SEC1

Company Confidential

Access for Capgemini users and authorized third parties

# Relational Databases

Maciej Lipski

2021-04-06

# Agenda

SEC1

- Before we Begin
- Relational Databases (DB)
- Structured Query Language (SQL)
- Data locking strategies
- Advanced Databases Topics
- Modeling databases
- Assignments

# Agenda

SEC1

- **Before we Begin**
- Relational Databases (DB)
- Structured Query Language (SQL)
- Data locking strategies
- Advanced Databases Topics
- Modeling databases
- Assignments

# Agenda


- Before we Begin
- **Relational Databases (DB)**
- Structured Query Language (SQL)
- Data locking strategies
- Advanced Databases Topics
- Modeling databases
- Assignments

# What is this database thing?

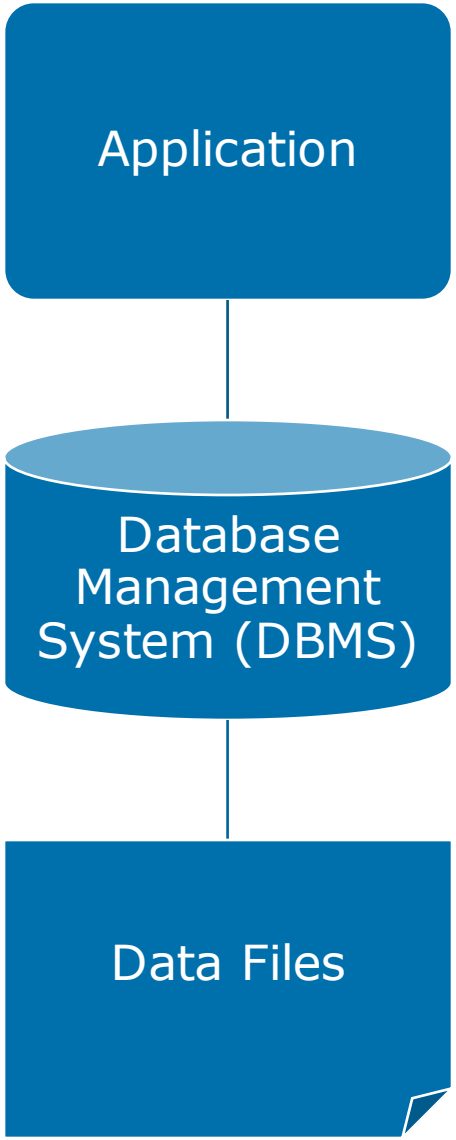
- **Database (DB)** – organized collection of data
- **Database Management System (DBMS)** – computer software, allows the definition, creation, querying, update, and administration of databases.
- In our daily work we usually mix those two terms.

# Why do we need databases?

SEC1

## Is it really needed? Do we care?

- Abstraction layer between data and application
- It's about speed, scalability, integrity, security, transaction support, availability, consistency… You don't want to implement it yourself.
- NOTE: Our development environments might give an impression, that the database is irrelevant (in-memory database, initialized on applications start and destroyed when application stops). It is not



# Who is using DB?

SEC1

- **Everyone!**

- End-users
- Developers of the applications
- Testers
- Designers of the database
- System analytics
- Business Analytics
- Database Administrators
- …

# Different database types




**Relational**

- **Oracle**
- **MySQL**
- **MS SQL Server**
- PostgreSQL
- IBM Db2
- MariaDB

**Object oriented**

- InterSystem Caché
- Db4o
- Matisse
- Perst
- ZopeDB

**Column**

- Cassandra
- HBase
- Accumulo
- Scylla
- HBase

**Document**

- MongoDB
- Amazon DynamoDB
- Microsoft Azure Cosmos DB
- CouchDB
- DocumentDB
- MemcacheDB

**Key-Value**

- Redis
- Amazon DynamoDB
- Oracle NoSQL
- Apache Ignite

**Graph**

- Neo4j
- Microsoft Azure Cosmos DB
- OrientDB
- ArangoDB
- AllegroGraph

Source:https://www.alooma.com/blog/types-of-modern-databases ; https://db-engines.com/en/ranking

# Rankings of popularity

URL: https://db-engines.com/en/ranking

361 systems in ranking, January 2021

| Rank | | | DBMS | Database Model | Score | | |
|---|---|---|---|---|---|---|---|
| Jan 2021 | Dec 2020 | Jan 2020 | | | Jan 2021 | Dec 2020 | Jan 2020 |
| 1. | 1. | 1. | Oracle | Relational, Multi-model | 1322.93 | -2.66 | -23.75 |
| 2. | 2. | 2. | MySQL | Relational, Multi-model | 1252.06 | -3.40 | -22.60 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model | 1031.23 | -6.85 | -67.31 |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model | 552.23 | +4.65 | +45.03 |
| 5. | 5. | 5. | MongoDB | Document, Multi-model | 457.22 | -0.51 | +30.26 |
| 6. | 6. | 6. | IBM Db2 | Relational, Multi-model | 157.17 | -3.26 | -11.53 |
| 7. | 7. | ↑ 8. | Redis | Key-value, Multi-model | 155.01 | +1.38 | +6.26 |
| 8. | 8. | ↓ 7. | Elasticsearch | Search engine, Multi-model | 151.25 | -1.24 | -0.19 |
| 9. | 9. | ↑ 10. | SQLite | Relational | 121.89 | +0.21 | -0.25 |
| 10. | 10. | ↑ 11. | Cassandra | Wide column | 118.08 | -0.76 | -2.59 |
| 11. | 11. | ↓ 9. | Microsoft Access | Relational | 115.33 | -1.41 | -13.24 |
| 12. | 12. | ↑ 13. | MariaDB | Relational, Multi-model | 93.79 | +0.18 | +6.34 |
| 13. | 13. | ↓ 12. | Splunk | Search engine | 87.66 | +0.66 | -1.01 |
| 14. | 14. | ↑ 15. | Teradata | Relational, Multi-model | 72.59 | -1.24 | -5.70 |
| 15. | ↑ 16. | ↑ 25. | Microsoft Azure SQL Database | Relational, Multi-model | 71.36 | +1.87 | +43.16 |
| 16. | ↓ 15. | ↓ 14. | Hive | Relational | 70.43 | +0.16 | -13.81 |
| 17. | 17. | ↓ 16. | Amazon DynamoDB | Multi-model | 69.14 | +0.01 | +7.11 |

# Cloud Computing, Cloud Databases


Database as a Service

SaaS IaaS PaaS

Source:https://www.supportpro.com/blog/dbms-and-dbaas/

# Relational databases

SEC1

- Based on the mathematical model of relation (you know, a subset of a cartesian product).
- A database consists of multiple tables.

Column

Each column has name and a defined type (like string, number, enum, date…)

Table name

employee

| id | first_name | last_name | hired | email | created_by | created_at |
|---|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 2017-01-01 | mkowalski@capg.com | hr1 | 2019-02-17 |
| 2 | Armando | Roggio | 2014-03-13 | armando@some.com | hr1 | 2019-01-01 |
| 3 | Barack | Obama | 2017-01-20 | barack@usa.com | admin | 2019-01-02 |
| 4 | Bono | Vox | 2014-07-13 | bono@u2.com | admin | 2019-01-01 |
| 5 | Bill | Gates | 2009-02-20 | bill@microsoft.com | admin | 2019-01-01 |
| 6 | Steve | Jobs | 1988-02-08 | steve@apple.com | null | null |
| 7 | Larry | Ellison | 2000-08-09 | larry@oracle.com | admin | 2019-01-01 |
| 8 | Maciej | Nowak | 2017-02-01 | mnowak@capg.com | hr2 | 2017-02-01 |

Row, tuple

# Primary Key


- Attribute or set of attributes, which identifies row
- Can be natural or technical (some ‚id' field)

| first_name | last_name | hired | email | created_by | created_at |
|---|---|---|---|---|---|
| Maciej | Kowalski | 2017-01-01 | mkowalski@capg.com | hr1 | 2019-02-17 |
| Armando | Roggio | 2014-03-13 | armando@some.com | hr1 | 2019-01-01 |
| Barack | Obama | 2017-01-20 | barack@usa.com | admin | 2019-01-02 |
| Bono | Vox | 2014-07-13 | bono@u2.com | admin | 2019-01-01 |
| Bill | Gates | 2009-02-20 | bill@microsoft.com | admin | 2019-01-01 |
| Steve | Jobs | 1988-02-08 | steve@apple.com | null | null |
| Larry | Ellison | 2000-08-09 | larry@oracle.com | admin | 2019-01-01 |
| Maciej | Nowak | 2017-02-01 | mnowak@capg.com | hr2 | 2017-02-01 |

# Foreign Key


- Attribute or set of attributes, which points to row in other (or same) table

Employee

| id | first_name | last_name | position_id | manager_id |
|---|---|---|---|---|
| 1 | Maciej | Kowalski | 1 | null |
| 2 | Armando | Roggio | 2 | 1 |
| 3 | Barack | Obama | 2 | 1 |
| 4 | Bono | Vox | 3 | 2 |
| 5 | Bill | Gates | 3 | 2 |
| 6 | Steve | Jobs | 3 | 2 |
| 7 | Larry | Ellison | 3 | 3 |
| 8 | Maciej | Nowak | 3 | 3 |

Position

| id | position |
|---|---|
| 1 | CEO |
| 2 | Head of department |
| 3 | Developer |

Mails of employees

| employee_id | email |
|---|---|
| 1 | kowalski@capg.com |
| 2 | roggio@some.com |
| 3 | barack@usa.com |
| 1 | m.kowalski@test.pl |

# Normalizations and relations


- Re-structure tables and relations in order to satisfy so called normal forms
- Goal: reduce data duplications, improve integrity etc

| id | name | email |
|---|---|---|
| 1 | Maciej Kowalski | mkowalski@capg.com |
| 2 | Armando Roggio | armando@some.com |
| 3 | Barack Obama | barack@usa.com |
| 4 | Bono Vox | bono@u2.com |

| id | first_name | last_name | email |
|---|---|---|---|
| 1 | Maciej | Kowalski | mkowalski@capg.com |
| 2 | Armando | Roggio | armando@some.com |
| 3 | Barack | Obama | barack@usa.com |
| 4 | Bono | Vox | bono@u2.com |

| id | first_name | last_name | address |
|---|---|---|---|
| 1 | Maciej | Kowalski | Legnicka 48H, 54-202 Wrocław |
| 2 | Armando | Roggio | Tęczowa 301, 53-601 Wrocław |

| id | first_name | last_name | zip_code | city | street | apartment |
|---|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 54-202 | Wrocław | Legnicka | 48H |
| 2 | Armando | Roggio | 53-601 | Wrocław | Tęczowa | 301 |

