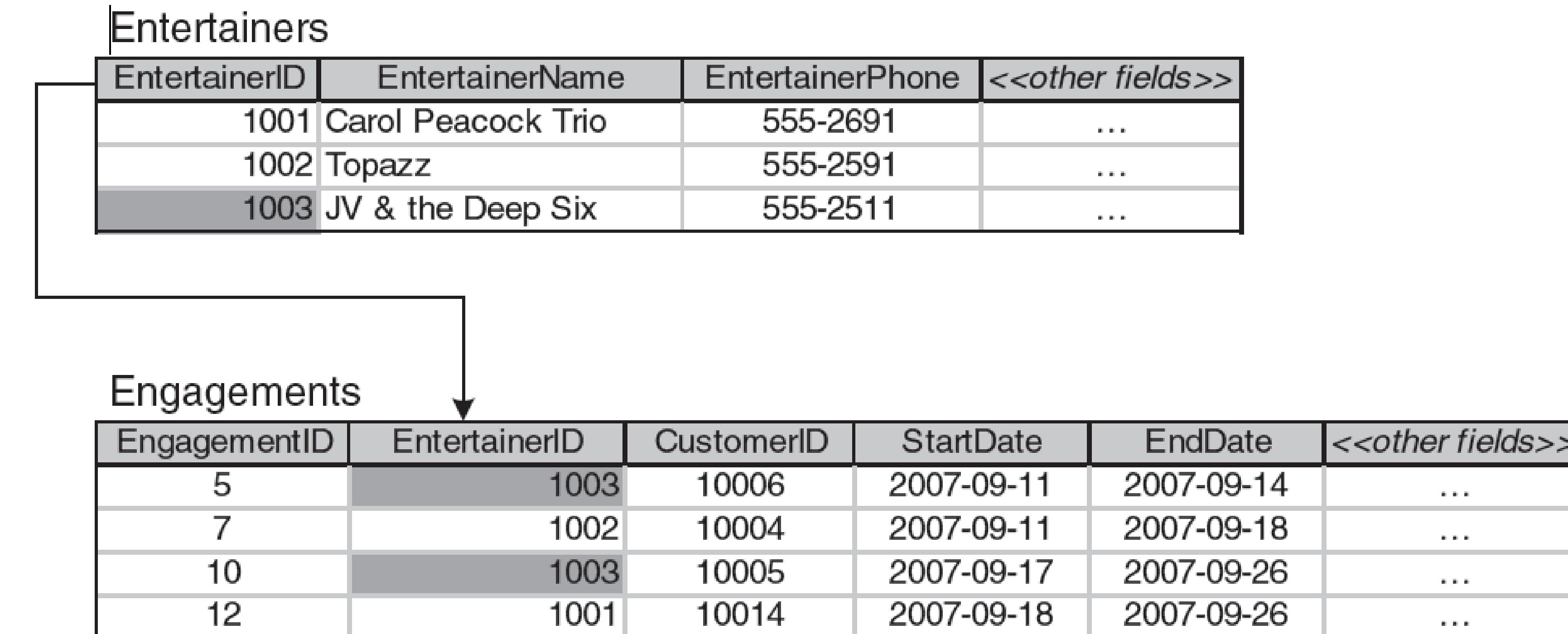


Table B

The image shows a 4x4 grid of colored squares. The colors are as follows: Row 1: Dark Gray, Light Gray, Light Gray, Light Gray. Row 2: Light Gray, Light Gray, Light Gray, Light Gray. Row 3: Light Gray, White, White, White. Row 4: Light Gray, Light Gray, Light Gray, Light Gray. A red arrow points to the top-left square of the grid.

Relational Databases - Relationships

Example:



to each Entertainer can be linked multiple rows in Engagements,
but each Engagement is only linked to a single Entertainer

Relational Databases - Relationships

Table A

A 4x4 grid of squares. The top row consists of four dark gray squares. The second row has the first square in dark gray and the remaining three in light gray. The third row has the first square in light gray and the remaining three in medium gray. The bottom row consists of four medium gray squares.

Many to Many



Table B

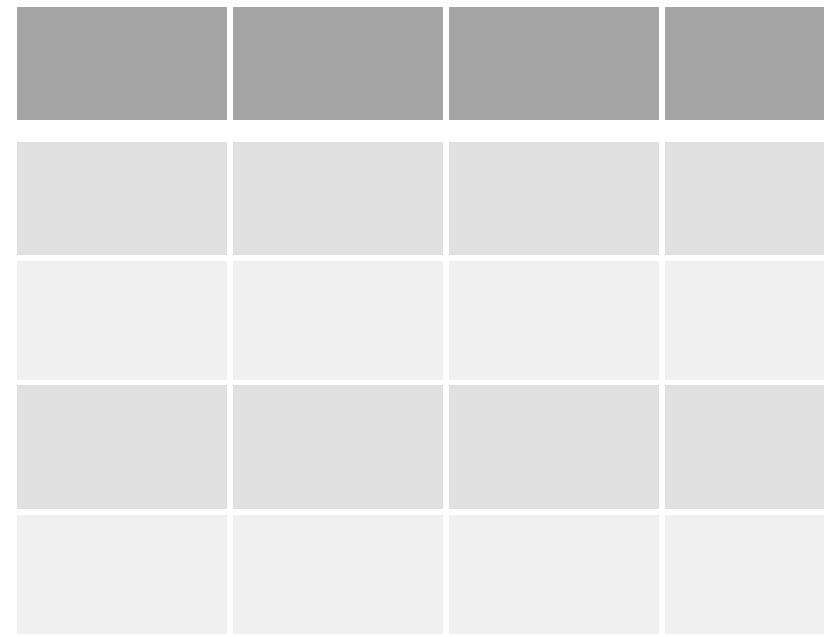
The image shows a 5x4 grid of colored squares. The top row consists of four dark gray squares. The second row has the first square light gray, followed by three white squares. The third row has the first square white, followed by three light gray squares. The fourth row has the first square light gray, followed by three white squares. The fifth row consists of four white squares. An orange arrow points from the bottom-left towards the center of the grid.

Relational Databases - Relationships

Problem: how to create a database structure with a many-to-many relationship type?

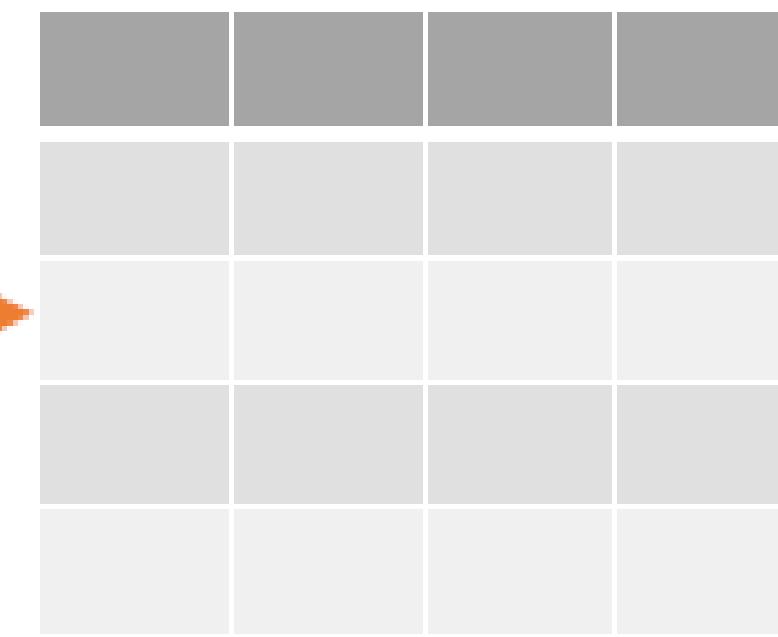
Relational Databases - Relationships

Table A



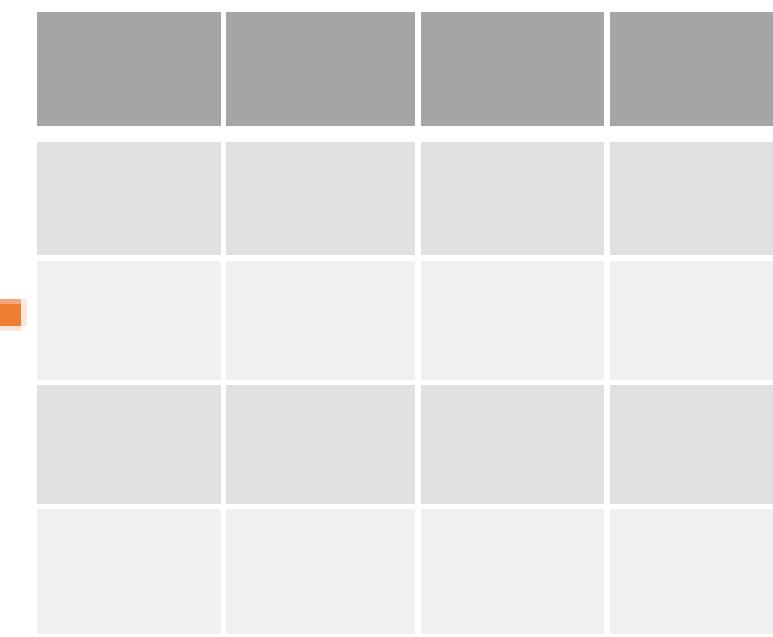
One to Many

Linking Table



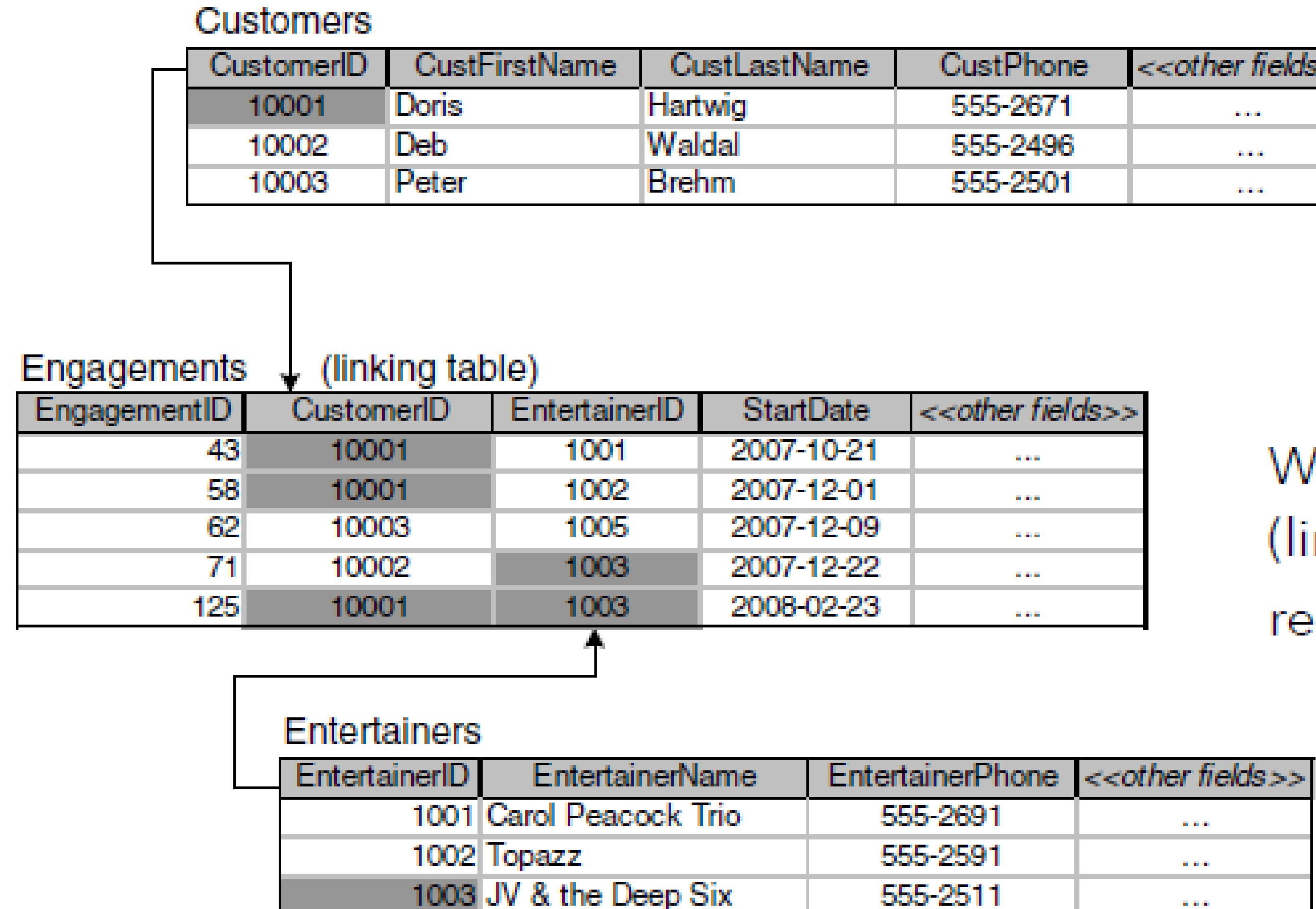
One to Many

Table B



Relational Databases - Relationships

Example:



We create an additional table (linking table) with one to many relationships with the original tables

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

**Let's have
some fun!**

END OF LECTURE 2

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA

IMS

Information
Management
School

Storing and Retrieving Data

Lecture 3

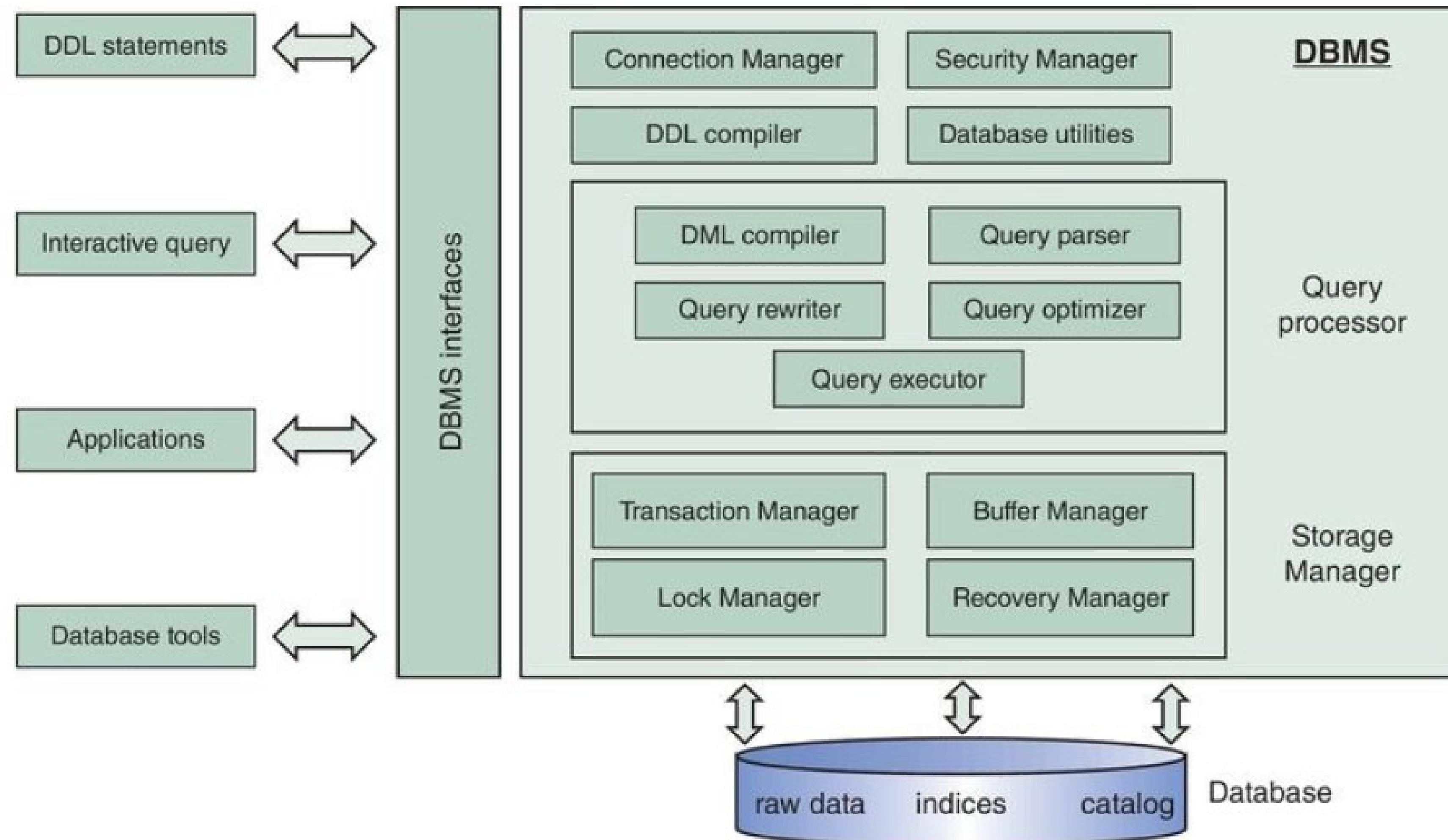
Structured Query Language (SQL) – Part 1

Lecturer: Mijail Naranjo-Zolotov

Email: mijail.naranjo@novaims.unl.pt

Previous class

Architecture of a DBMS

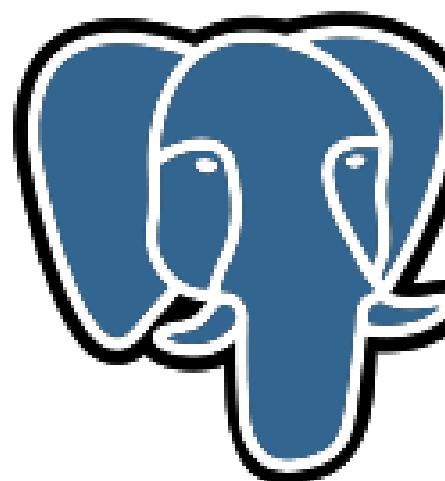


Relational Database Management System (RDBMS)

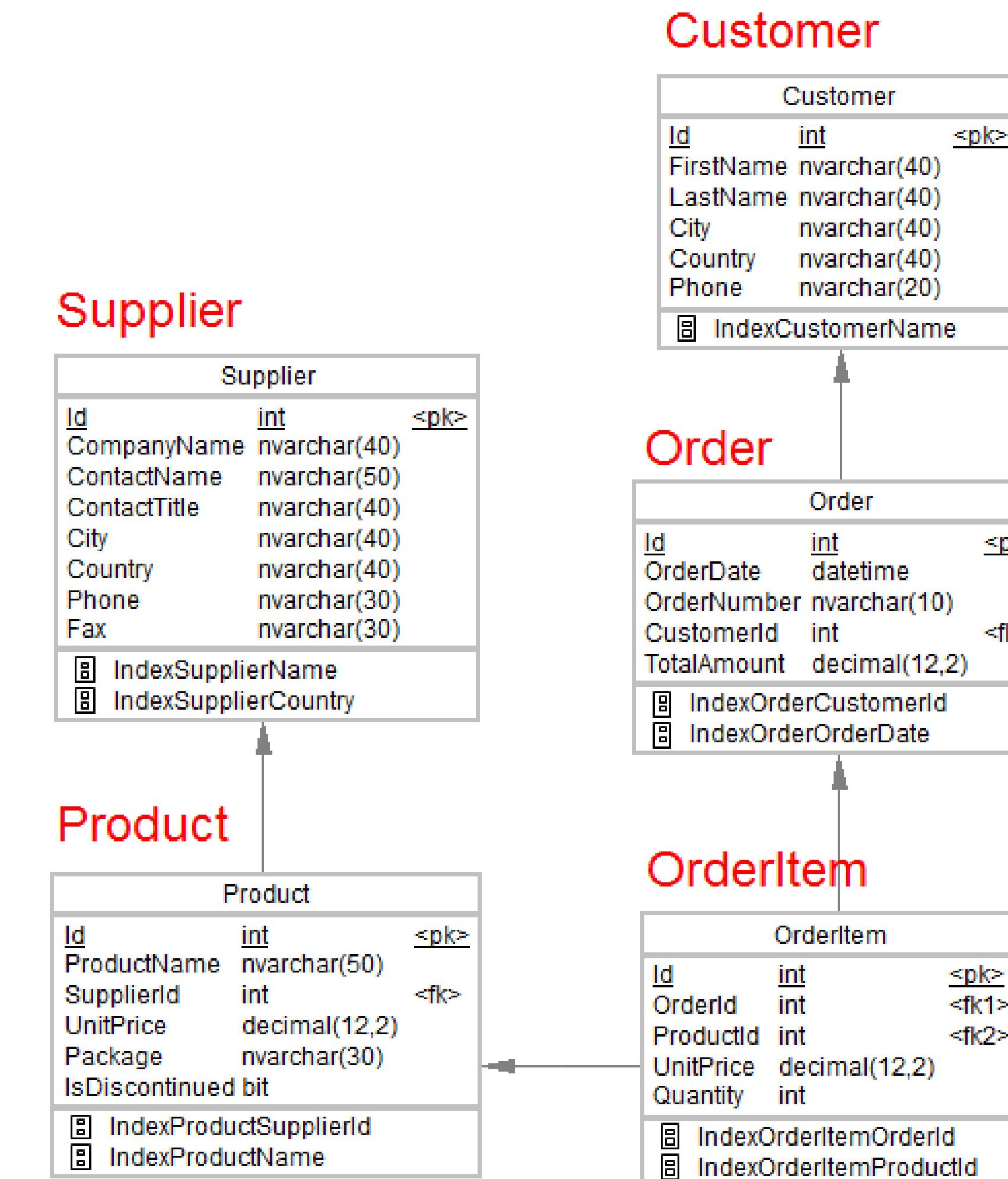
Information is stored in **tables**, and relations between data materialize in relations between tables



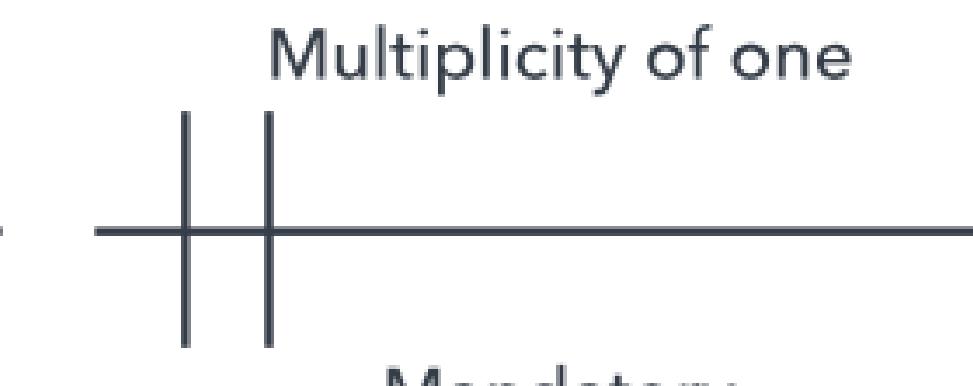
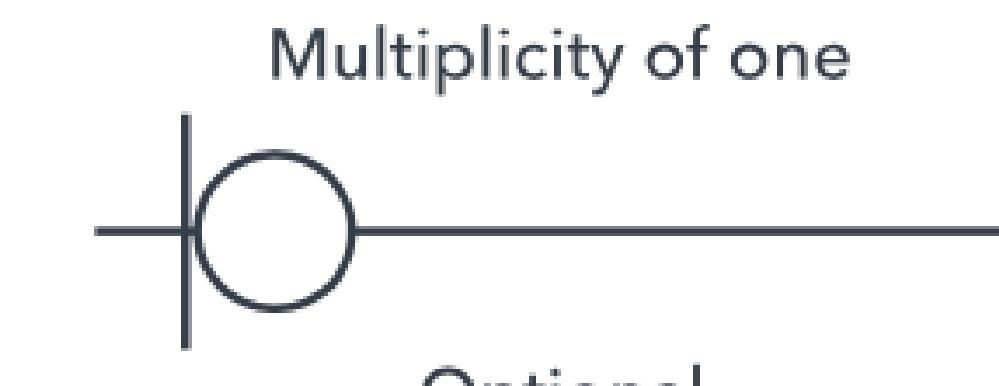
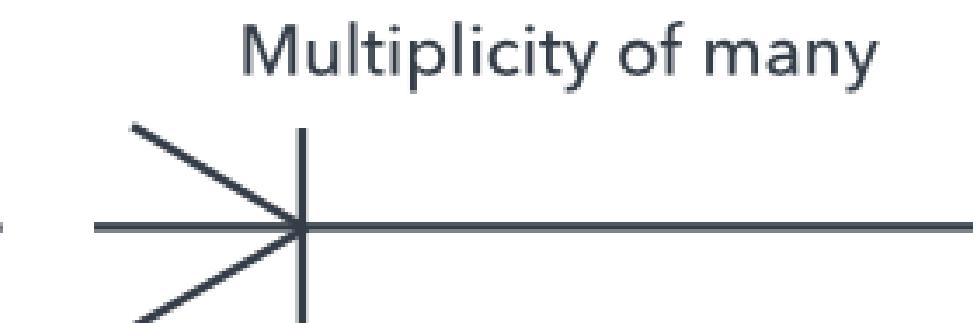
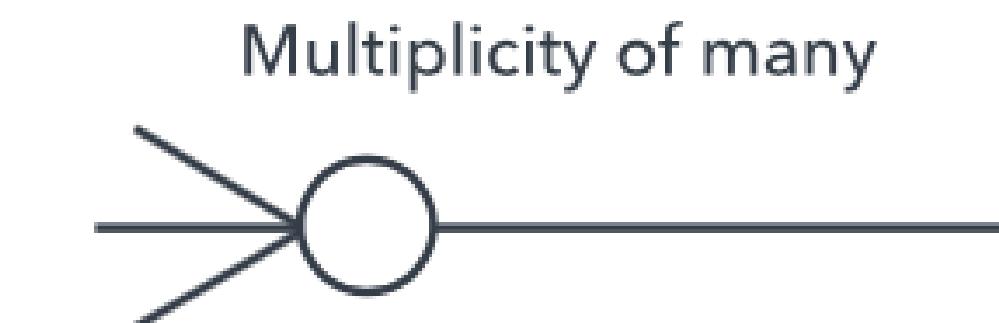
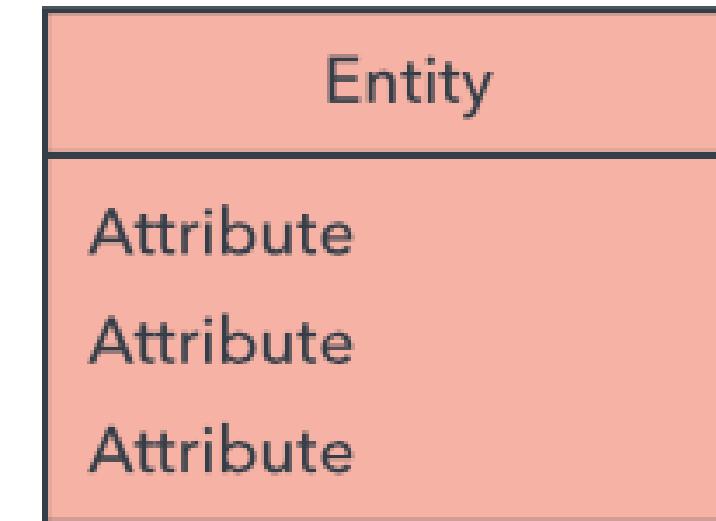
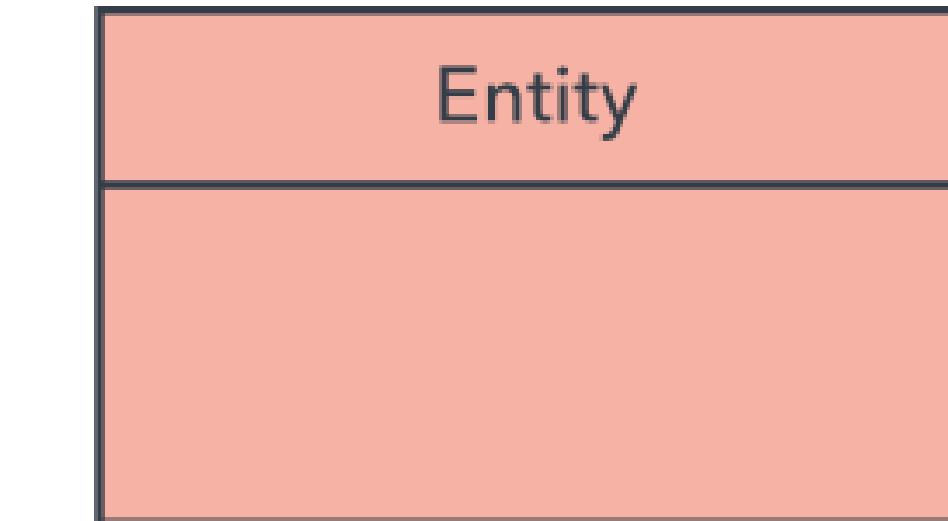
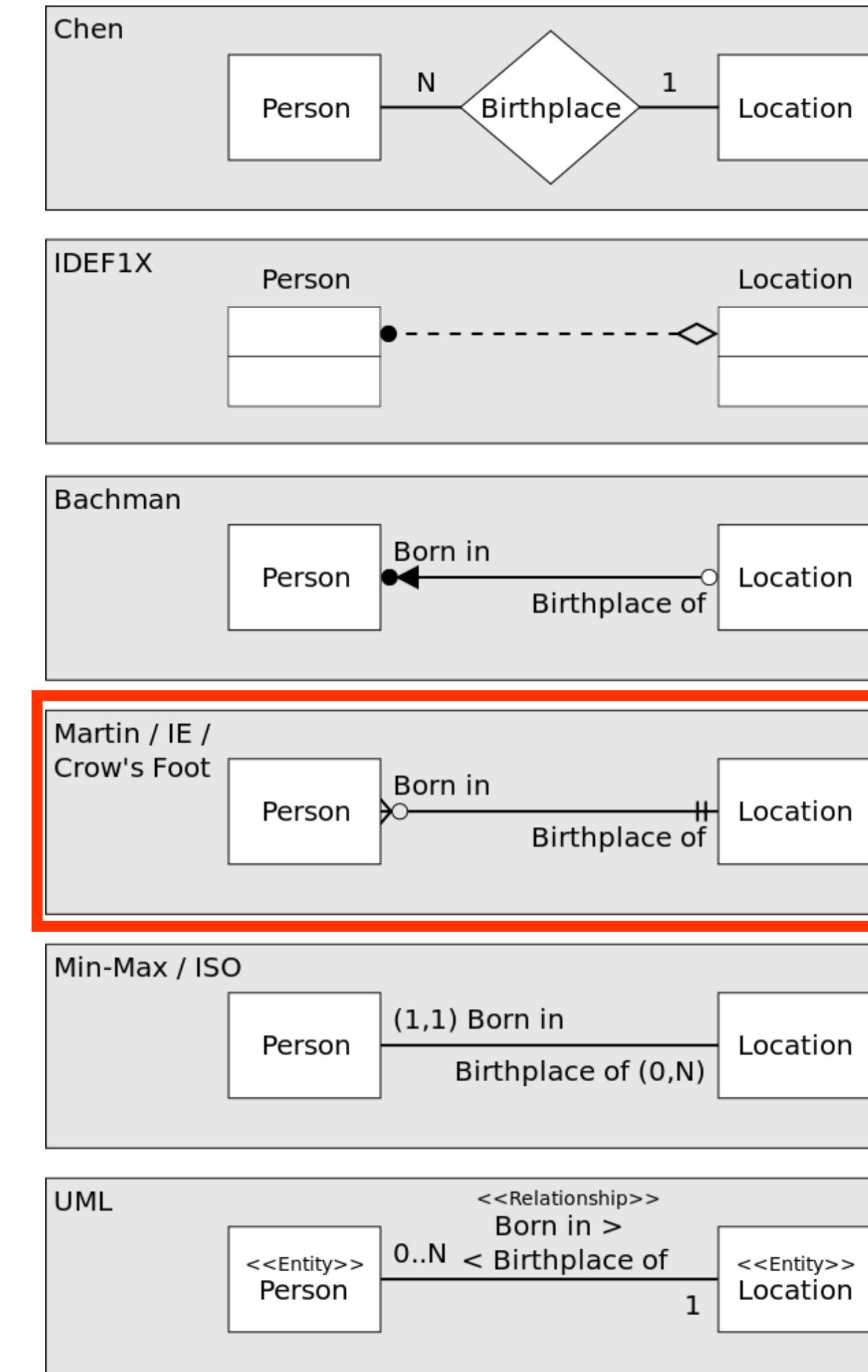
PostgreSQL



ORACLE
D A T A B A S E



Crow's foot Notation for the ERD



- Is SQL dead?
- Create a database
- Create tables
- Read, Update and Delete

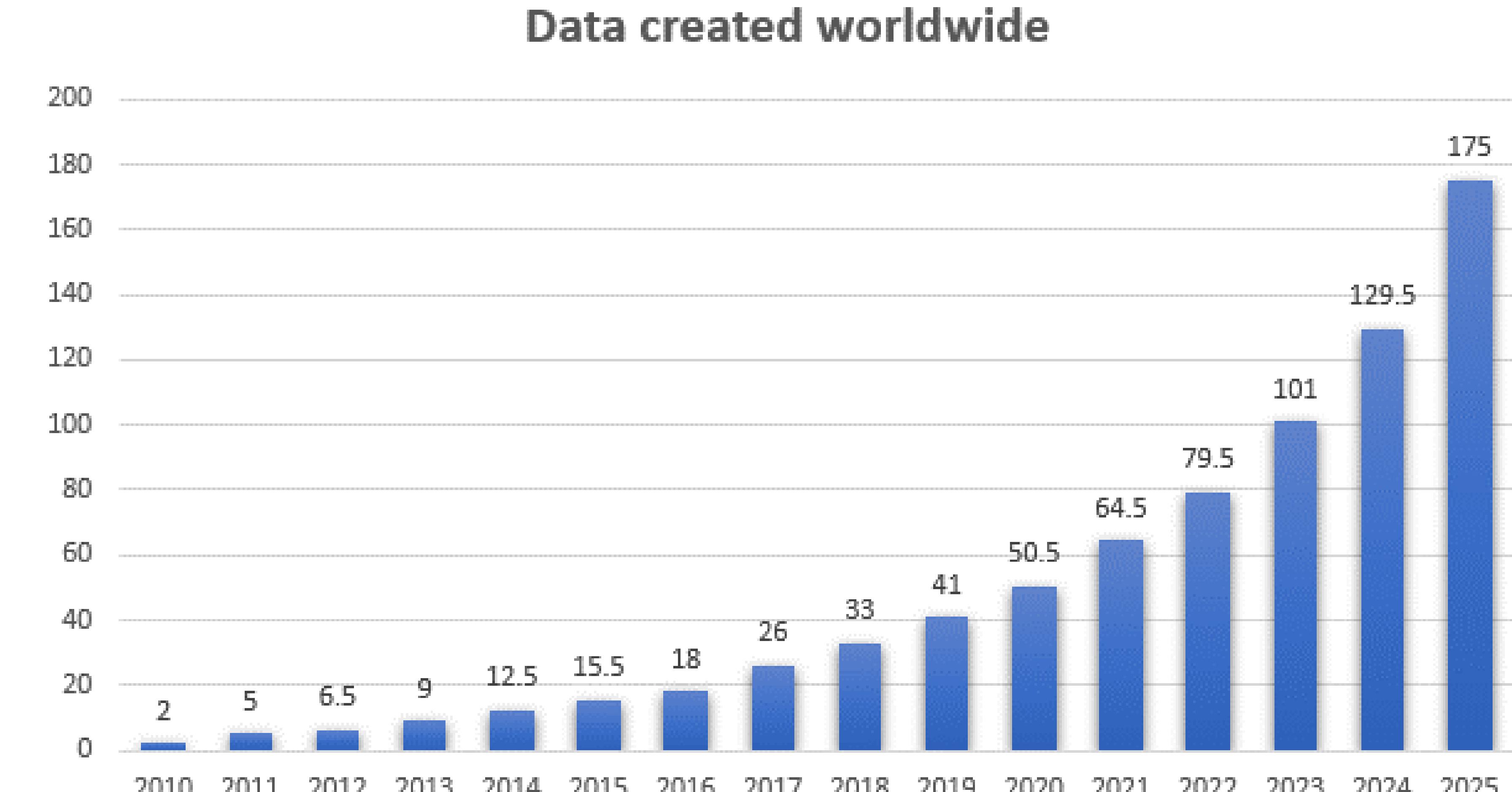
Is SQL dead?



Image source: <https://www.rd.com/wp-content/uploads/2018/09/Dinosaur.jpg>

Is SQL dead?

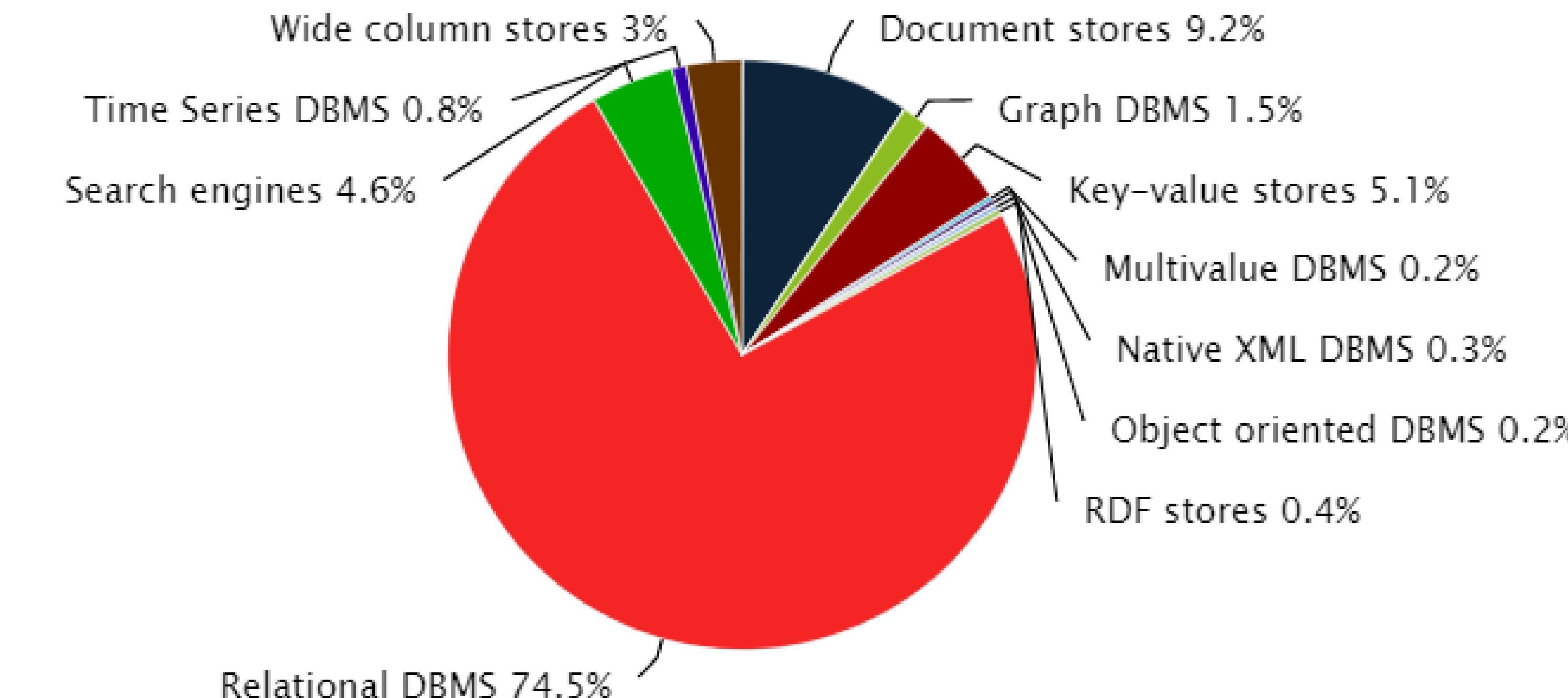
(in
zettabytes)



Source: <https://www.statista.com/statistics/871513/worldwide-data-created/>

Is SQL dead?