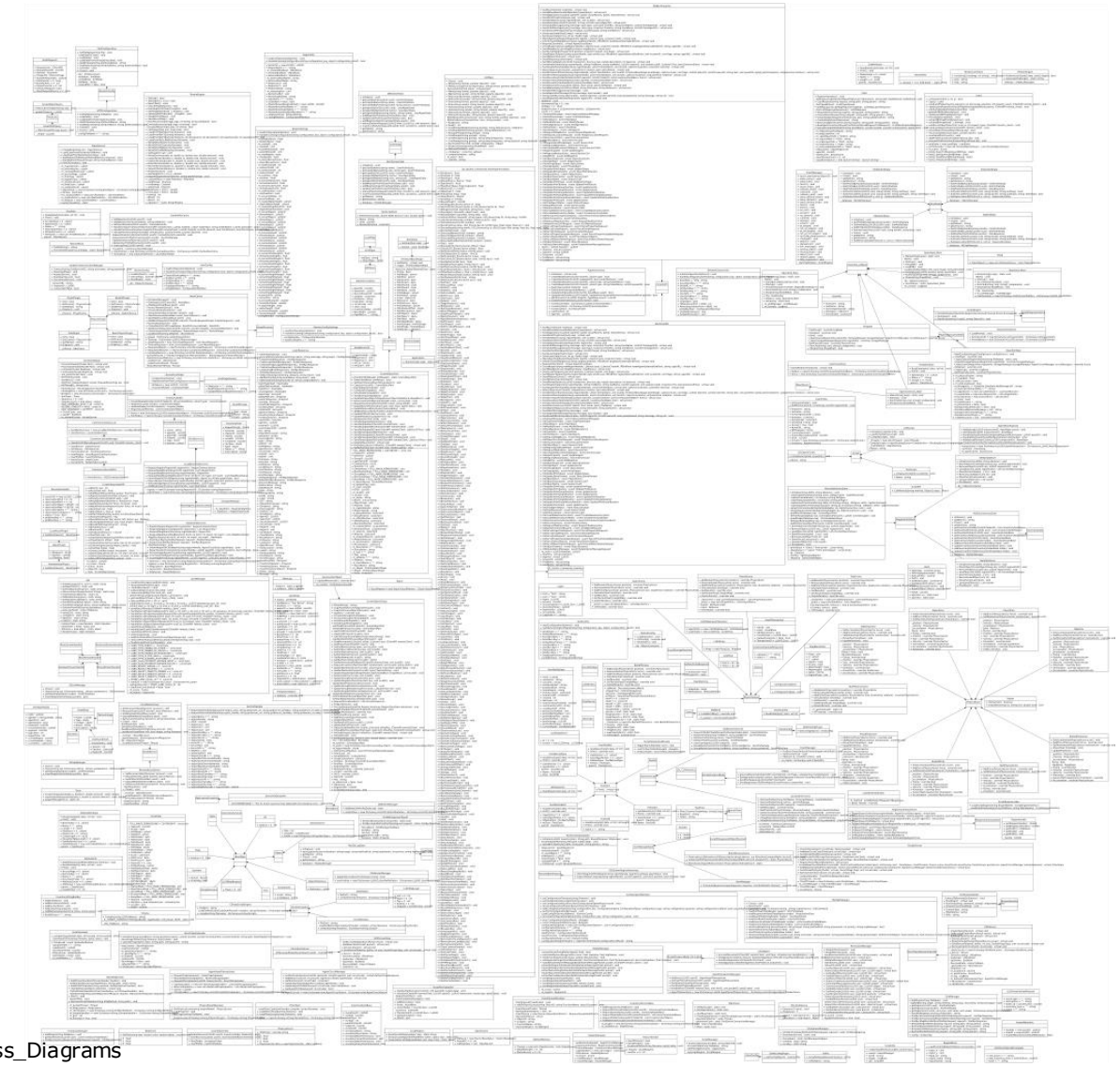
| id | last_name | email1 | email2 |
|---|---|---|---|
| 1 | Kowalski | kowal@capg.com | mkowalski@test.pl |
| 2 | Roggio | roggio@some.com | |
| 3 | Obama | barack@usa.com | |
| 4 | Vox | | |

| id | last_name |
|---|---|
| 1 | Kowalski |
| 2 | Roggio |
| 3 | Obama |
| 4 | Vox |

| emp_id | email |
|---|---|
| 1 | kowal@capg.com |
| 2 | roggio@some.com |
| 3 | barack@usa.com |
| 1 | mkowalski@test.pl |

https://en.wikipedia.org/wiki/Database_normalization

# Relations… everywhere… relations



Source:http://opensimulator.org/wiki/OpenSim:_Class_Diagrams

# Agenda

SEC1

- Before we Begin
- Relational Databases (DB)
- **Structured Query Language (SQL)**
- Data locking strategies
- Advanced Databases Topics
- Modeling databases
- Assignments

# SQL


Source:https://learnsql.com/blog/sql-or-sequel/

# Structured Query Langauge (SQL)

SEC1

## Standardized

Current version: SQL:2016

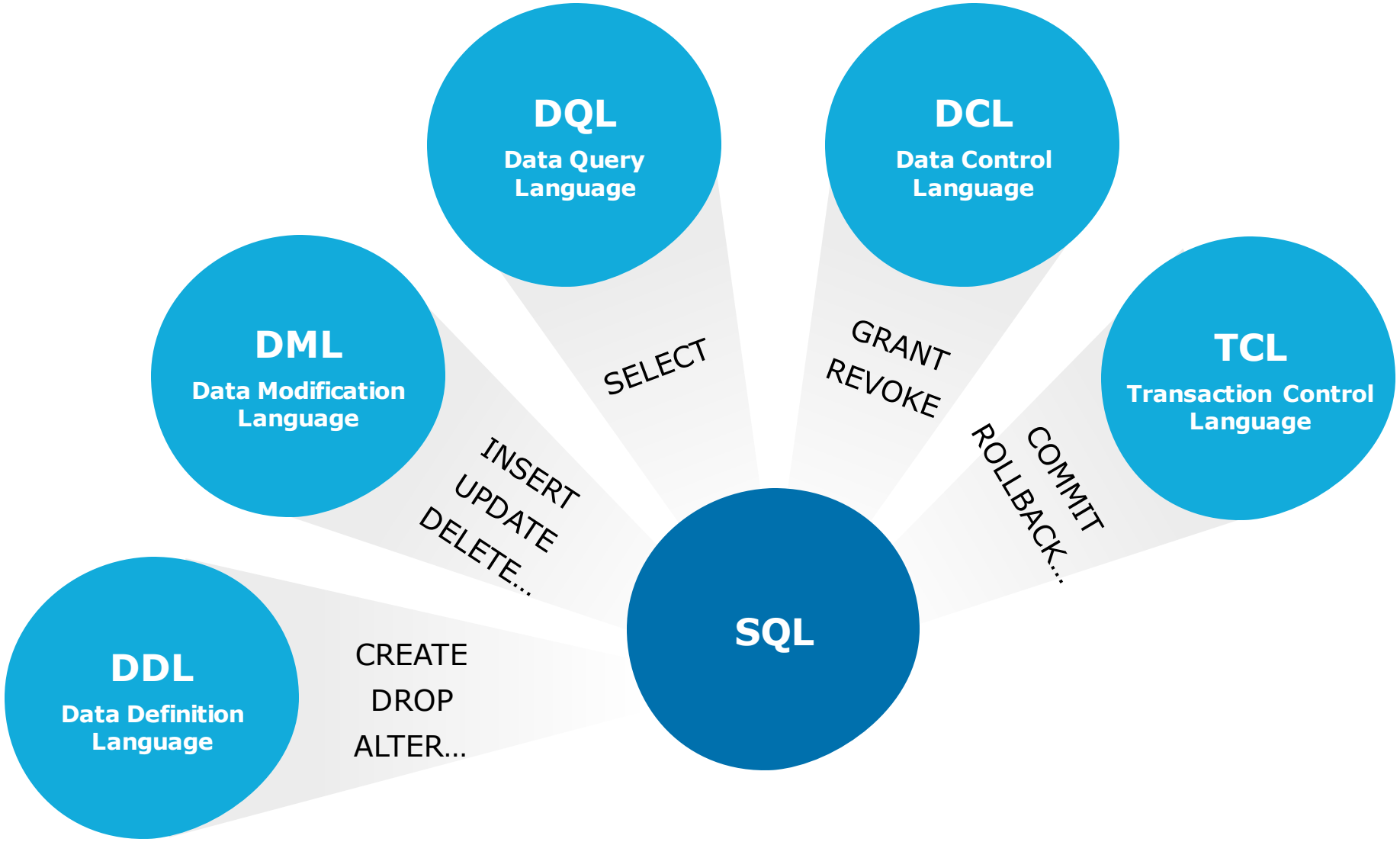
## Every vendor adds his own flavors

## Declarative language

Define how the output should look like.

in contrast with imperative programming, where you define an algorithm.