Ranking scores per category in percent, August 2020



© 2020, DB-Engines.com

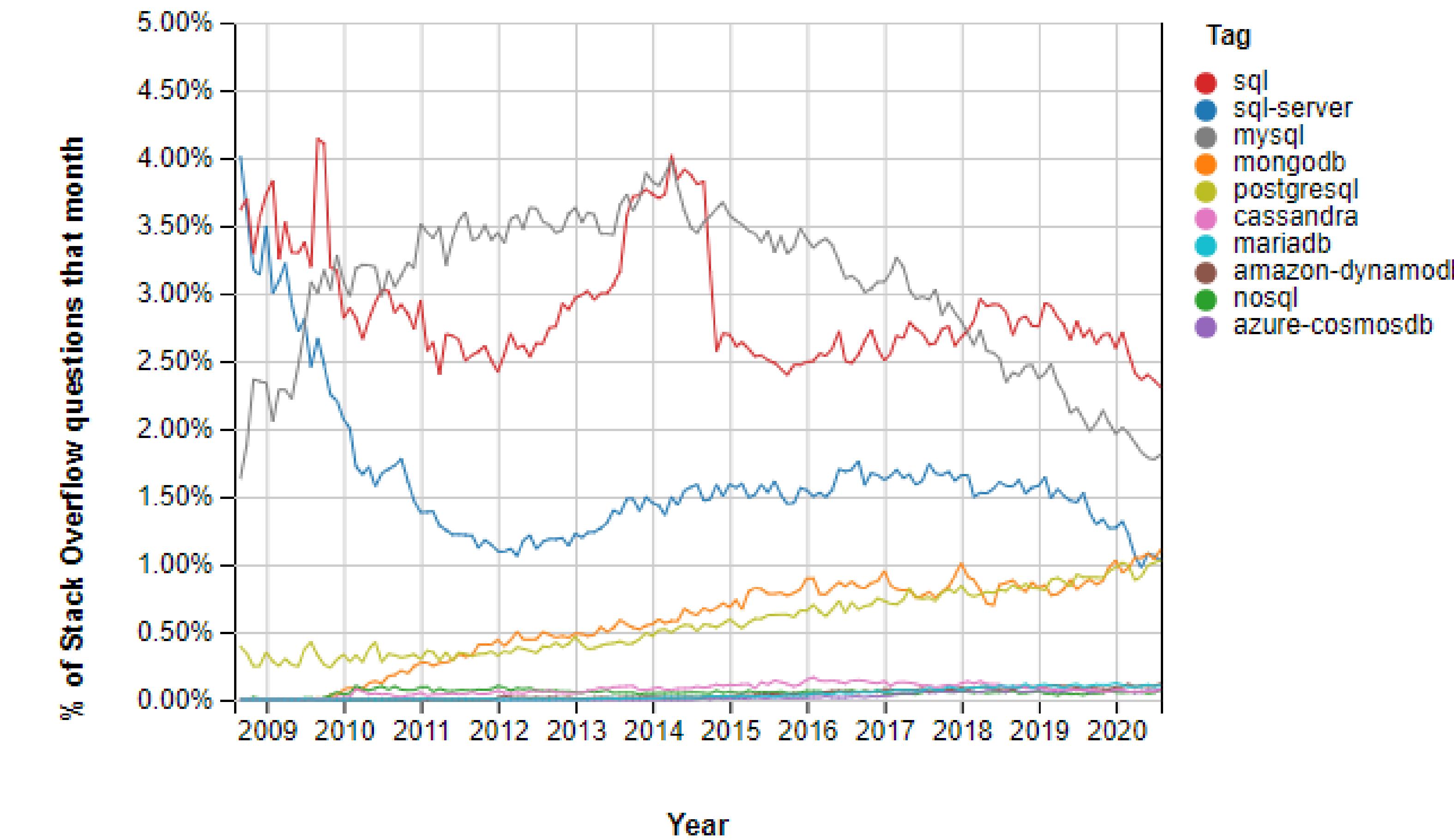
Source: https://db-engines.com/en/ranking_categories

Is SQL dead?

	Rank			DBMS	Database Model	Score		
	Oct 2018	Sep 2018	Oct 2017			Oct 2018	Sep 2018	Oct 2017
	1.	1.	1.	Oracle 	Relational DBMS	1319.27	+10.15	-29.54
	2.	2.	2.	MySQL 	Relational DBMS	1178.12	-2.36	-120.71
	3.	3.	3.	Microsoft SQL Server 	Relational DBMS	1058.33	+7.05	-151.99
	4.	4.	4.	PostgreSQL 	Relational DBMS	419.39	+12.97	+46.12
	5.	5.	5.	MongoDB 	Document store	363.19	+4.39	+33.79
	6.	6.	6.	DB2 	Relational DBMS	179.69	-1.38	-14.90
	7.	↑ 8.	↑ 9.	Redis 	Key-value store	145.29	+4.35	+23.24
	8.	↓ 7.	↑ 10.	Elasticsearch 	Search engine	142.33	-0.28	+22.09
	9.	9.	↓ 7.	Microsoft Access	Relational DBMS	136.80	+3.41	+7.35
	10.	10.	↓ 8.	Cassandra 	Wide column store	123.39	+3.83	-1.40

Source: <https://db-engines.com/en/ranking>

Is SQL dead?



Source: <https://insights.stackoverflow.com/trends?tags=sql-server%2Cmongodb%2Cnosql%2Csql%2Cazure-cosmosdb%2Camazon-dynamodb%2Ccassandra%2Cmysql%2Cpostgresql%2Cmariadb>

Key Characteristics of SQL

- First version, SQL-86 in 1986, most recent version in 2011 (SQL:2011)
- Accepted by the American National Standards Institute (ANSI) in 1986 and by the International Organization for Standardization (ISO) in 1987
- Each vendor provides its own implementation (also called SQL dialect) of SQL

Key Characteristics of SQL

- Set-oriented and declarative
- Free-form language
- Case insensitive
- Can be used both interactively from a command prompt or executed by a program

Key Characteristics of SQL

Example: Executed from MySQL workbench

The screenshot shows the MySQL Workbench interface with a query window titled "Query 1". The query is:

```
1  SELECT P.PRODNR, P.PRODNAME FROM PRODUCT P
2    WHERE 1 <
3        (SELECT COUNT(*)
4         FROM PO_LINE POL
5         WHERE P.PRODNR = POL.PRODNR)
```

The results grid displays the following data:

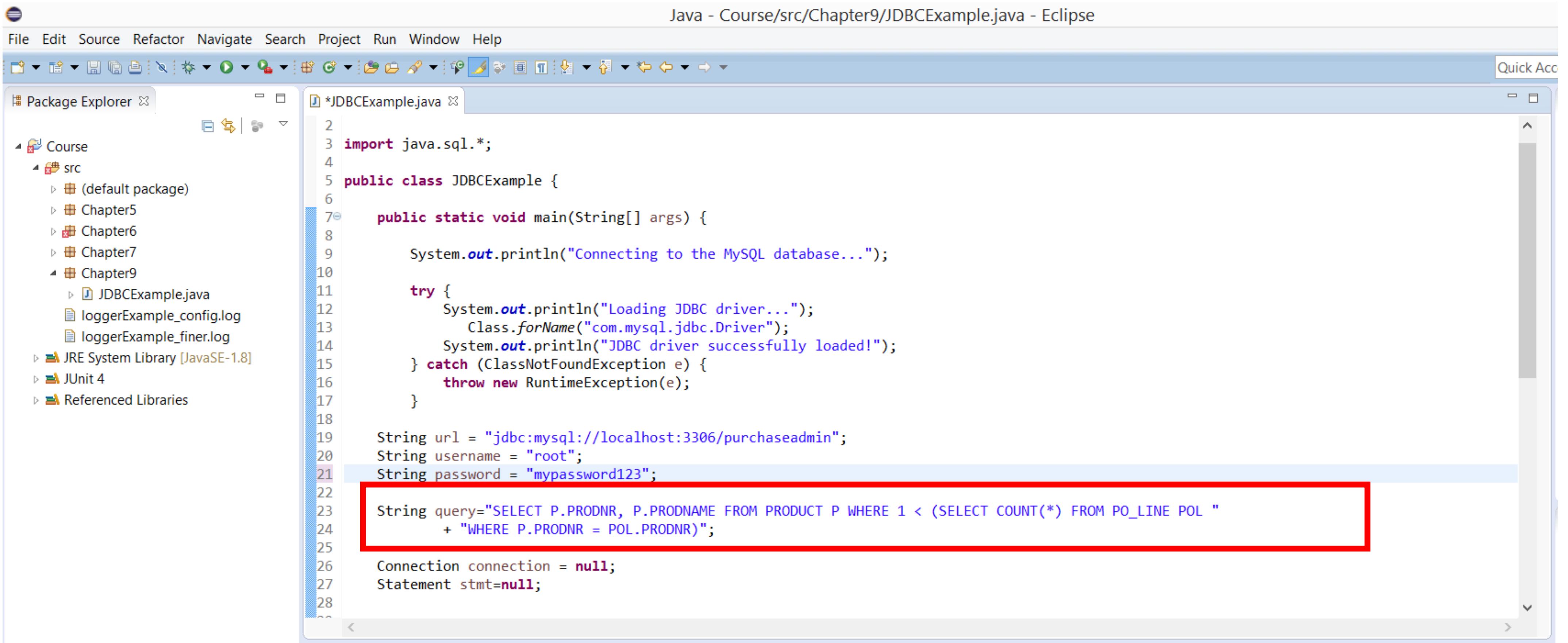
PRODNR	PRODNAME
0212	Billecart-Salmon, Brut Réserve, 2014
0977	Chateau Batailley, Grand Cru Classé, 1975
0900	Chateau Cheval Blanc, Saint Emilion, Grand Cru Classé, 1972
0306	Chateau Coupe Roses, Granaxa, 2011
0783	Clos D'Opleeuw, Chardonnay, 2012
0668	Gallo Family Vineyards, Grenache, 2014
0766	GH Mum, Brut, 2012
0178	Meerdael, Methode Traditionnelle Chardonnay, 2014
HULL	HULL

The output pane shows the executed query and its duration:

Time	Action	Message	Duration / Fetch
16:47:29	SELECT P.PRODNR, P.PRODNAME FROM PRODUCT P WHERE 1 < (SELECT COUNT(*) FROM PO_LINE P ...	8 row(s) returned	0.000 sec / 0.000 sec

Key Characteristics of SQL

Example: Embedded in java code.



The screenshot shows the Eclipse IDE interface with the title "Java - Course/src/Chapter9/JDBCExample.java - Eclipse". The left side features the "Package Explorer" view, which displays a project structure under the "Course" package. The "src" folder contains several chapters (Chapter5, Chapter6, Chapter7, Chapter9) and their corresponding Java files (JDBCExample.java). Other files like loggerExample_config.log and loggerExample_finer.log are also listed. The "JRE System Library [JavaSE-1.8]" and "JUnit 4" are also visible. The main editor window on the right shows the Java code for "JDBCExample.java". A red rectangular box highlights the SQL query at the bottom of the code. The code itself is as follows:

```
2
3 import java.sql.*;
4
5 public class JDBCExample {
6
7     public static void main(String[] args) {
8
9         System.out.println("Connecting to the MySQL database...");
10
11     try {
12         System.out.println("Loading JDBC driver...");
13         Class.forName("com.mysql.jdbc.Driver");
14         System.out.println("JDBC driver successfully loaded!");
15     } catch (ClassNotFoundException e) {
16         throw new RuntimeException(e);
17     }
18
19     String url = "jdbc:mysql://localhost:3306/purchaseadmin";
20     String username = "root";
21     String password = "mypassword123";
22
23     String query="SELECT P.PRODNR, P.PRODNAME FROM PRODUCT P WHERE 1 < (SELECT COUNT(*) FROM PO_LINE POL "
24     + "WHERE P.PRODNR = POL.PRODNR)";
25
26     Connection connection = null;
27     Statement stmt=null;
```

Recommended tutorial!!

Many of the examples are from:

<https://www.mysqltutorial.org/>



We encourage you to visit the site and explore further!! There is a lot to learn!!

Create a Database



Check existing
databases

SHOW DATABASES;

Create Database

CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name]

A MySQL character set is a set
of characters that are legal in a
string

A MySQL collation is a set of rules used
to compare characters in a
particular character set.

```
use Database | USE database_name;
```

Drop Database | **DROP DATABASE [IF EXISTS] database_name;**

See tables present
in the database

SHOW TABLES;

See the columns and
properties of a table

DESCRIBE database_name

Create Tables

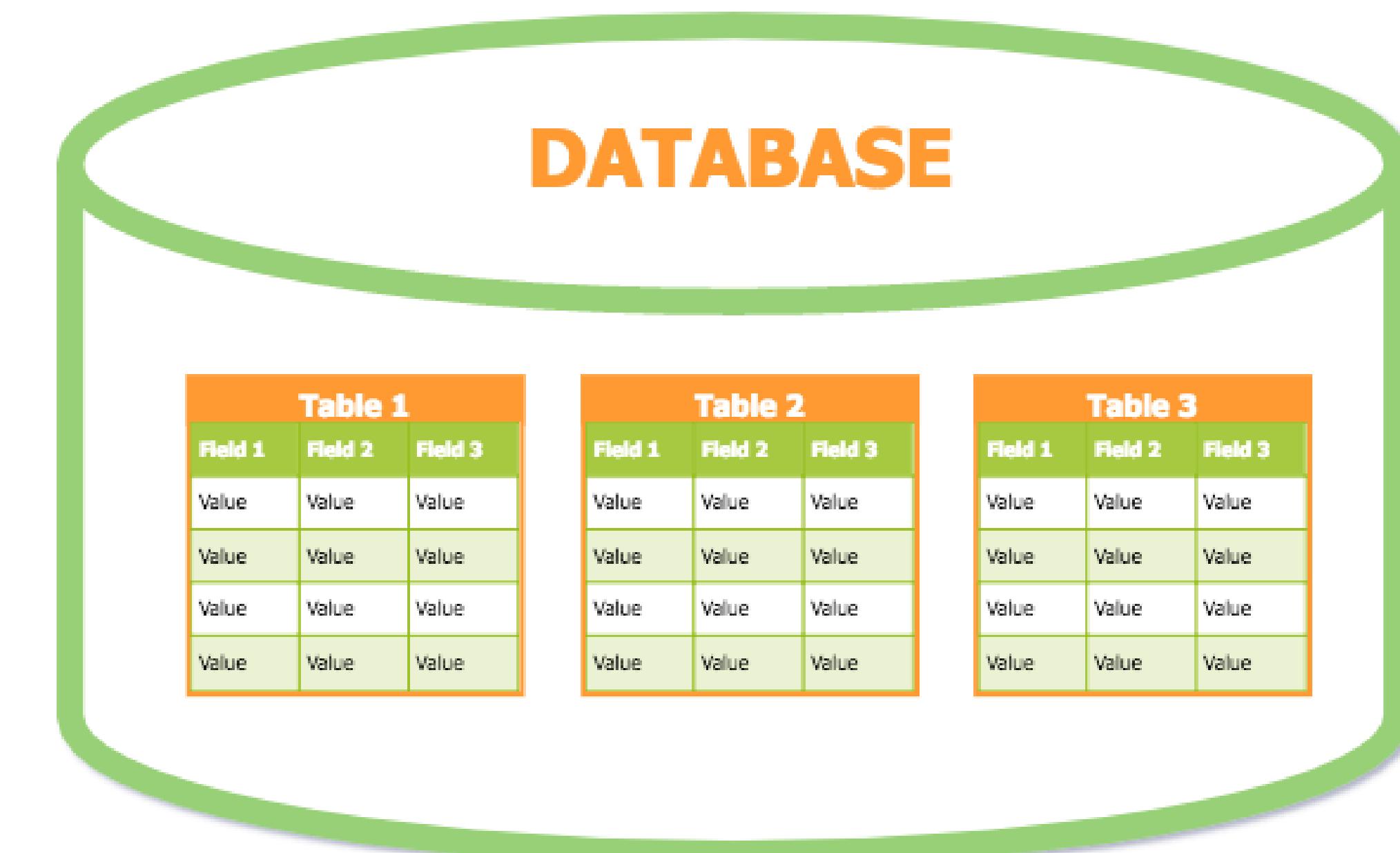


Image source: <https://database.guide/database-tutorial-part-1-about-databases-creating-databases-tables/>

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(
    col1 data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]
    col2 data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]
    col3 data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]
) ENGINE=storage_engine;
```

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(  
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]  
) ENGINE=storage_engine;
```

The column_name specifies the name of the column.
Each column has a specific data type and maximum length e.g.,VARCHAR(255)

Table creation – data types

The Data Type specifies what type of data the column can hold.

CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TEXT	Holds a string with a maximum length of 65,535 characters
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
FLOAT(size,d)	A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DOUBLE(size,d)	A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(  
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]  
) ENGINE=storage_engine;
```

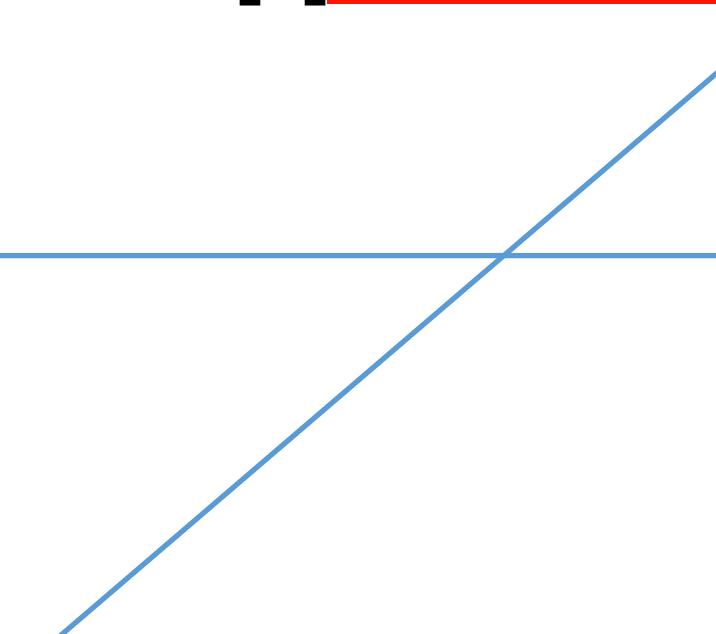


The **NOT NULL** indicates that the column does not allow **NULL**

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(  
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]  
) ENGINE=storage_engine;
```



The **DEFAULT** value is used to specify the default value of the column

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(  
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]  
) ENGINE=storage_engine;
```

The **AUTO_INCREMENT** indicates that the value of the column is incremented by one automatically whenever a new row is inserted into the table. Each table has one and only one **AUTO_INCREMENT** column

Table creation

Create a Table

```
CREATE TABLE [IF NOT EXISTS] table_name(  
column_name data_type(length) [NOT NULL] [DEFAULT value] [AUTO_INCREMENT]  
) ENGINE=storage_engine;
```

Third, you can optionally specify the storage engine for the table in the **ENGINE** clause. You can use any storage engine such as InnoDB and MyISAM. If you don't explicitly declare the storage engine, MySQL will use InnoDB by default.

Table creation – primary key

Primary Keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length) [PRIMARY KEY]  
    col2 data_type(length) [PRIMARY KEY]  
    col3 data_type(length) [PRIMARY KEY]  
);
```

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    col2 data_type(length)  
    col3 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
);
```

Table creation - example

Person

pid int
firstname varchar(25)
surname varchar(25)

```
CREATE TABLE Person (
    pid int AUTO_INCREMENT NOT NULL,
    firstname varchar(25) NOT NULL,
    surname varchar(25) NOT NULL,
);
```

What are we missing?

Person

pid int
firstname varchar(25)
surname varchar(25)

```
CREATE TABLE Person (
    pid int AUTO_INCREMENT NOT NULL,
    firstname varchar(25) NOT NULL,
    surname varchar(25) NOT NULL,
    PRIMARY KEY (pid)
);
```

What are we missing? The Primary Key

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
PRIMARY KEY(col1, col2, ...)  
CONSTRAINT constraint_name  
FOREIGN KEY foreign_key_name (columns)  
REFERENCES parent_table(columns)  
ON DELETE action  
ON UPDATE action  
);
```

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
CONSTRAINT constraint_name  
    FOREIGN KEY foreign_key_name (columns)  
    REFERENCES parent_table(columns)  
    ON DELETE action  
    ON UPDATE action  
);
```

The **CONSTRAINT** clause allows you to define constraint name for the foreign key constraint. If you omit it, MySQL will generate a name automatically.

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
    CONSTRAINT constraint_name  
    FOREIGN KEY foreign key name (columns)  
    REFERENCES parent_table(columns)  
    ON DELETE action  
    ON UPDATE action  
);
```

The **FOREIGN KEY** clause specifies the columns in the child table that refers to primary key columns in the parent table. You can put a foreign key name after **FOREIGN KEY** clause or leave it to let MySQL create a name for you. Notice that MySQL automatically creates an index with the `foreign_key_name` name.

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
    CONSTRAINT constraint_name  
    FOREIGN KEY foreign_key_name (columns)  
    REFERENCES parent_table(columns)  
    ON DELETE action  
    ON UPDATE action  
);
```

The **REFERENCES** clause specifies the parent table and its columns to which the columns in the child table refer. The number of columns in the child table and parent table specified in the **FOREIGN KEY** and **REFERENCES** must be the same.

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
    CONSTRAINT constraint_name  
    FOREIGN KEY foreign_key_name (columns)  
    REFERENCES parent_table(columns)  
    ON DELETE action  
    ON UPDATE action  
);
```

possible actions :
CASCADE, SET NULL, NO ACTION

The **ON DELETE** clause allows you to define what happens to the records in the child table when the records in the parent table are deleted. If you omit the **ON DELETE** clause and delete a record in the parent table that has records in the child table refer to, MySQL will reject the deletion.

Table creation - Foreign keys

```
CREATE TABLE [IF NOT EXISTS] table_name(  
    col1 data_type(length)  
    PRIMARY KEY(col1, col2, ...)  
    CONSTRAINT constraint_name  
    FOREIGN KEY foreign_key_name (columns)  
    REFERENCES parent_table(columns)  
    ON DELETE action  
ON UPDATE action  
);
```

possible actions :
CASCADE, SET NULL, NO ACTION

The **ON UPDATE** clause enables you to specify what happens to the rows in the child table when rows in the parent table are updated. You can omit the **ON UPDATE** clause to let MySQL reject any updates to the rows in the child table when the rows in the parent table are updated.

Table creation - Foreign keys - Example

```
CREATE TABLE category(
```

```
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
```

```
    categoryName VARCHAR(100) NOT NULL);
```

```
CREATE TABLE products(
```

```
    productId INT AUTO_INCREMENT PRIMARY KEY,
```

```
    productName varchar(100) not null,
```

```
    categoryId INT NOT NULL,
```

```
    CONSTRAINT fk_category
```

```
    FOREIGN KEY (categoryId)
```

```
        REFERENCES category(categoryId)
```

```
        ON UPDATE SET NULL
```

```
        ON DELETE SET NULL );
```

Source: <https://www.mysqltutorial.org/mysql-foreign-key/>

Drop a table

DROP TABLE [IF EXISTS] table_name [, table_name] ...

The **DROP TABLE** statement removes a table and its data permanently from the database. In MySQL, you can also remove multiple tables using a single **DROP TABLE** statement, each table is separated by a comma (,).

Drop a table

DROP TABLE [IF EXISTS] table_name [, table_name] ...

The **IF EXISTS** addition helps you prevent from attempt of removing non-existent tables. When you use **IF EXISTS** addition, MySQL generates a NOTE, which can be retrieved by using the SHOW WARNING statement. It is important to note that the **DROP TABLE** statement removes all existing tables and issues an error message or a NOTE when you have a non-existent table in the list.

Create, Read, Update and Delete

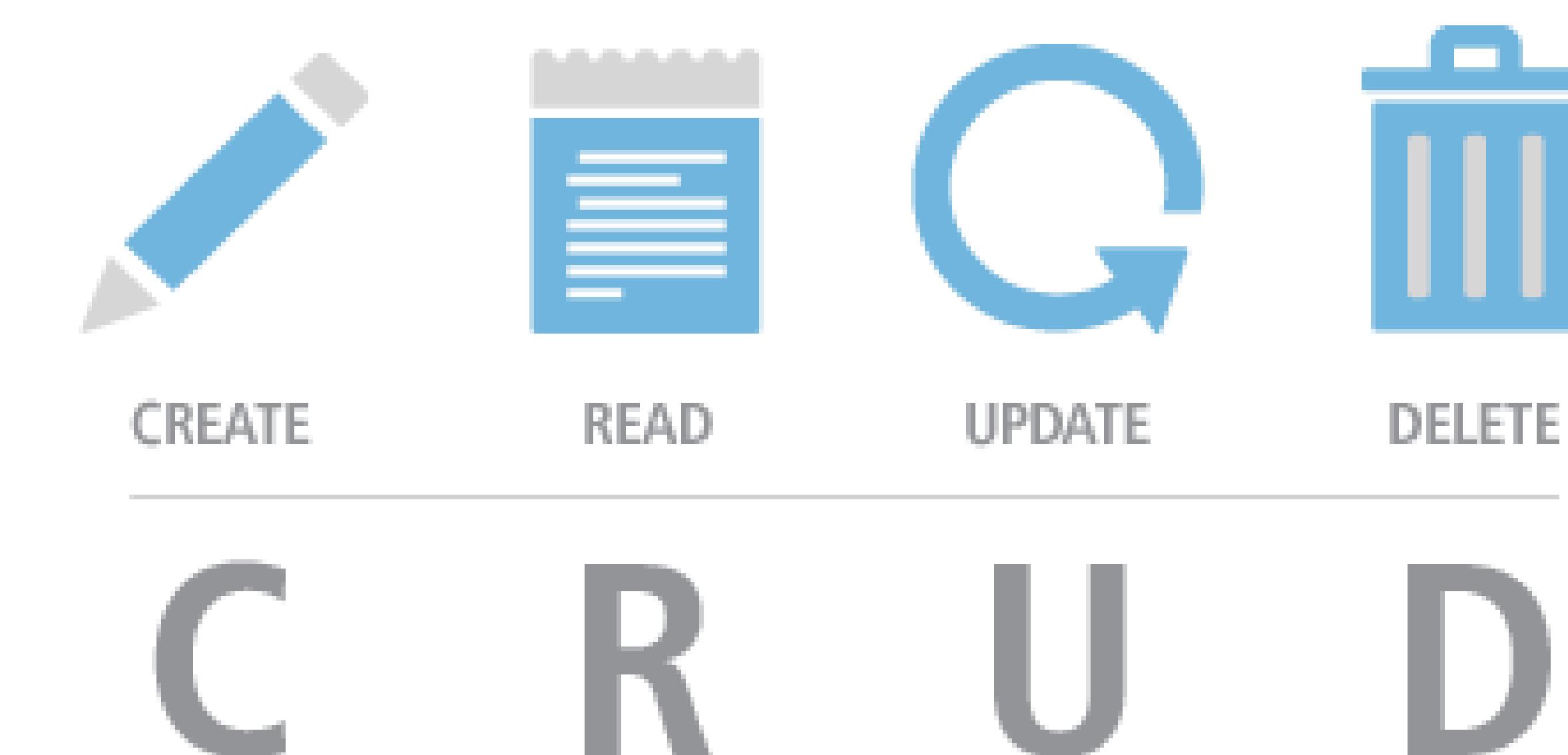


Image source: <https://clofusinnovations.com/blog/post/using-mysql-for-crud-operations-with-nodejs/>

Insert (Create)

```
INSERT INTO table_name(c1,c2,...)  
VALUES(v1,v2,..);
```

Insert example

Insert one task into the table **tasks**:

```
INSERT INTO tasks(title,priority)  
VALUES('Learn MySQL INSERT Statement',1);
```

	task_id	title	start_date	due_date	priority	description
▶	1	Learn MySQL INSERT Statement	NULL	NULL	1	NULL

Source: <https://www.mysqltutorial.org/mysql-insert-statement.aspx>

Read

```
SELECT
    column_1, column_2, ...
FROM
    table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;
```

Do not worry too much about joins for now, we will see more details next class

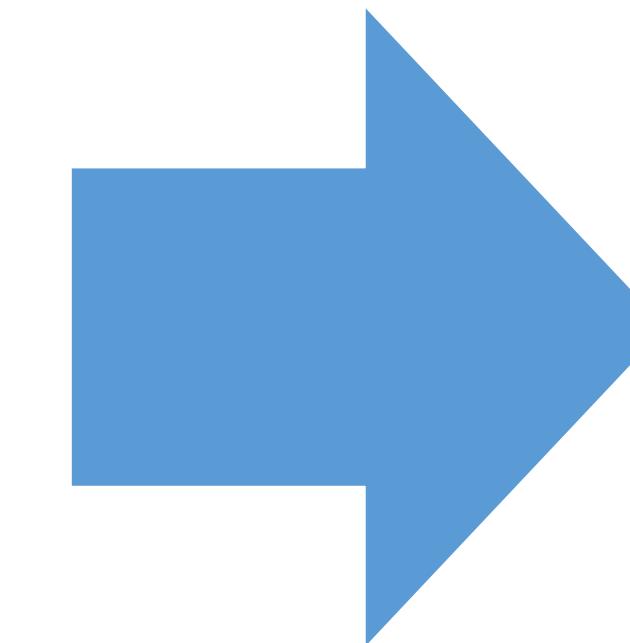
Read example

How to get the last name, first name, and job title from employee table?

	employeeNum	lastName	firstName	extension	email	officeCode	reportsTo	jobTitle
▶	1002	Murphy	Diane	x5800	dmurphy@classicmodelcars.com	1	NULL	President
	1056	Patterson	Mary	x4611	mpatterso@classicmodelcars.com	1	1002	VP Sales
	1076	Fimelli	Jeff	x9273	jfimelli@classicmodelcars.com	1	1002	VP Marketing
	1088	Patterson	William	x4871	wpatterson@classicmodelcars.com	6	1056	Sales Manager (APAC)
	1102	Bondur	Gerard	x5408	gbondur@classicmodelcars.com	4	1056	Sale Manager (EMEA)
	1143	Bow	Anthony	x5428	abow@classicmodelcars.com	1	1056	Sales Manager (NA)
	1165	Jennings	Leslie	x3291	ljennings@classicmodelcars.com	1	1143	Sales Rep
	1166	Thompson	Leslie	x4065	lthompson@classicmodelcars.com	1	1143	Sales Rep
	1188	Fimelli	Julie	x2173	jfimelli@classicmodelcars.com	2	1143	Sales Rep
	1216	Patterson	Steve	x4334	spatterson@classicmodelcars.com	2	1143	Sales Rep
	1286	Tseng	Foon Yue	x2248	ftseng@classicmodelcars.com	3	1143	Sales Rep
	1323	Vanauf	George	x4102	gvanauf@classicmodelcars.com	3	1143	Sales Rep
	1337	Bondur	Loui	x6493	lbondur@classicmodelcars.com	4	1102	Sales Rep
	1370	Hemandez	Gerard	x2028	ghemande@classicmodelcars.com	4	1102	Sales Rep
	1401	Castillo	Pamela	x2759	pcastillo@classicmodelcars.com	4	1102	Sales Rep
	1501	Rott	I am	x2311	lhrott@classicmodelcars.com	7	1102	Sales Rep

Read example

```
SELECT
    lastname,
    firstname,
    jobtitle
FROM
    employees;
```



	lastname	firstname	jobtitle
▶	Murphy	Diane	President
	Patterson	Mary	VP Sales
	Fimelli	Jeff	VP Marketing
	Patterson	William	Sales Manager (APAC)
	Bondur	Gerard	Sale Manager (EMEA)
	Bow	Anthony	Sales Manager (NA)
	Jennings	Leslie	Sales Rep
	Thompson	Leslie	Sales Rep
	Fimelli	Julie	Sales Rep
	Patterson	Steve	Sales Rep
	Tseng	Foon Yue	Sales Rep
	Vanauf	George	Sales Rep
	Bondur	Loui	Sales Rep
	Hemandez	Gerard	Sales Rep

Update

```
UPDATE [LOW_PRIORITY] [IGNORE] table_name
SET
column_name1 = expr1,
column_name2 = expr2,
[WHERE
    condition];
```

Update example

You want to update the email of an employee in table **employees**:

```
UPDATE employees
SET
    email = 'mary.patterson@classicmodelcars.com'
WHERE
    employeeNumber = 1056;
```

	firstname	lastname	email
▶	Mary	Patterson	mary.patterson@classicmodelcars.com

Source: <https://www.mysqltutorial.org/mysql-update-data.aspx>

Delete

```
DELETE FROM table_name  
WHERE condition;
```

Delete example

Delete the first 5 **customers** in France:

```
DELETE FROM customers  
WHERE country = 'France'  
LIMIT 5;
```

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

Let's have
some fun!

END OF LECTURE 3

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa



Information
Management
School

Gestão e armazenamento de dados

Lecture 4

Structured Query Language (SQL) – Part 2

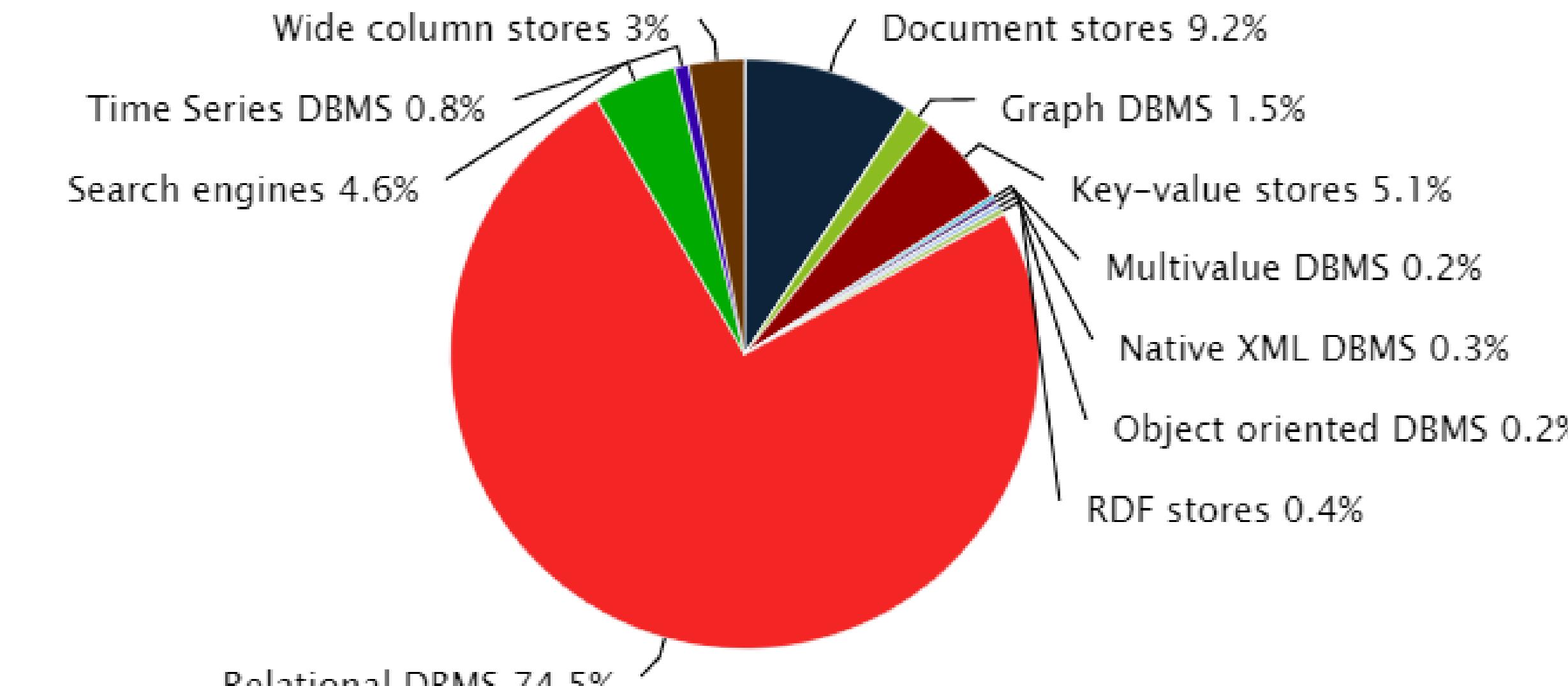
Previous class



Previous class

Is SQL dead?

Ranking scores per category in percent, August 2020



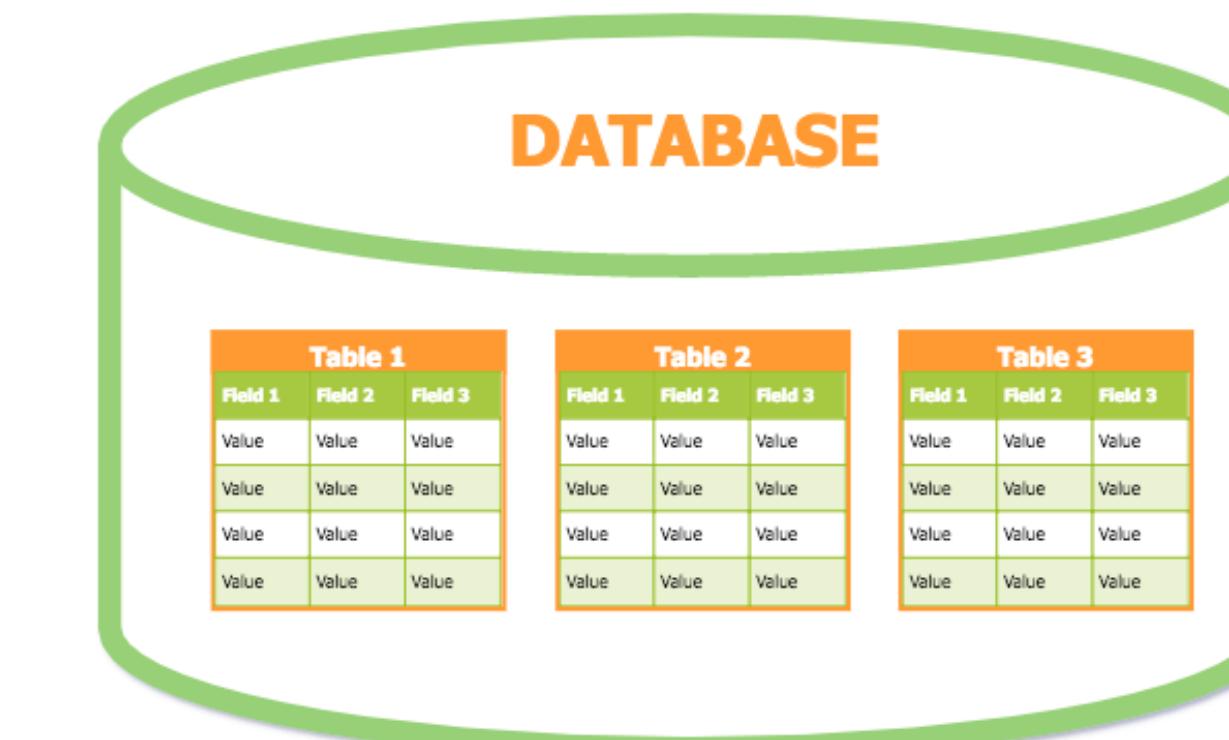
© 2020, DB-Engines.com

Source: https://db-engines.com/en/ranking_categories

Create a Database



Create Tables



CREATE READ UPDATE DELETE

C R U D

Today

- SQL clauses and operators
- SQL Joins

Clauses and operators

LIMIT
DISTINCT
BETWEEN
[NOT] LIKE
ORDER BY

LIMIT clause

The LIMIT clause is used in the SELECT statement to constrain the number of rows to return.