# SQL commands are categorized into categories



# Our trip…

SEC1

**Other**

subqueries, views,
functions, procedures

**DQL**

SELECT with JOIN

**DQL**

SELECT with GROUP BY

**DQL**

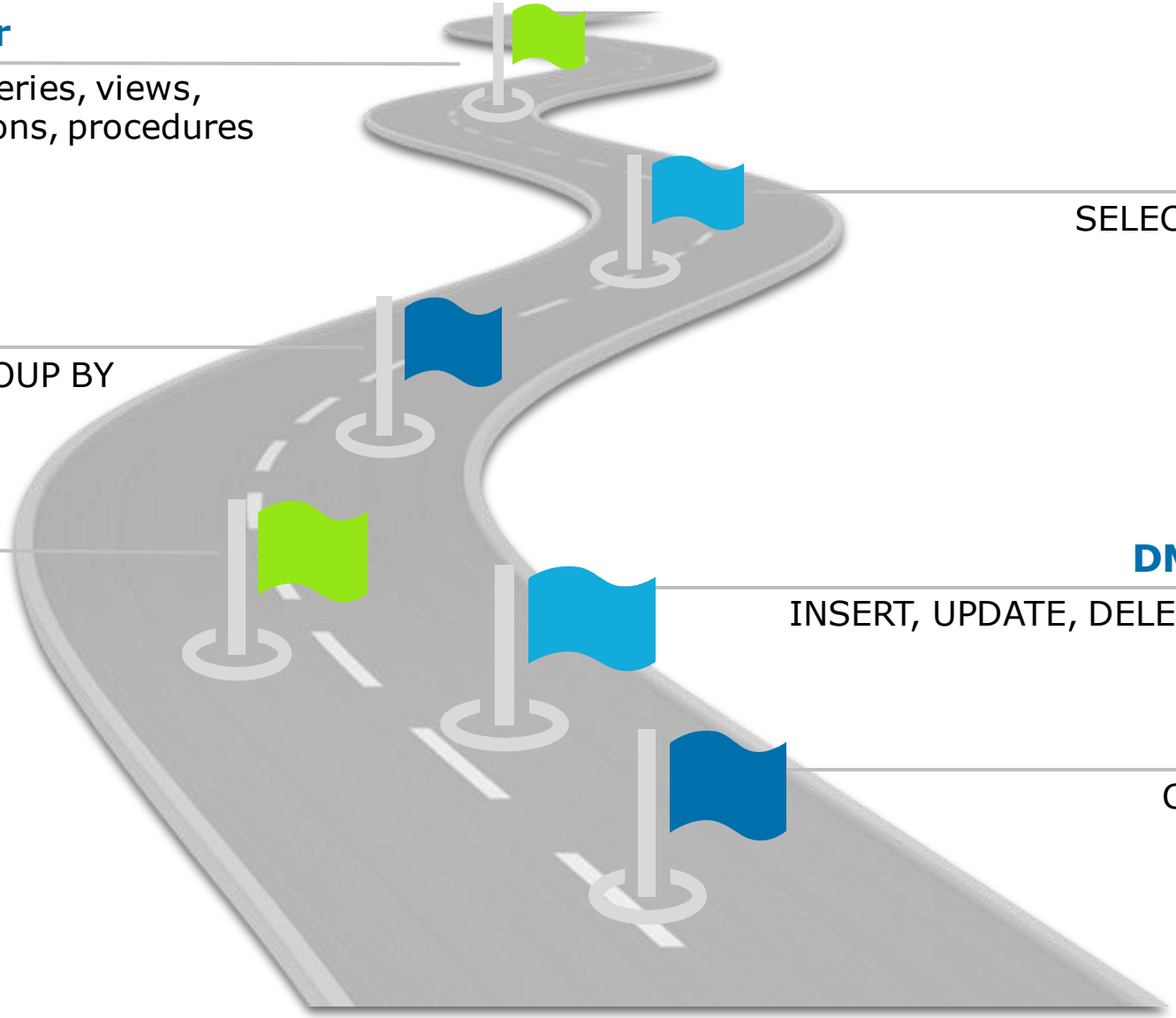
Simple SELECT

**DML**

INSERT, UPDATE, DELETE

**DDL**

CREATE TABLE



# DDL: CREATE TABLE

SEC1

Table name

```
CREATE TABLE employee (
  id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(45) NOT NULL,
  last_name VARCHAR(45) NOT NULL,
  hired DATE NOT NULL,
  mail VARCHAR(50) NOT NULL,
  created_by VARCHAR(50) NULL,
  created_at DATETIME NULL
    DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  UNIQUE INDEX uq_employee_mail (mail ASC)
);
```

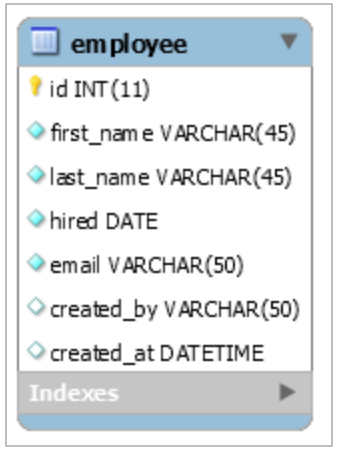
Table column definition:
<name> <type> <options>

Optional constraints

https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html

## Common types:

- CHAR
- VARCHAR
- INT
- ENUM
- DATE, DATETIME
- TIMESTAMP…




## Common options:

- NULL / NOT NULL
- DEFAULT <value>
- AUTO_INCREMENT
- ON UPDATE…

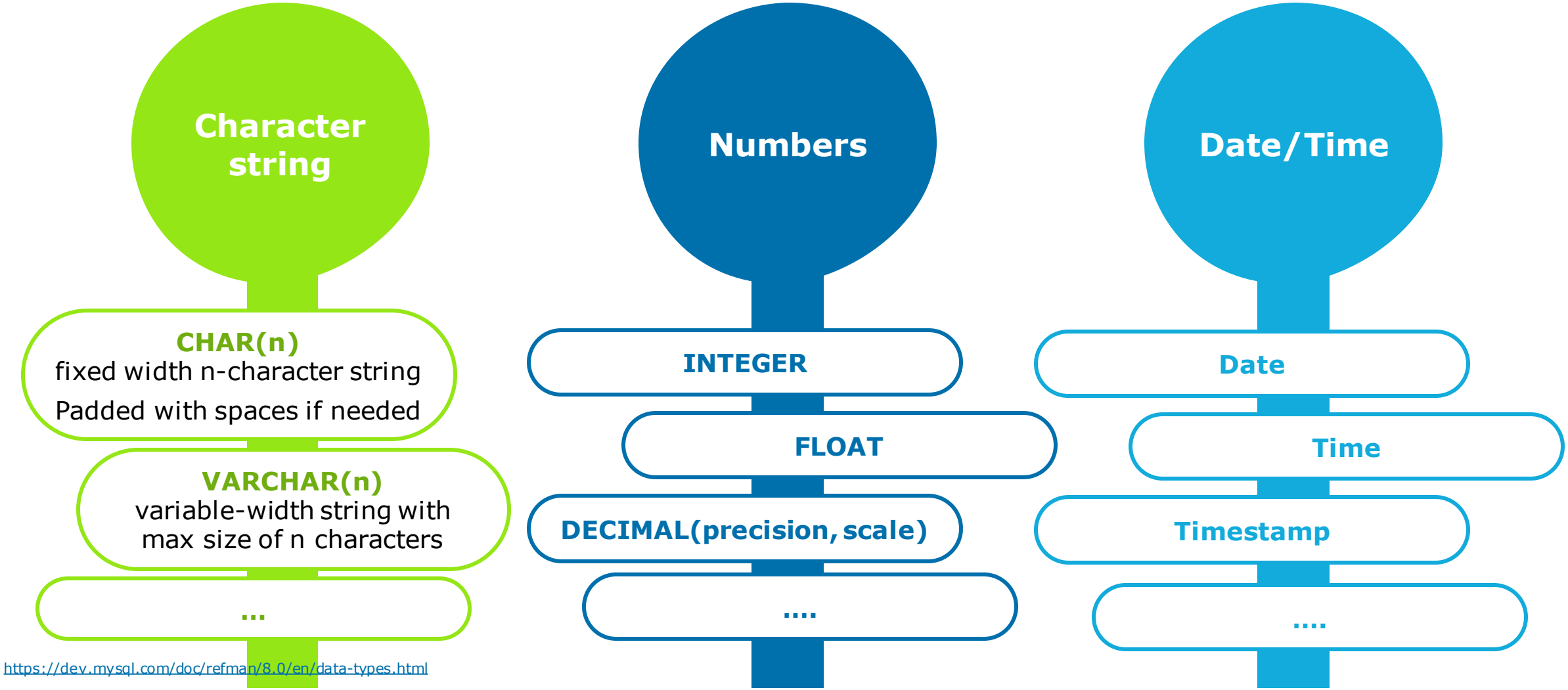
## Significance of Primary Key (PK):

- PK identifies one table row
- PK can span over multiple columns
- Usually, all tables have a PK
- DB usually creates an index for a PK

# Data types of columns

SEC1

Every database can have its own data types




https://dev.mysql.com/doc/refman/8.0/en/data-types.html

# DML: INSERT INTO


Schema name

Table name

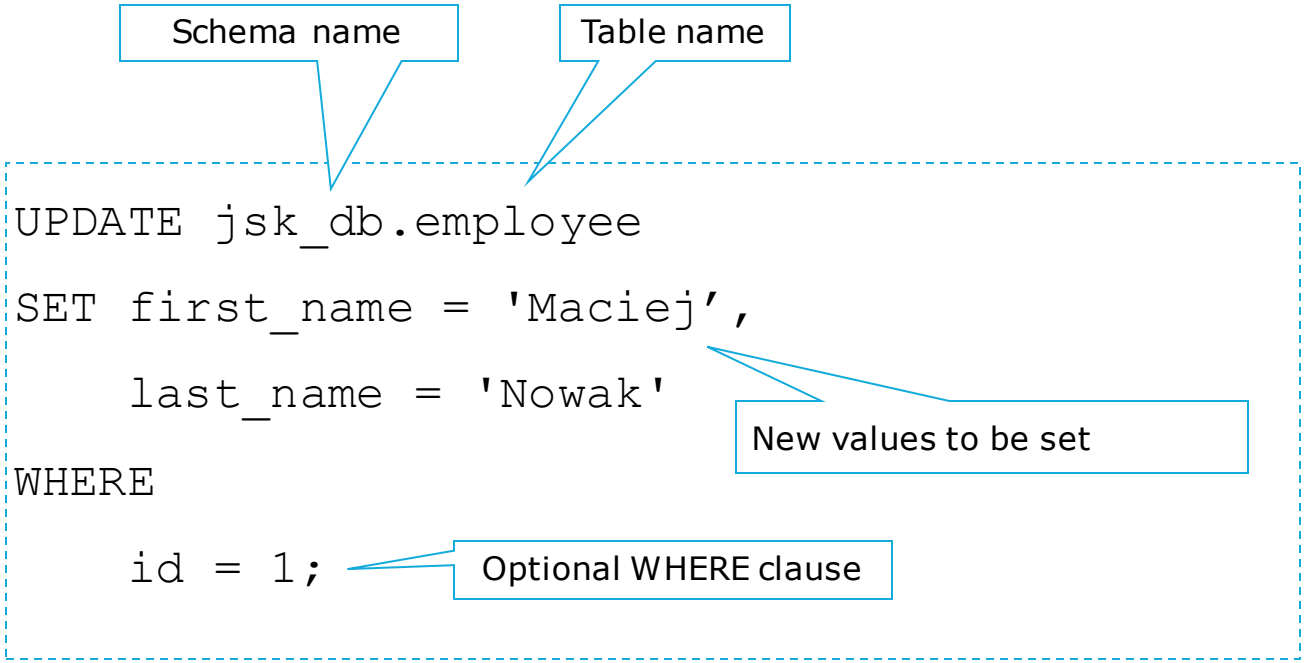
Optional, comma separated list of columns.
All NOT NULL columns without DEFAULT or AUTO_INCREMENT value must be present.

```
INSERT INTO jsk_db.employee(first_name, last_name, hired, email)
VALUES
        ('Armando', 'Roggio', '2014-03-13', 'armando@some.com'),
        ('Barack', 'Obama', '2017-01-20', 'barack@usa.com'),
        ('Bono', 'Vox', '2014-07-13', 'bono@u2.com', 'admin');
```

Comma separated list of values for each row.

https://dev.mysql.com/doc/refman/8.0/en/insert.html

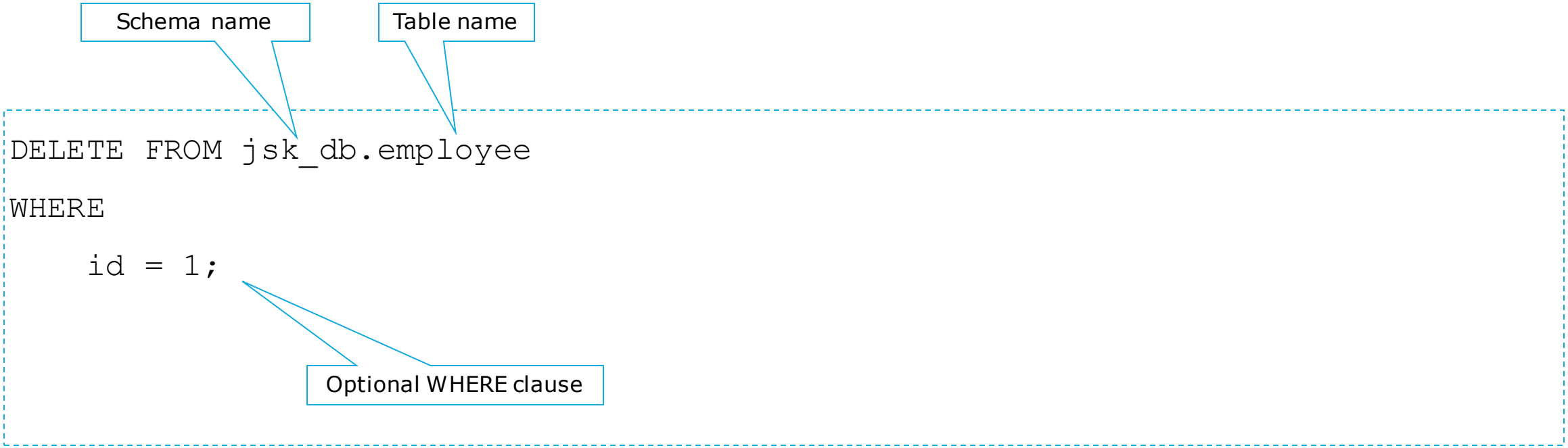
# DML: UPDATE




- **DANGER**: Since WHERE clause is optional, running the statement without it will change all rows in the table.
- NOTE: there is no UNDO button in the DB. And it's surprisingly easy to screw things up (BUT there are Transactions)
- When doing difficult migrations it might be wise to store table data as backup (CREATE TABLE … AS SELECT …).

https://dev.mysql.com/doc/refman/8.0/en/update.html

# DML: DELETE FROM




https://dev.mysql.com/doc/refman/8.0/en/delete.html

# Let’s do some excersises…


Sample table „employee”

| id | first_name | last_name | hired | email | created_by | created_at |
|---|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 2017-01-01 | mkowalski@capg.com | hr1 | 2019-02-17 |
| 2 | Armando | Roggio | 2014-03-13 | armando@some.com | hr1 | 2019-01-01 |
| 3 | Barack | Obama | 2017-01-20 | barack@usa.com | admin | 2019-01-02 |
| 4 | Bono | Vox | 2014-07-13 | bono@u2.com | admin | 2019-01-01 |
| 5 | Bill | Gates | 2009-02-20 | bill@microsoft.com | admin | 2019-01-01 |
| 6 | Steve | Jobs | 1988-02-08 | steve@apple.com | null | null |
| 7 | Larry | Ellison | 2000-08-09 | larry@oracle.com | admin | 2019-01-01 |
| 8 | Maciej | Nowak | 2017-02-01 | mnowak@capg.com | hr2 | 2017-02-01 |

https://dev.mysql.com/doc/refman/8.0/en/select.html

# Understanding GROUP BY


| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 2017-01-01 | hr1 | 2019-02-17 |
| 2 | Armando | Roggio | 2014-03-13 | hr1 | 2019-01-01 |
| 3 | Barack | Obama | 2017-01-20 | admin | 2019-01-02 |
| 4 | Bono | Vox | 2014-07-13 | admin | 2019-01-01 |
| 5 | Bill | Gates | 2009-02-20 | admin | 2019-01-01 |
| 6 | Steve | Jobs | 1988-02-08 | null | null |
| 7 | Larry | Ellison | 2000-08-09 | admin | 2019-01-01 |
| 8 | Maciej | Nowak | 2017-02-01 | hr2 | 2017-02-01 |

```
SELECT e.created_by,
       COUNT(*),
       MAX(e.id)
FROM employee AS e
GROUP BY e.created_by;
```

| created_by | COUNT(*) | MAX(e.id) |
|---|---|---|
| hr1 | 2 | 2 |
| admin | 4 | 7 |
| null | 1 | 6 |
| hr2 | 1 | 8 |

| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 2017-01-01 | hr1 | 2019-02-17 |
| 2 | Armando | Roggio | 2014-03-13 | hr1 | 2019-01-01 |

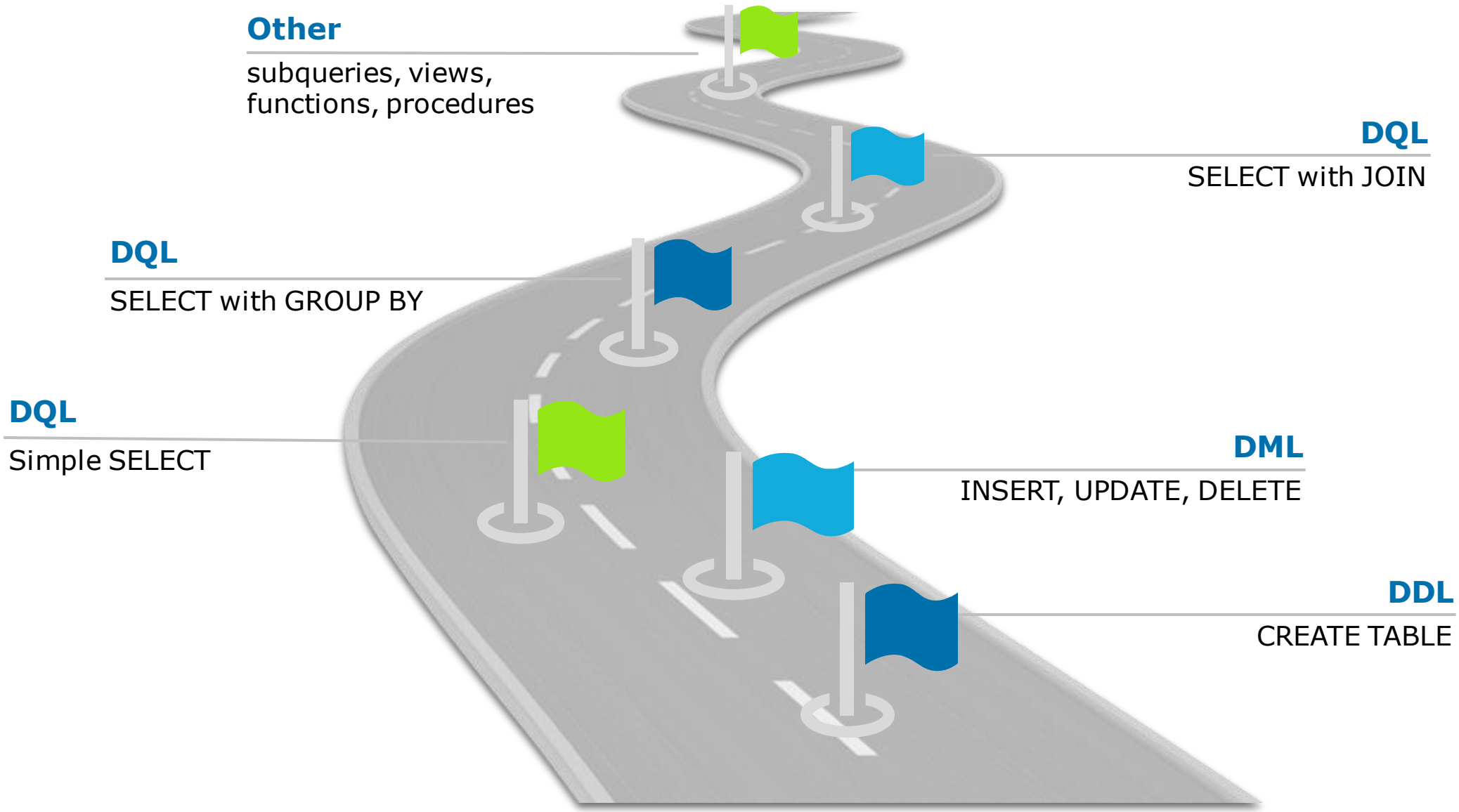
| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 3 | Barack | Obama | 2017-01-20 | admin | 2019-01-02 |
| 4 | Bono | Vox | 2014-07-13 | admin | 2019-01-01 |
| 5 | Bill | Gates | 2009-02-20 | admin | 2019-01-01 |
| 7 | Larry | Ellison | 2000-08-09 | admin | 2019-01-01 |

| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 6 | Steve | Jobs | 1988-02-08 | null | null |