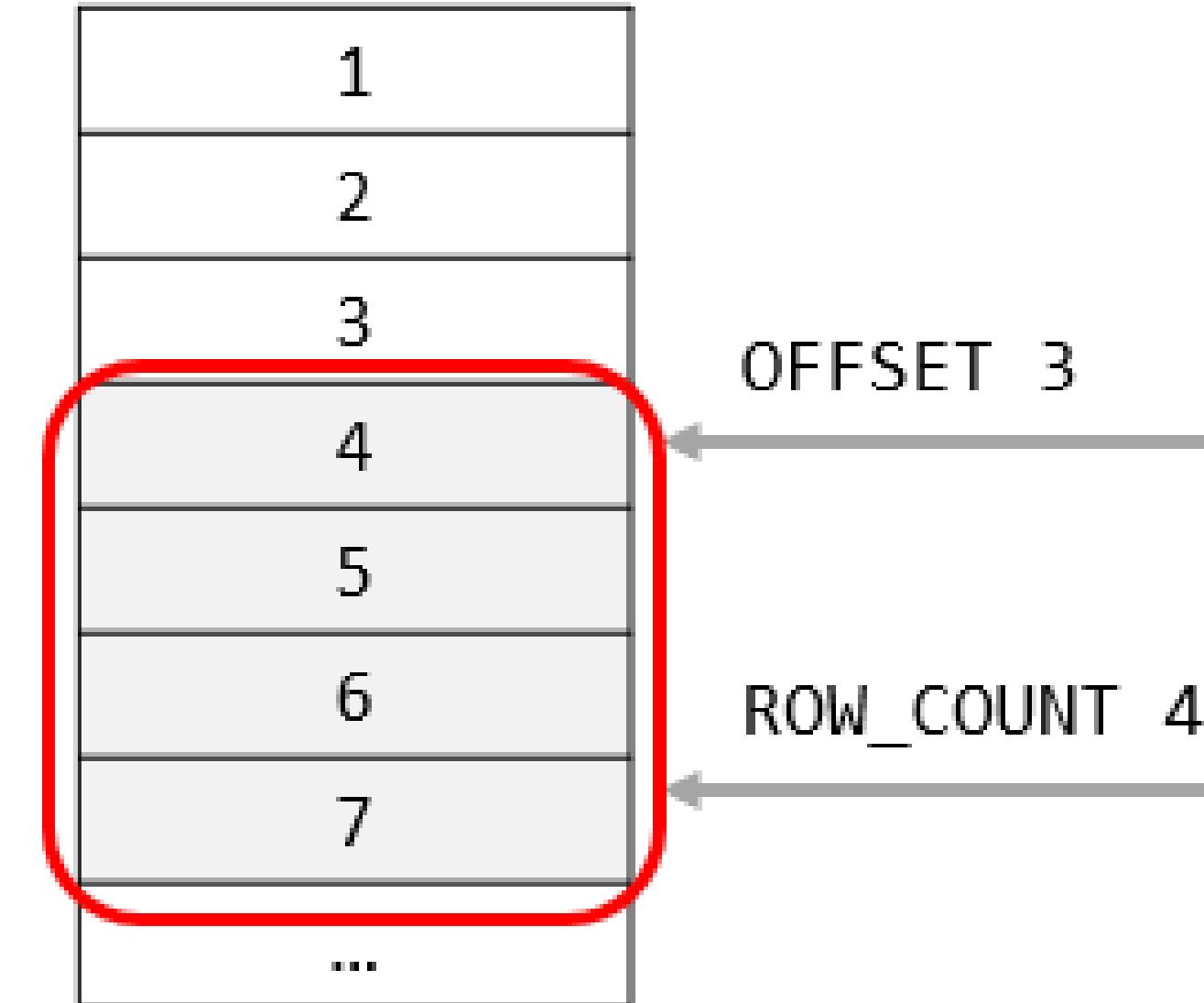


Source: <https://www.mysqltutorial.org/mysql-limit.aspx>

LIMIT clause

```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
```

```
SELECT n FROM t
ORDER BY n
LIMIT 3, 4;
```



- The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1.
- The row_count specifies the maximum number of rows to return.

Source: <https://www.mysqltutorial.org/mysql-limit.aspx>

Examples

Extract the first 10 rows from SALARIES WHERE salary is higher than 8000 dollars.

```
SELECT * FROM salaries WHERE salary > 8000 LIMIT 10;
```



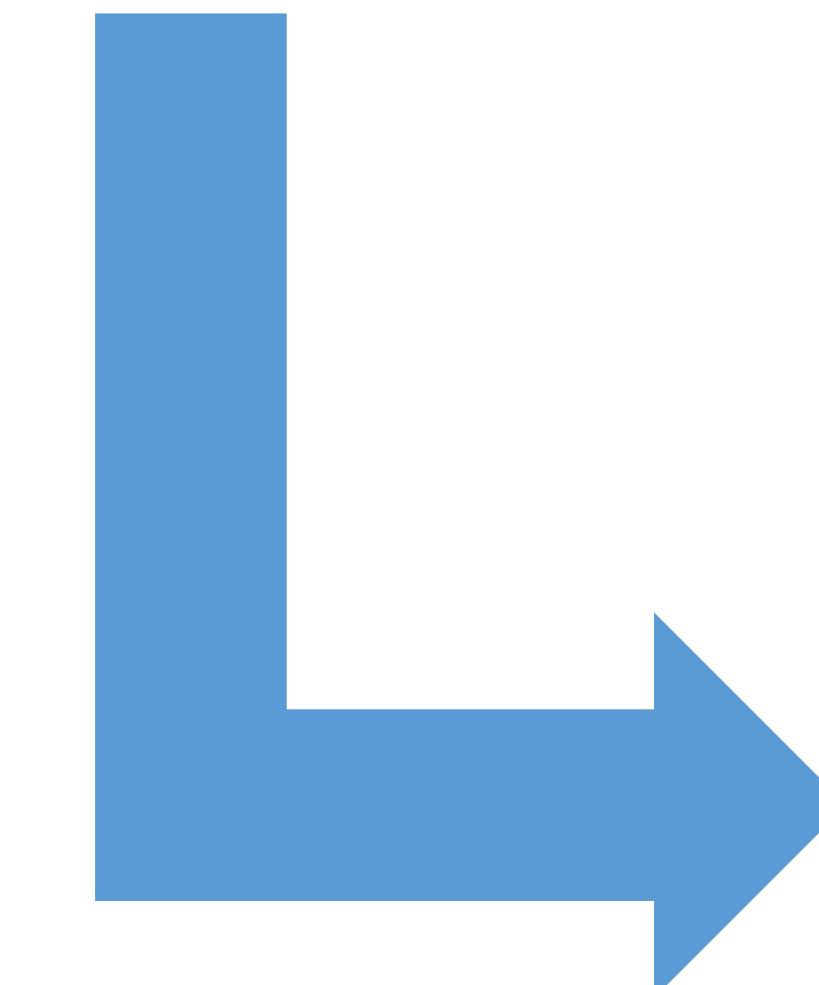
emp_no	salary	from_date	to_date
10001	60117	1986-06-26	1987-06-26
10001	62102	1987-06-26	1988-06-25
10001	66074	1988-06-25	1989-06-25
10001	66596	1989-06-25	1990-06-25
10001	66961	1990-06-25	1991-06-25
10001	71046	1991-06-25	1992-06-24
10001	74333	1992-06-24	1993-06-24
10001	75286	1993-06-24	1994-06-24
10001	75994	1994-06-24	1995-06-24
10001	76884	1995-06-24	1996-06-23

DISTINCT clause

Use DISTINCT to remove duplicates.

Example: Get the first 10 unique names.

```
SELECT DISTINCT e.first_name FROM employees AS e LIMIT 10;
```



first_name

Georgi

Bezalel

Parto

Chirstian

Kyoichi

Anneke

Tzvetan

Saniya

Sumant

Duangkaew

BETWEEN clause

The BETWEEN operator is a logical operator that allows you to specify whether a value is within a range or not.

Example:

Get the employees that were hired between 1999-01-01 AND 1999-12-31

BETWEEN clause - Example

```
SELECT e.emp_no, e.first_name, e.last_name, e.hire_date  
FROM employees AS e  
WHERE e.hire_date BETWEEN '1999-01-01' AND '1999-12-31';
```



emp_no	first_name	last_name	hire_date
10019	Lillian	Haddadi	1999-04-30
10105	Hironobu	Piveteau	1999-03-23
10298	Dietrich	DuCasse	1999-03-30
10684	Aimee	Tokunaga	1999-10-28
11315	Neven	Meriste	1999-07-17
11325	Samphel	Rossi	1999-01-10
11697	JoAnne	Merey	1999-11-06
11754	Malu	Magliocco	1999-08-23
11829	Chaosheng	Pettis	1999-02-17
12015	Kensei	Guenter	1999-01-04

LIKE operator

The LIKE operator is a logical operator that tests whether a string contains a specified pattern or not.

Example:

Retrieve the employees whose last names begin with 'T'

```
SELECT e.first_name, e.last_name  
FROM employees AS e  
WHERE e.last_name LIKE 'T%';
```

string wildcard

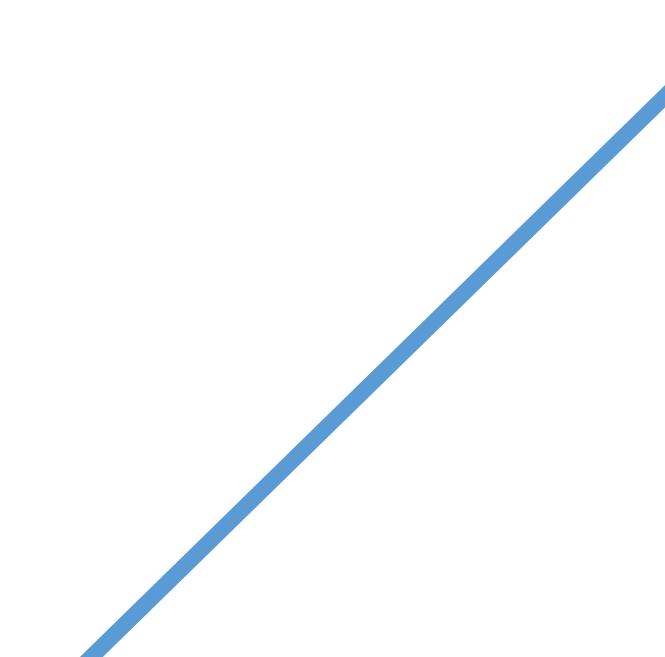
first_name	last_name
Eberhardt	Terkki
Domenick	Tempesti
Yishay	Tzvieli
Basil	Tramer
Abdulah	Thibadeau
Herbert	Trachtenberg
Leon	Trogemann
Ulises	Takanami
Basem	Teitelbaum
Frederique	Tempesti

LIKE operator - Example

Retrieve the employees whose first name has 3 characters and begin with 'Ul'

```
SELECT e.first_name, e.last_name  
FROM employees AS e  
WHERE e.first_name LIKE 'Ul_';
```

Character wildcard



first_name	last_name
Ulf	Siepmann
Uli	Keustermans
Ult	Farrar
Ulf	Basawa
Ulf	Erde
Ult	Anido
Ulf	Hofstetter
Ult	Berstel
Ulf	Skogmar
Uli	Servieres

NOT LIKE operator - Example

Retrieve the first 10 rows of employees where the last name does not begin with 'Tr'

```
SELECT e.first_name, e.last_name
FROM employees AS e
WHERE e.last_name NOT LIKE 'Tr%'  

LIMIT 10;
```



first_name	last_name
Georgi	Facello
Bezalel	Simmel
Parto	Bamford
Chirstian	Koblick
Kyoichi	Maliniak
Anneke	Preusig
Tzvetan	Zielinski
Saniya	Kalloufi
Sumant	Peac
Duangkaew	Piveteau

MySQL alias

For tables:

```
table_name AS table_alias
```

The AS keyword is optional, so you can omit it.

For columns:

```
SELECT  
    [column_1 | expression] AS descriptive_name  
FROM table_name;
```

MySQL alias - Examples

For tables:

```
SELECT
    e.firstName,
    e.lastName
FROM
    employees e
ORDER BY e.firstName;
```

For columns:

```
SELECT
    CONCAT_WS(', ', lastName, firstname) AS Full_name
FROM
    employees;
```

MySQL IN operator

The IN operator allows you to determine if a specified value matches any value in a set of values or returned by a subquery.

```
SELECT
    column1,column2, ...
FROM
    table_name
WHERE
    (expr|column_1) IN ('value1','value2',...);
```

Source: <https://www.mysqltutorial.org/sql-in.aspx>

MySQL IN operator - Example

Retrieve employees that have titles as 'Senior Staff' and 'Senior Engineer'

```
SELECT e.first_name, t.title  
FROM employees AS e, titles AS t  
WHERE e.emp_no = t.emp_no  
AND t.title IN ('Senior Staff', 'Senior Engineer')  
LIMIT 10;
```

first_name	title
Georgi	Senior Engineer
Parto	Senior Engineer
Chirstian	Senior Engineer
Kyoichi	Senior Staff
Anneke	Senior Engineer
Tzvetan	Senior Staff
Sumanth	Senior Engineer
Patricio	Senior Engineer
Eberhardt	Senior Staff
Guoxiang	Senior Staff

ORDER BY clause

The SELECT statement does not provide the result set sorted.

To sort the result set, you add the ORDER BY clause to the SELECT statement.

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    column1 [ASC|DESC],
    column2 [ASC|DESC],
    ...;
```



Note that the ORDER BY clause is always evaluated after the FROM and SELECT clause.

Source: <https://www.mysqltutorial.org/mysql-order-by/>

ORDER BY clause - Examples

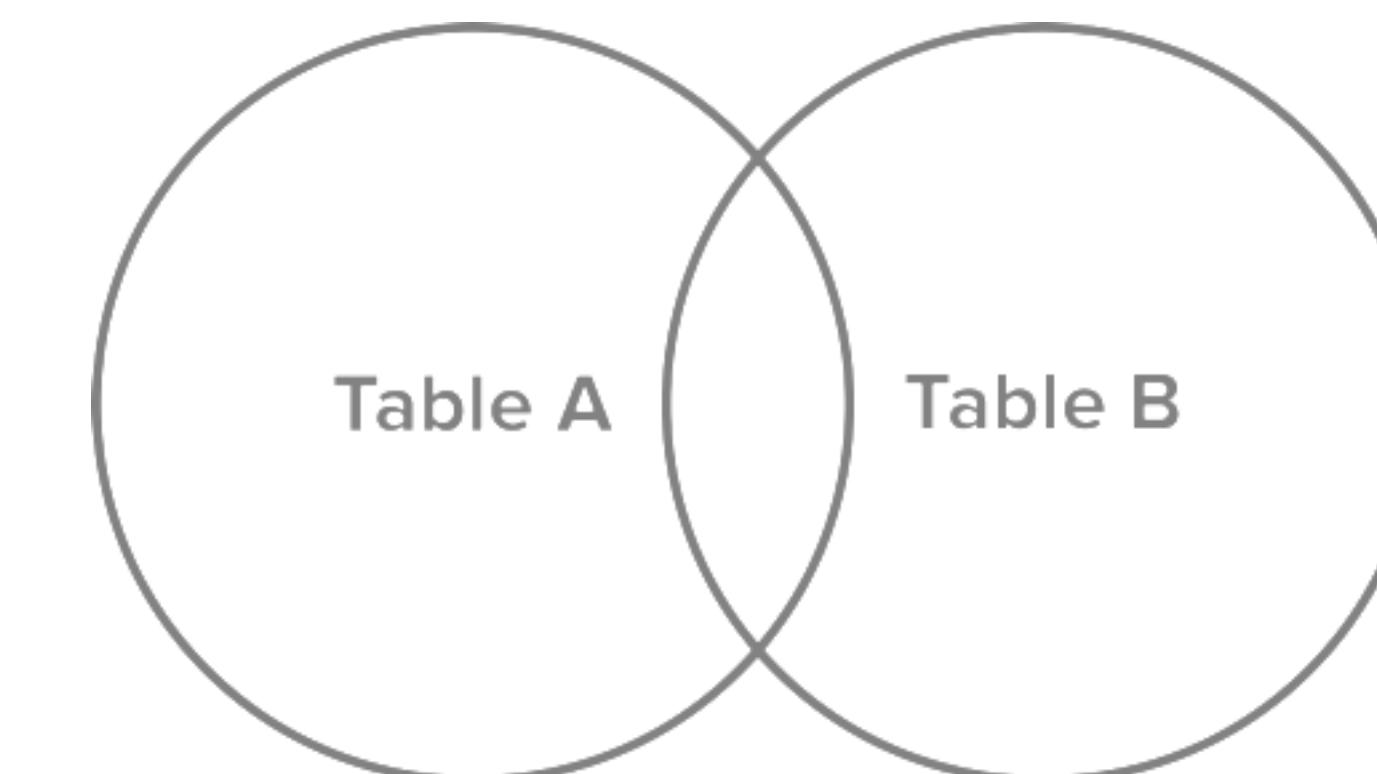
Extract the top five lowest salaries from the list of employees

```
SELECT a.emp_no, a.salary, b.first_name, b.last_name  
FROM salaries AS a, employees AS b  
WHERE a.emp_no=b.emp_no  
AND a.from_date <= current_date  
AND a.to_date >= current_date  
ORDER BY a.salary ASC  
LIMIT 5;
```

emp_no	salary	first_name	last_name
253406	38623	Olivera	Baek
245832	38936	Pascal	Lueh
401786	38942	Sachar	Nicolson
15830	39012	Yurij	Narwekar
230890	39036	Oldrich	Schmiedel

Understand JOIN while Thinking about Sets

There are four basic types of SQL joins: **inner**, **left**, **right**, and **full**. The easiest and most intuitive way to explain the difference between these four types is by using a Venn diagram, which shows all possible logical relations between data sets.



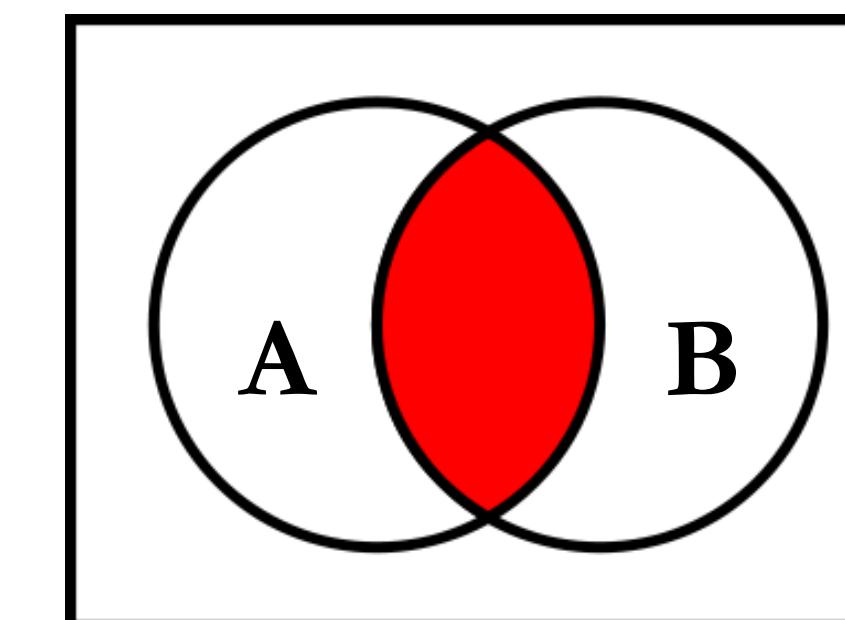
talking about sets, the three most common operations are: Intersection, Difference, and Union

A:{1,5,8,9,32,55,78}

B:{3,7,8,22,55,71,99}

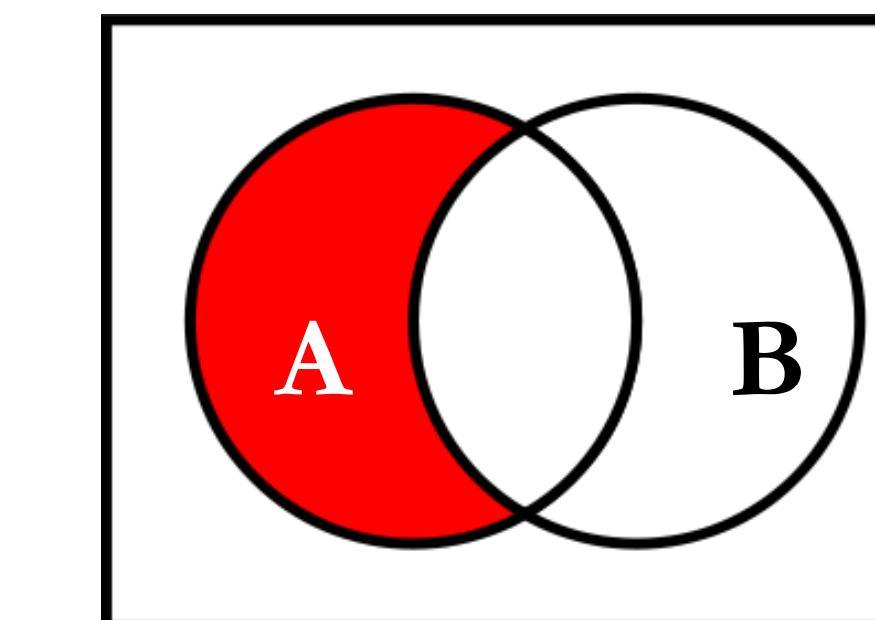
Intersection

{8, 55}



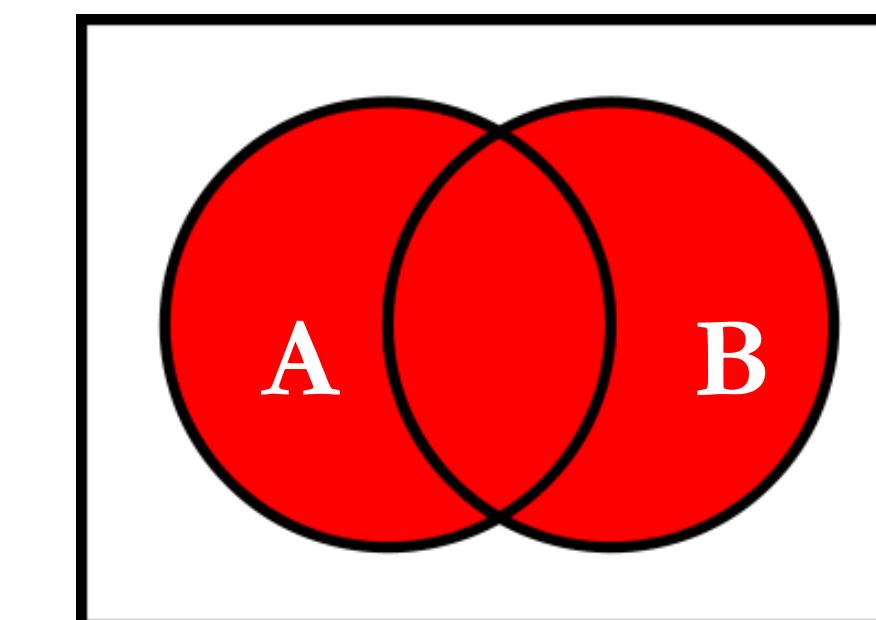
Difference

{1,5,9,32,78}

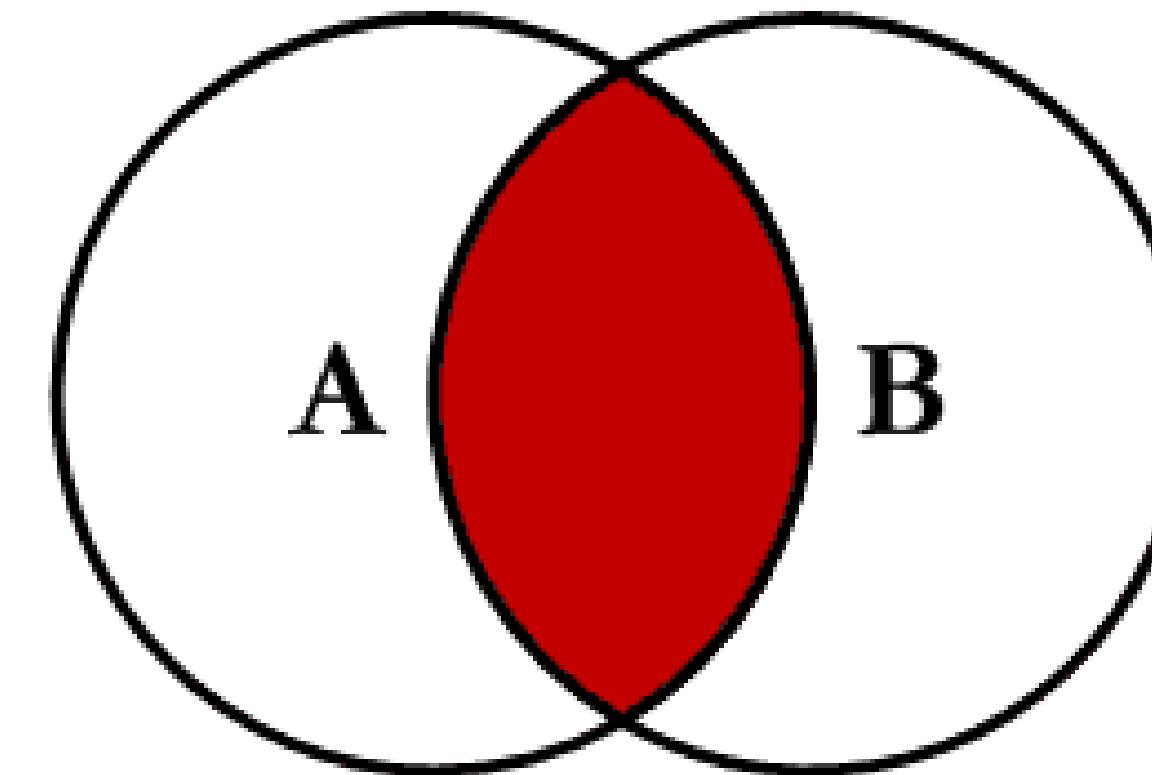


Union

{1,5,8,9,32,55,78,3,7,22,71,99 }



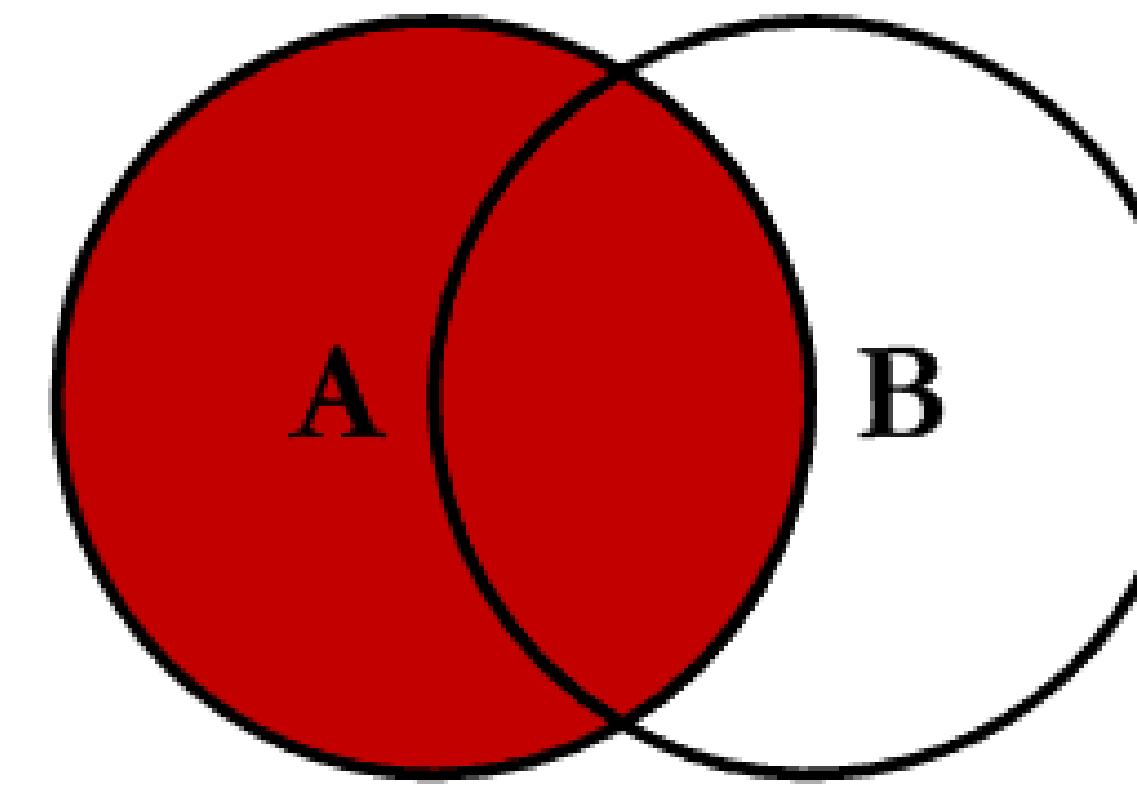
Inner Join



Returns all of the records in the left table (table A) that have a matching record in the right table (table B).

```
SELECT <select_list>  
FROM Table_A A  
INNER JOIN Table_B B  
ON A.Key = B.Key;
```

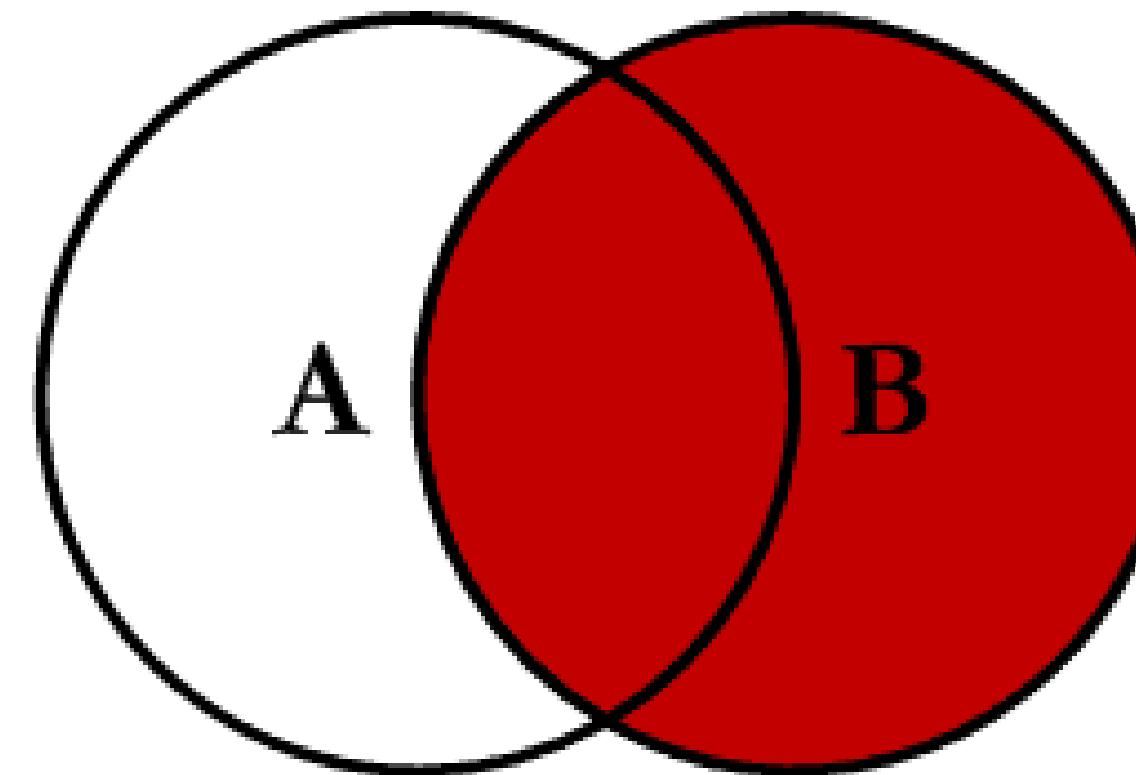
Left Join



```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key;
```

Returns all of the records in the left table (table A) regardless if any of those records have a match in the right table (table B). It will also return any matching records from the right table.

Right Join

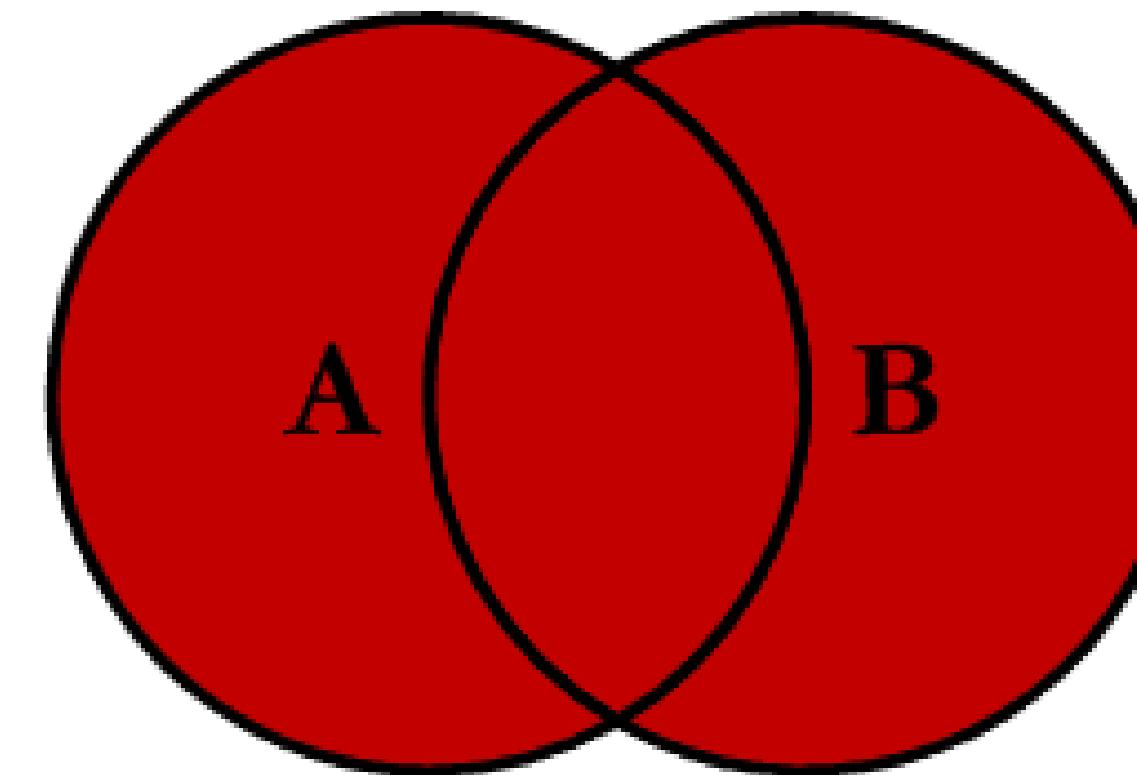


```
SELECT <select_list>
FROM Table_A A
RIGHT JOIN Table_B B
ON A.Key = B.Key;
```

Returns all of the records in the right table (table B) regardless if any of those records have a match in the left table (table A). It will also return any matching records from the right table.

Emulating Full outer join with UNION

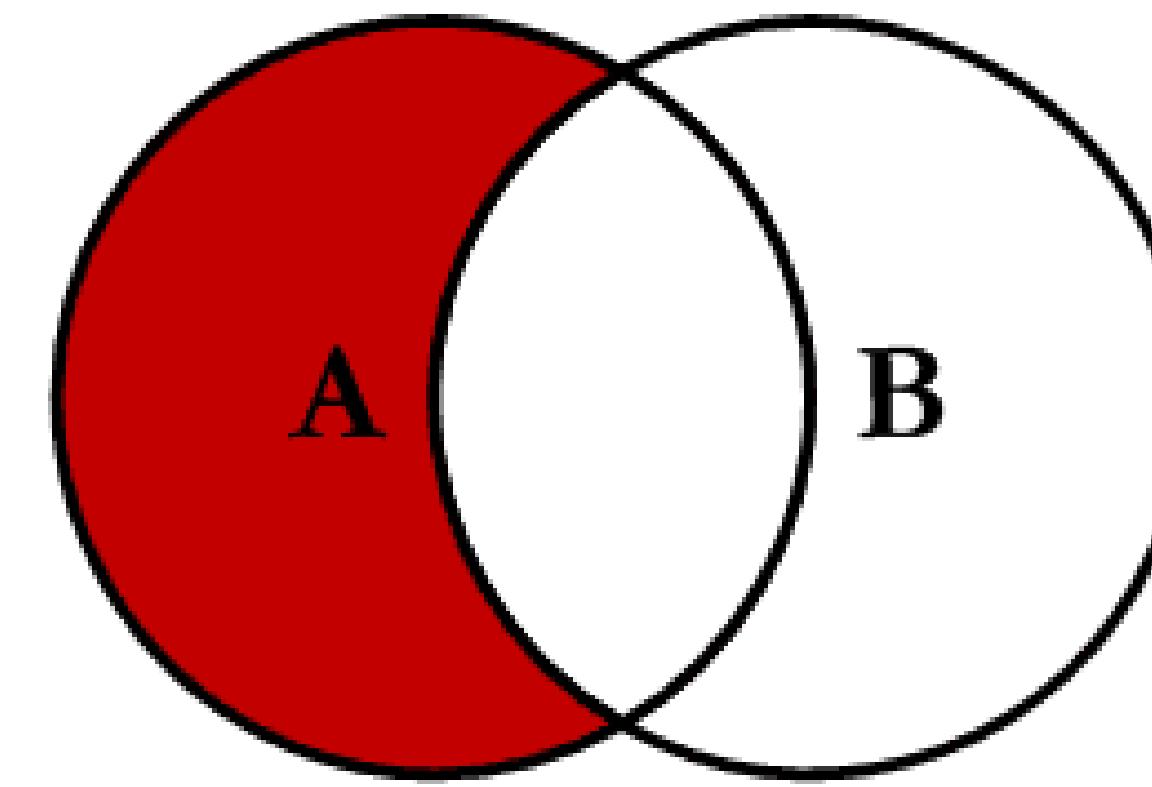
MySQL does not support full outer join, we need to emulate it.