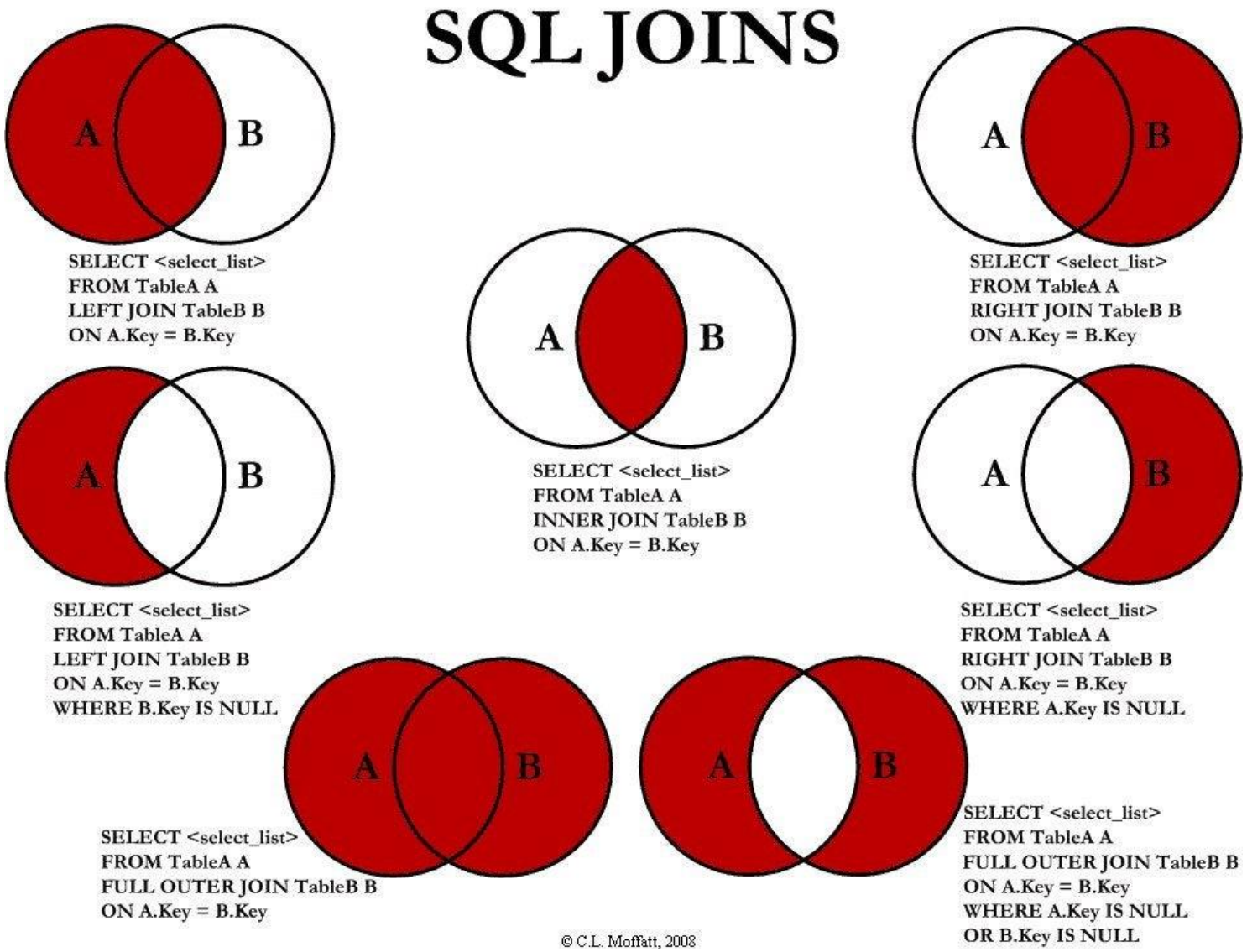
| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 8 | Maciej | Nowak | 2017-02-01 | hr2 | 2017-02-01 |

https://dev.mysql.com/doc/refman/8.0/en/select.html

# Almost there…



# Join - Types

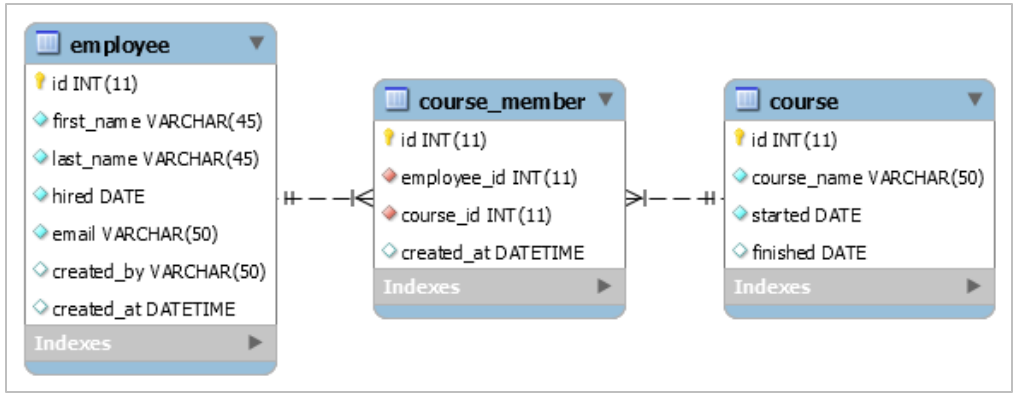



https://dev.mysql.com/doc/refman/8.0/en/join.html ; Source: https://pawelwiejkut.net/sql-on-conditions-in-left-join/

# Let’s create more tables

SEC1

```
CREATE TABLE jsk_db.course (
  id INT NOT NULL AUTO_INCREMENT,
  course_name VARCHAR(50) NOT NULL,
  started DATE NOT NULL,
  finished DATE DEFAULT NULL,
  PRIMARY KEY (id),
  UNIQUE INDEX uq_course_name (course_name)
);
```




```
CREATE TABLE jsk_db.course_member (
  id INT NOT NULL AUTO_INCREMENT,
  employee_id INT NOT NULL,
  course_id INT NOT NULL,
  PRIMARY KEY (id),
  UNIQUE INDEX uq_course_member
         (employee_id, course_id),
  CONSTRAINT fk_employee_id
     FOREIGN KEY (employee_id)
     REFERENCES employee (id),
  CONSTRAINT fk_course_id
     FOREIGN KEY (course_id)
     REFERENCES course (id)
);
```

https://dev.mysql.com/doc/refman/8.0/en/sql-data-definition-statements.html

# JOIN


Table: employee

| id | first_name | last_name | hired | created_by | created_at |
|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | 2017-01-01 | hr1 | 2019-02-17 |
| 2 | Armando | Roggio | 2014-03-13 | hr1 | 2019-01-01 |
| 3 | Barack | Obama | 2017-01-20 | admin | 2019-01-02 |
| 4 | Bono | Vox | 2014-07-13 | admin | 2019-01-01 |
| 5 | Bill | Gates | 2009-02-20 | admin | 2019-01-01 |
| 6 | Steve | Jobs | 1988-02-08 | null | null |
| 7 | Larry | Ellison | 2000-08-09 | admin | 2019-01-01 |
| 8 | Maciej | Nowak | 2017-02-01 | hr2 | 2017-02-01 |

Table: course

| id | course_name | started | finished |
|---|---|---|---|
| 1 | Deutschkurs 101 | 2019-02-20 | null |
| 2 | Deutschkurs 102 | 2019-02-20 | null |
| 3 | English course 101 | 2019-01-21 | 2019-01-31 |
| 4 | English course 102 | 2019-02-20 | null |
| 5 | English course 201 | 2019-02-20 | |

Table: course_member

| id | employee_id | course_id | created_at |
|---|---|---|---|
| 1 | 1 | 1 | 2019-02-20 |
| 2 | 1 | 2 | 2019-02-20 |
| 3 | 1 | 3 | 2019-02-20 |
| 4 | 1 | 4 | 2019-02-20 |
| … | … | … | … |

| employee_id | first_name | last_name | Deutschkrus 101 | Deutshkurs 102 | English course 101 | English course 102 | English course 201 |
|---|---|---|---|---|---|---|---|
| 1 | Maciej | Kowalski | | | | | |
| 2 | Armando | Roggio | | | | | |
| 3 | Barack | Obama | | | | | |
| 4 | Bono | Vox | | | | | |
| 5 | Bill | Gates | | | | | |
| 6 | Steve | Jobs | | | | | |
| 7 | Larry | Ellison | | | | | |
| 8 | Maciej | Nowak | | | | | |

https://dev.mysql.com/doc/refman/8.0/en/join.html

# Oracle syntax vs ANSI syntax


```
SELECT e.*
FROM
    employee e,
    course_member cm
WHERE
    cm.employee_id = e.id;
```

```
SELECT e.*
FROM employee e
JOIN course_member cm ON (cm.employee_id = e.id);
```

Source: https://asktom.oracle.com/pls/apex/asktom.search?tag=recommended-join-style ; https://docs.oracle.com/en/database/oracle/oracle-database/12.2/sqlrf/Joins.html#GUID-794F7DD5-FB18-4ADC-9E46-ADDA8C30C3C6

# DQL: SELECT - subqueries


```
SELECT c.*
FROM curse c
WHERE c.id IN (
  SELECT cm.course_id
  FROM course_member cm
  WHERE cm.created_at < '2019-02-01'
);
```

```
SELECT c.*
FROM course c
WHERE EXISTS (
  SELECT 1
  FROM course_member cm
  WHERE cm.course_id = c.id
    AND cm.created_at < '2019-02-01'
);
```

```
SELECT c.*
FROM comments c
JOIN properties p ON (p.comments_id = c.id)
WHERE p.property = 'OLD';
```

https://dev.mysql.com/doc/refman/8.0/en/subqueries.html ; https://dev.mysql.com/doc/refman/8.0/en/optimizing-subqueries.html ; https://dev.mysql.com/doc/refman/8.0/en/subquery-optimization-with-exists.html

# VIEW

SEC1

```
CREATE OR REPLACE VIEW v_employees AS
SELECT e.id, e.first_name, e.last_name
FROM employee e;
```

```
SELECT e.* FROM v_employees e;
```

```
CREATE OR REPLACE VIEW v_courses_and_attendants
SELECT c.id AS course_id,
  c.course_name,
  COUNT(cm.id) AS num_participiants
FROM course c
LEFT JOIN course_member cm ON (cm.course_id = c.id)
GROUP BY c.id, c.course_name
ORDER BY c.course_name ASC;
```

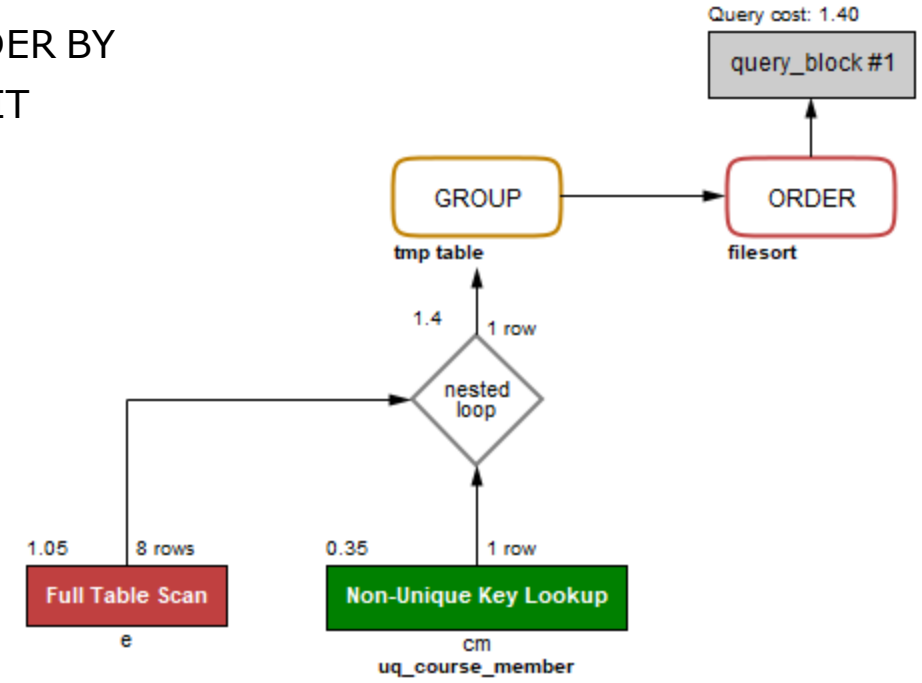
```
SELECT c.*,
  v.num_participiants
FROM course c
JOIN v_courses_and_attendants v
  ON (v.course_id = c.id)
WHERE v.num_participiants > 2;
```

https://dev.mysql.com/doc/refman/8.0/en/create-view.html ; https://dev.mysql.com/doc/refman/8.0/en/with.html