```
SELECT * FROM t1  
LEFT JOIN t2 ON t1.id = t2.id  
UNION ALL  
SELECT * FROM t1  
RIGHT JOIN t2 ON t1.id = t2.id
```

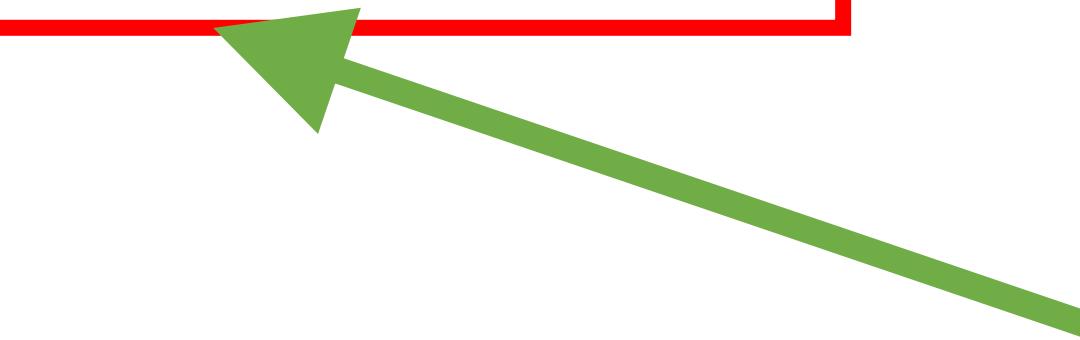
Returns all of the records from both tables, joining records from the left table (table A) that match records from the right table

Left excluding Join



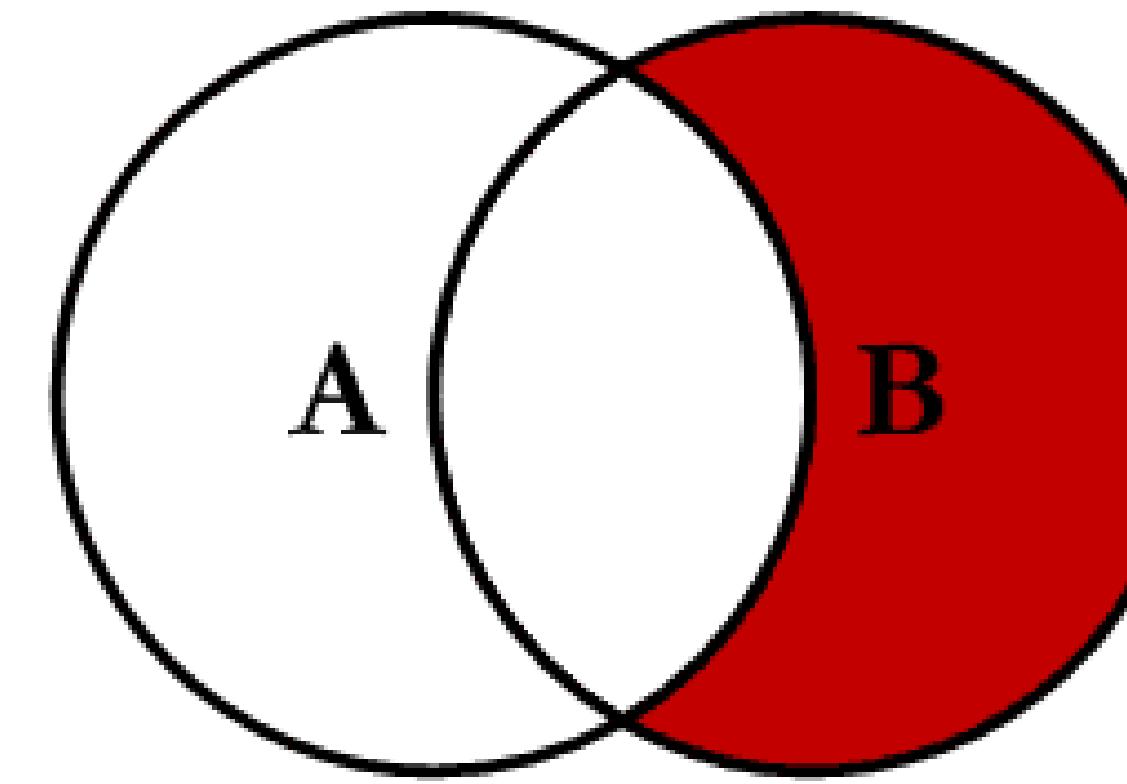
This query will return all of the records in the left table (table A) that do not match any records in the right table (table B).

```
SELECT <select_list>
FROM Table_A A
LEFT JOIN Table_B B
ON A.Key = B.Key
WHERE B.Key IS NULL;
```



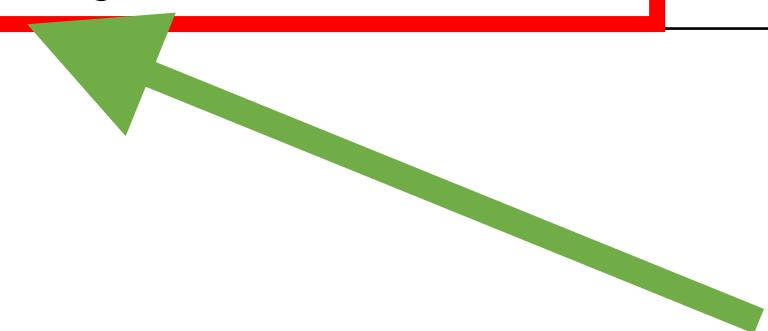
This where statement makes the difference with the Left Join (not excluding)

Right excluding Join



This query will return all of the records in the right table (table B) that do not match any records in the left table (table A).

```
SELECT <select_list>  
FROM Table_A A  
RIGHT JOIN Table_B B  
ON A.Key = B.Key  
WHERE A.Key IS NULL;
```



This where statement makes the difference with the Right Join (not excluding)

Join

Left, Inner, Right, Left with Null Right, Right with Null Left

Joins operate at the select level

```
SELECT
    column_1, column_2, ...
FROM
    table_1
[INNER | LEFT |RIGHT] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1
LIMIT offset, length;
```

Examples of Joins

Example of an INNER JOIN or just JOIN

Retrieve all information from employees and their salaries

```
SELECT * FROM salaries AS a, employees AS b  
WHERE a.emp_no = b.emp_no  
LIMIT 10;
```

emp_no	salary	from_date	to_date	emp_no	birth_date	first_name	last_name	gender	hire_date
10001	60117	1986-06-26	1987-06-26	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	62102	1987-06-26	1988-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66074	1988-06-25	1989-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66596	1989-06-25	1990-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66961	1990-06-25	1991-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	71046	1991-06-25	1992-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	74333	1992-06-24	1993-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	75286	1993-06-24	1994-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	75994	1994-06-24	1995-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	76884	1995-06-24	1996-06-23	10001	1953-09-02	Georgi	Facello	M	1986-06-26

Example of an INNER JOIN or just JOIN

A different way of doing the same query as above:

```
SELECT *
FROM salaries AS a
JOIN employees AS b ON a.emp_no = b.emp_no
LIMIT 10;
```

emp_no	salary	from_date	to_date	emp_no	birth_date	first_name	last_name	gender	hire_date
10001	60117	1986-06-26	1987-06-26	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	62102	1987-06-26	1988-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66074	1988-06-25	1989-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66596	1989-06-25	1990-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	66961	1990-06-25	1991-06-25	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	71046	1991-06-25	1992-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	74333	1992-06-24	1993-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	75286	1993-06-24	1994-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	75994	1994-06-24	1995-06-24	10001	1953-09-02	Georgi	Facello	M	1986-06-26
10001	76884	1995-06-24	1996-06-23	10001	1953-09-02	Georgi	Facello	M	1986-06-26

Example of a JOIN

Select a list of cities and their countries

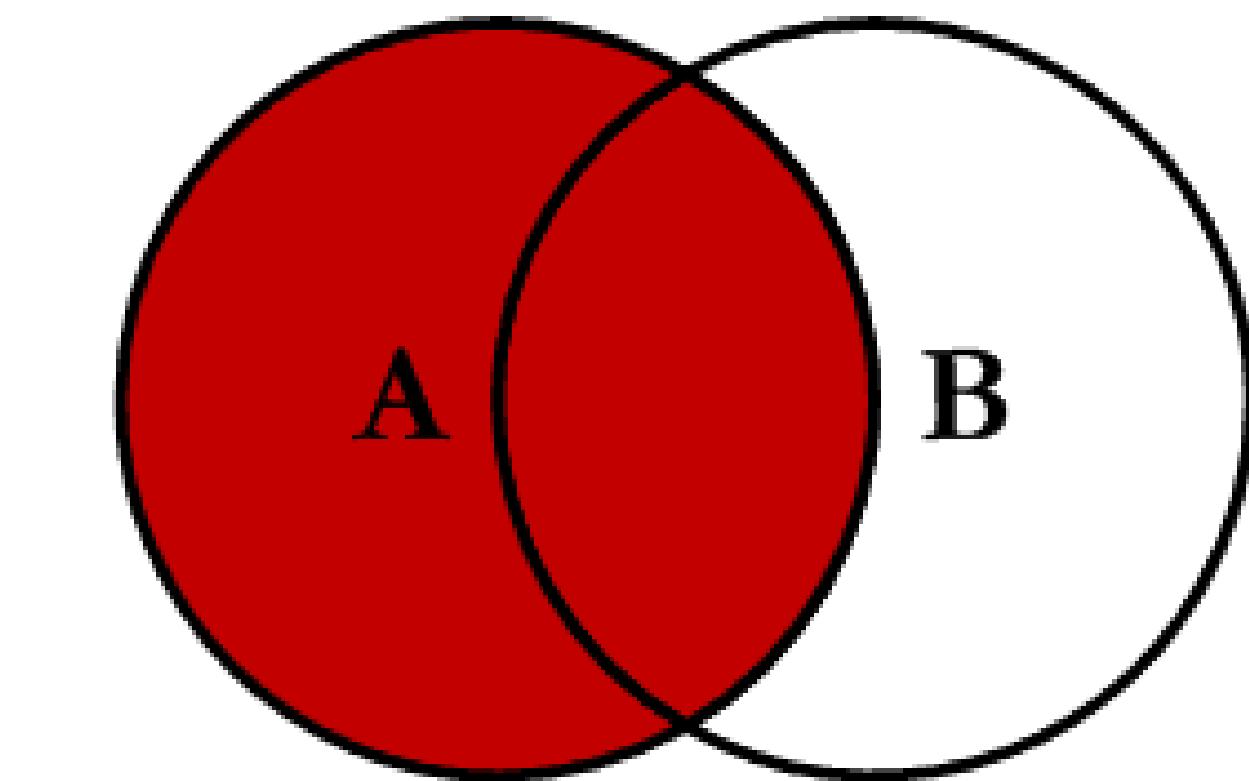
```
SELECT city, country
FROM city
INNER JOIN country
ON city.country_id = country.country_id
LIMIT 10;
```

city_id	city	country_id	last_update	country_id	country	last_update
1	A Corua (La Coruña)	87	2006-02-15 04:45:25	1	Afghanistan	2006-02-15 04:44:00
2	Abha	82	2006-02-15 04:45:25	2	Algeria	2006-02-15 04:44:00
3	Abu Dhabi	101	2006-02-15 04:45:25	3	American Samoa	2006-02-15 04:44:00
4	Acua	60	2006-02-15 04:45:25	4	Angola	2006-02-15 04:44:00
5	Adana	97	2006-02-15 04:45:25	5	Anguilla	2006-02-15 04:44:00
6	Addis Abeba	31	2006-02-15 04:45:25	6	Argentina	2006-02-15 04:44:00
7	Aden	107	2006-02-15 04:45:25	7	Armenia	2006-02-15 04:44:00
8	Adoni	44	2006-02-15 04:45:25	8	Australia	2006-02-15 04:44:00
9	Ahmadnagar	44	2006-02-15 04:45:25	9	Austria	2006-02-15 04:44:00
10	Akishima	50	2006-02-15 04:45:25	10	Azerbaijan	2006-02-15 04:44:00

city	country
Kabul	Afghanistan
Batna	Algeria
Bchar	Algeria
Skikda	Algeria
Tafuna	American Samoa
Benguela	Angola
Namibe	Angola
South Hill	Anguilla
Almirante Braga	Argentina
Avellaneda	Argentina

Example Left join

Retrieve all the customers and actors who have the same last name. Customers should be retrieved even if there is not an actor with that name



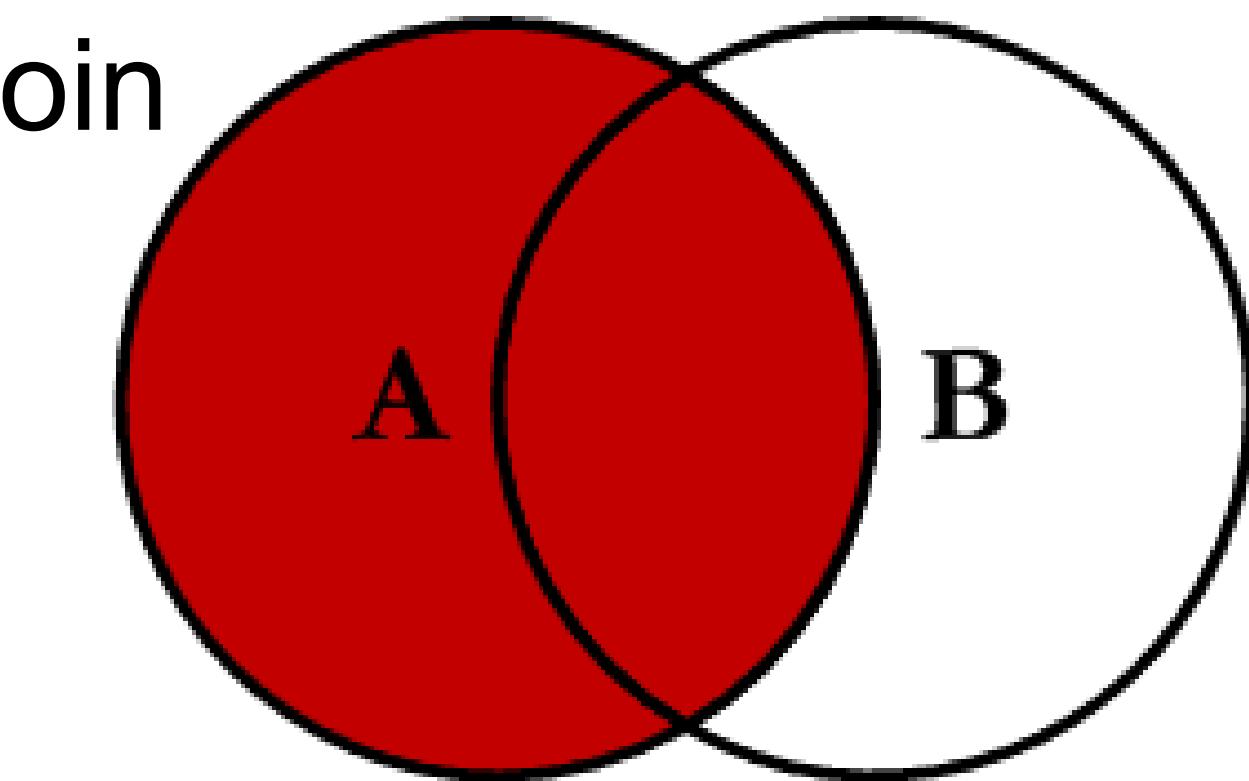
customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1	2006-02-14 22:04:36	2006-02-15 04:57:20
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	1	2006-02-14 22:04:36	2006-02-15 04:57:20
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	1	2006-02-14 22:04:36	2006-02-15 04:57:20
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	1	2006-02-14 22:04:36	2006-02-15 04:57:20
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	1	2006-02-14 22:04:36	2006-02-15 04:57:20

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINNESS	2006-02-15 04:34:33
2	NICK	WAHLBERG	2006-02-15 04:34:33
3	ED	CHASE	2006-02-15 04:34:33
4	JENNIFER	DAVIS	2006-02-15 04:34:33
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33

Example Left join

```
SELECT c.customer_id, c.first_name, c.last_name  
a.actor_id, a.first_name, a.last_name  
FROM customer AS c  
LEFT JOIN actor AS a  
ON c.last_name = a.last_name  
ORDER BY c.last_name;
```

Left Join



customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1	2006-02-14 22:04:36	2006-02-15 04:57:20
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	1	2006-02-14 22:04:36	2006-02-15 04:57:20
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	1	2006-02-14 22:04:36	2006-02-15 04:57:20
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	1	2006-02-14 22:04:36	2006-02-15 04:57:20
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	1	2006-02-14 22:04:36	2006-02-15 04:57:20
actor_id	first_name	last_name	last_update					
1	PENELOPE	GUINNESS	2006-02-15 04:34:33					
2	NICK	WAHLBERG	2006-02-15 04:34:33					
3	ED	CHASE	2006-02-15 04:34:33					
4	JENNIFER	DAVIS	2006-02-15 04:34:33					
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33					

Example Left join

```
SELECT c.customer_id, c.first_name,  
c.last_name,  
a.actor_id, a.first_name, a.last_name  
FROM customer AS c  
LEFT JOIN actor AS a  
ON c.last_name = a.last_name  
ORDER BY c.last_name;
```

customer_id	actor first_name	actor last_name	actor_id	first_name	last_name
505	RAFAEL	ABNEY	NULL	NULL	NULL
504	NATHANIEL	ADAM	NULL	NULL	NULL
36	KATHLEEN	ADAMS	NULL	NULL	NULL
96	DIANA	ALEXANDER	NULL	NULL	NULL
470	GORDON	ALLARD	NULL	NULL	NULL
27	SHIRLEY	ALLEN	118	CUBA	ALLEN
27	SHIRLEY	ALLEN	145	KIM	ALLEN
27	SHIRLEY	ALLEN	194	MERYL	ALLEN
220	CHARLENE	ALVAREZ	NULL	NULL	NULL
11	LISA	ANDERSON	NULL	NULL	NULL
326	JOSE	ANDREW	NULL	NULL	NULL
183	IDA	ANDREWS	NULL	NULL	NULL
449	OSCAR	AQUINO	NULL	NULL	NULL
368	HARRY	ARCE	NULL	NULL	NULL
560	JORDAN	ARCHULETA	NULL	NULL	NULL
188	MELANIE	ARMSTRONG	NULL	NULL	NULL
170	BEATRICE	ARNOLD	NULL	NULL	NULL
591	KENT	ARSENault	NULL	NULL	NULL
345	CARL	ARTIS	NULL	NULL	NULL
530	DARRYL	ASHCRAFT	NULL	NULL	NULL
540	TYRONE	ASHER	NULL	NULL	NULL
196	ALMA	AUSTIN	NULL	NULL	NULL
60	MILDRED	BAILEY	67	JESSICA	BAILEY
60	MILDRED	BAILEY	190	AUDREY	BAILEY
37	PAMELA	BAKER	NULL	NULL	NULL

Example – Comparing inner join with left join

Inner Join

customer_id	actor first_name	actor last_name	actor_id	first_name	last_name
27	SHIRLEY	ALLEN	145	KIM	ALLEN
27	SHIRLEY	ALLEN	118	CUBA	ALLEN
27	SHIRLEY	ALLEN	194	MERYL	ALLEN
60	MILDRED	BAILEY	190	AUDREY	BAILEY
60	MILDRED	BAILEY	67	JESSICA	BAILEY
168	REGINA	BERRY	12	KARL	BERRY
168	REGINA	BERRY	60	HENRY	BERRY
168	REGINA	BERRY	91	CHRISTOPHER	BERRY
132	ESTHER	CRAWFORD	129	DARYL	CRAWFORD
132	ESTHER	CRAWFORD	26	RIP	CRAWFORD
118	KIM	CRUZ	80	RALPH	CRUZ
6	JENNIFER	DAVIS	101	SUSAN	DAVIS
6	JENNIFER	DAVIS	110	SUSAN	DAVIS
6	JENNIFER	DAVIS	4	JENNIFER	DAVIS
236	MARCIA	DEAN	35	JUDY	DEAN
236	MARCIA	DEAN	143	RIVER	DEAN
116	VICTORIA	GIBSON	154	MERYL	GIBSON
154	MICHELE	GRANT	71	ADAM	GRANT
15	HELEN	HARRIS	141	CATE	HARRIS
15	HELEN	HARRIS	152	BEN	HARRIS
15	HELEN	HARRIS	56	DAN	HARRIS
252	MATTIE	HOFFMAN	28	WOODY	HOFFMAN
252	MATTIE	HOFFMAN	169	KENNETH	HOFFMAN
252	MATTIE	HOFFMAN	79	MAE	HOFFMAN
263	HILDA	HOPKINS	50	NATALIE	HOPKINS

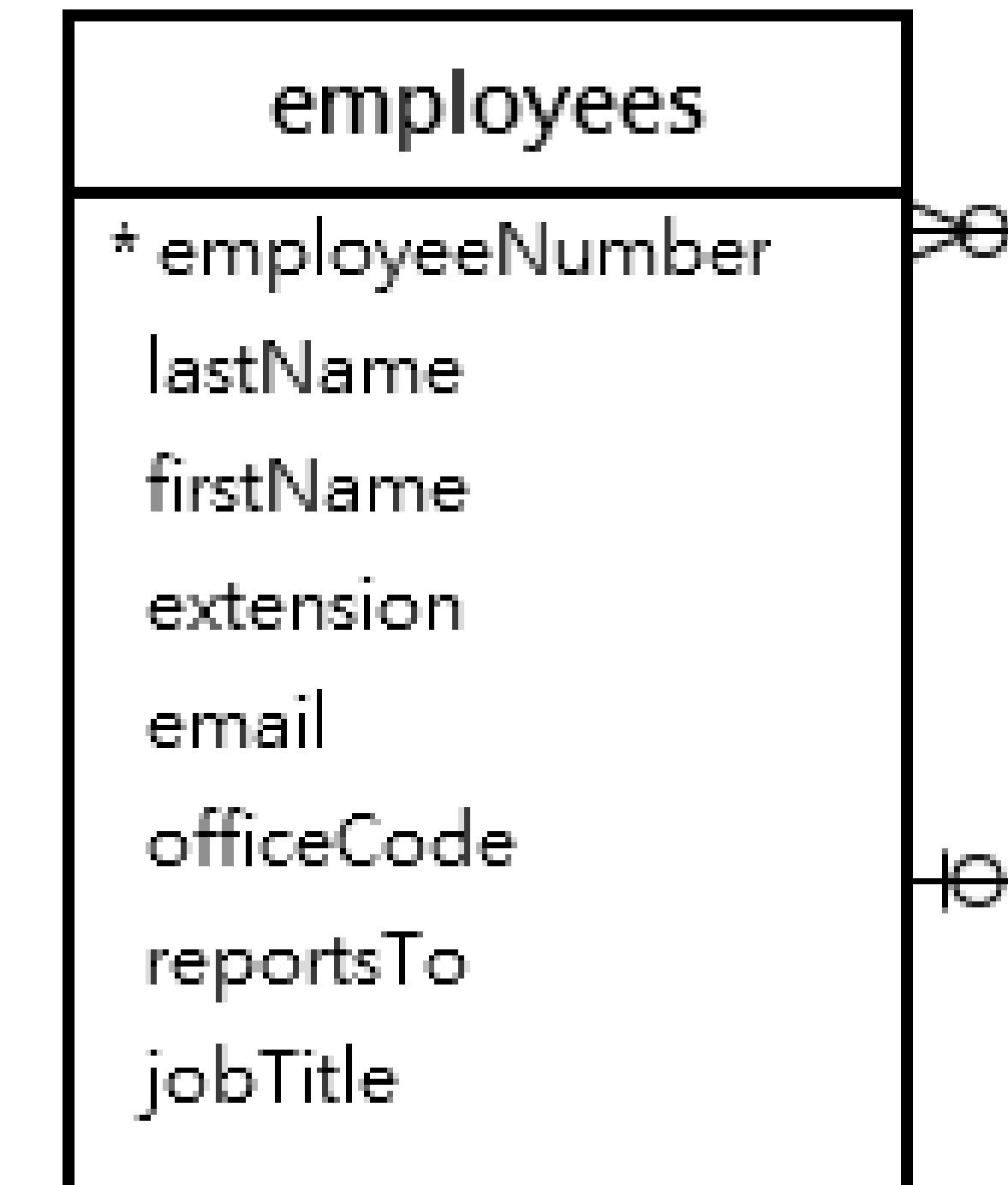
Left Join

customer_id	actor first_name	actor last_name	actor_id	first_name	last_name
505	RAFAEL	ABNEY	NULL	NULL	NULL
504	NATHANIEL	ADAM	NULL	NULL	NULL
36	KATHLEEN	ADAMS	NULL	NULL	NULL
96	DIANA	ALEXANDER	NULL	NULL	NULL
470	GORDON	ALLARD	NULL	NULL	NULL
27	SHIRLEY	ALLEN	118	CUBA	ALLEN
27	SHIRLEY	ALLEN	145	KIM	ALLEN
27	SHIRLEY	ALLEN	194	MERYL	ALLEN
220	CHARLENE	ALVAREZ	NULL	NULL	NULL
11	LISA	ANDERSON	NULL	NULL	NULL
326	JOSE	ANDREW	NULL	NULL	NULL
183	IDA	ANDREWS	NULL	NULL	NULL
449	OSCAR	AQUINO	NULL	NULL	NULL
368	HARRY	ARCE	NULL	NULL	NULL
560	JORDAN	ARCHULETA	NULL	NULL	NULL
188	MELANIE	ARMSTRONG	NULL	NULL	NULL
170	BEATRICE	ARNOLD	NULL	NULL	NULL
591	KENT	ARSENault	NULL	NULL	NULL
345	CARL	ARTIS	NULL	NULL	NULL
530	DARRYL	ASHCRAFT	NULL	NULL	NULL
540	TYRONE	ASHER	NULL	NULL	NULL
196	ALMA	AUSTIN	NULL	NULL	NULL
60	MILDRED	BAILEY	67	JESSICA	BAILEY
60	MILDRED	BAILEY	190	AUDREY	BAILEY
37	PAMELA	BAKER	NULL	NULL	NULL

Self join – A table joins to itself

The self join is often used to query hierarchical data. For instance, the table employees contains the employees and their supervisors (which are also employees)

To perform a self join, you must use table aliases.



Source: <https://www.mysqltutorial.org/mysql-self-join/>

Self join – Example

Retrieve the last name of employees and their supervisors

```
SELECT
    m.lastName AS Manager,
    e.lastName AS 'Direct report'
FROM
    employees e
INNER JOIN employees m ON
    m.employeeNumber = e.reportsTo;
```

Source: <https://www.mysqltutorial.org/mysql-self-join/>

Examples – complex joins

What if you want to retrieve data from tables that are not directly linked?

Payment

payment_id
customer_id
staff_id
rental_id
amount
payment_date
last_update

rental

rental_id
rental_date
inventory_id
customer_id
return_date
staff_id
last_update

inventory

inventory_id
film_id
store_id
last_update

store

store_id
manager_staff_id
address_id
last_update

address

address_id
address
address2
district
city_id
postal_code
phone
location
last_update

city

city_id
city
country_id
last_update

Country

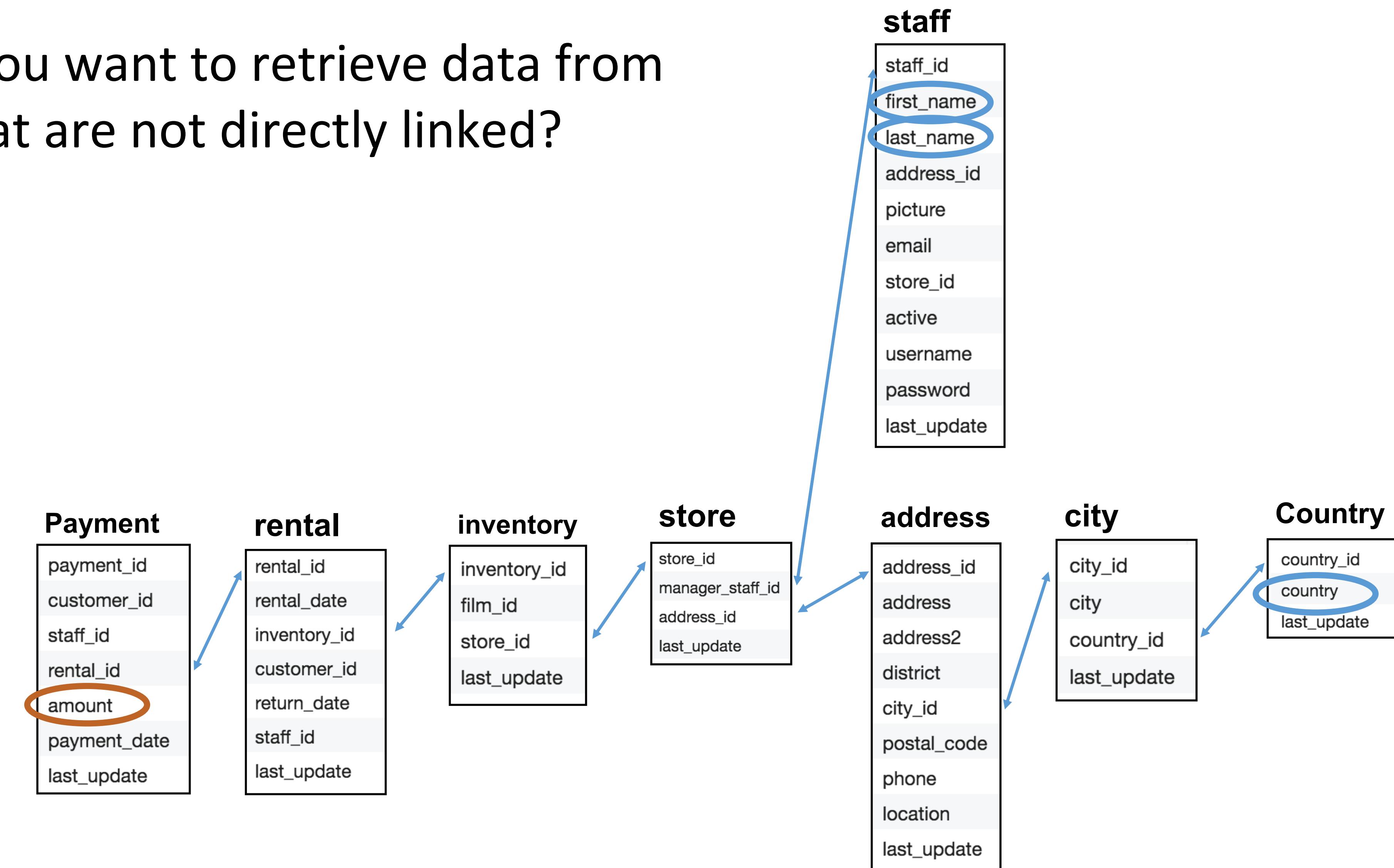
country_id
country
last_update

staff

staff_id
first_name
last_name
address_id
picture
email
store_id
active
username
password
last_update

Examples – complex joins

What if you want to retrieve data from tables that are not directly linked?



Examples – complex joins

```
SELECT c.city, cy.country, m.first_name, m.last_name,  
SUM(p.amount) AS total_sales  
FROM payment AS p  
INNER JOIN rental AS r ON p.rental_id = r.rental_id  
INNER JOIN inventory AS i ON r.inventory_id = i.inventory_id  
INNER JOIN store AS s ON i.store_id = s.store_id  
INNER JOIN address AS a ON s.address_id = a.address_id  
INNER JOIN city AS c ON a.city_id = c.city_id  
INNER JOIN country AS cy ON c.country_id = cy.country_id  
INNER JOIN staff AS m ON s.manager_staff_id = m.staff_id ;
```

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

Let's have
some fun!

END OF LECTURE 4

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



eduniversal



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Gestão e armazenamento de dados

Lecture 5

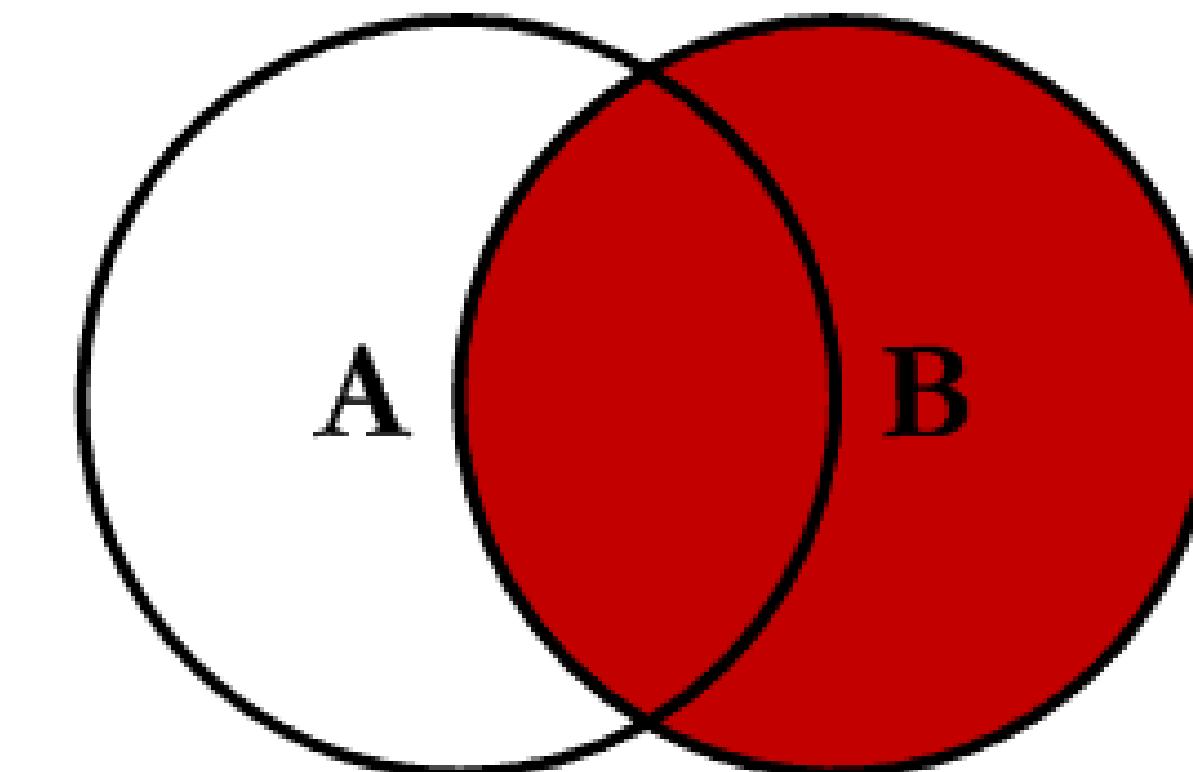
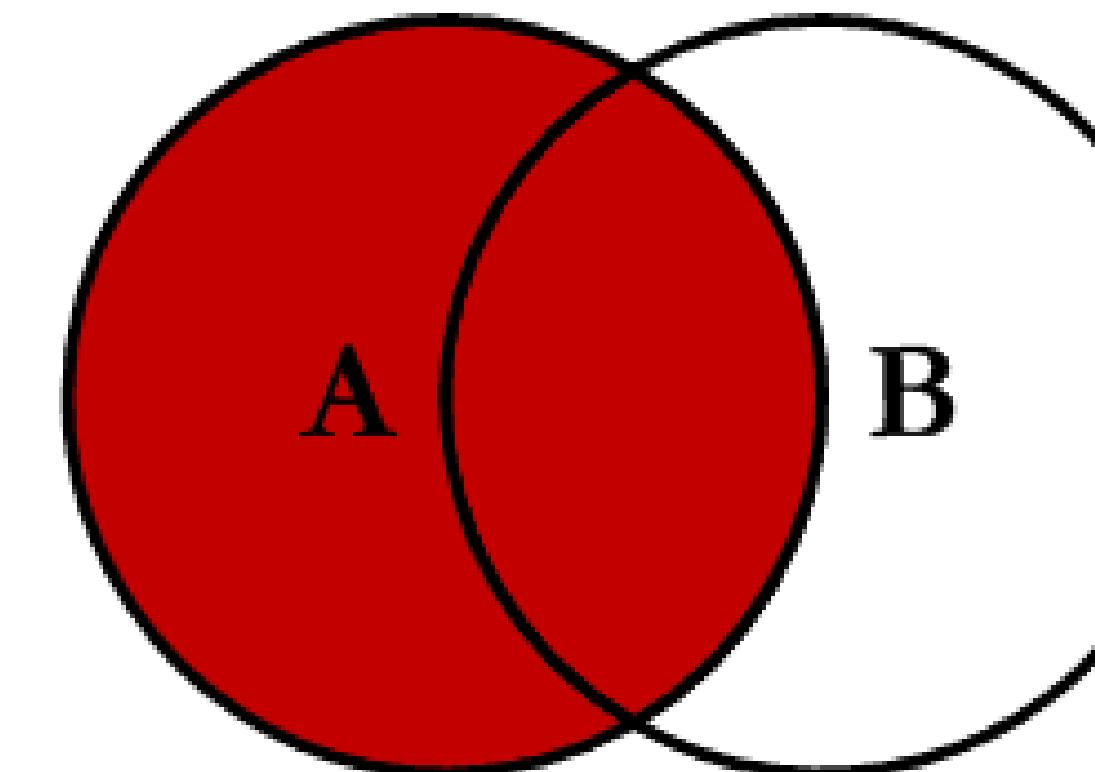
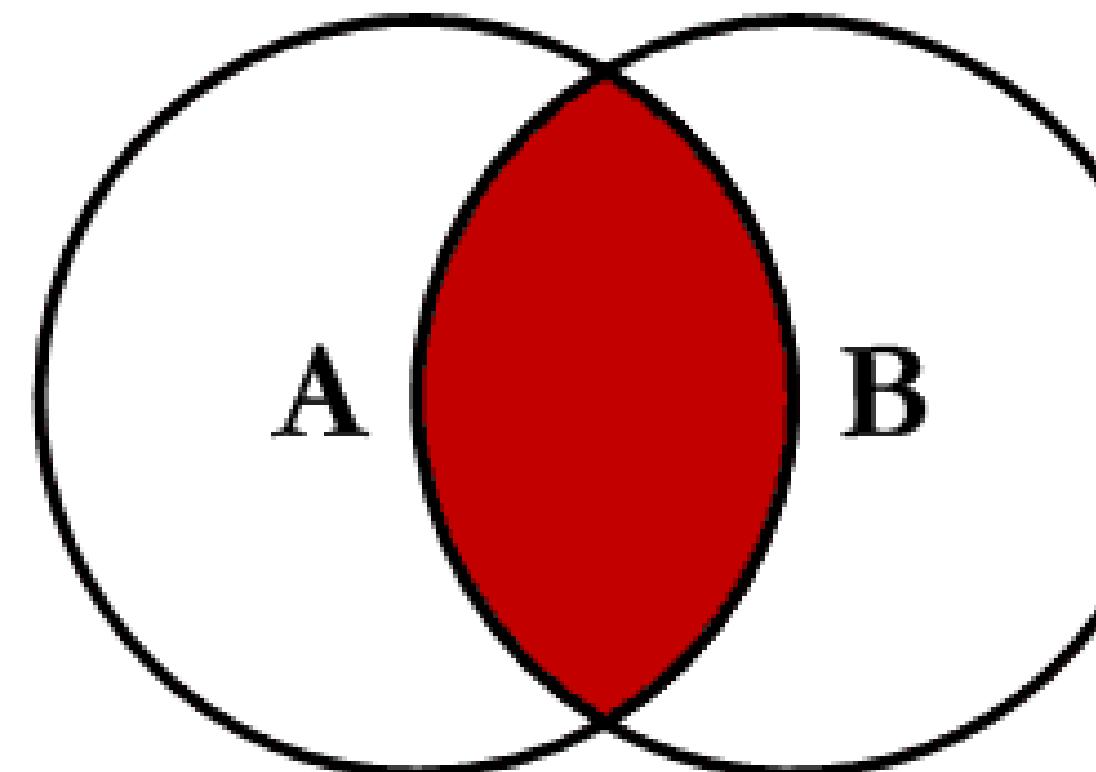
More clauses
Views and Triggers

Previous class



Previous class

- Clauses and operators
- Joins



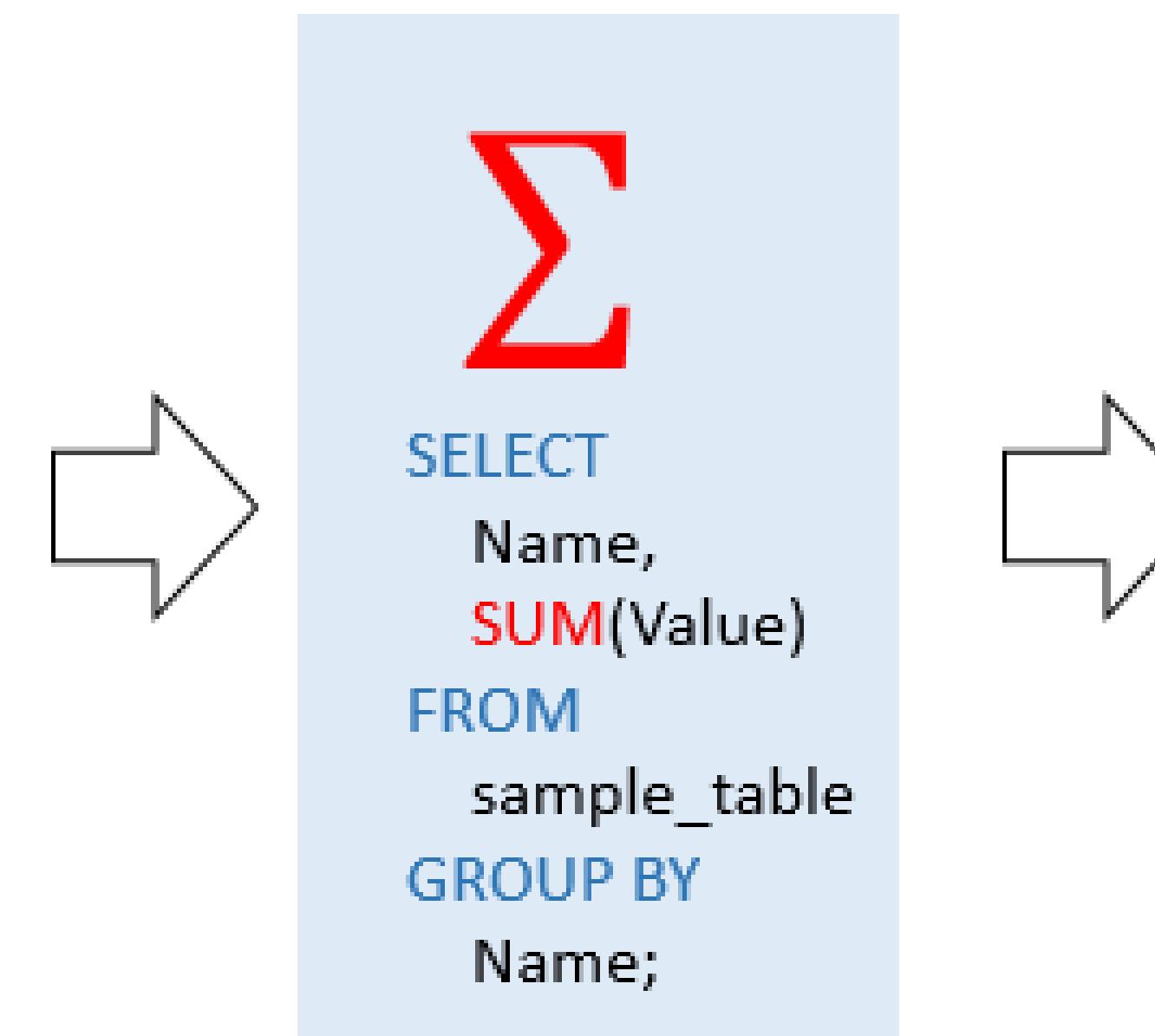
- MySQL functions
 - Aggregate
 - Math
 - Date
 - String
- Group by
- Having
- Views
- Triggers

MySQL functions

Aggregate functions

An aggregate function performs a calculation on multiple values and returns a single value.

Name	Value
A	10
A	20
B	40
C	20
C	50



The aggregate functions are often used with the GROUP BY clause to calculate an aggregate value for each group.

Source: <https://www.mysqltutorial.org/mysql-aggregate-functions.aspx>

Aggregate functions

Some of the most used aggregate functions:

Aggregate function	Description
AVG()	Return the average of non-NULL values.
MAX()	Return the highest value (maximum) in a set of non-NULL values.
MIN()	Return the lowest value (minimum) in a set of non-NULL values.
STDEV()	Return the population standard deviation.
COUNT()	Return the number of rows in a group, including rows with NULL values.
SUM()	Return the summation of all non-NULL values a set.

Source: <https://www.mysqltutorial.org/mysql-aggregate-functions.aspx>

Aggregate functions - Examples

How many employees you have in the SHIPPING department
(department_id = 50)

```
SELECT count(employee_id) as ShippingEmployees  
FROM employee  
WHERE department_id = 50;
```



	ShippingEmployees
→	45

Aggregate functions - Examples

Find the average salary of female employees

```
SELECT avg(s.salary)
FROM employees AS e, salaries AS s
WHERE e.emp_no = s.emp_no
AND e.gender = 'F';
```

e.gender = 'F'

avg(s.salary)

55579.4444

A few MySQL Math functions

MOD()	Returns the remainder of a number divided by another
ROUND()	Rounds a number to a specified number of decimal places.
TRUNCATE()	Truncates a number to a specified number of decimal places
SIN(n)	Returns the sine of n
SQRT(n)	Returns the square root of n
RAND()	Returns a random floating-point value
LOG(n)	Returns the natural logarithm of the first argument
POW()	Returns the argument raised to the specified power

There are many more...

Math functions - Round Example

Round the salary of an employee to 2 decimals.

```
SELECT round(e.salary, 2)  
FROM employee AS e;
```

Number of decimals

A few MySQL Date functions

CURDATE or CURRENT_DATE	Returns the current date.
DATEDIFF	Calculates the number of days between two DATE values.
DAY	Gets the day of the month of a specified date.
DAYOFWEEK	Returns the weekday index for a date.
NOW	Returns the current date and time at which the statement executed.
MONTH	Returns an integer that represents a month of a specified date.
TIMEDIFF	Calculates the difference between two TIME or DATETIME values.
YEAR	Return the year for a specified date

Source: <https://www.mysqltutorial.org/mysql-date-functions/>

A few MySQL String functions

CONCAT	Concatenate two or more strings into a single string
LENGTH	Get the length of a string in bytes and in characters
LOWER	Convert a string to lowercase
REPLACE	Search and replace a substring in a string
SUBSTRING	Extract a substring starting from a position with a specific length.
TRIM	Remove unwanted characters from a string.
UPPER	Convert a string to uppercase
INSTR	Return the position of the first occurrence of a substring in a string

String functions - CONCAT Example

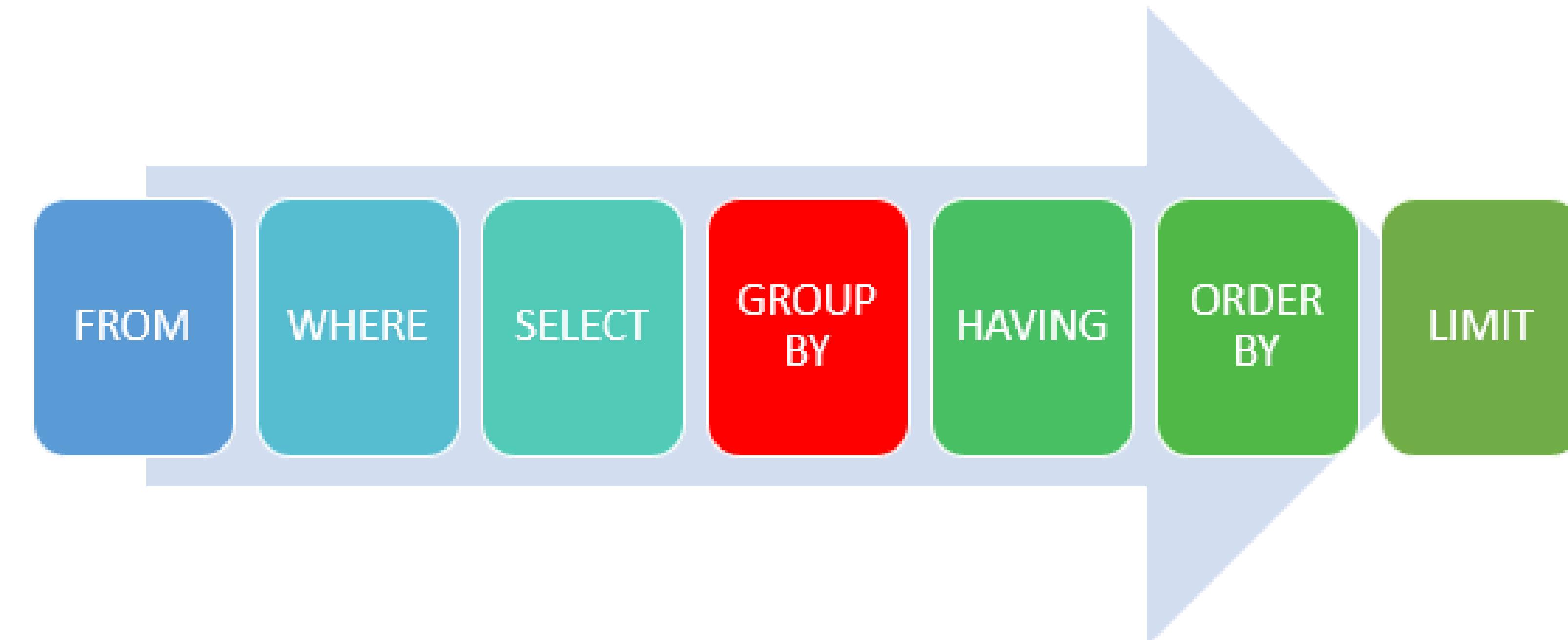
Concatenate the first and last name of the employee separated by a blank space

```
SELECT concat(e.first_name, ' ', e.last_name)  
FROM employee AS e;
```

Group By clause

Group by clause

The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions. The GROUP BY clause returns one row for each group.



Source: <https://www.mysqltutorial.org/mysql-group-by.aspx>

Group by clause

Usually GROUP BY clause is used with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT.

```
SELECT  
    c1, c2,..., cn, aggregate_function(ci)  
FROM  
    table_name  
WHERE  
    where_conditions  
GROUP BY c1 , c2,...,cn;
```

Source: <https://www.mysqltutorial.org/mysql-group-by.aspx>

Examples

Select the average salary by gender

```
SELECT e.gender, avg(s.salary)  
FROM employees AS e, salaries AS s, dept_emp AS d  
WHERE e.emp_no = s.emp_no  
AND d.emp_no = e.emp_no  
AND d.to_date >= current_date  
AND e.hire_date BETWEEN '2000-01-01' AND current_date  
GROUP BY e.gender;
```

gender	avg(s.salary)
M	58126.6667
F	55579.4444

Examples

Select the average salary by year

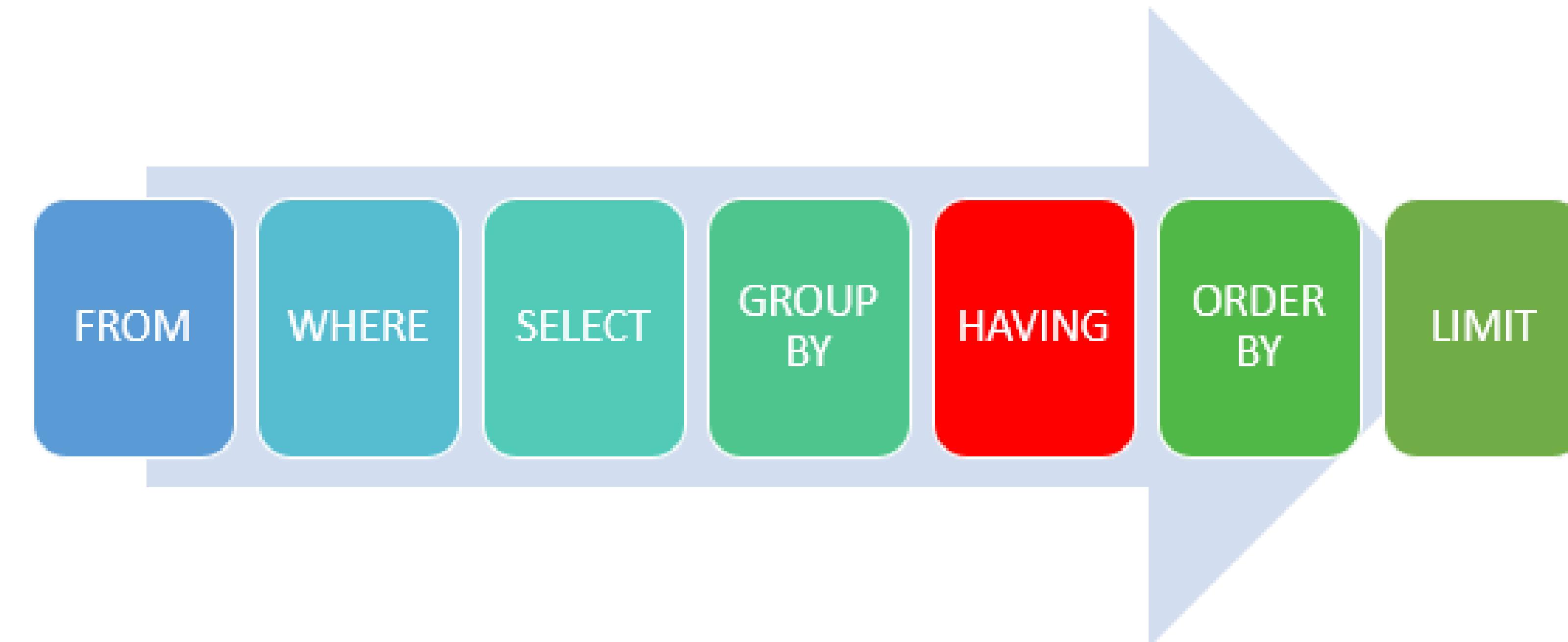
```
SELECT YEAR(e.hire_date) AS year, AVG(s.salary)
FROM employees AS e, salaries AS s
WHERE e.emp_no = s.emp_no
GROUP BY YEAR(e.hire_date)
```

year	AVG(s.salary)
1986	66187.3453
1985	66966.7550
1989	63658.8510
1994	59372.7106
1990	62736.4975
1992	60962.9784
1987	65199.4887
1995	58369.3347
1993	60393.9920
1999	55561.4406
1991	61765.4281
1988	64205.4734
1997	56797.7335
1996	57724.8363
1998	56390.4280
2000	54084.6389

Having clause

Having clause

The HAVING clause is used in the SELECT statement to specify filter conditions for a group of rows or aggregates.



Source: <https://www.mysqltutorial.org/mysql-having.aspx>

Having clause

The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified condition. Notice that the HAVING clause applies a filter condition to each group of rows, while the WHERE clause applies the filter condition to each individual row.

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
GROUP BY
    group_by_expression
HAVING
    group_condition;
```

Source: <https://www.mysqltutorial.org/mysql-having.aspx>

Examples

Retrieve the number of employees by gender hired after 1995

```
SELECT YEAR(e.hire_date) AS year, e.gender, COUNT(e.gender)
FROM employees AS e, salaries AS s
WHERE e.emp_no = s.emp_no
GROUP BY YEAR(e.hire_date), e.gender
HAVING year > 1995
LIMIT 10;
```

year	gender	COUNT(e.gender)
1999	M	2899
1997	F	11708
1996	M	30139
1996	F	19381
1998	F	6593
1997	M	18253
1998	M	9399
1999	F	1942
2000	M	16
2000	F	20

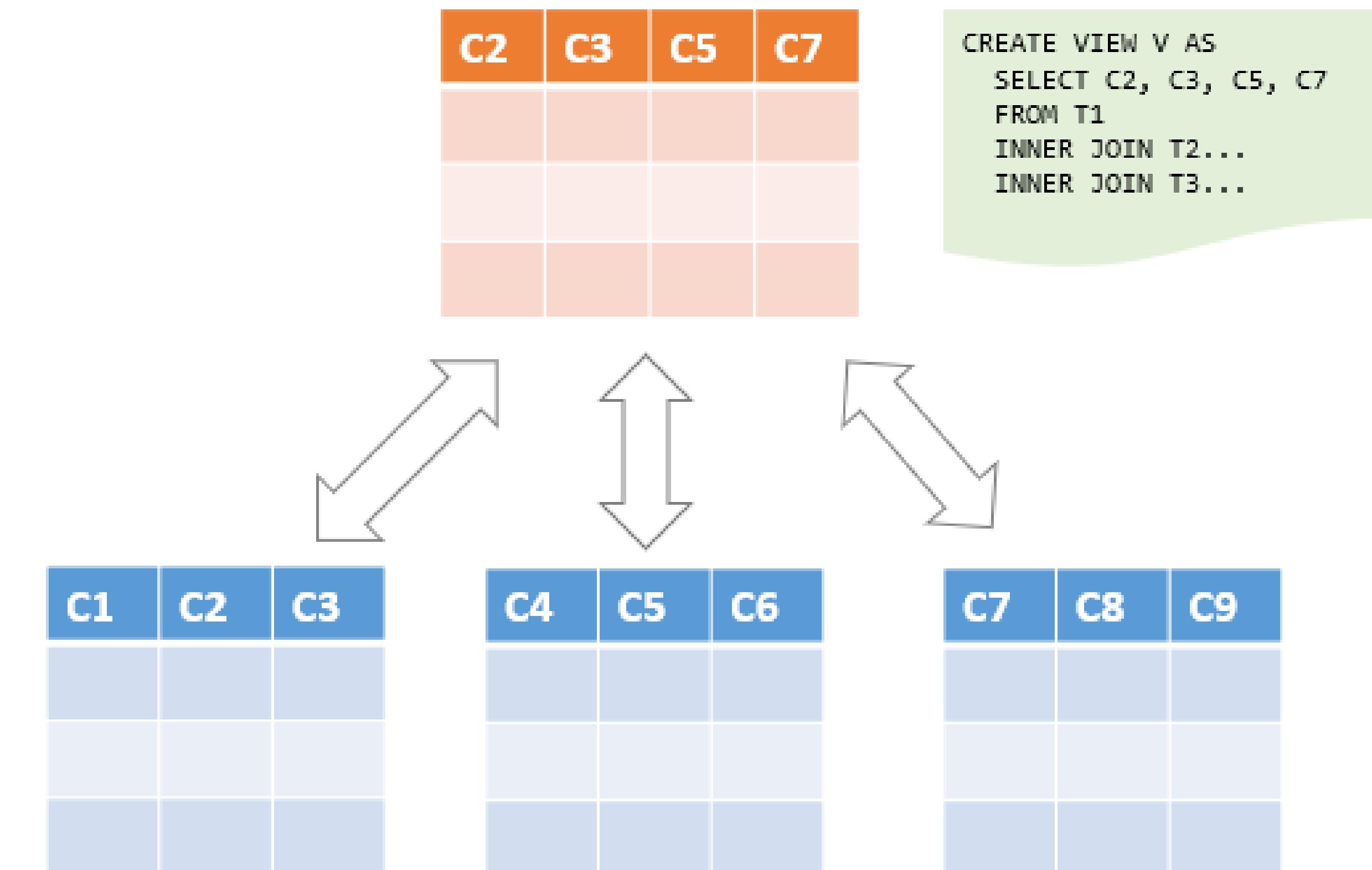
MySQL VIEWS

MySQL Views

- A view is defined by means of an **SQL query** and its content is generated upon invocation of the view by an application or other query.
- A view is a virtual table without physical tuples
- You can run queries against the view.

Why MySQL Views?

A view does not physically store the data. Each time you query the view, the underlying query (view definition) is executed



Source: <https://www.mysqltutorial.org/mysql-views-tutorial.aspx>

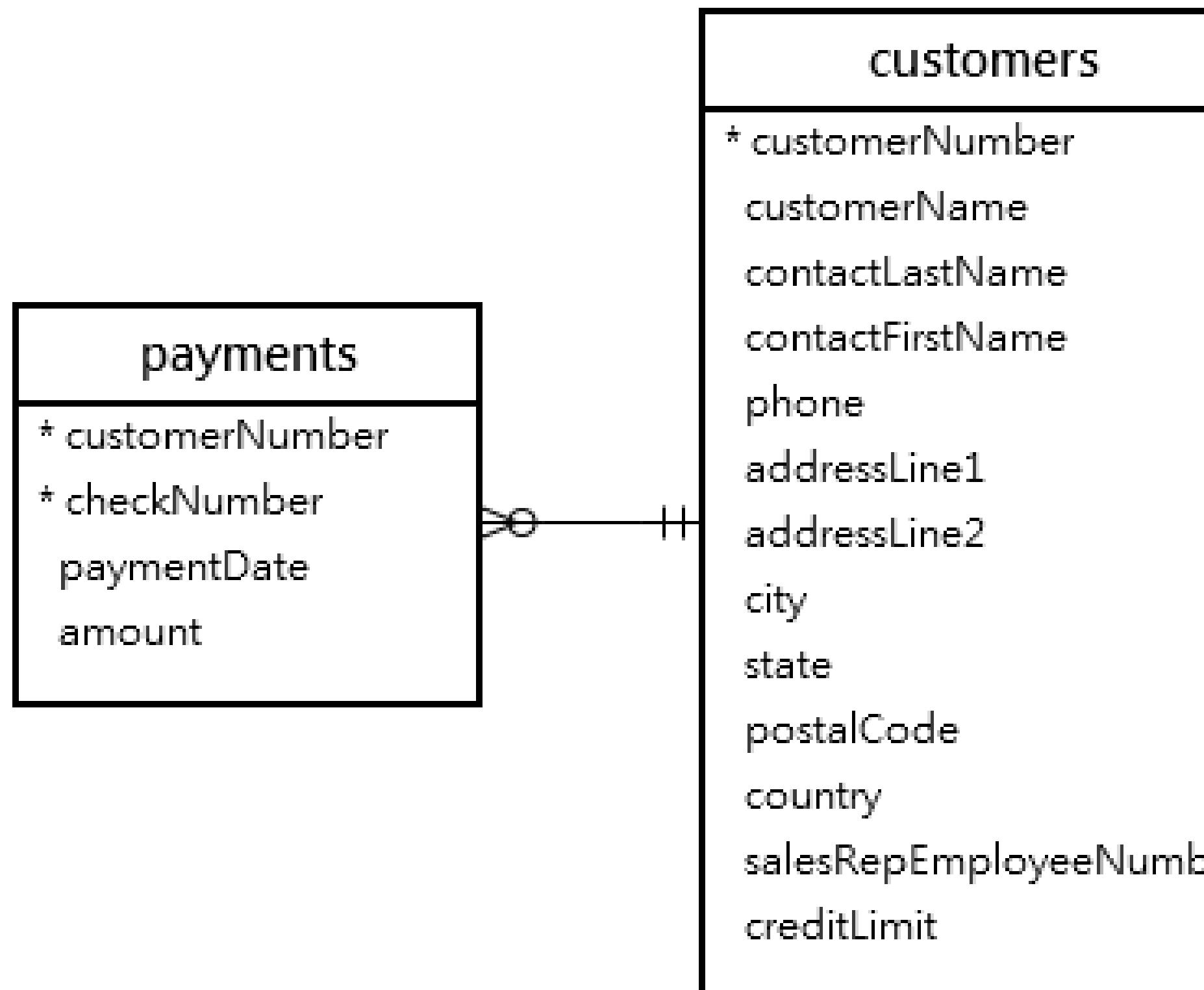
CREATE VIEW SINTAX

```
CREATE [OR REPLACE] VIEW [db_name.]view_name  
[(column_list)]  
AS  
select-statement;
```

NOTE: The name a view cannot be the same as the name of an existing table.

Using a VIEW - Example

This query joins customers and payments. What if you want to do it frequently?



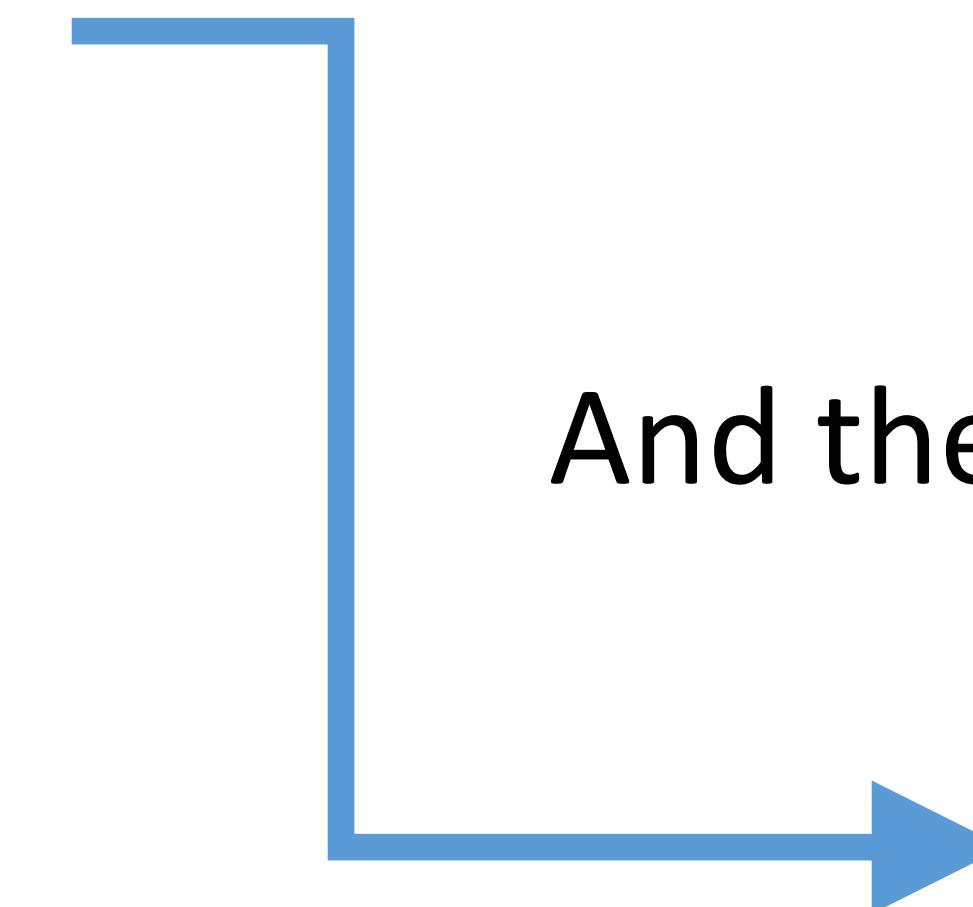
	customerName	checkNumber	paymentDate	amount
▶	Atelier graphique	HQ336336	2004-10-19	6066.78
	Atelier graphique	JM555205	2003-06-05	14571.44
	Atelier graphique	OM314933	2004-12-18	1676.14
	Signal Gift Stores	BO864823	2004-12-17	14191.12
	Signal Gift Stores	HQ55022	2003-06-06	32641.98
	Signal Gift Stores	ND748579	2004-08-20	33347.88
	Australian Collectors, Co.	GG31455	2003-05-20	45864.03
	Australian Collectors, Co.	MA765515	2004-12-15	82261.22
	Australian Collectors, Co.	NP603840	2003-05-31	7565.08
	Australian Collectors, Co.	NR27552	2004-03-10	44894.74
	La Rochelle Gifts	DB933704	2004-11-14	19501.82
	La Rochelle Gifts	LN373447	2004-08-08	47924.19

Source: <https://www.mysqltutorial.org/mysql-views-tutorial.aspx>

Using a VIEW - Example

Just create a view:

```
CREATE VIEW customerPayments
AS
SELECT
    customerName,
    checkNumber,
    paymentDate,
    amount
FROM
    customers
INNER JOIN
    payments USING
        (customerNumber);
```