# Order of execution

```
SELECT e.id, e.first_name, e.last_name,
        COUNT(cm.id) AS num_courses
FROM employee e
JOIN course_member cm ON (cm.employee_id = e.id)
WHERE e.first_name LIKE '%a%'
GROUP BY e.id, e.first_name, e.last_name
HAVING COUNT(*) > 1
ORDER BY e.first_name ASC, e.last_name ASC;
```

1. FROM & JOIN
2. WHERE
3. GROUP BY & aggregate functions
4. HAVING
5. WINDOW functions
6. ORDER BY
7. LIMIT




Source:https://www.eversql.com/sql-order-of-operations-sql-query-order-of-execution/

# Functions and procedures


- Functions – processing input data and returns result / modify function's parameters
- Procedures – processing data / returns rowset

```
DROP FUNCTION IF EXISTS fn_hello;

DELIMITER $$

CREATE FUNCTION fn_hello(name VARCHAR(50))
  RETURNS VARCHAR(100) DETERMINISTIC
BEGIN
  RETURN CONCAT('Hello ', name, '!');
END
$$

DELIMITER ;
```

```
-- Sample call
SELECT fn_hello('Jan');

-- Use with data from table
SELECT e.id,
  e.first_name,
  e.last_name,
  fn_hello(e.first_name) AS hello
FROM employee e
ORDER BY e.first_name ASC;
```

https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

# Why it is worth to alias tables/columns?

SEC1

```
SELECT DISTINCT
              serial_no AS BARCODE_NO
              , org_id AS ORG_ID
              , wip_entity_id AS WOID
              , model_suffix AS MTRLID
              , model AS MODLID
              , suffix AS SFFX_NAME
              , line_id AS PCSGID
              , buyer_code AS CUSTOMERID
              , serial AS LBL_SERIAL_NO
              , quantity
              , wip_entity_name
              , serial_no AS BUYER_SERIAL_NO
              , TO_CHAR(sysdate, 'YYYYMMDD') AS PUB_YMD
              , TO_TIMESTAMP(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') AS CREATED_AT
FROM prodplan.daily_prod_plan pln
JOIN matmgmt.workorder wo ON (WOID = wip_entity_id)
LEFT JOIN matmgmt.serialnumbers snb ON (
              division_code = division
              AND work_order = mfg_order)
LEFT JOIN matmgmt.serialnumbers_sent sns ON (
              division_code = division
              AND work_order = mfg_order
              AND serial = serial)
LEFT JOIN prodplan.xxx_register_inf lph ON (barcode_no = serial_no)
WHERE 1=1
              AND org_id = '1111'
              AND division = 'ABC'
              AND item_type = 'F'
              AND barcode_no is null
              AND mfg_order = '111X0123'
              AND serial = 'KP001'
              AND ROWNUM = 1;
```

```
SELECT DISTINCT
              snb.serial_no AS BARCODE_NO
              , pln.org_id AS ORG_ID
              , wo.wip_entity_id AS WOID
              , wo.model_suffix AS MTRLID
              , wo.model AS MODLID
              , wo.suffix AS SFFX_NAME
              , pln.line_id AS PCSGID
              , wo.buyer_code AS CUSTOMERID
              , snb.serial AS LBL_SERIAL_NO
              , wo.quantity
              , wo.wip_entity_name
              , sns.serial_no AS BUYER_SERIAL_NO
              , TO_CHAR(sysdate, 'YYYYMMDD') AS PUB_YMD
              , TO_TIMESTAMP(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') AS CREATED_AT
FROM prodplan.daily_prod_plan pln
JOIN matmgmt.workorder wo ON (pln.WOID = wo.wip_entity_id)
LEFT JOIN matmgmt.serialnumbers snb ON (
              snb.division_code = wo.division
              AND snb.work_order = pln.mfg_order)
LEFT JOIN matmgmt.serialnumbers_sent sns ON (
              sns.division_code = wo.division
              AND sns.work_order = pln.mfg_order
              AND sns.serial = snb.serial)
LEFT JOIN prodplan.xxx_register_inf lph ON (lph.barcode_no = snb.serial_no)
WHERE 1=1
              AND pln.org_id = '1111'
              AND wo.division = 'ABC'
              AND wo.item_type = 'F'
              AND lph.barcode_no is null
              AND pln.mfg_order = '111X0123'
              AND snb.serial = 'KP001'
              AND ROWNUM = 1;
```

# Window Functions (MySQL) / Analytic Functions (Oracle)


```
mysql> SELECT
         val,
         ROW_NUMBER() OVER w AS 'row_number',
         RANK()       OVER w AS 'rank',
         DENSE_RANK() OVER w AS 'dense_rank'
       FROM numbers
       WINDOW w AS (ORDER BY val);
+------+------------+------+------------+
| val  | row_number | rank | dense_rank |
+------+------------+------+------------+
|    1 |          1 |    1 |          1 |
|    1 |          2 |    1 |          1 |
|    2 |          3 |    3 |          2 |
|    3 |          4 |    4 |          3 |
|    3 |          5 |    4 |          3 |
|    3 |          6 |    4 |          3 |
|    4 |          7 |    7 |          4 |
|    4 |          8 |    7 |          4 |
|    5 |          9 |    9 |          5 |
+------+------------+------+------------+
```

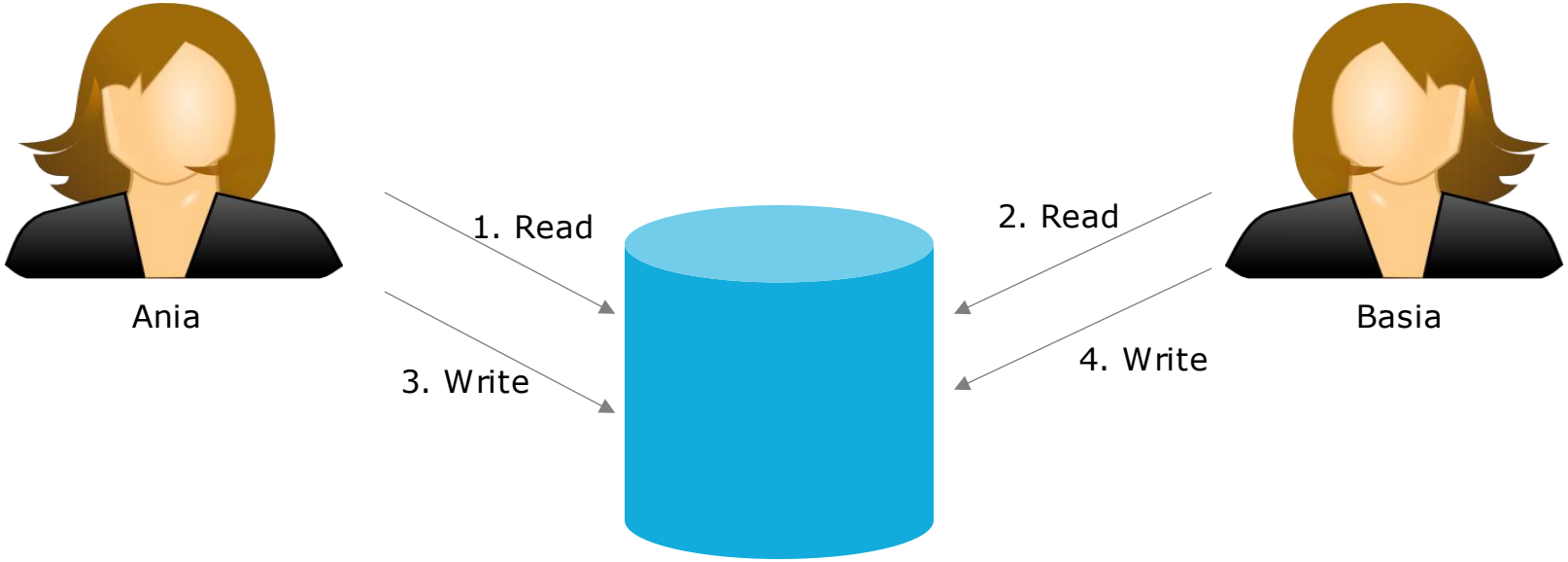
MySQL: https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html ; Oracle: https://docs.oracle.com/cd/E11882_01/server.112/e41084/functions004.htm#SQLRF06174

# Agenda


- Before we Begin
- Relational Databases (DB)
- Structured Query Language (SQL)
- **Data locking strategies**
- Advanced Databases Topics
- Modeling databases
- Assignments

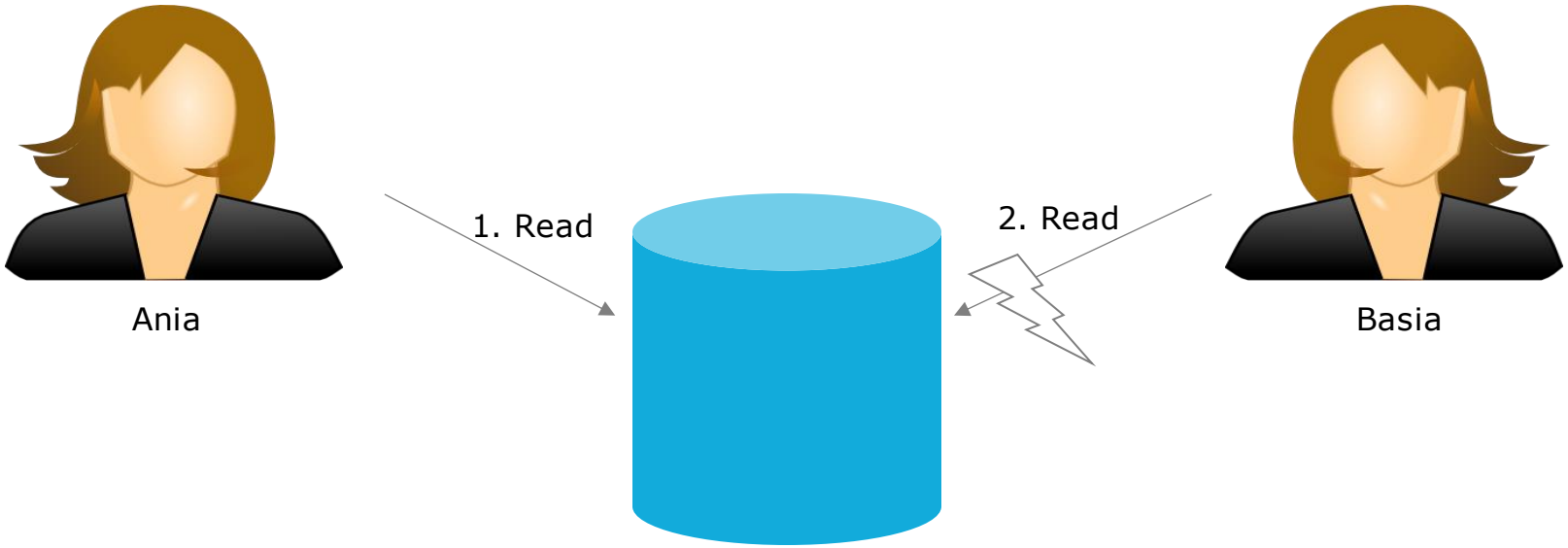
# No locking

- Effect: Last one wins.
- Problematic from the business point of view.