And then.... Just query the view:

```
SELECT * FROM
customerPayments;
```

SQL Views – More Examples

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50
```

```
CREATE VIEW TOPSUPPLIERS_SF
AS SELECT * FROM TOPSUPPLIERS
WHERE SUPCITY = 'San Francisco'
```

TRIGGERS

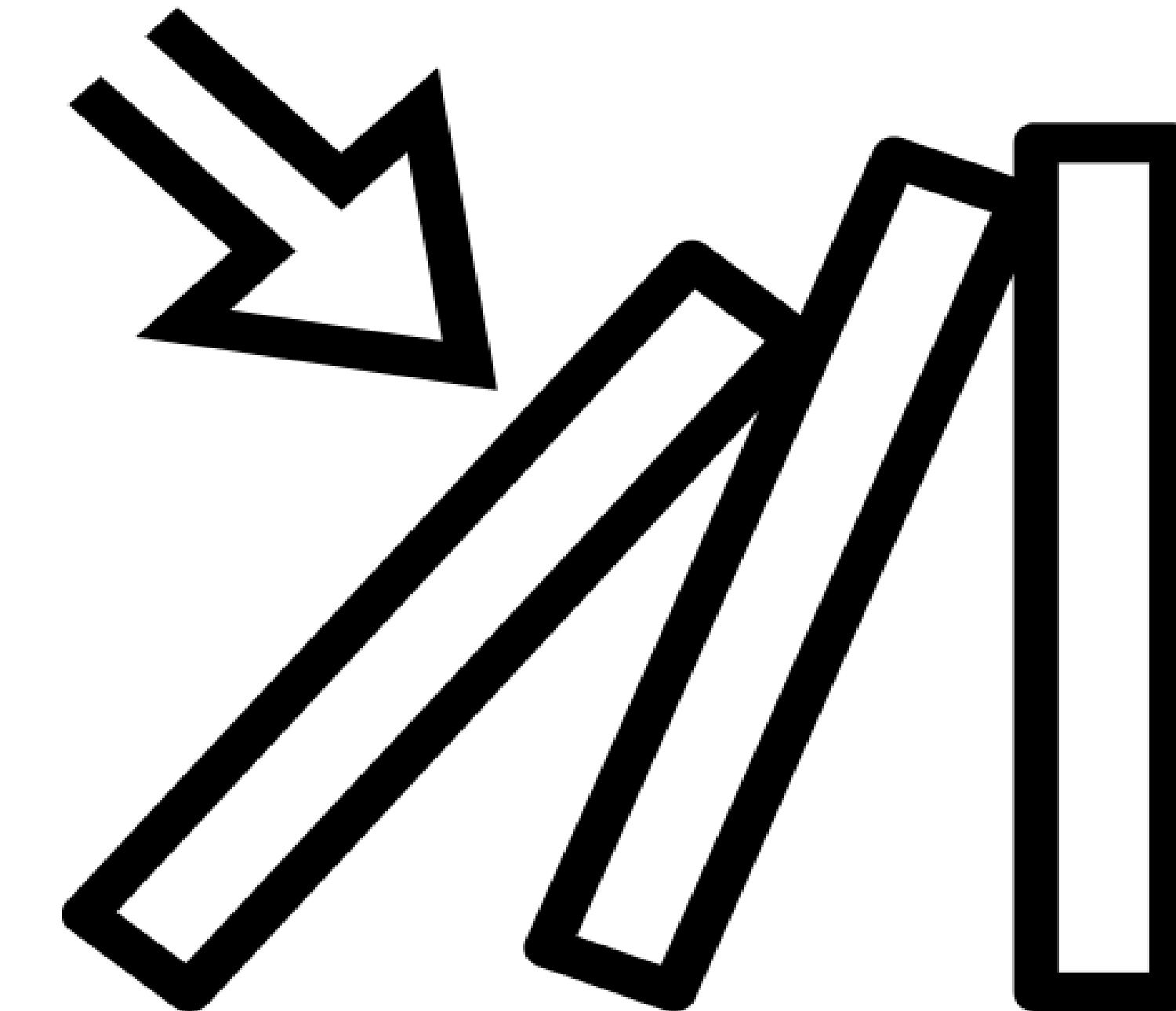


Image source: <https://icons8.com/icon/2158/trigger>

Triggers

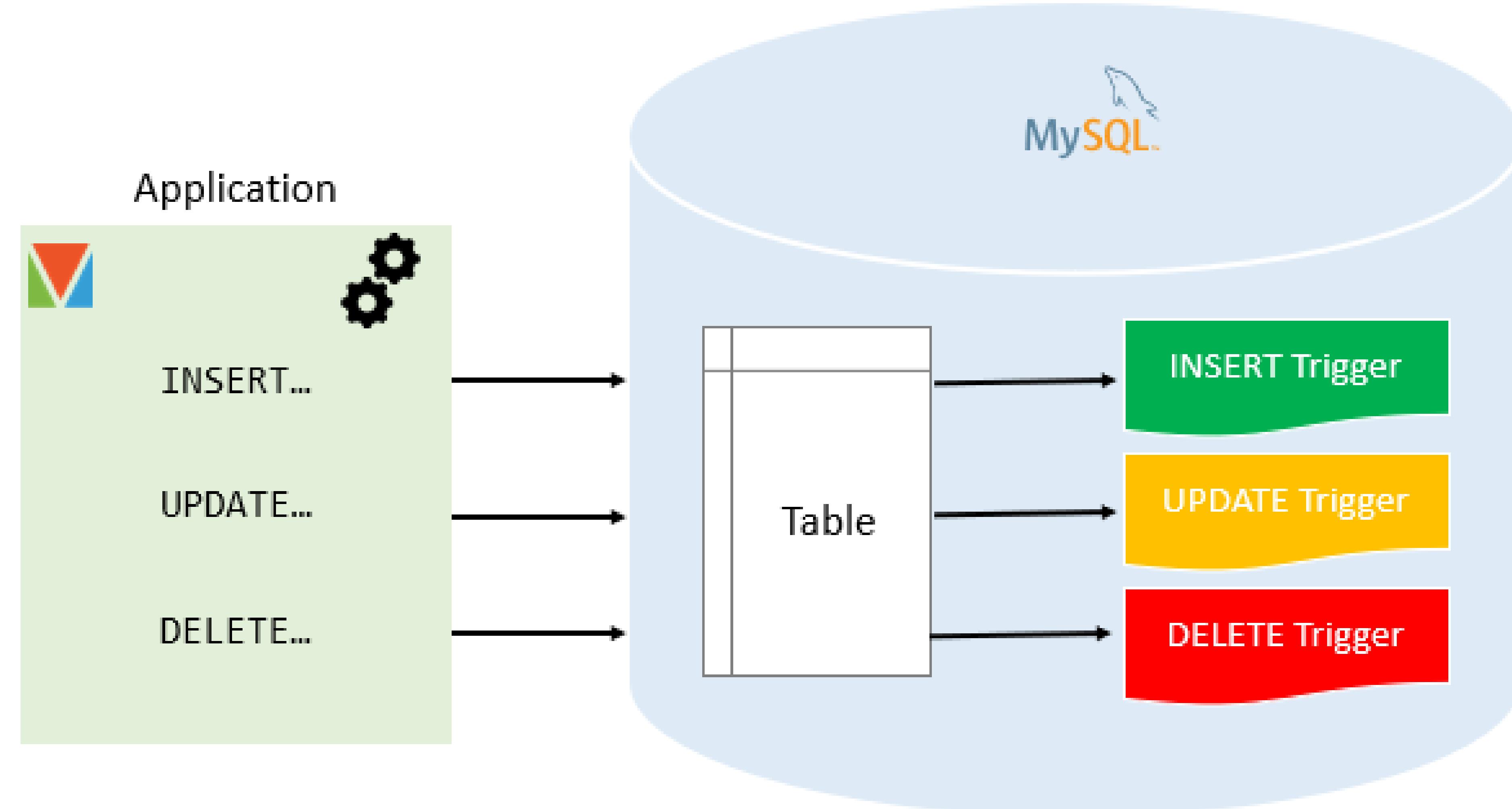


Image source: <https://www.mysqltutorial.org/mysql-triggers.aspx>

Triggers

- A **trigger** is a piece of SQL code consisting of declarative and/or procedural instructions and stored in the catalog of the RDBMS.
- It is automatically activated and run by the RDBMS whenever a specific event (e.g. INSERT, UPDATE, DELETE) occurs and a specific condition is evaluated as true.

Image source: <https://www.mysqltutorial.org/mysql-triggers.aspx>

Triggers

- Two types of triggers:
 - A row-level trigger is activated for each row that is inserted, updated, or deleted.
 - A statement-level trigger is executed once for each transaction regardless of how many rows are inserted, updated, or deleted.
- **MySQL supports only row-level triggers. It doesn't support statement-level triggers.**

Image source: <https://www.mysqltutorial.org/mysql-triggers.aspx>

Triggers - Syntax

CREATE TRIGGER trigger_name trigger_time trigger_event

ON table_name

FOR EACH ROW

BEGIN

Here your code of what your trigger will do

END;

Image source: <https://www.mysqltutorial.org/mysql-triggers.aspx>

Triggers - Syntax

CREATE TRIGGER trigger_name trigger_time trigger_event

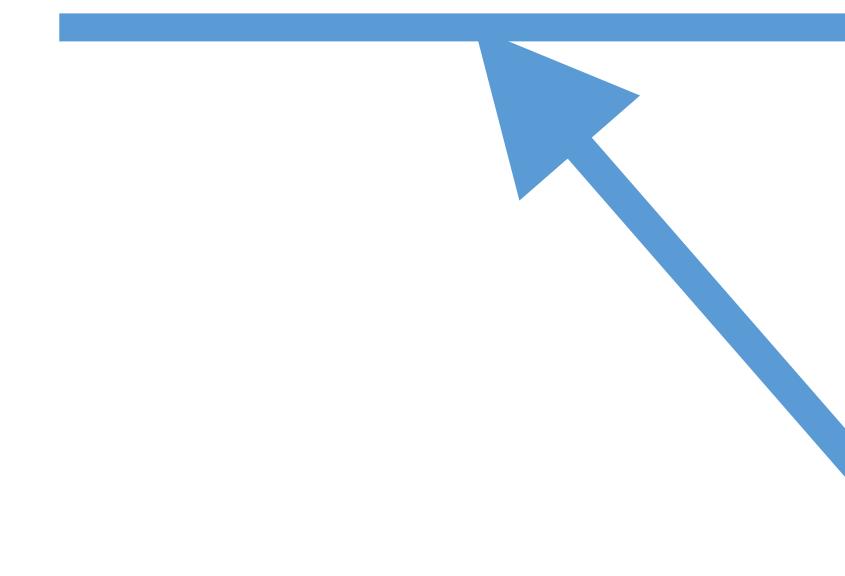
ON table_name

FOR EACH ROW

BEGIN

Here your code of what your trigger will do

END;



**BEFORE or
AFTER**

Triggers - Syntax

CREATE TRIGGER trigger_name trigger_time trigger_event

ON table_name

FOR EACH ROW

BEGIN

Here your code of what your trigger will do

END;

INSERT
UPDATE
DELETE

Image source: <https://www.mysqltutorial.org/mysql-triggers.aspx>

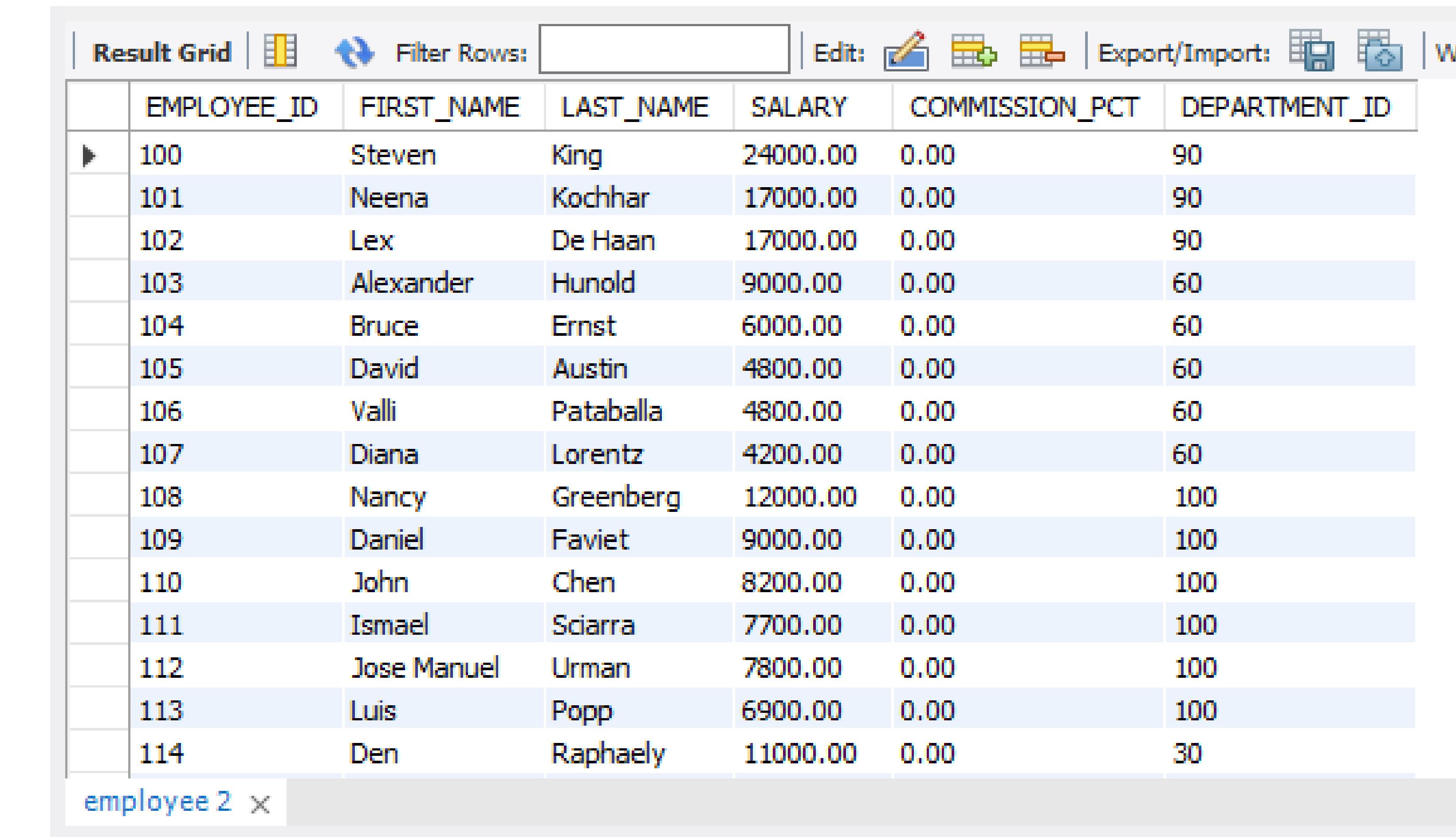
BEFORE INSERT Trigger Example

When a new employee is added to the table **EMPLOYEES** and if the employee belongs to the **SALES** department (**DEPARTMENT_ID = 80**), you will give a commission of 15%.

Create a trigger to implement this functionality. Use **BEFORE TRIGGER** condition

BEFORE INSERT Trigger Example

Table EMPLOYEE:



The screenshot shows a database grid interface with the following features at the top:

- Result Grid
- Filter Rows: (with a search bar)
- Edit: (with icons for edit, insert, delete, and refresh)
- Export/Import: (with icons for export to various formats)

The table structure is defined by the following columns:

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
▶	100	Steven	King	24000.00	0.00	90
	101	Neena	Kochhar	17000.00	0.00	90
	102	Lex	De Haan	17000.00	0.00	90
	103	Alexander	Hunold	9000.00	0.00	60
	104	Bruce	Ernst	6000.00	0.00	60
	105	David	Austin	4800.00	0.00	60
	106	Valli	Pataballa	4800.00	0.00	60
	107	Diana	Lorentz	4200.00	0.00	60
	108	Nancy	Greenberg	12000.00	0.00	100
	109	Daniel	Faviet	9000.00	0.00	100
	110	John	Chen	8200.00	0.00	100
	111	Ismael	Sciarra	7700.00	0.00	100
	112	Jose Manuel	Urman	7800.00	0.00	100
	113	Luis	Popp	6900.00	0.00	100
	114	Den	Raphaely	11000.00	0.00	30

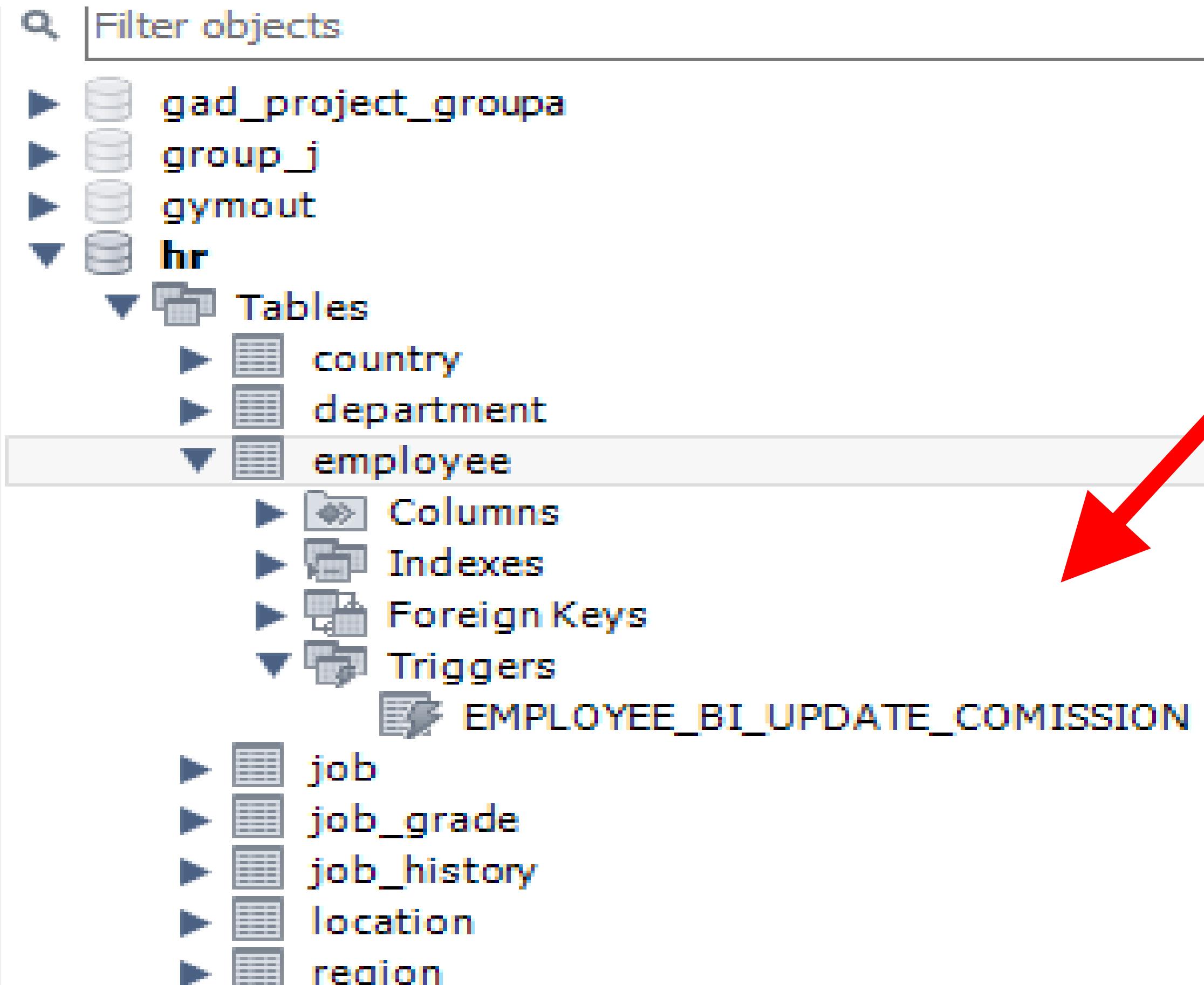
At the bottom of the grid, there is a footer bar with the text "employee 2" and a close button "x".

BEFORE INSERT Trigger Example

```
DELIMITER $$

CREATE TRIGGER EMPLOYEE_BI_UPDATE_COMISSION
BEFORE INSERT
ON employee
FOR EACH ROW
BEGIN
    IF NEW.department_id = 80 THEN
        SET NEW.commission_pct = 0.15;
    END IF;
END $$
```

BEFORE INSERT Trigger Example



The new trigger appears in the schema tree for the employee table

BEFORE INSERT Trigger Example

Let's insert a new employee. Notice we are not specifying any commission percentage in the insert:

```
INSERT INTO
EMPLOYEE (EMPLOYEE_ID,FIRST_NAME,LAST_NAME,
SALARY,DEPARTMENT_ID)
VALUES (208, 'Mijail', 'Naranjo', 9000.00, 80);
```

BEFORE INSERT Trigger Example

```
SELECT FIRST_NAME, LAST_NAME, SALARY,  
COMMISSION_PCT, DEPARTMENT_ID  
FROM employee  
WHERE EMPLOYEE_ID = 208;
```

The screenshot shows a database query results grid. At the top, there are buttons for 'Result Grid' (selected), 'Filter Rows' (with an input field), 'Export' (with a dropdown menu), and 'Wrap Cell Content'. The table has five columns: FIRST_NAME, LAST_NAME, SALARY, COMMISSION_PCT, and DEPARTMENT_ID. The data for employee ID 208 is displayed: Mijail Naranjo with a salary of 9000.00, a commission percentage of 0.15, and department ID 80.

FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
Mijail	Naranjo	9000.00	0.15	80

Triggers - Advantages

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

Triggers - Disadvantages

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

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

Let's have
some fun!

END OF LECTURE 6

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



eduniversal



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa



Information
Management
School

Storing and Retrieving Data

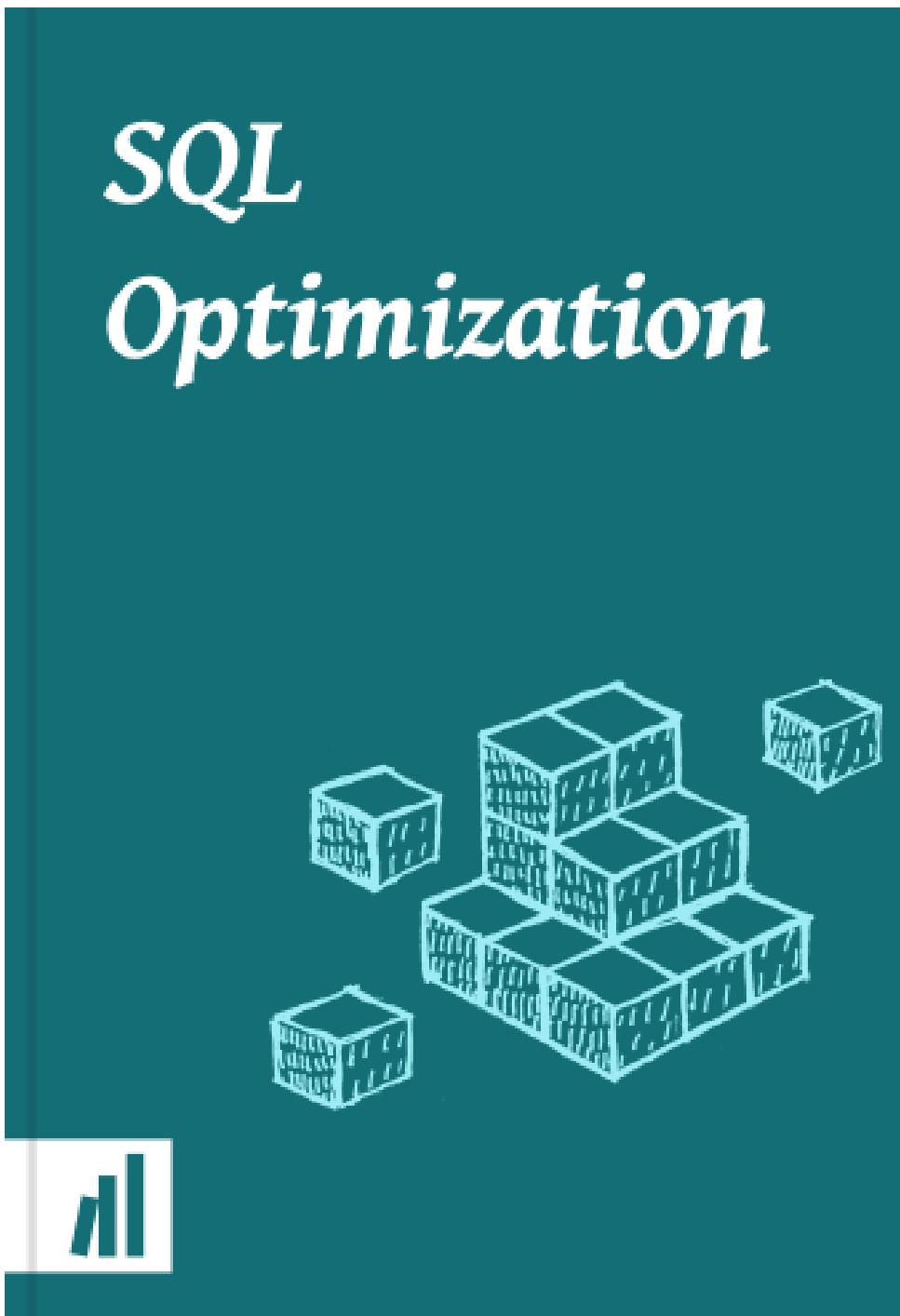
Lecture 6

MySQL query optimization

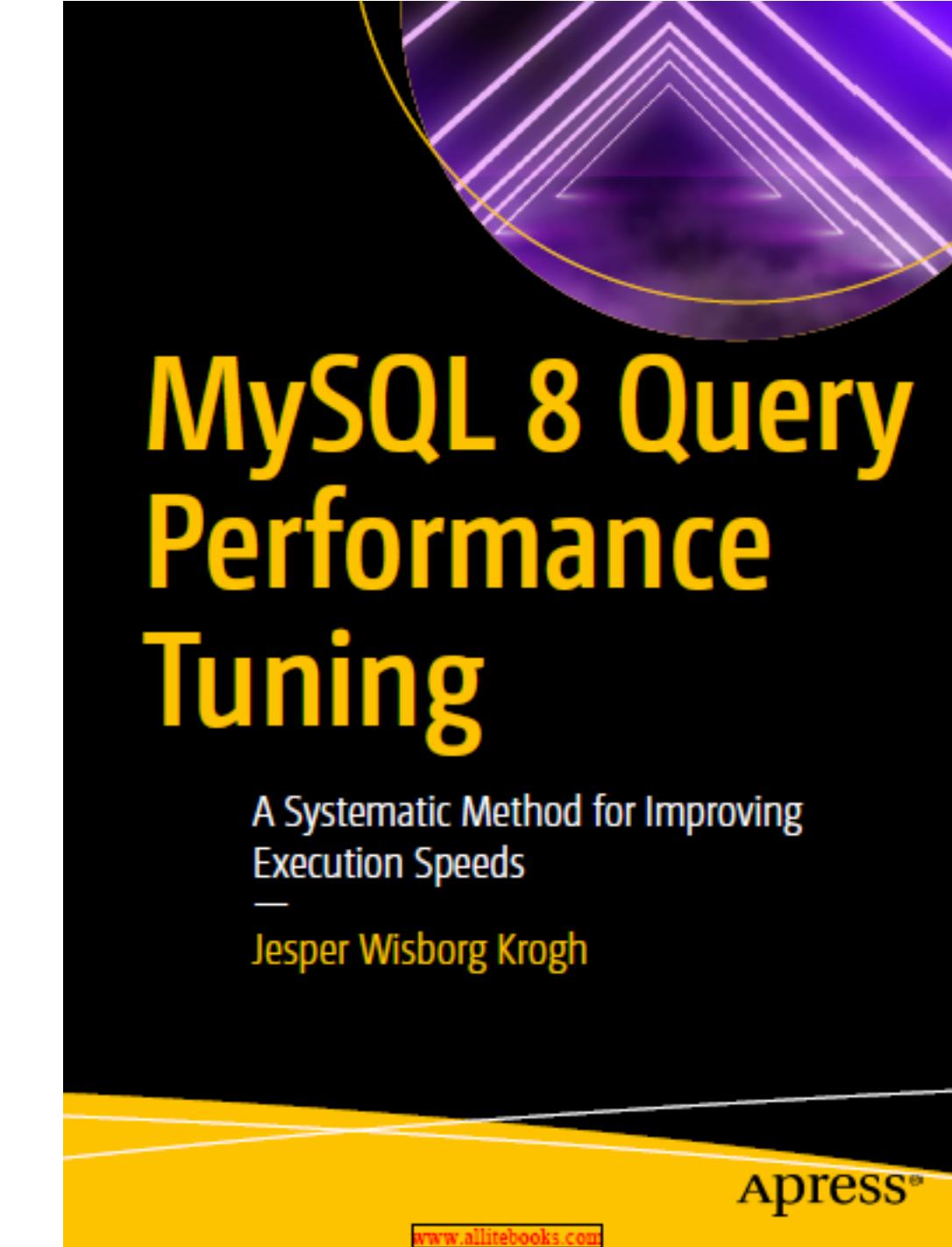
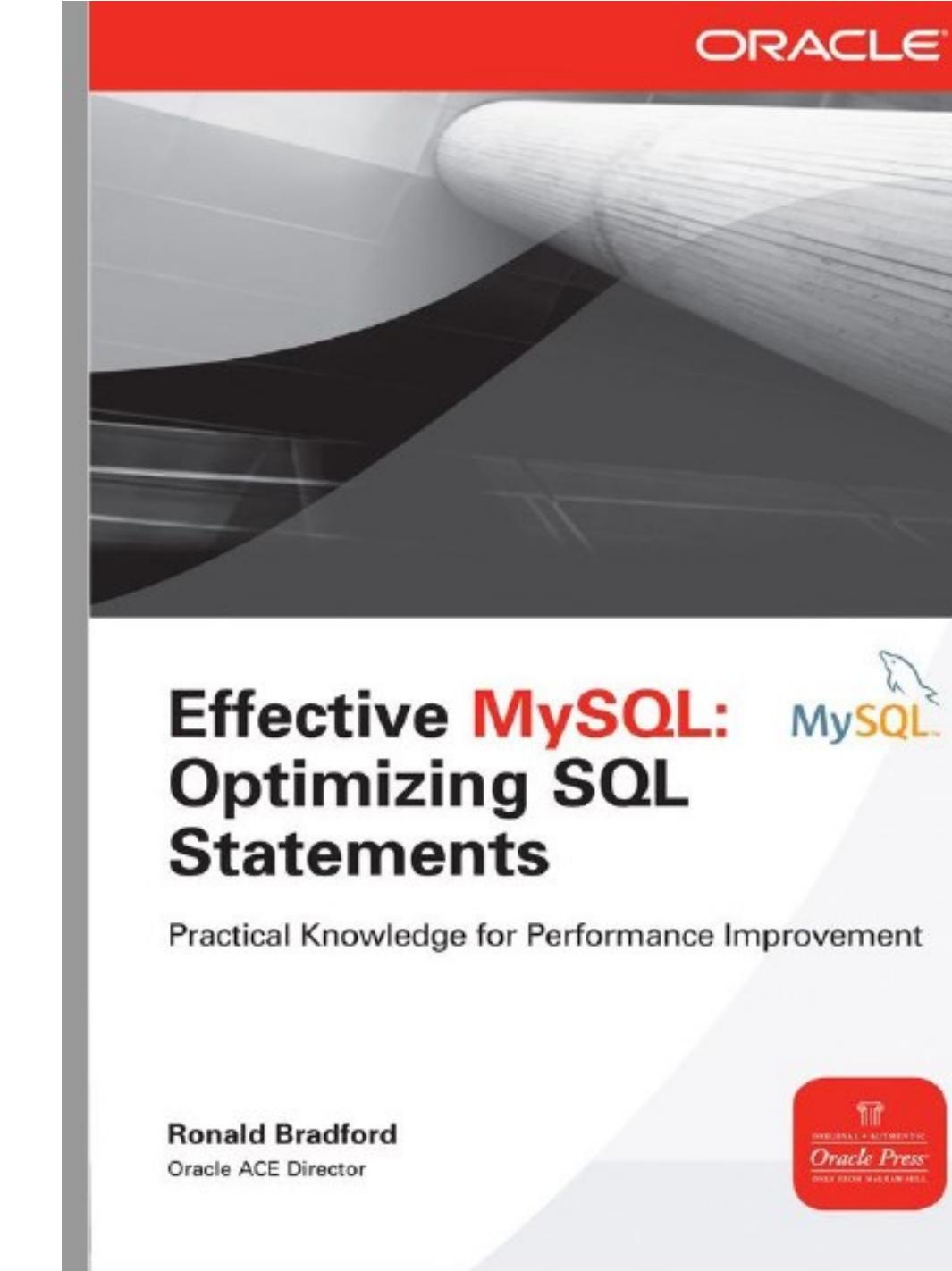
MySQL query optimization

- The problem
- The EXPLAIN command
- Indexes
- Impact of data types in performance
- Final tips for fast querying

Recommended literature



<https://dataschool.com/sql-optimization/>



Introduction

PROBLEM:

Users are complaining that the application is **slow!**



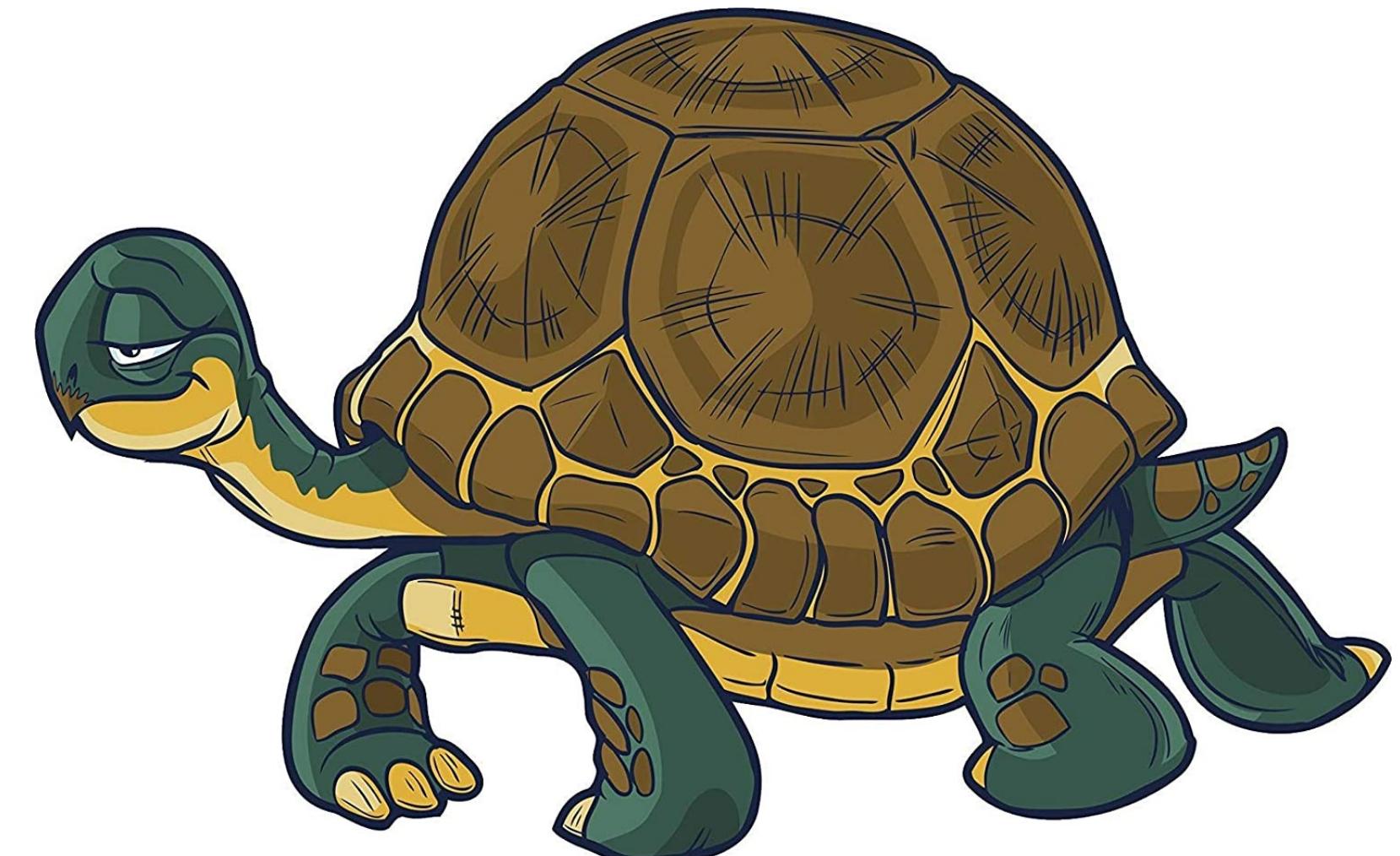
Introduction

Users are complaining that the application is **slow!**

This could be (among dozens of reasons) because of **a slow running SQL query** in the database



But... what means
“slow”?



What means “slow”?

User's feeling about response time:

0.1 second → the user feels that the system is reacting instantaneously (no feedback needed).

1.0 second → limit for the user's flow of thought to stay uninterrupted (no feedback needed, but the user loses the feeling of operating directly on the data).

10 seconds → limit for keeping the user's attention (users will want to perform other tasks, feedback is needed!)

Source: <https://www.nngroup.com/articles/response-times-3-important-limits/>

SQL query optimization

For instance, is this query slow?



```
mysql> SELECT * FROM inventory WHERE item_id = 16102176;  
Empty set (3.19 sec)
```

SQL query optimization

For instance, is this query slow?

```
mysql> SELECT * FROM inventory WHERE item_id = 16102176;  
Empty set (3.19 sec)
```



Yes!! Very slow!!

But....How do we know the query has a problem?

SQL query optimization

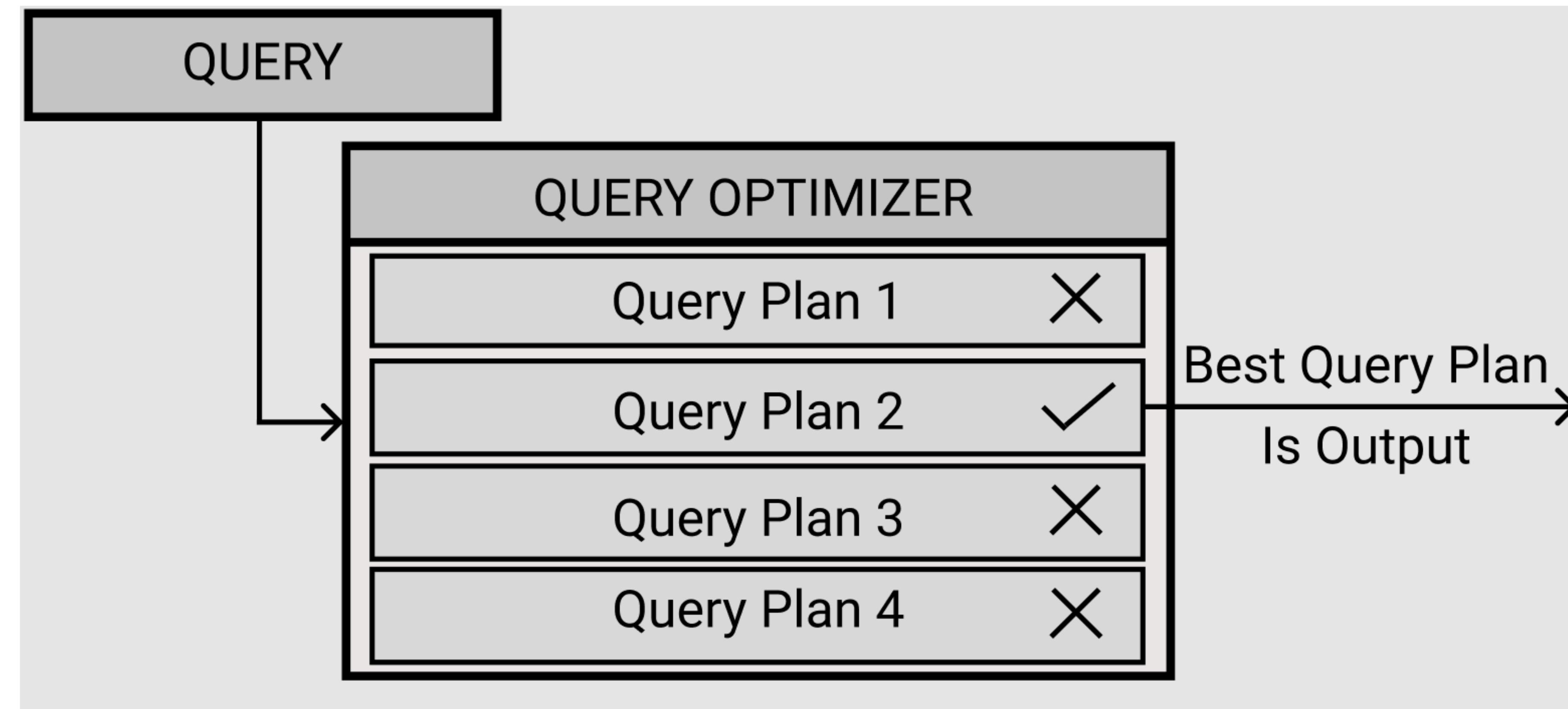
But....How do we know the query has a problem?

Generating a Query Execution Plan (QEP)

A Query plan is a list of instructions that the database needs to follow in order to execute a query on the data.

SQL query optimization

The **Query Optimizer** generates **multiple** Query Plans for a single query and determines the most efficient plan to run.



Example... Let's compare two queries

SQL query optimization - Example

Example:

We want to find a particular tool from table TOOLS:

```
SELECT *
FROM tools
WHERE name='Screwdriver' ;
```

Both queries return the same thing:

```
SELECT *
FROM tools
WHERE id=3 ;
```

ID	Name
1	Callipers
2	Hammer
3	Screwdriver
4	Wrench
5	Hammer

But...Which one is more efficient?

SQL query optimization - Example

Same results, but may have different final query plans:

The first query **checks all five rows of the table TOOLS**

```
SELECT *
FROM tools
WHERE name='Screwdriver' ;
```

ID	Name
1	Callipers
2	Hammer
3	Screwdriver
4	Wrench
5	Hammer

SQL query optimization - Example

Same results, but may have different final query plans:

The second query **uses a sequential seek**. It will stop once a matching is found.

```
SELECT *
FROM tools
WHERE id=3;
```

ID	Name
1	Callipers
2	Hammer
3	Screwdriver
4	Wrench
5	Hammer

The second query is faster!

So... How to examine the query plan?

Use the **EXPLAIN** command

SQL query optimization – The EXPLAIN command

- The EXPLAIN command is used to show the query plan.
- The EXPLAIN does not run the actual SQL statement (except subqueries).



The EXPLAIN command **does not provide any tuning recommendations**, but it does provide valuable information to help you make tuning decisions.!!

More detailed info about EXPLAIN:

<https://dev.mysql.com/doc/refman/8.0/en/explain.html>

Source: Bradford, R. (2011). *Effective MySQL Optimizing SQL Statements*.

Example 1 using the EXPLAIN command

SQL query optimization – The EXPLAIN command

Example of EXPLAIN command:

```
mysql> EXPLAIN SELECT * FROM inventory WHERE item_id = 16102176;
***** 1. row *****
id: 1
select_type: SIMPLE
table: inventory
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 787338
Extra: Using where
```

Any query that **does not use an index** signified by the **key column** in the preceding output can be considered a **poorly tuned SQL query**.

approximate number of rows read in order to find the occurrences matching item_id=16102176.

Source: Bradford, R. (2011). *Effective MySQL Optimizing SQL Statements*.

SQL query optimization – The EXPLAIN command

Clearly, the query is poorly tuned!

Solution: We need to optimize the query **creating an index** based on the WHERE clause.

```
mysql> ALTER TABLE inventory ADD INDEX (item_id);  
Query OK, 734787 rows affected (54.22 sec)  
Records: 734787  Duplicates: 0  Warnings: 0
```



Be careful when creating an index in production environment. For larger tables, an ALTER statement can takes hours, or even days, to complete!

SQL query optimization – The EXPLAIN command

Let's confirm our optimization.