# Pesimistic locking

- Effect: First one locks the data.
- Can result in long lasting locks that disturb the work.



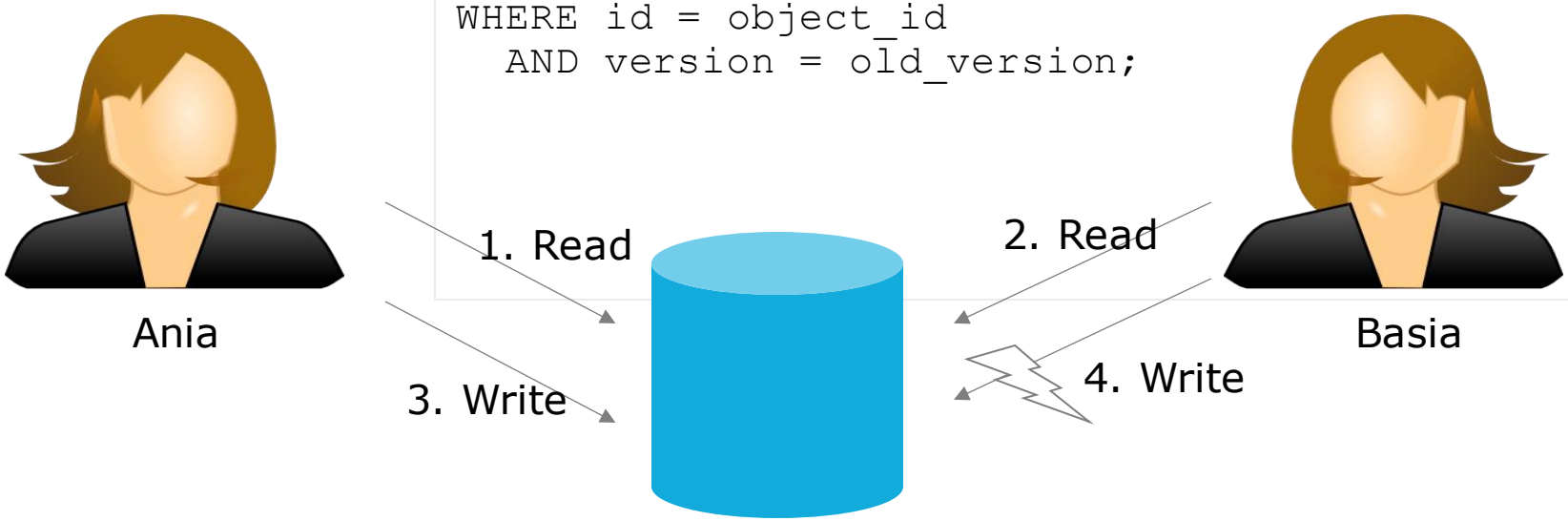
# Optimistic locking

- Effect: Only the first write is successful.
- Standard approach

Implementation:

1. Table has an attribute: version (int)
2. When you update the row, you increment the version
3. You can only update if row's version matches Yours's version

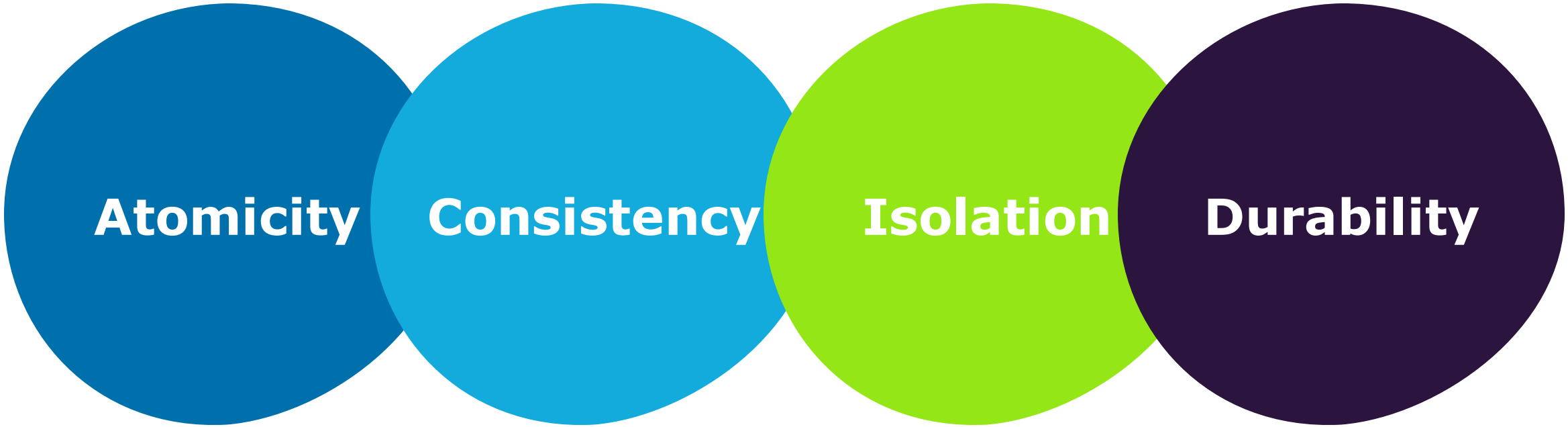
```
UPDATE table
SET attribute = new_value
  , version = version+1
WHERE id = object_id
  AND version = old_version;
```



# Agenda


- Before we Begin
- Relational Databases (DB)
- Structured Query Language (SQL)
- Data locking strategies
- **Advanced Databases Topics**
- Modeling databases
- Assignments

# Transactions. ACID Properties




Isolation levels:

1. Serializable
2. Repeatable read
3. **Read committed**
4. Read uncommitted

# Transactions


## Connection 1

```
SHOW VARIABLES LIKE 'transaction_isolation’;
-- REPEATABLE-READ
```

```
INSERT INTO test (comment) VALUES ('test2');
COMMIT; -- or ROLLBACK
```

```
BEGIN; -- start transaction
INSERT INTO test (comment) VALUES ('test1');
```

```
SELECT * FROM test;
-- test1 not visible
```

```
SELECT * FROM test;
-- test1 and test2 visible
```

## Connection 2

# Agenda


- Before we Begin
- Relational Databases (DB)
- Structured Query Language (SQL)
- Data locking strategies
- Advanced Databases Topics
- **Modeling databases**
- Assignments

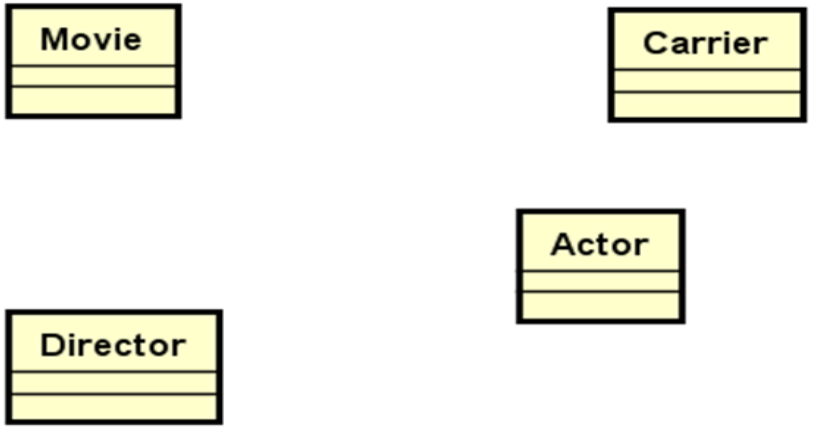
# Consider this simple excercise


## Description of the domain:

1. The database is to be used for personal movie organization purposes.
2. All movies are stored on some kind of a data carrier (e.g. VideoCD, DVD, BlueRay).
3. More than one movie can be stored on a carrier.
4. Each carrier is identified by a unique number.
5. The database must contain a photo of the carriers front cover.
6. For each and every movie a title, year of production and one director must be present and all actors must be listed.
7. …

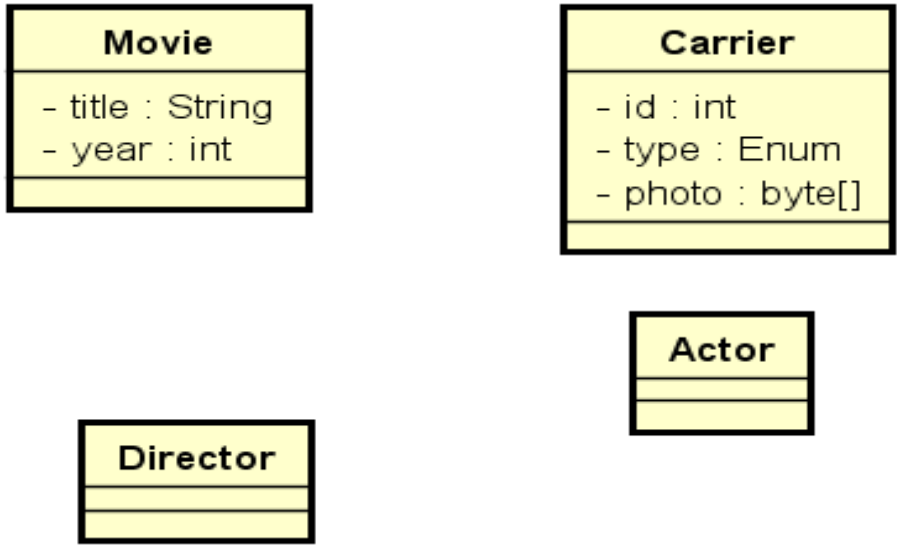
# Identify all entities


The database is to be used for personal movie organization purposes. All **movies** are stored on some kind of a data **carrier** (e.g. VideoCD, DVD, BlueRay), more than one **movie** can be stored on a **carrier**. Each **carrier** is identified by a unique number. The database must contain a photo of the **carriers** front cover. For each and every movie a title, year of production and one **director** must be present and all **actors** must be listed. ...



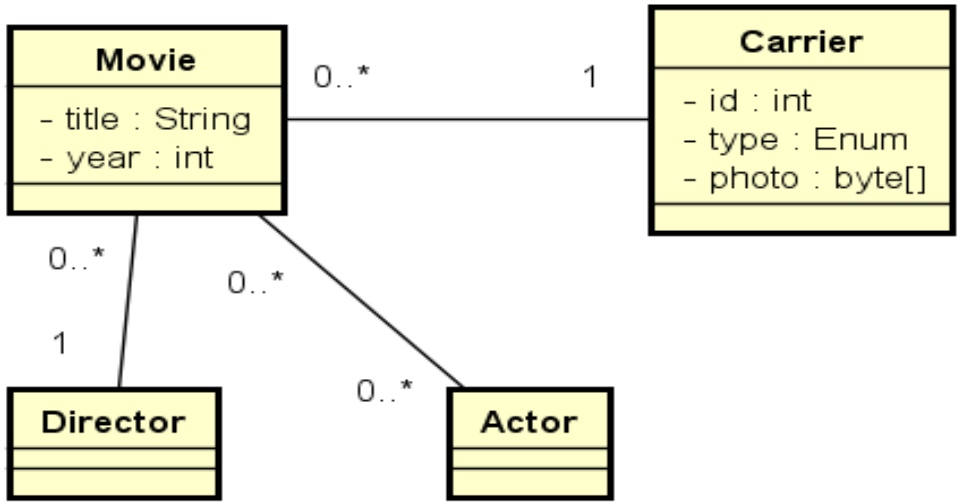
# Identify entity attributes

The database is to be used for personal movie organization purposes. All **movies** are stored on some kind of a data **carrier** (e.g. VideoCD, DVD, BlueRay), more than one **movie** can be stored on a **carrier**. Each **carrier** is identified by a unique number. The database must contain a photo of the **carriers** front cover. For each and every movie a title, year of production and one **director** must be present and all **actors** must be listed. ...

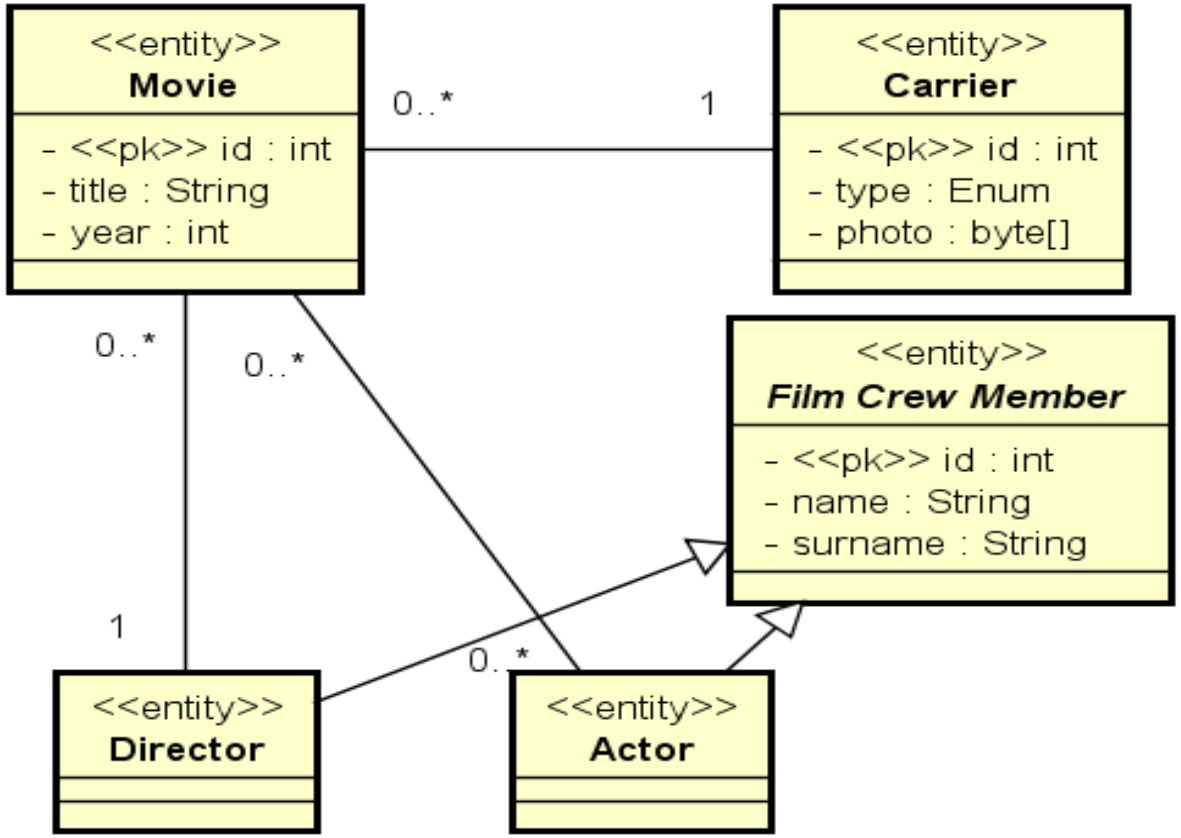


# Identify the relations between the entities

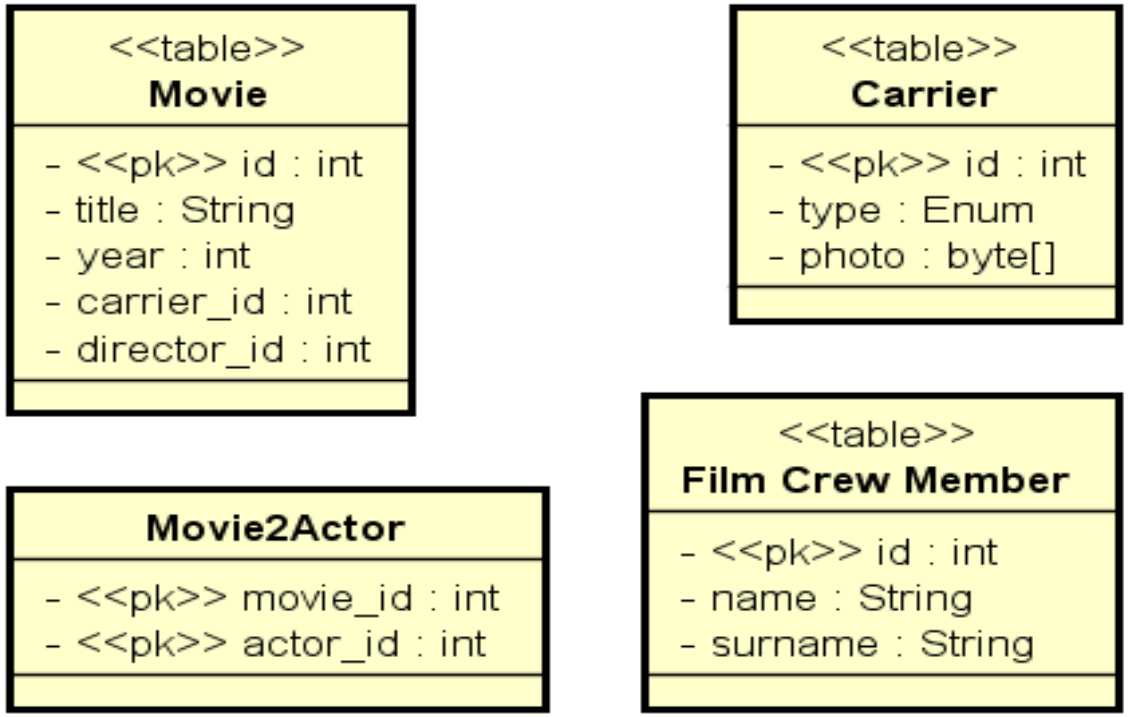
The database is to be used for personal movie organization purposes. All **movies** are stored on some kind of a data **carrier** (e.g. VideoCD, DVD, BlueRay), more than one **movie** can be stored on a **carrier**. Each **carrier** is identified by a unique number. The database must contain a photo of the **carriers** front cover. For each and every movie a title, year of production and one **director** must be present and all **actors** must be listed. ...



# Validate, normalize, do a sanity check.
# What you get is a business database model.



# Create a technical database model



# Agenda

SEC1

- Before we Begin
- Relational Databases (DB)
- Structured Query Language (SQL)
- Data locking strategies
- Advanced Databases Topics
- Modeling databases
- **Assignments**

# Assignments

1. Proszę zwrócić uwagę na informacje zawarte w mailu powitalnym
   1. zwróćcie uwagę na sugestie
   2. pomocne linki
   3. pytajcie trenerów!

2. Niektóre zadania pozwalają na interpretacje
   1. proponujcie rozwiązania, argumentujcie „dlaczego tak a nie inaczej”
   2. doprecyzujcie niektóre tematy na konsultacjach
   3. brońcie Waszego rozwiązania podczas konsultacji

3. Przydatny plik do analizy: „JSK  DB, live sample*.sql”

# Things to remember

1. Carefully read questions
2. Listen Your trainer
3. Assignments:
    1. Use aliases on tables / subqueries
    2. Do not use subqueries in places, where they are inefficient
    3. Sample scripts were prepared in a way, in which they could be easily re-created; do the same

# Questions?

SEC1

## About Capgemini

Capgemini is a global leader in consulting, digital transformation, technology and engineering services. The Group is at the forefront of innovation to address the entire breadth of clients' opportunities in the evolving world of cloud, digital and platforms. Building on its strong 50-year+ heritage and deep industry-specific expertise, Capgemini enables organizations to realize their business ambitions through an array of services from strategy to operations. Capgemini is driven by the conviction that the business value of technology comes from and through people. Today, it is a multicultural company of 270,000 team members in almost 50 countries. With Altran, the Group reported 2019 combined revenues of €17billion.

Learn more about us at

www.capgemini.com

**People matter, results count.**