```
mysql> EXPLAIN SELECT * FROM inventory WHERE item_id = 16102176
*****
1. row ****
id: 1
select_type: SIMPLE
table: inventory
type: ref
possible_keys: item_id
key: item_id
key_len: 4
ref: const
rows: 1
Extra:
```

The query uses an index now! The estimated rows decreased to 1 !!!

SQL query optimization – The EXPLAIN command

How to identify primary issues quickly using EXPLAIN command?

- No index used (NULL specified in the *key* column)
- A large number of rows processed (the *rows* column)
- A large number of indexes evaluated (the *possible_keys* column)

Understanding the MySQL EXPLAIN statement is an art that can take years of experience to perfect.!!

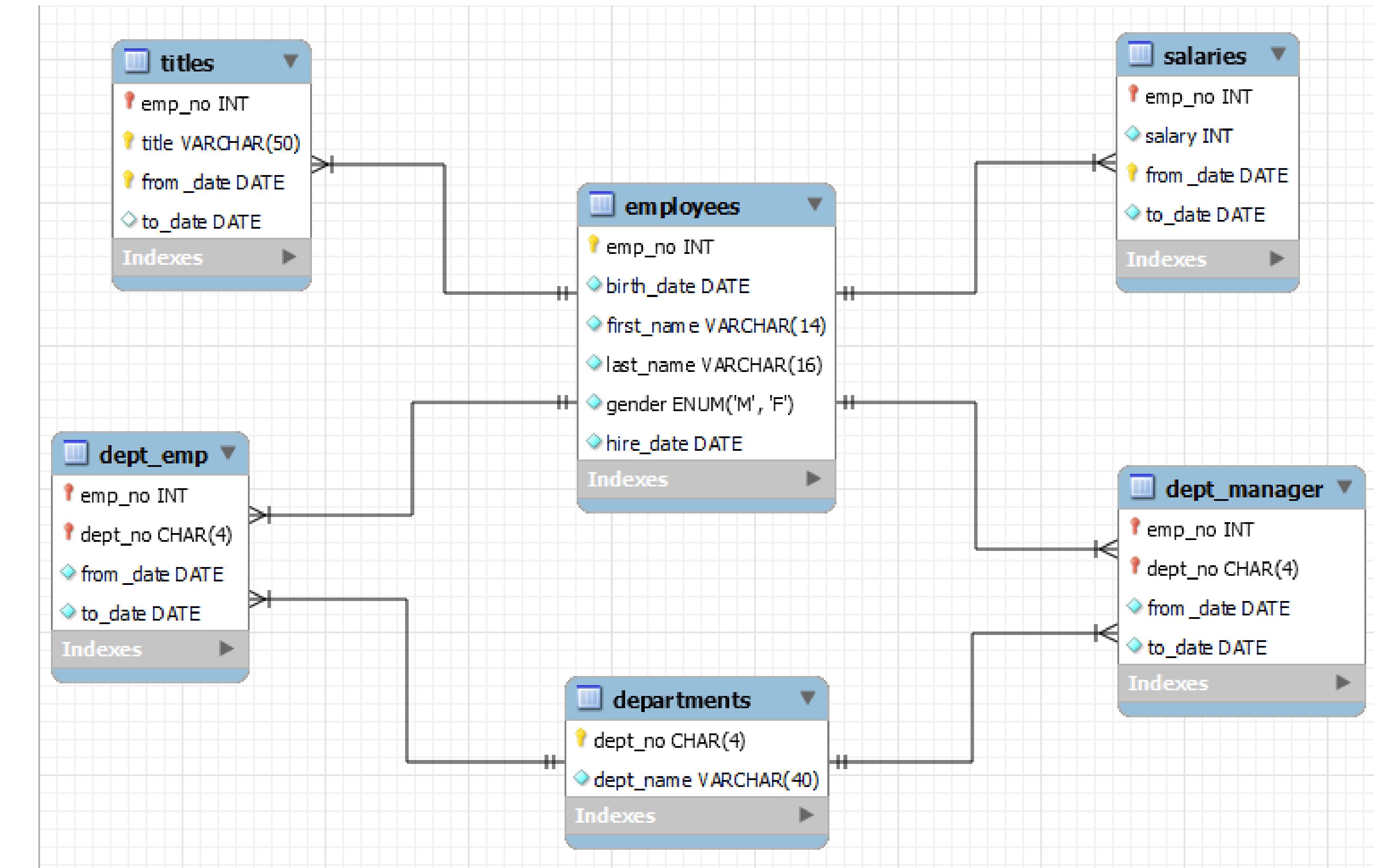
Example 2 using EXPLAIN

Using the database Employees in Workbench

https://github.com/databarmer/test_db

SQL query optimization – Example

The EMPLOYEES ERD:



SQL query optimization – Example

We run a query on EMPLOYEES looking for a specific first name:

```
3 • SELECT * FROM employees WHERE first_name = 'Elvis'
```

Timing (as measured at client side):
Execution time: 0:00:0.3900000

Timing (as measured by the server):
Execution time: 0:00:0.39520460
Table lock wait time: 0:00:0.00015000

SQL query optimization – Example

Now we create an index by first_name on EMPLOYEES, and re-run the query:

```
2  
3 • SELECT * FROM employees WHERE first_name = 'Elvis'  
4
```

Timing (as measured at client side):
Execution time: 0:00:0.03200000

Timing (as measured by the server):
Execution time: 0:00:0.04579960
Table lock wait time: 0:00:0.00017400

SQL query optimization – Example

Now we create an index by first_name on EMPLOYEES, and re-run the query:

```
<  
3 • SELECT * FROM employees WHERE first_name = 'Elvis'  
>
```

Timing (as measured at client side):
Execution time: 0:00:0.03200000

Timing (as measured by the server):
Execution time: 0:00:0.04579960
Table lock wait time: 0:00:0.00017400

If we keep running the same query about 10 more times:

SQL query optimization – Example

Now we create an index by first_name on EMPLOYEES, and re-run the query:

```
2  
3 • SELECT * FROM employees WHERE first_name = 'Elvis'  
4
```

Timing (as measured at client side):

Execution time: 0:00:0.03200000

Timing (as measured by the server):

Execution time: 0:00:0.04579960

Table lock wait time: 0:00:0.00017400

If we keep running the same query about 10 more times:

Timing (as measured at client side):

Execution time: 0:00:0.00000000

Timing (as measured by the server):

Execution time: 0:00:0.00441360

Table lock wait time: 0:00:0.00012700

SQL query optimization – Example

Now we create an index by first_name on EMPLOYEES, and re-run the query:

```
<  
3 • SELECT * FROM employees WHERE first_name = 'Elvis'  
>
```

Timing (as measured at client side):
Execution time: 0:00:0.03200000

Timing (as measured by the server):
Execution time: 0:00:0.04579960
Table lock wait time: 0:00:0.00017400

**Why is getting faster?
Query statistics are being updated!!**

Let's keep running the same query about 10 times:

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00441360
Table lock wait time: 0:00:0.00012700

SQL query optimization – Example

Comparing the times differences:

Without optimization

Timing (as measured at client side):

Execution time: 0:00:0.39000000

Timing (as measured by the server):

Execution time: 0:00:0.39520460

Table lock wait time: 0:00:0.00015000

With optimization

Timing (as measured at client side):

Execution time: 0:00:0.00000000

Timing (as measured by the server):

Execution time: 0:00:0.00441360

Table lock wait time: 0:00:0.00012700

About 100 times faster! → Huge improvement!!!

SQL query optimization – The EXPLAIN command



Be careful!! Optimizing SQL statements is not about just adding an index!!



The generated Query Plan is not guaranteed and can change, depending on several factors.



For UPDATE and DELETE statements, you need to rewrite the query as a SELECT statement to confirm index usage.

Indexes

Creating the right indexes is one of the most significant techniques
for SQL performance tuning

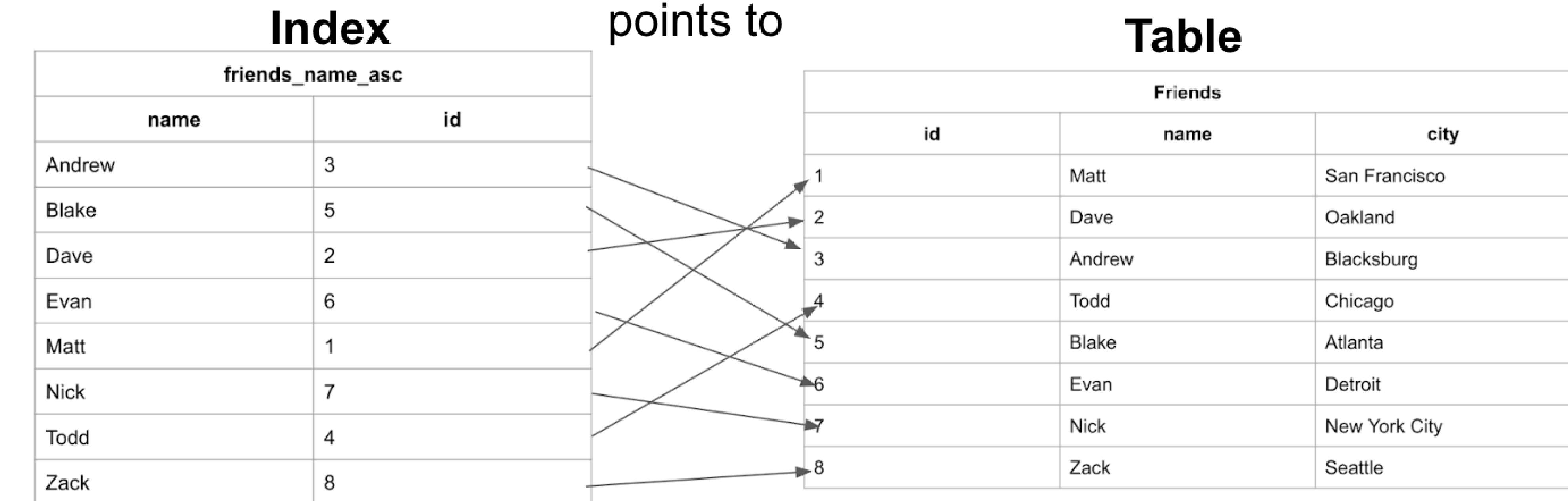
Complete documentation about index creation syntax:

<https://dev.mysql.com/doc/refman/8.0/en/create-index.html>

SQL query optimization – Understanding Indexes

What is an Index?

An index is a structure that holds the field the index is sorting and a pointer from each record to their corresponding record in the original table where the data is stored.

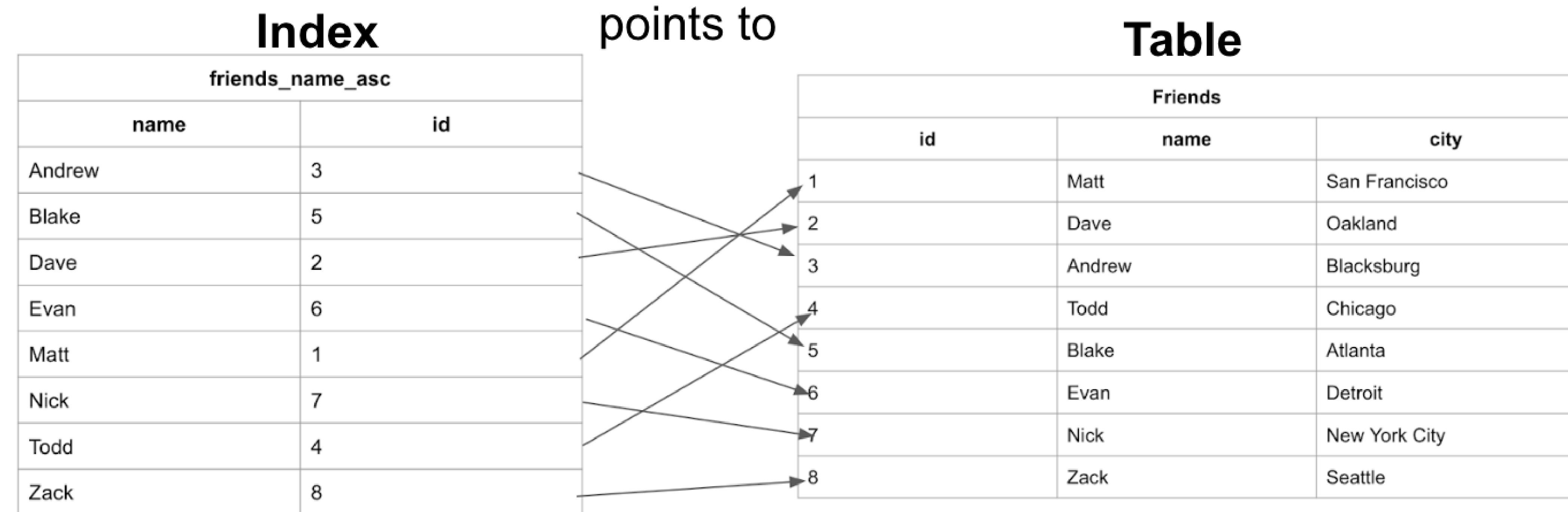


Source: David, Matt. (2021). SQL Optimization. <https://dataschool.com/sql-optimization/>

SQL query optimization – Understanding Indexes

The Index has the names stored in alphabetical order

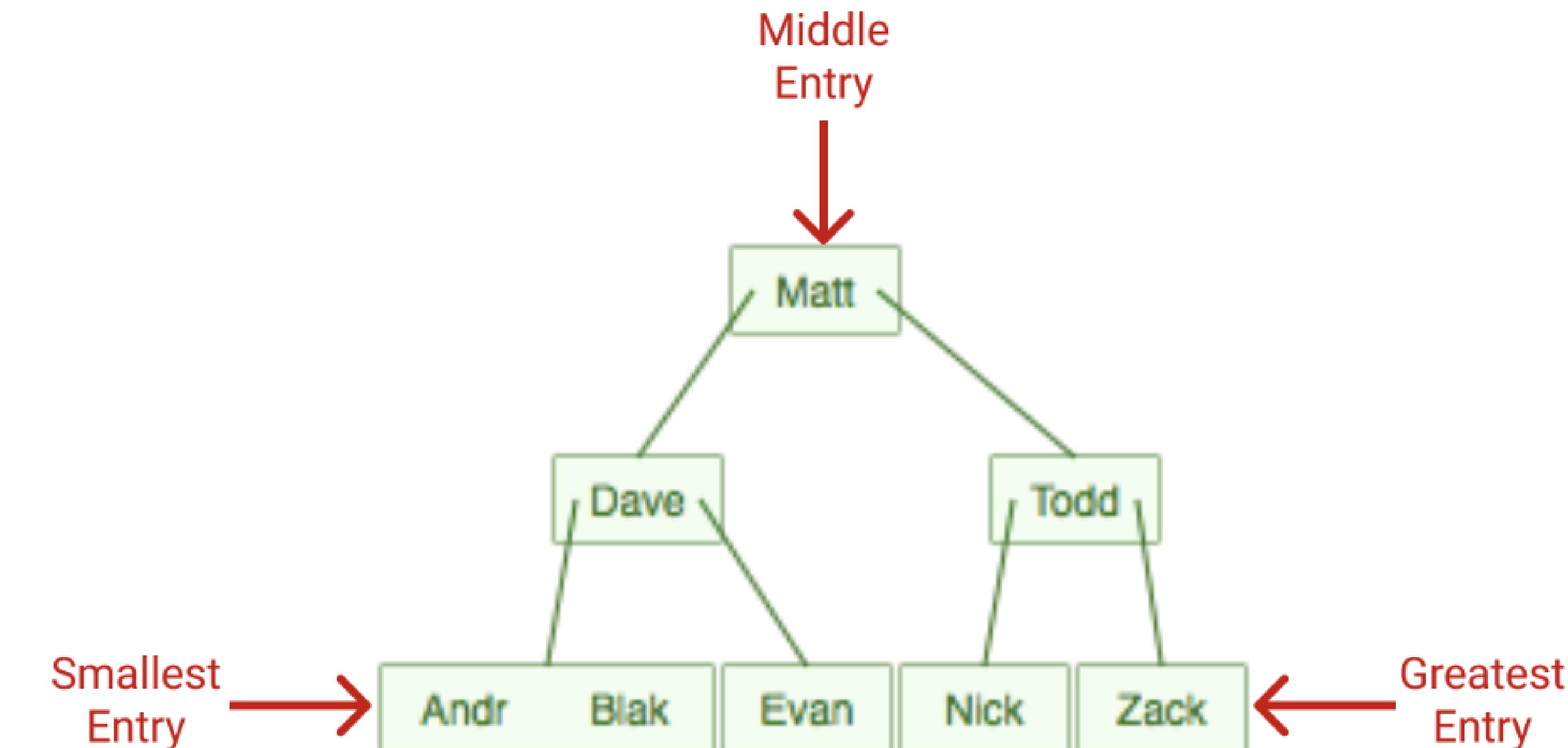
The table has the data ordered by an incrementing id (the order in which the data was added).



Source: David, Matt. (2021). SQL Optimization. <https://dataschool.com/sql-optimization/>

SQL query optimization – Understanding Indexes

Indexes are stored and searched as B-trees. It creates a tree-like structure that sorts data for quick searching.



Source: David, Matt. (2021). SQL Optimization. <https://dataschool.com/sql-optimization/>

SQL query optimization – Understanding Indexes

Indexing makes columns faster to query by creating pointers to where data is stored within a database.

With no index

```
SELECT * FROM friends WHERE name = 'Zack';
```

friends		
id	name	city
1	Matt	San Francisco
2	Dave	Oakland
3	Andrew	Blacksburg
4	Todd	Chicago
5	Blake	Atlanta
6	Evan	Detroit
7	Nick	New York City
8	Zack	Seattle

Indexes use a binary search

```
SELECT * FROM friends WHERE name = 'Zack';
```

friends_name_asc	
Name	Index
Andrew	3
Blake	5
Dave	2
Evan	6
Matt	1
Nick	7
Todd	4
Zack	8

Suggested video about indexing: <https://www.youtube.com/watch?v=kv3jC0P4gOc>

Source: David, Matt. (2021). SQL Optimization. <https://dataschool.com/sql-optimization/>

SQL query optimization – Understanding Indexes

Understanding Index cardinality (number of unique values)

- When you have multiple different indexes (listed in *possible_keys* of the query plan), MySQL tries to identify the most effective index for the query.
- The optimizer chooses an index based on the **estimated cost** to do the least amount of work, not what a human considers the right order. (EXPLAIN ANALYZE and EXPLAIN FORMAT = TREE to see the cost
<https://dev.mysql.com/doc/refman/8.0/en/explain.html>)
- We can use the **index cardinality** to confirm the likely reason for this decision.

Source: Bradford, R. (2011). *Effective MySQL Optimizing SQL Statements*. McGraw Hill Professional.

SQL query optimization – Understanding Indexes

Understanding Index cardinality (number of unique values)



Source: <https://www.youtube.com/watch?v=jx5vUSxn3m8>

SQL query optimization – Understanding Indexes

Example: MySQL has to make a choice between the possible indexes of table *artist*. **What is the best index for the query?**

```
mysql> EXPLAIN SELECT artist_id, name, country_id
   -> FROM artist
   -> WHERE type='Band'
   -> AND founded = 1980\G
*****
1. row *****
    id: 1
  select_type: SIMPLE
        table: artist
       type: ref
possible_keys: founded,founded_2,type
      key: founded
    key_len: 2
      ref: const
     rows: 1216
    Extra: Using where
```

```
mysql> SHOW INDEXES FROM artist\G
...
***** 3. row *****
      Table: artist
Non_unique: 1
      Key_name: founded
Seq_in_index: 1
Column_name: founded
Collation: A
Cardinality: 846
...
***** 5. row *****
      Table: artist
Non_unique: 1
      Key_name: type
Seq_in_index: 1
Column_name: type
Collation: A
Cardinality: 10
```

Source: Bradford, R. (2011). *Effective MySQL Optimizing SQL Statements*. McGraw Hill Professional.

SQL query optimization – Understanding Indexes

Example: MySQL has to make a choice between the possible indexes of table *artist*. **What is the best index for the query?**

```
mysql> SHOW INDEXES FROM artist\G
...
***** 3. row ****
      Table: artist
    Non_unique: 1
      Key_name: founded
Seq_in_index: 1
Column_name: founded
      Collation: A
Cardinality: 846
...
***** 5. row ****
      Table: artist
    Non_unique: 1
      Key_name: type
Seq_in_index: 1
Column_name: type
      Collation: A
Cardinality: 10
```

there is a higher likelihood of finding the needed records in fewer reads from the index “founded”

Source: Bradford, R. (2011). *Effective MySQL Optimizing SQL Statements*. McGraw Hill Professional.

SQL query optimization – Understanding Indexes

Normally, you create all indexes on a table at the time the table itself is created.

If you need to create an index after table creation, you can use:

CREATE INDEX index_name ON table_name (column_list)

Source: <https://dev.mysql.com/doc/refman/8.0/en/create-index.html>.

What about data types for
query performance?

SQL query optimization – Data types

How **data types** impact performance?

Different data types have different performance characteristics.



In general, the simpler the data type, the better it performs.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types

How **data types** impact performance?

Different data types have different performance characteristics.



In general, the simpler the data type, the better it performs.

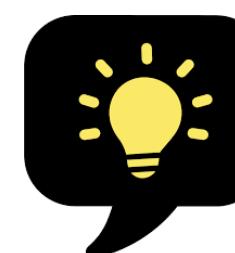
So which data types should you use?

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types

So which data types should you use?

There is not a magical formula. But here are some tips:



Choose a data type that is native for the data you need to store.

Examples:

- Choose a tinyint, int, or bigint if you need to store integers. **Example: what datatype for number of children?**
- Use the json type instead of longtext or longblob if you want to store JSON.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types

So which data types should you use?

There is not a magical formula. But here are some tips:



Choose a data type that is native for the data you need to store.

Examples:

- Choose a tinyint, int, or bigint if you need to store integers. **Example: what datatype for number of children? tinyint**
- Use the json type instead of longtext or longblob if you want to store JSON.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types



For the size of the data type, you need to consider both the current need and the future need.

Examples: If you expect to need larger values in short term, better to choose the larger data type. If the expected change is years away, it may be better to go with the smaller data type.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types



If you need to store data such as monetary values that must be exact, you should always choose the decimal data type.



If you do not need the data to be exact, the float and double data types perform better.



If you only have a few unique string values, it can also be worth considering using the enum data type.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

SQL query optimization – Data types



Remember that optimization is also about knowing when you have optimized enough.



Is the value included in indexes? The larger the values, the larger the index.

Source: Jesper Wisborg Krogh (2020). MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds

Final tips for fast querying

SQL query optimization – Final tips for faster querying

1. Define SELECT fields instead of SELECT *
2. Avoid SELECT DISTINCT if possible
3. Use WHERE instead of HAVING to define Filters
4. Use WILDCARDS at the end of the phrase

```
SELECT City FROM Customers  
WHERE City LIKE '%Char%'
```

A more efficient query would be:

```
SELECT City FROM Customers  
WHERE City LIKE 'Char%'
```

SQL query optimization – Final tips for faster querying

5. Use LIMIT to sample query results

6. Run queries during Off-Peak times

7. Replace SUBQUERIES with JOIN

8. Index your tables properly.

- The column is queried frequently
- Foreign key column(s) that reference other tables
- A unique key exists on the column(s)

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

Let's have
some fun!

END OF LECTURE 6

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



eduniversal



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa



Information
Management
School

Storing and Retrieving Data

Lecture 7

CAP Theorem and NoSQL databases

Transactions

Collection of actions that make consistent transformations of system states while preserving consistency

Transaction Begins

```
UPDATE savings_accounts  
SET balance = balance - 500  
WHERE account = 3209;
```

Decrement Savings Account

```
UPDATE checking_accounts  
SET balance = balance + 500  
WHERE account = 3208;
```

Increment Checking Account

```
INSERT INTO journal VALUES  
(journal_seq.NEXTVAL, '1B'  
3209, 3208, 500);
```

Record in Transaction Journal

```
COMMIT WORK;
```

End Transaction

Transaction Ends

ACID properties

ACID Compliance

Atomicity

Consistency

Isolation

Durability



ACID properties

ACID Compliance

Atomicity

Consistency

Isolation

Durability

Transactions are often composed of multiple [statements](#). Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely: if any of the statements constituting a transaction fails to complete, the entire transaction fails and the database is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors and crashes.

when you do something to change a database the change should work or fail as a whole

ACID properties

ACID Compliance

Atomicity

Consistency

Isolation

Durability

Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants: any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This prevents database corruption by an illegal transaction, but does not guarantee that a transaction is correct.

Any given database transaction must change affected data only in allowed ways

ACID properties

ACID Compliance

Atomicity

Consistency

Isolation

Durability

Transactions are often executed concurrently (e.g., reading and writing to multiple tables at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially. Isolation is the main goal of concurrency control; depending on the method used, the effects of an incomplete transaction might not even be visible to other transactions.

Isolation defines how/when the changes made by one operation become visible to other

Revisiting ACID properties

ACID Compliance

Atomicity

Consistency

Isolation

Durability

Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash). This usually means that completed transactions (or their effects) are recorded in non-volatile memory.

Durability guarantees that transactions that have committed will survive permanently

RDBMS and ACID properties

Relational database management systems (RDBMSs) are **ACID Compliant**

- RDBMSs put a lot of emphasis on keeping data consistent.
- They require a formal database schema.
- New data or modifications are not accepted unless they comply with this schema in terms of data type, referential integrity, etc.

The NoSQL movement

Cons of RDBMSs

Sometimes this focus on consistency may become a burden:

- May induce overhead and hampers scalability and flexibility.
- RDBMS can not handle ‘Data Variety’ (all types of data under a unified schema of tables).
- Addition of a new functionality would need all the elements to support the new structure. Change is inevitable.

Source 1: <https://www.freecodecamp.org/news/nosql-databases-5f6639ed9574/>

Source 2: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

The NoSQL Movement

As the data volumes or number of parallel transactions increase, capacity can be increased by

- Vertical scaling: extending storage capacity and/or CPU power of the database server¹¹
- Horizontal scaling: multiple DBMS servers being arranged in a cluster

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

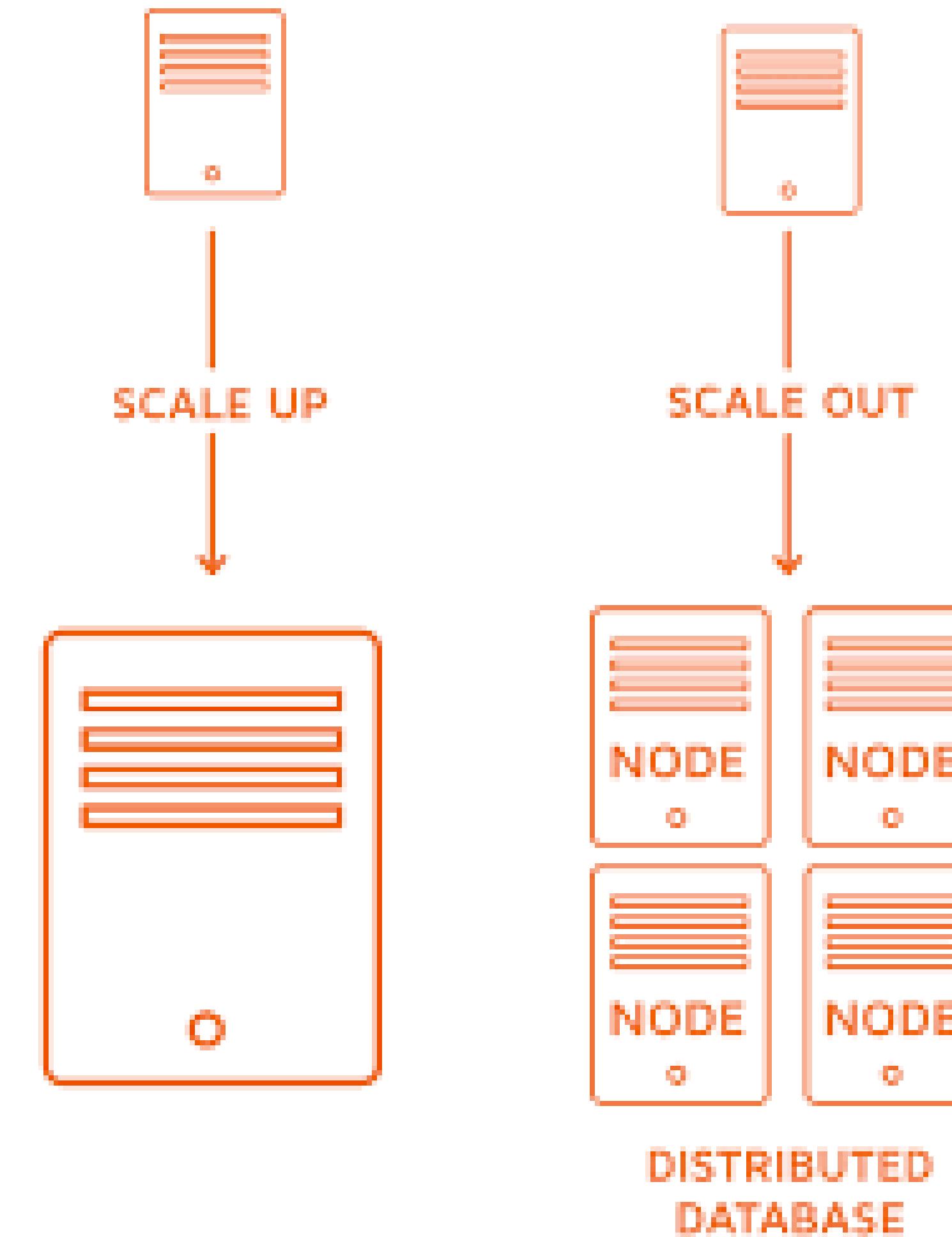
The NoSQL Movement

- RDBMSs are not good at extensive horizontal scaling
 - Coordination overhead because of focus on consistency
 - Rigid database schemas
- Other types of DBMSs needed for situations with massive volumes, flexible data structures¹², and where scalability and availability are more important → NoSQL databases

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

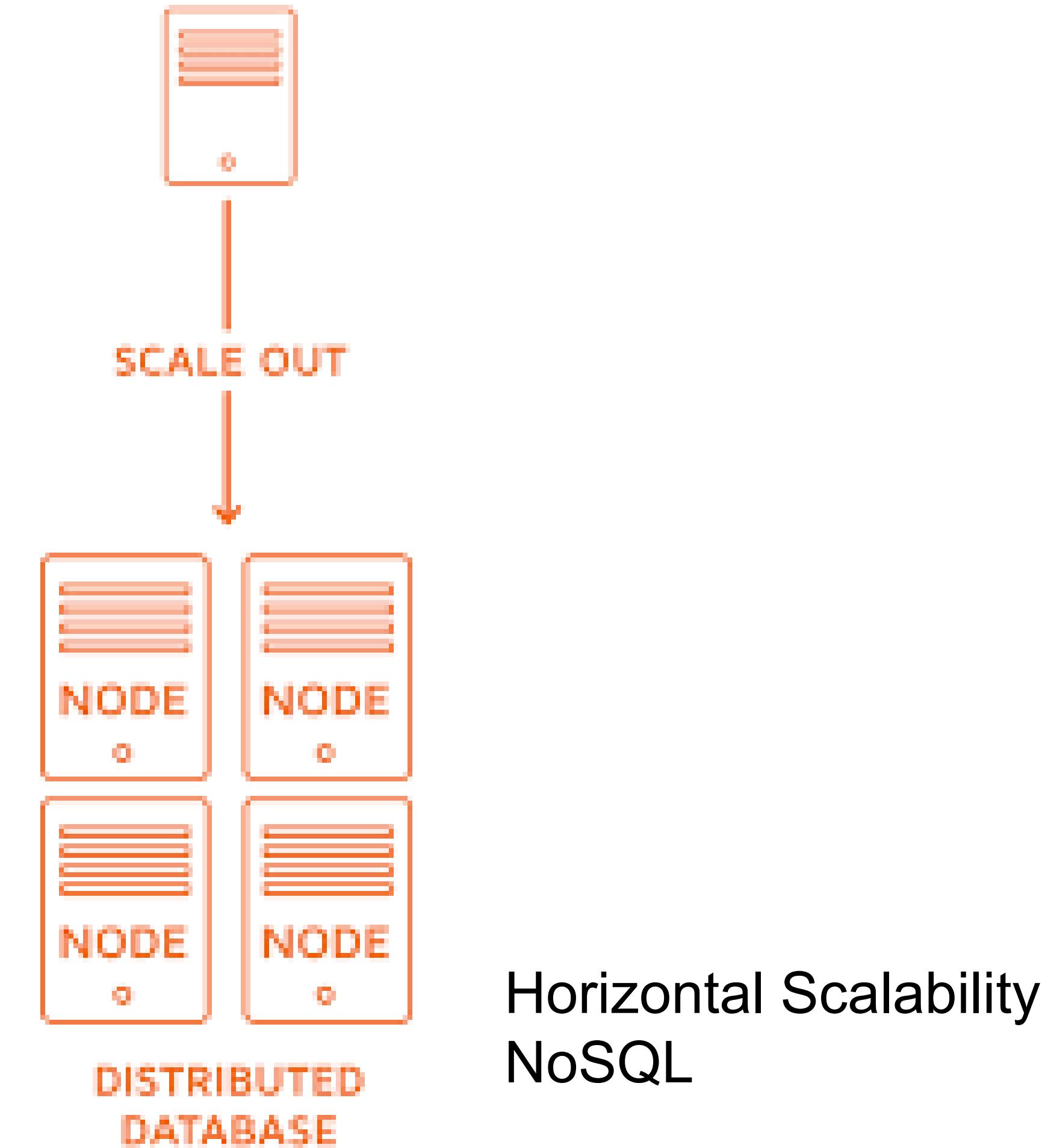
Scaling up VS Scaling out

Vertical Scalability
RDBMS/SQL



Horizontal Scalability
NoSQL

**It needs to be Tolerant
to Partition Failures!**



The NoSQL Movement

- NoSQL databases
 - Store and manipulate data in formats **other than tabular relations**, i.e., non-relational databases
 - NoSQL databases aim at near-linear horizontal scalability by distributing data over a cluster of database nodes for the sake of performance and availability
 - Eventual consistency: the data (and its replicas) will become consistent at some point in time after each transaction

The NoSQL Movement

	Relational Databases	NoSQL Databases
Data paradigm	Relational tables	Key-value (tuple) based Document based Column based Graph based XML, object based Others: time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Vertical scaling, harder to scale horizontally	Easy to scale horizontally, easy data replication
Openness	Closed and open source	Mainly open source
Schema role	Schema-driven	Mainly schema-free or flexible schema
Query language	SQL as query language	No or simple querying facilities, or special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically Available, Soft state, Eventual consistency
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Capable of handling normal-sized datasets	Capable of handling huge amounts of data and/or very high frequencies of read/write requests

NoSQL databases
(rule of thumb)

NOT
ACID Compliant



Not suitable for critical
transactions as in a
Bank system

NoSQL relies on BASE model

BASE Model

Basically available

Soft state

Eventual consistency

Guarantees the availability of the data . There will be a response to any request (can be failure too).

Source: <https://www.freecodecamp.org/news/nosql-databases-5f6639ed9574/>

NoSQL relies on BASE model

BASE Model

Basically available

Soft state

Eventual consistency

The system can change over time, even without receiving input (since nodes continue to update each other)

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

NoSQL relies on BASE model

BASE Model

Basically available

Soft state

Eventual consistency

The system will become consistent over time but might not be consistent at a particular moment..

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

CAP theorem

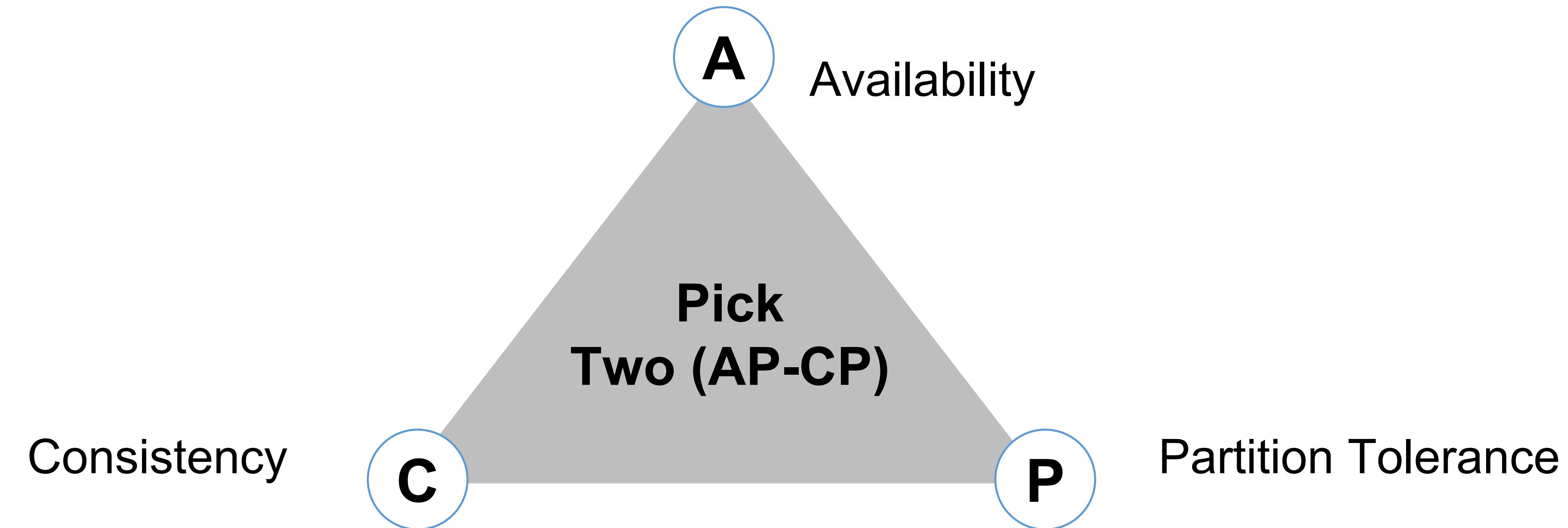
(or Brewer's theorem)

CAP Theorem

“A distributed computer system, such as the one we’ve looked at above, cannot guarantee the following three properties simultaneously: **consistency** (all nodes see the same data simultaneously); **availability** (guarantees that every request receives a response indicating a success or failure result); and **partition tolerance** (the system continues to work even if nodes go down or are added).”

Source: Lemahieu, W., vanden Broucke, S., & Baesens, B. (2018). Principles of Database Management

CAP Theorem (or Brewer's theorem)



The “pick two” should not be interpreted literally, please see:

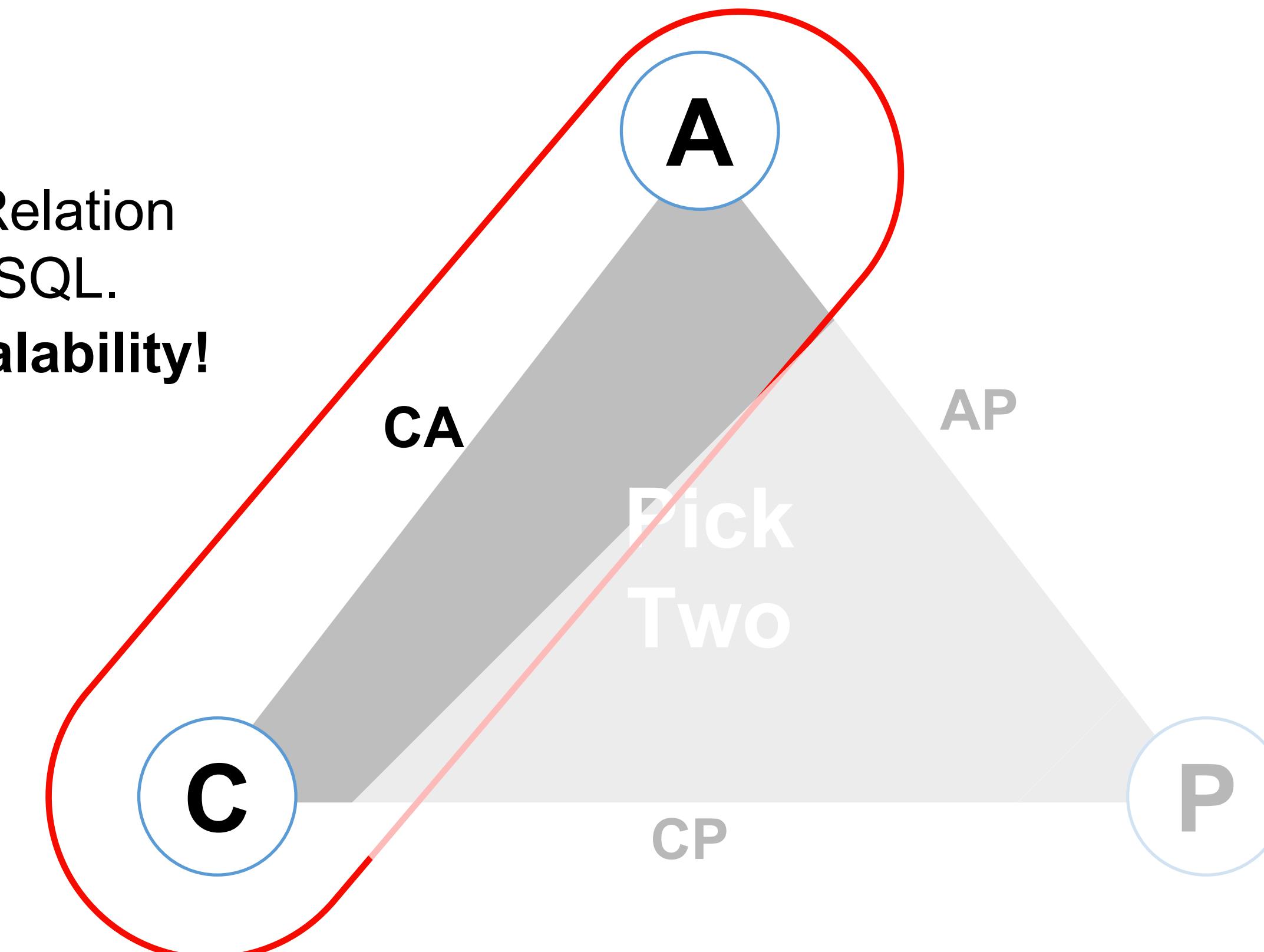
<https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/>

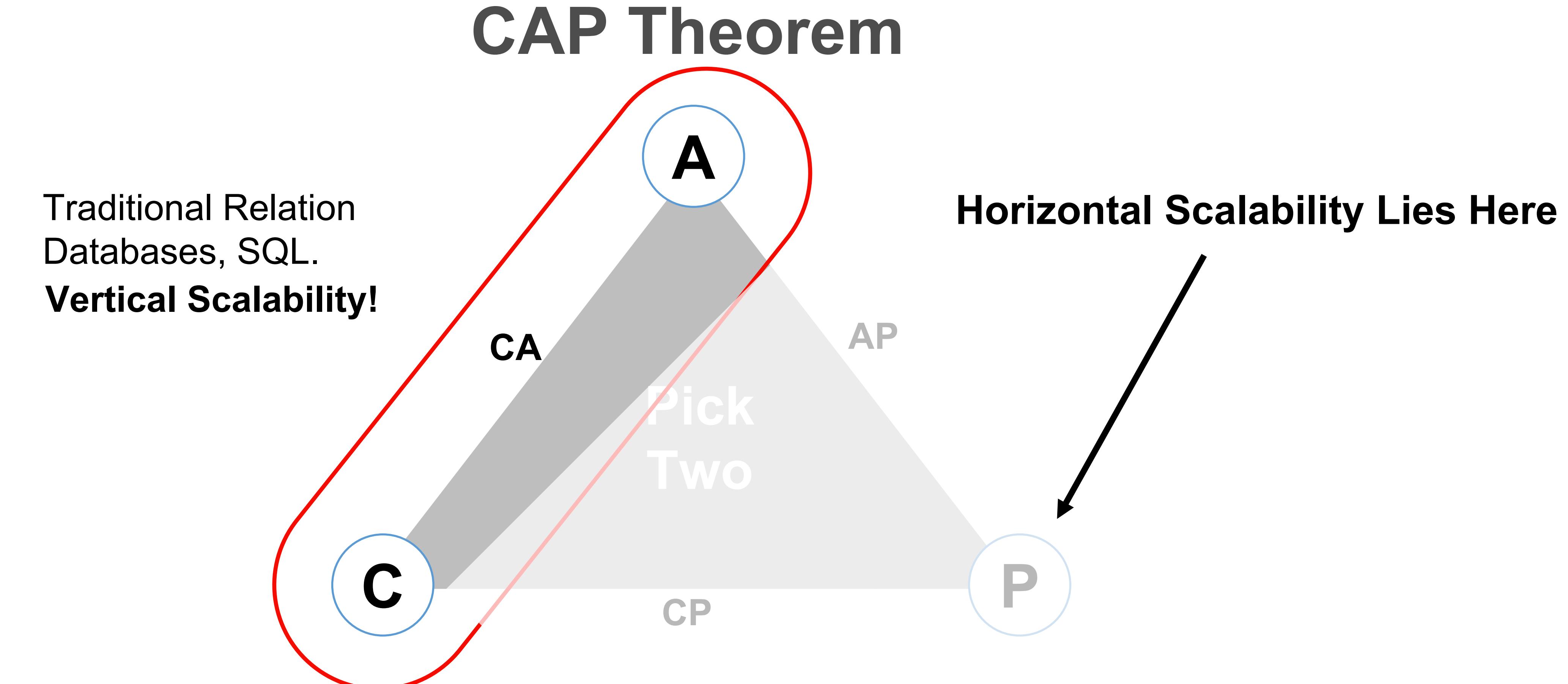


More critical voices: <https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html>

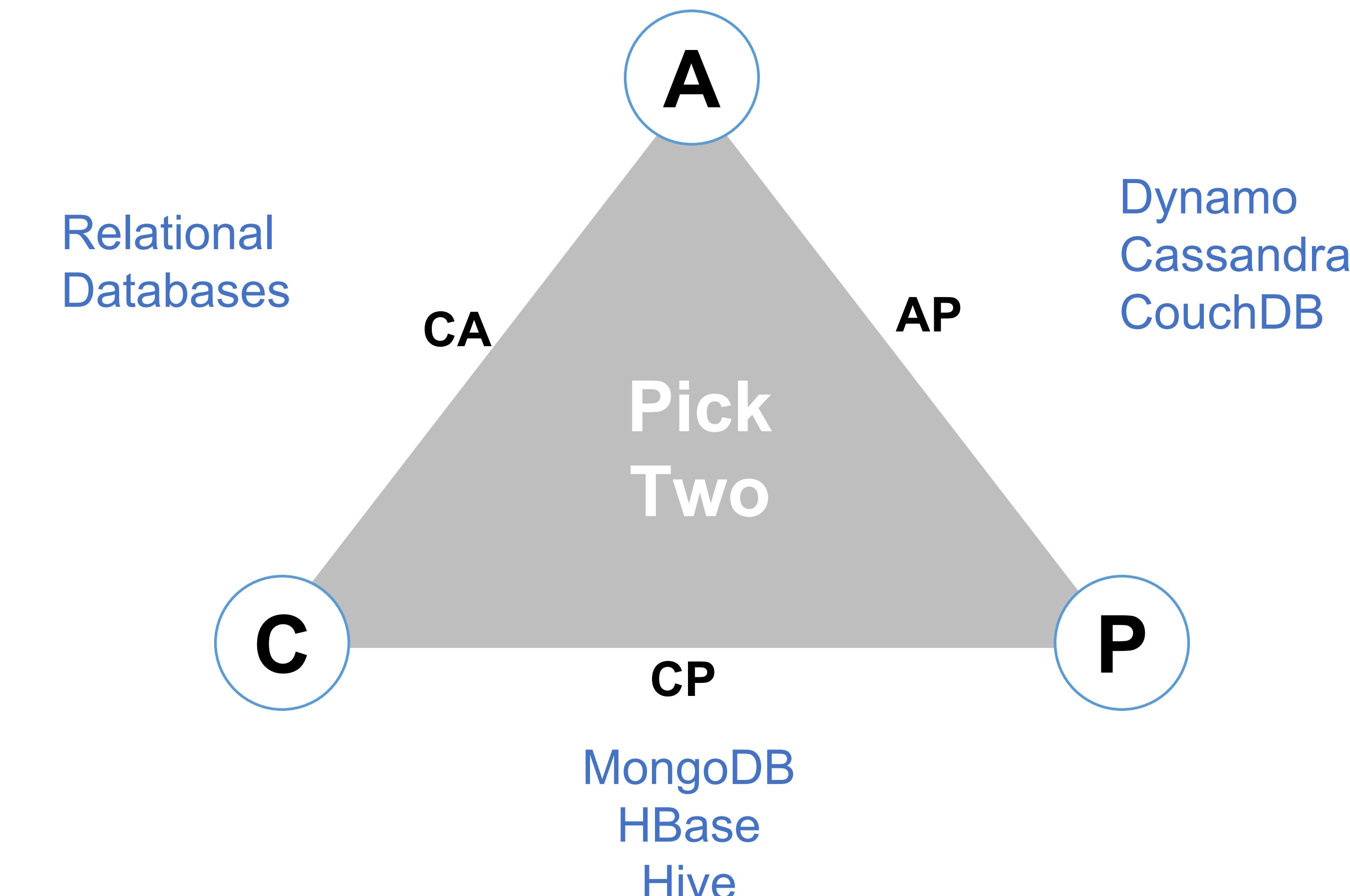
CAP Theorem

Traditional Relation
Databases, SQL.
Vertical Scalability!





What is the best trade off: Availability or Consistency?



NoSQL Types

NoSQL Types

- Key–value stores
- Document stores
- Column-oriented databases
- Graph-based databases
- Other NoSQL categories

Key–Value Stores

- Key-value stores provide much higher performances than RDBMS
- Key–value-based database stores data as (key, value) pairs

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Key–Value Stores - Comparison

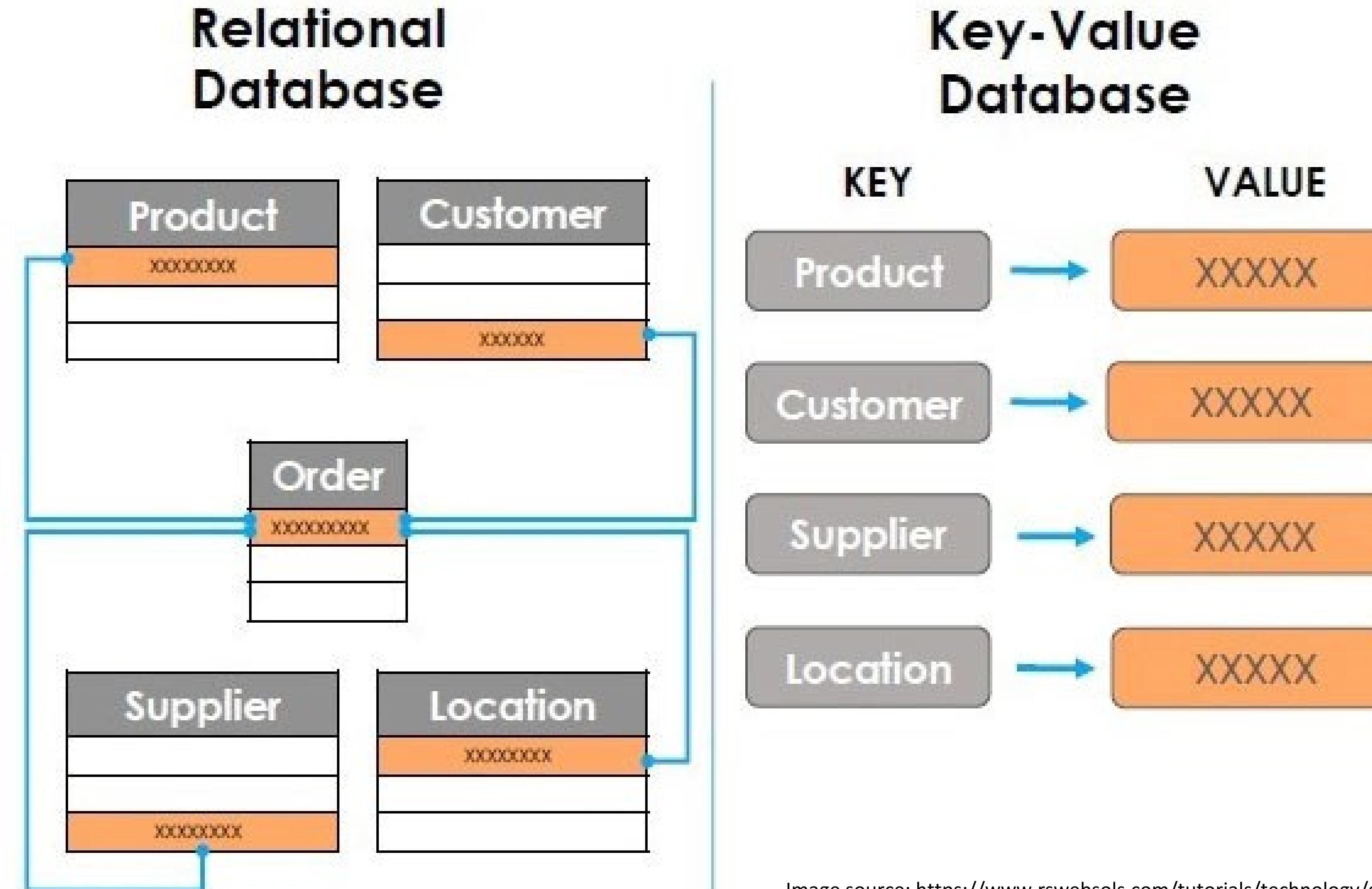


Image source: <https://www.rswebsols.com/tutorials/technology/graph-database-future-technologies>

Key–Value Stores – Application examples

Can you provide some use cases of Key–Value Stores:



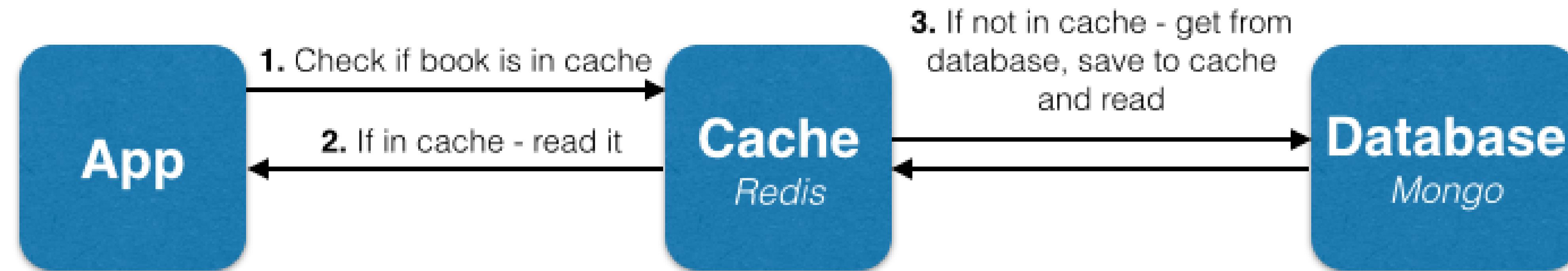
Key–Value Stores – Application examples

Can you provide some use cases of Key–Value Stores:

- Web applications may store user session details and preferences.
- Real-time recommendations and advertising.
- In-memory data caching to speed up applications by minimizing reads and writes that slower disk-based systems.

Key–Value Stores - Example

Key Value can be quite useful to build Cache layers and speed up response times



Document Stores

- **Document stores** store a collection of attributes that are labeled and unordered, representing items that are semi-structured
- Example:

```
{  
    Title = "Harry Potter"  
    ISBN = "111-1111111111"  
    Authors = [ "J.K. Rowling" ]  
    Price = 32  
    Dimensions = "8.5 x 11.0 x 0.5"  
    PageCount = 234  
    Genre = "Fantasy"  
}
```

Document Stores - Example



Document Based NoSQL Solution

Document Stores - Example

Document - JavaScript Object Notation (JSON)

```
{  
    name: "sue",  
    age: 26,  
    status: "A",  
    groups: [ "news", "sports" ]  
}
```



The diagram illustrates a JSON document structure. It starts with an opening brace '{' followed by four key-value pairs. Each pair consists of a field name (e.g., 'name', 'age', 'status', 'groups') in blue text followed by its value in black text. To the right of each pair, a black arrow points from the field name to its corresponding value. The field names are aligned with the arrows.

- name: "sue" → field: value
- age: 26 → field: value
- status: "A" → field: value
- groups: ["news", "sports"] → field: value

Document Stores - Example

Collection

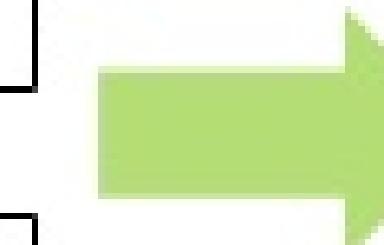
```
{  
    name: "al",  
    age: 18,  
    status: "D",  
    groups: [ "politics", "news" ]  
}
```

Relational vs documental

Relational

Customer ID	First Name	Last Name	City
0	John	Doe	New York
1	Mark	Smith	San Francisco
2	Jay	Black	Newark
3	Meagan	White	London
4	Edward	Daniels	Boston

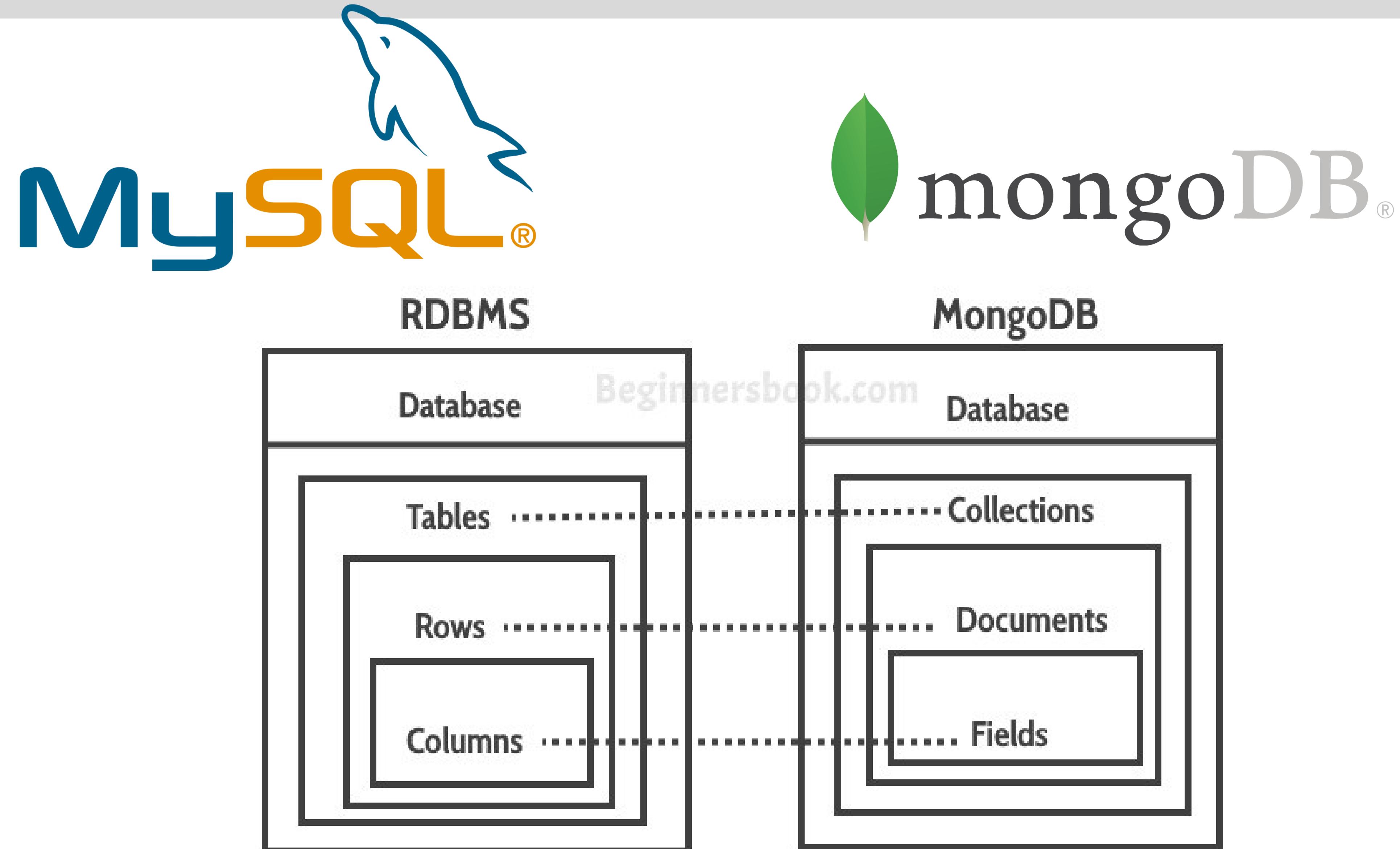
Phone Number	Type	DNC	Customer ID
1-212-555-1212	home	T	0
1-212-555-1213	home	T	0
1-212-555-1214	cell	F	0
1-212-777-1212	home	T	1
1-212-777-1213	cell	(null)	1
1-212-888-1212	home	F	2



MongoDB

```
{   customer_id : 1,  
    first_name : "Mark",  
    last_name : "Smith",  
    city : "San Francisco",  
    phones: [ {  
        number : "1-212-777-1212",  
        dnc : true,  
        type : "home"  
    },  
    {  
        number : "1-212-777-1213",  
        type : "cell"  
    } ]  
}
```

MySQL vs MongoDB



Document Stores - Example

Can you provide some use cases of Document Stores:



Document Stores - Example

Can you provide some use cases of Document Stores:

- Content management systems
- Online profiles in which different users provide different types of information

Column-Oriented Databases

- A **column-oriented DBMS** stores data tables as sections of columns of data
- Useful if:
 - Aggregates are regularly computed over large numbers of similar data items
 - Data are sparse, i.e., columns with many null values

Column-Oriented Databases

- Example

Id	Genre	Title	Price	Audiobook price
1	fantasy	My first book	20	30
2	education	Beginners guide	10	null
3	education	SQL strikes back	40	null
4	fantasy	The rise of SQL	10	null

- RDBMS are not efficient at performing operations that apply to the entire dataset
 - Need indexes which add overhead

Column-Oriented Databases

- In a column-oriented database, all values of a column are placed together on disk

Genre: fantasy:1,4 education:2,3

Title: My first book:1Beginners guide:2 SQL strikes back:3 The rise of SQL:4

Price: 20:1 10:2,4 40:3

Audiobook price: 30:1

- Operations such as find all records with price equal to 10 can now be executed directly
- Null values do not take up storage space anymore

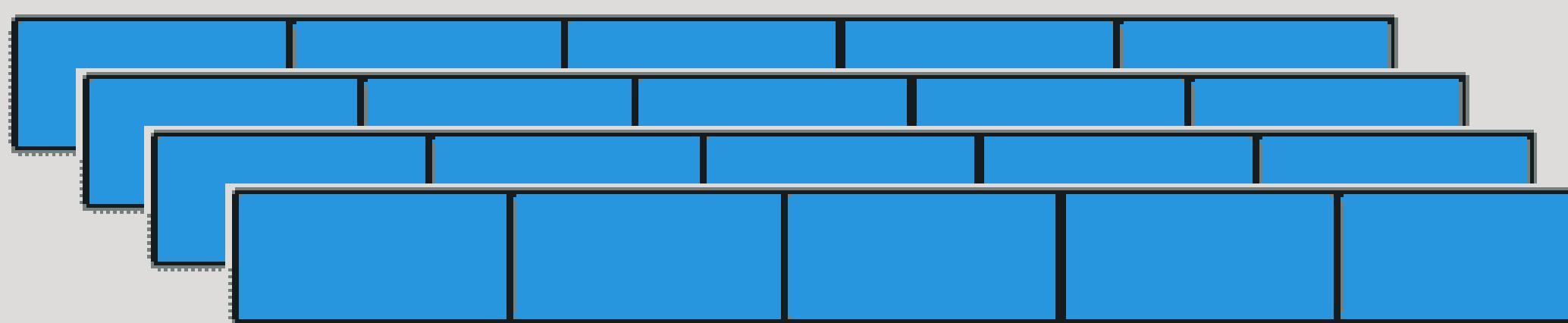
Column-Oriented Databases

- Disadvantages
 - Retrieving all attributes pertaining to a single entity becomes less efficient
 - Join operations will be slowed down
- Examples
 - Google BigTable, Cassandra, HBase, and Parquet

Column-Oriented Databases

row-store

ID	Name	City	Country	Order_no
----	------	------	---------	----------



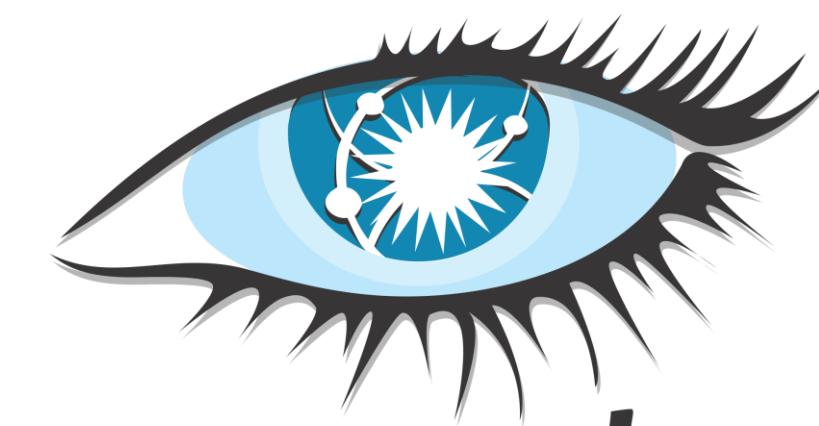
- + easy to add/modify a record
- might read in unnecessary data

column-store

ID	Name	City	Country	Order_no

- + only need to read in relevant data
- tuple writes require multiple accesses

=>suitable for read-mostly, read-intensive, large data repositories



cassandra

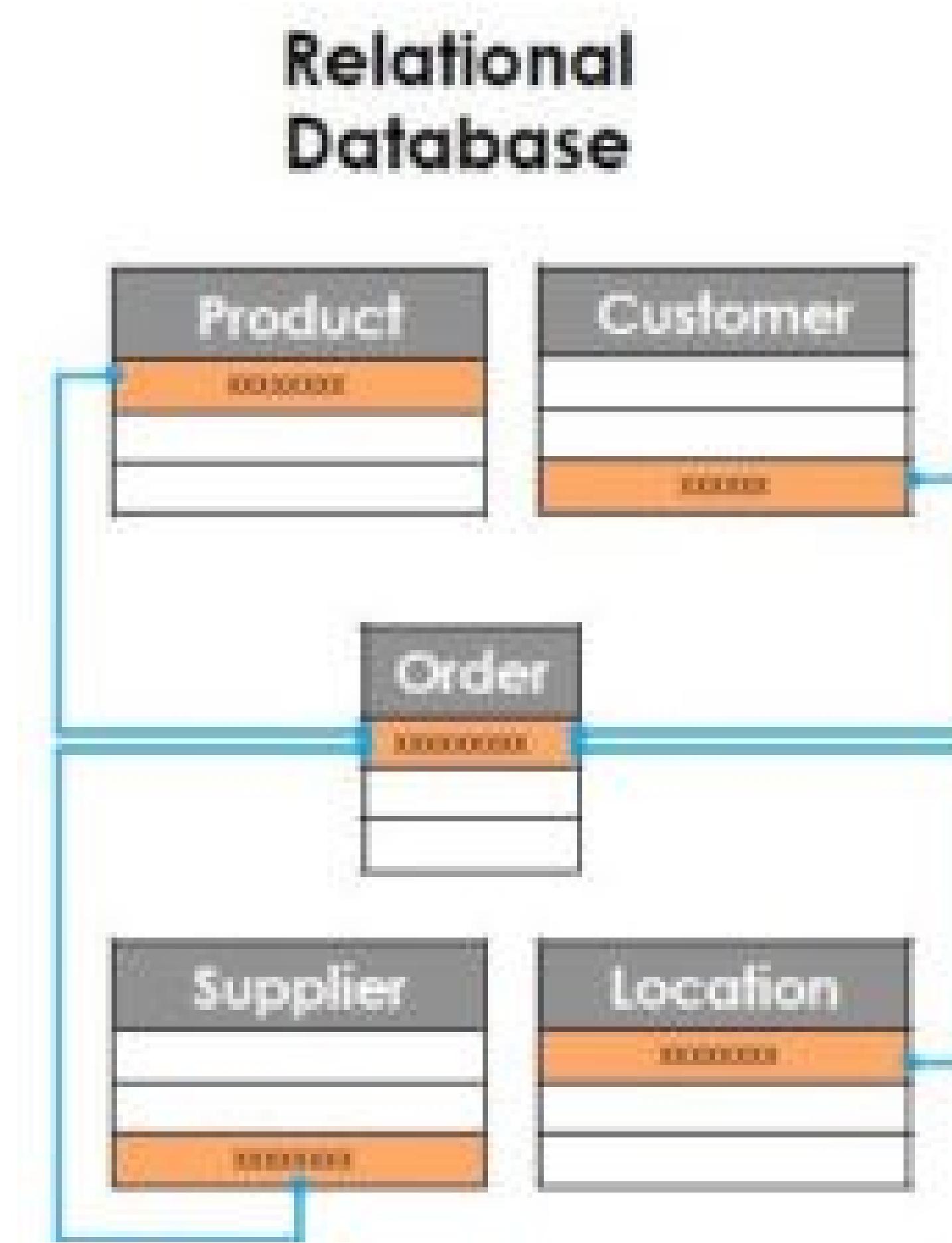
APACHE
HBASE

Graph-Based Databases

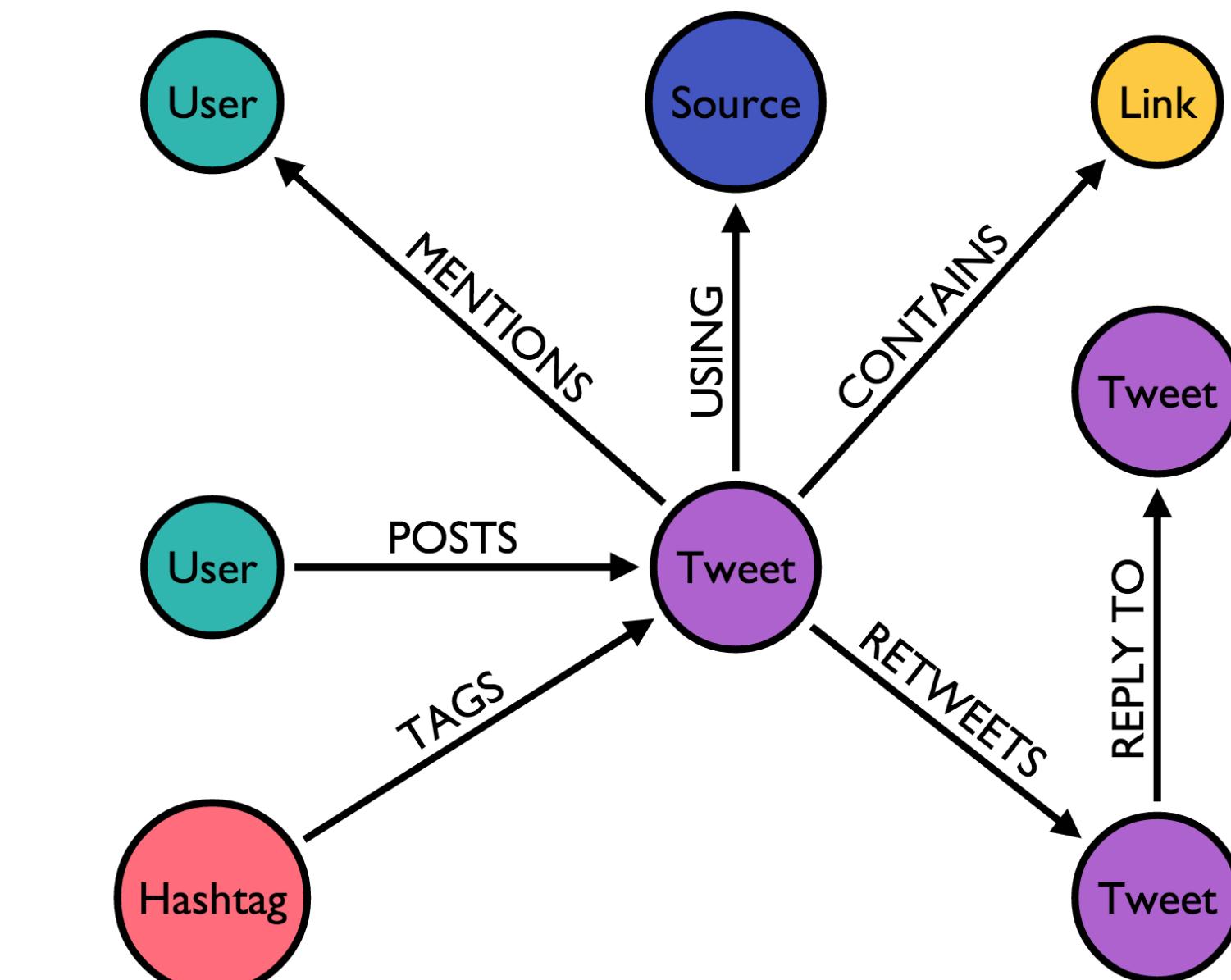
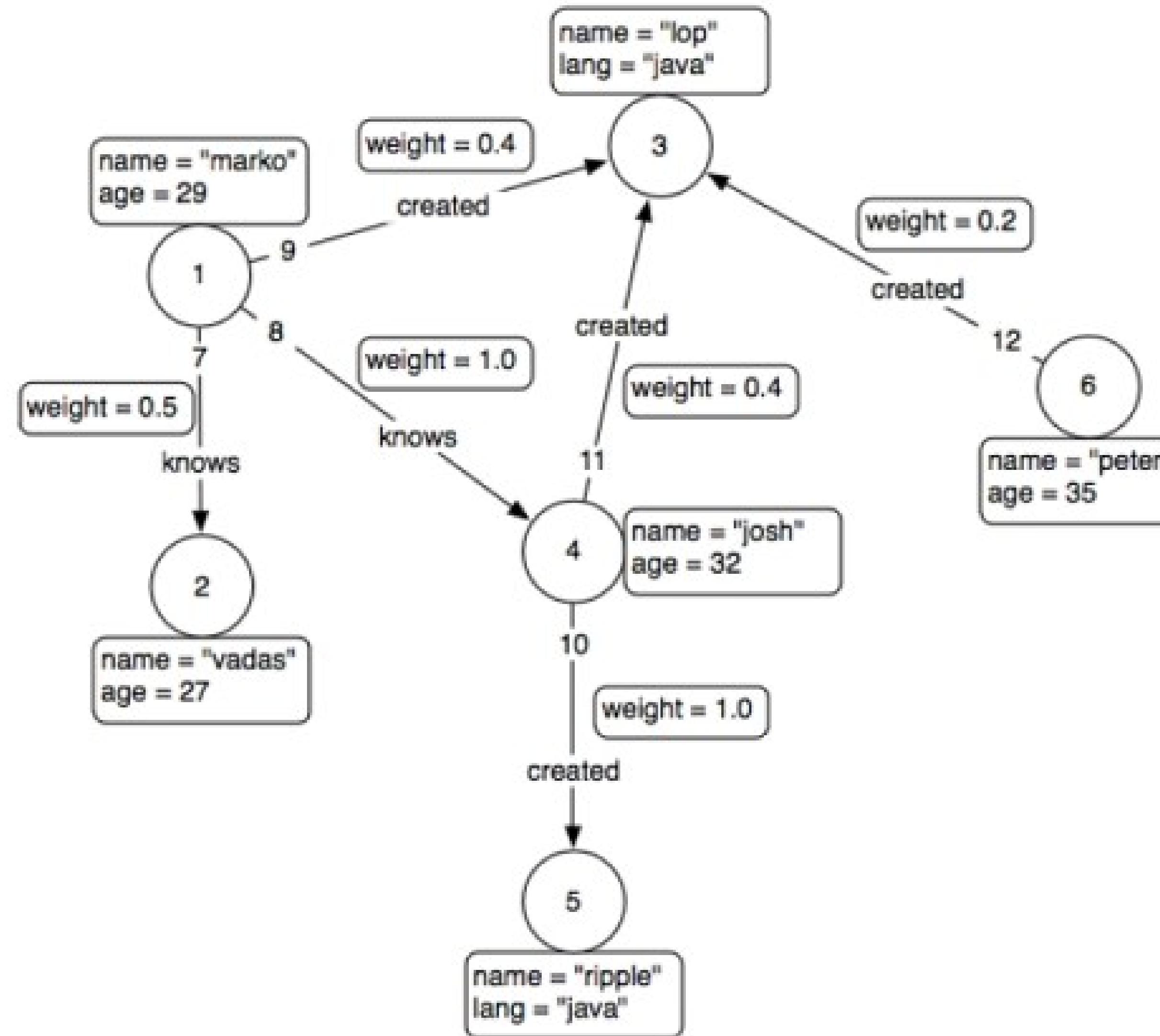
- **Graph databases** apply graph theory to the storage of information of records
- Graphs consist of **nodes** and **edges**
- A graph database is a hyper-relational database
- JOIN tables are replaced by semantically meaningful relationships.
- Relationships that can be navigated and/or queried using graph traversal based on graph pattern matching.

Graph-Based Databases

- One-to-one, one-to-many, and many-to-many structures can easily be modeled in a graph



Graph-Based Databases - Example



neo4j

Graph Databases – Examples

Think about some use cases for graph databases:



Graph Databases – Examples

Think about some use cases for graph databases:

- Location-based services
- Recommender systems
- Social media (e.g., Twitter and FlockDB)
- Knowledge-based systems

Other NoSQL Categories

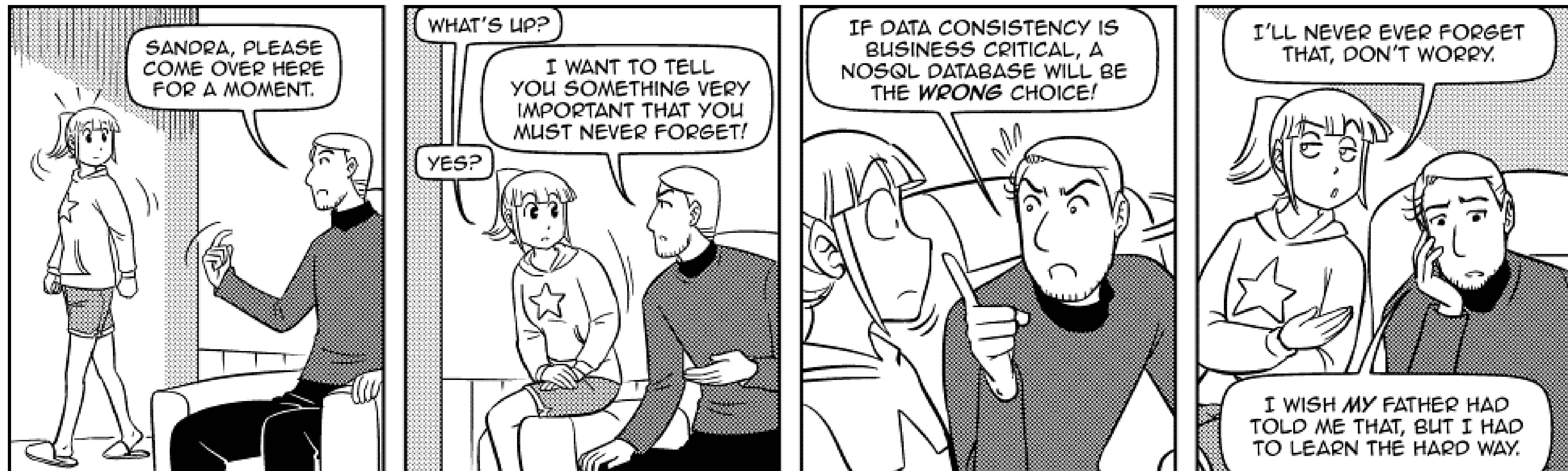
- XML databases
- OO databases
- Database systems to deal with time series and streaming events
- Database systems to store and query geospatial data
- Database systems such as BayesDB which let users query the probable implication of their data

Evaluating NoSQL DBMSs

- Most NoSQL implementations have yet to prove their true worth in the field
- Some queries or aggregations are particularly difficult; map-reduce interfaces are harder to learn and use
- Some early adopters of NoSQL were confronted with some sour lessons
 - e.g., Twitter and HealthCare.gov

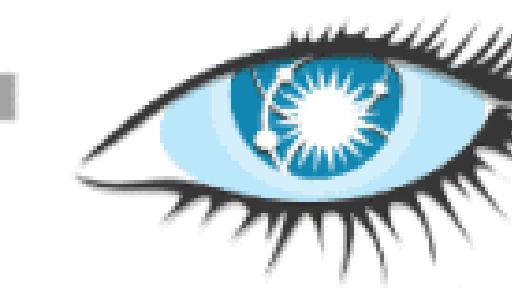
Evaluating NoSQL DBMSs

- NoSQL vendors start focusing again on robustness and durability, whereas RDBMS vendors start implementing features to build schema-free, scalable data stores
- NewSQL: blend the scalable performance and flexibility of NoSQL systems with the robustness guarantees of a traditional RDBMS



Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

NOSQL is a big family, you should always look for the one that best fits your needs!



KeyValue Databases e.g Riak, Redis, Berkeley DB, CouchBase etc.

Document Databases e.g MongoDB, CouchDB, OrientDB etc.

Column Family Databases e.g Cassandra, HBase, Amazon DynamoDB etc.

Graph Databases e.g Neo4J, OrientDB etc.

Quick quiz

<https://b.socrative.com/login/student/>

Room: SRD2021



Quiz Time

Let's have
some fun!

THE END

Acreditações e Certificações



UNIGIS



A3ES



Double Degree
Master Course in
Information Systems
Management



eduniversal



Computing
Accreditation
Commission

Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa