Lab4 is up!

-2

These that need more = (Really \ All pairs) * A

MovieAward \ (Movie × Awards)

m_1 × A
m_2 × Awards

m_1
m_2

StudentCourse Took \ Student × Courses

s_1
s_2

s_1 × ⋞
s_2 ⋞

ALL → (:) → 6 up

Major ⋞ Awards

= Movie / Award

Student ⋞ Courses → Stud / Courses

Prof ⋞ Course

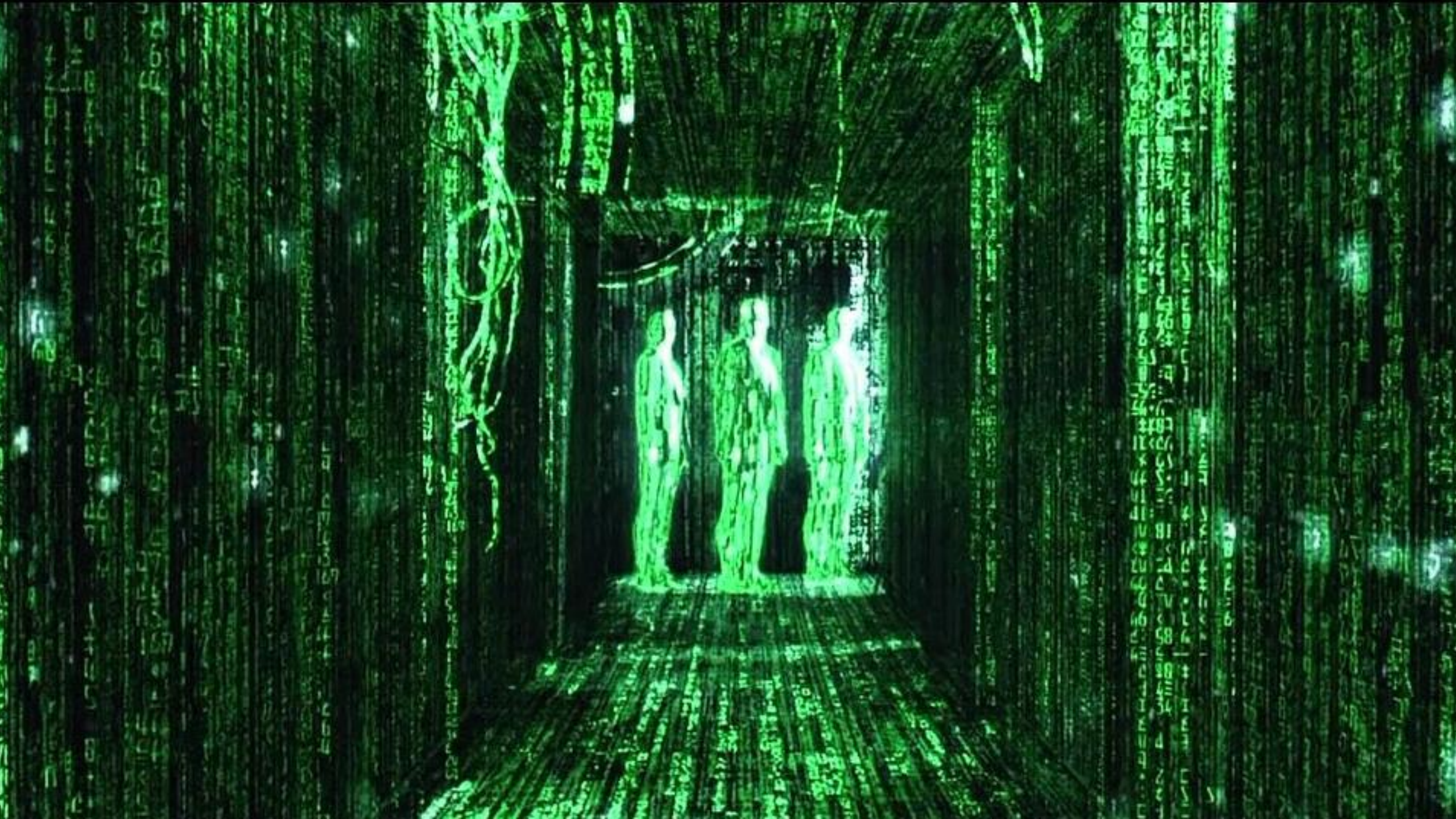
Student ⋞ Books

ALL = Really \ A

Q4Me

-1

Book vs. Slides	SQL	CH06 (1st & 2nd Ed.)
Lab		?
Last Weeks		?



Today

Data Modeling in RDBMS

Real World Entity

Conceptual Level | Entity-Relationship Model (E/R) Level

Conceptual Level | Logical Level | Relational Model

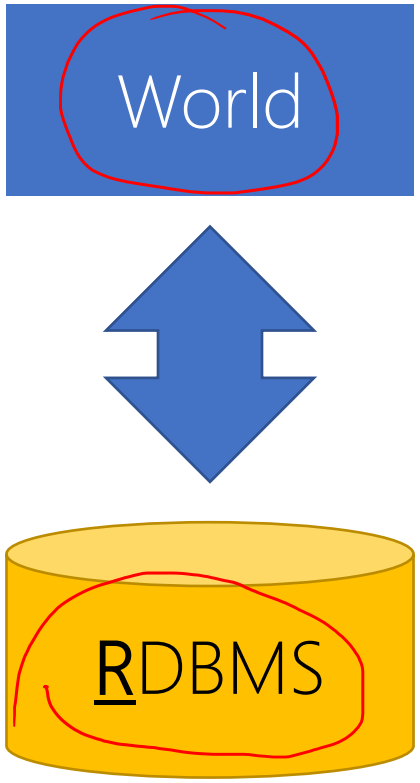
Conceptual Level | Logical Level | Physical Level | SQL

Conceptual Level | Logical Level | Computable Entity

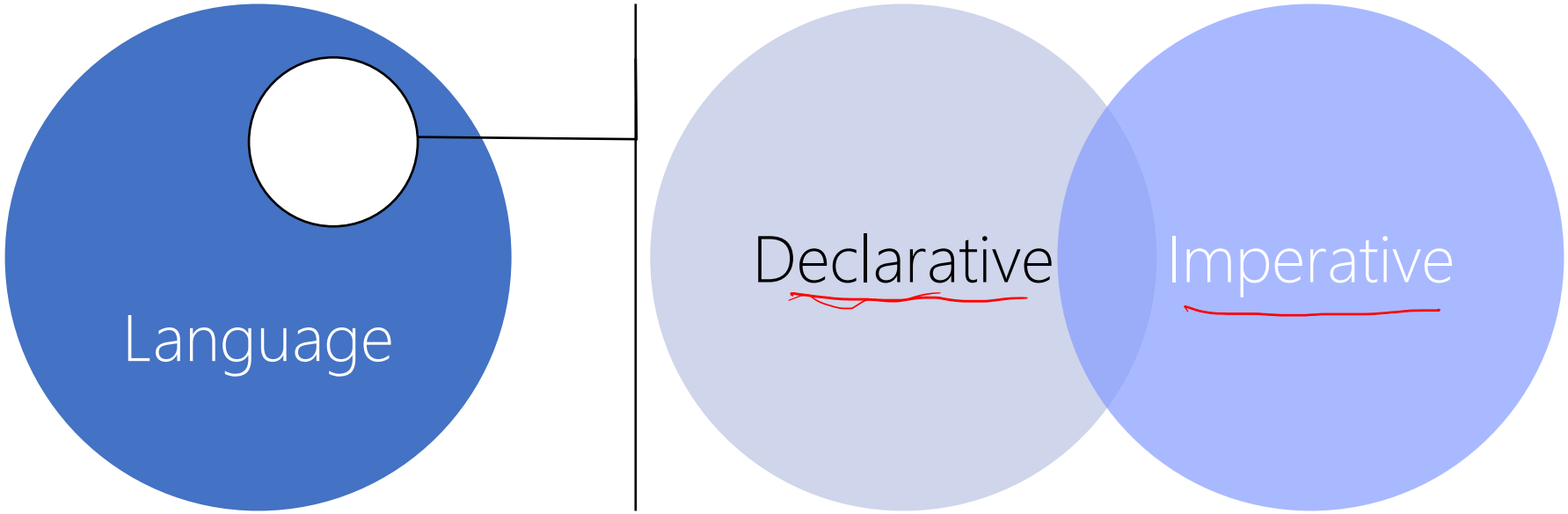
SQL × Language

S DBMS
O DBMS

Communicate with Relational Database Management Systems (RDBMS)



Language × Types



Language × Types

Imperative
(How)

```
#include <stdio.h>
void main() {
        int n;
        scanf("%d", &n)
        int r = 1;
        for(int i = n; i > 0; i--){
            r = r * i;
        }
        printf(r);
    }
}
```

10

100!

200!

Although correct, this program does not work in practice! (Why?)

Language × Types

Declarative
(What)

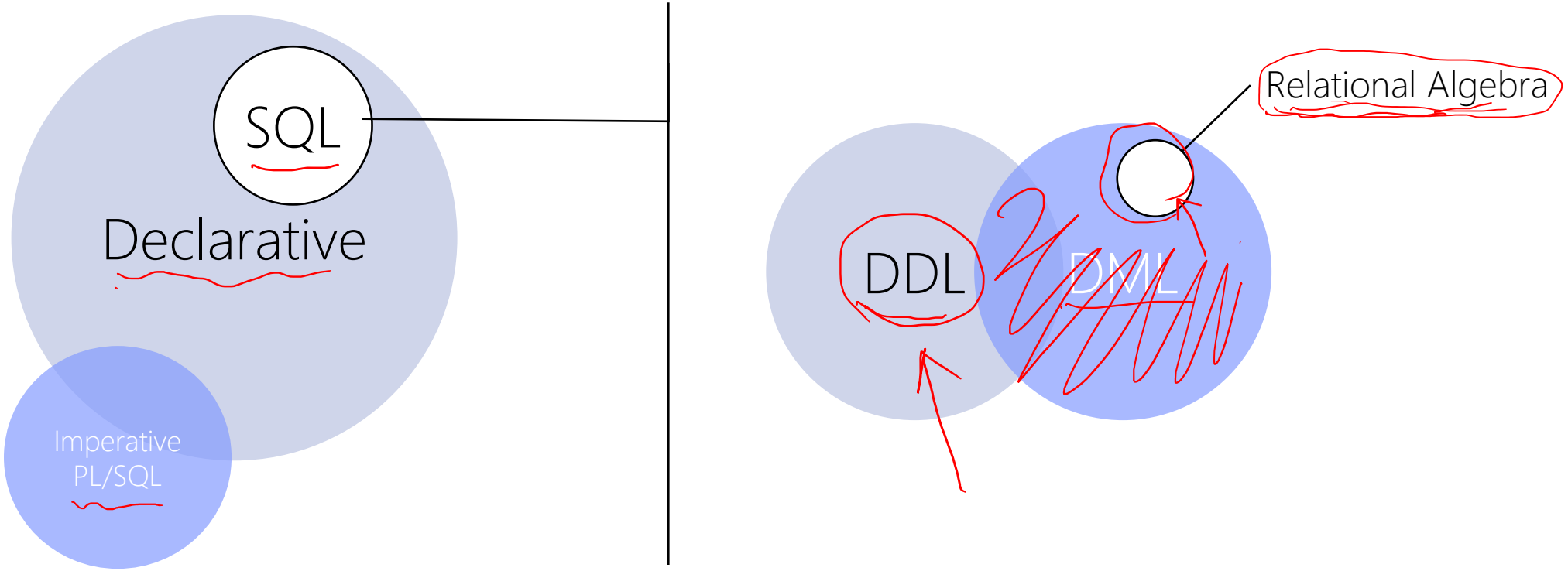
100

```
SELECT FACT(n);
```

10

The system figures out `best' way to execute query.

SQL × Declarative



SQL × Intro × History

The most commonly used relational DBMS's query and modify the database through a language called SQL (sometimes pronounced "sequel"). SQL stands for "Structured Query Language." The portion of SQL that supports queries has capabilities very close to that of relational algebra. However: SQL also includes statements for modifying the database (e.g., inserting and deleting tuples from relations) and for declaring a database schema. Thus, SQL serves as both a data-manipulation language and as a data definition language. SQL also standardizes many other database command. There are many different dialects of SQL. First, there are three major standards. There is ANSI (American National Standards Institute) SQL and an updated standard adopted in 1992, called SQL-92 or SQL2. The recent SQL-99 (previously referred to as SQL3) standard extends SQL2 with object-relational features and a number of other new capabilities. Then, there are versions of SQL produced by the principal DBMS vendors. These all include the capabilities of the original .ITS1 standard. They also conform to a large extent to the more recent SQL2, although each has its variations and extensions beyond SQL2, including some of the features in the SQL-99 standard. Herein, we consider SQL as a stand-alone query language.
SQLite does not fully support SQL-92.

SQL-2016

SQL × DDL

Data Definition Language to CREATE, ALTER, DROP relational s

(Table)

- Database
- Table

SQL × DDL × Database

CREATE DATABASE *DatabaseName*;
DROP DATABASE *DatabaseName*;
ALTER DATABASE *DatabaseName* … ;
RENAME DATABASE *DatabaseName* TO *NewDatebaseName*

F_1 F_2 F_3

In SQLite> sqlite3 *DatabaseName;*
In SQLite> simply delete database file
In SQLite> no command!
In SQLite> simply rename database file!

SQL × DDL × Table (Simple)


```
CREATE TABLE TableName(
        ColumnName1 DataType [NULL | NOT NULL],
        ColumnName2 DataType [NULL | NOT NULL],
        …,
        ColumnNameN DataType [NULL | NOT NULL],
);
```

SQL × DDL × Table (Simple) × Datatype

INTEGER | INT
DECIMAL(i, j): `i' total #digits, `j' #digits after decimal point (.), e.g., 12.351 ∈ DECIMAL(5,3)
FLOAT | REAL: single precision real number
DOUBLE: double precision real number

DATE: year month day, the format can be modified, e.g., yy-mm-dd | mm-dd-yyyy
TIME: hour:minute:second, the format can be modified, precision depends on DBMS
DATETIME
TIMESTAMP

CHAR(n): `n' character, if less, padded with space
VARCHAR(n): `n' character max, if less, no padding
TEXT: document, article, ...

BIT: only one bit, 0 | 1
BOOLEAN: TRUE | FALSE
BLOB: large object in binary format (voice, movie, image, ...)

SQL × DDL × Table (Simple) × Datatype

Data types are highly dependent on the underlying DBMS. See manual of the DBMS.

e.g., SQLite → https://www.sqlite.org/datatype3.html

SQL × DDL × Table × Surrogate Key


```
CREATE TABLE TableName(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        ColumnName1 DataType,
        ColumnName2 DataType,
        ...
        ColumnNameN DataType,
);
```

SQL × DDL × Table × Candidate Key


```
CREATE TABLE TableName(
        ColumnName1 DataType [UNIQUE],
        ColumnName2 DataType [UNIQUE],
        ...
        ColumnNameN DataType [UNIQUE],
);

CREATE TABLE TableName(
        ColumnName1 DataType,
        ColumnName2 DataType,
        ...
        ColumnNameN DataType,
        CONSTRAINT UK_Name UNIQUE(column names)
);
```

CK

1

SQL × DDL × Table × Candidate Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255) UNIQUE,
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        CONSTRAINT UK_Title UNIQUE(Title)
);

CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
```

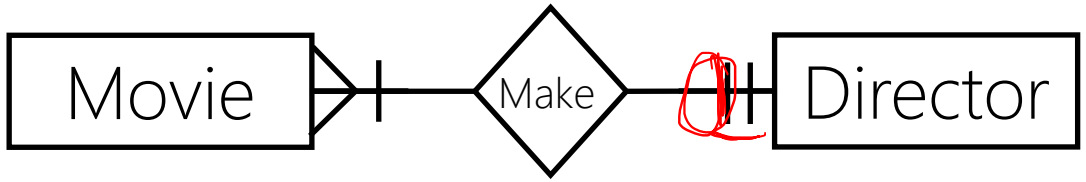
Could be any name, but by convention:
UK_ColumnName1_ColumnName2_...

SQL × DDL × Table × Foreign Key


```
CREATE TABLE TableName(
        ColumnName1 DataType,
        ColumnName2 DataType,
        ...
        ColumnNameN DataType,
        CONSTRAINT FK_Name FOREIGN KEY(foreign key columns)
        REFERENCES TargetTable(TargetTable's primary key columns)
);
```

Id

SQL × DDL × Table × Foreign Key



NOT NULL

R1: Movie(<u>Id</u>, Title, ReleaseDate, Language, RunningTime, DirectorId)
CK={Title}, FK={DirectorId}

R2: Director(<u>Id</u>, FirstName, LastName)
CK={FirstName, LastName}

SQL × DDL × Table × Foreign Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        DirectorId INTEGER NOT NULL,
        CONSTRAINT UK_Title UNIQUE(Title),
        CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
        REFERENCES Director(Id)
);
CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
```

abc

SQL × DDL × Table × Foreign Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        DirectorId INTEGER NOT NULL,
        CONSTRAINT UK_Title UNIQUE(Title),
        CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
        REFERENCES Director(Id)
);
```

Could be any name, but by convention we follow this:
FK_SourceTableName_ForeignKeyColumn_2_TargetTableName_PrimaryKeyColumn

SQL × DDL × Table × Foreign Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        DirectorId INTEGER NOT NULL,
        CONSTRAINT UK_Title UNIQUE(Title),
        CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
        REFERENCES Director(Id)
);
CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
```

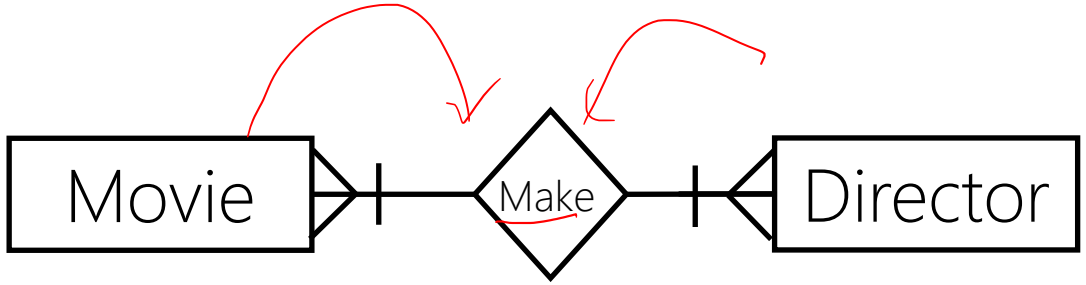
SQL × DDL × Table × Foreign Key


```
CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        DirectoId INTEGER NOT NULL,
        CONSTRAINT UK_Title UNIQUE(Title),
        CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
        REFERENCES Director(Id)
);
```

SQL × DDL × Table × Foreign Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        CONSTRAINT UK_Title UNIQUE(Title)
);
CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
ALTER TABLE Movie ADD COLUMN DirectorId INTEGER
ALTER TABLE Movie ADD CONSTRAINT FK_Movie_DirectorId_2_Director_Id FOREIGN
KEY(DirectorId) REFERENCES Director(Id)
```

SQL × DDL × Table × Foreign Key



R1: Movie(Id, Title, ReleaseDate, Language, RunningTime), CK={Title}
R2: Director(Id, FirstName, LastName), CK={FirstName, LastName}

R3: MovieDirector(Id, MovieId, DirectorId)
FK1={MovieId}, FK2={DirectorId}

SQL × DDL × Table × Foreign Key


```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER,
        CONSTRAINT UK_Title UNIQUE(Title)
);
CREATE TABLE Director(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        FirstName VARCHAR(255) NOT NULL,
        LastName VARCHAR(255) NOT NULL,
        CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
);
CREATE TABLE MovieDirector(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        MovieId INTEGER NOT NULL,
        DirectorId INTEGER NOT NULL,
        CONSTRAINT FK_MovieDirector_MovieId_2_Movie_Id FOREIGN KEY(MovieId)
        REFERENCES Movie(Id),
        CONSTRAINT FK_MovieDirector_DirectorId_2_Director_Id FOREIGN KEY(DirectorId)
        REFERENCES Director(Id)
);
```

SQL × DDL × Table

ALTER TABLE *TableName* ADD COLUMN *ColumnName DataType*
ALTER TABLE *TableName* DROP COLUMN *ColumnName*
ALTER TABLE *TableName* RENAME COLUMN *OldName* TO *NewName*
ALTER TABLE *TableName* RENAME TO *NewTableName*

DROP TABLE *TableName*

Truncate

SQLite does not fully support SQL-92
→ https://www.sqlite.org/omitted.html

SQL × DDL × Table × DEFAULT

What happens when add a column to a table which already has rows?

ALTER TABLE *TableName* ADD COLUMN *ColumnName*
ALTER TABLE *TableName* ADD COLUMN *ColumnName* NULL
ALTER TABLE *TableName* ADD COLUMN *ColumnName* DEFAULT NULL

ALTER TABLE *TableName* ADD COLUMN *ColumnName* NOT NULL ✗
ALTER TABLE *TableName* ADD COLUMN *ColumnName* NOT NULL DEFAULT *Value*

DEFAULT can be used in CREATE TABLE as well.

SQL × DDL × Table × DEFAULT

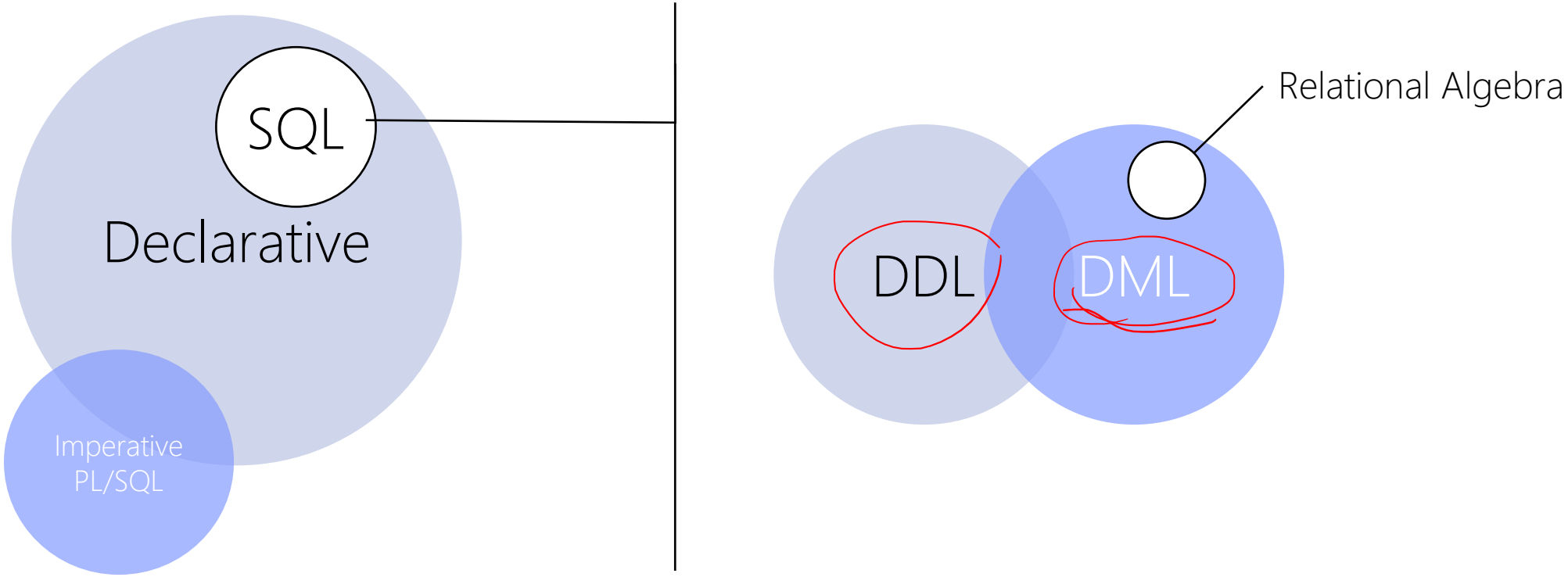

```
CREATE TABLE Movie(
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Title VARCHAR(255),
        ReleaseDate DATE,
        Language VARCHAR(255),
        RunningTime INTEGER DEFAULT 120,
        CONSTRAINT UK_Title UNIQUE(Title)
);
```

Default 2020

Or

```
ALTER TABLE Movie ADD COLUMN RunningTime NOT NULL DEFAULT 120
```

SQL × Declarative



SQL × DML

Data Manipulation Language to

INSERT
UPDATE
DELETE
SELECT

To/From tables.

SQL × DML × INSERT

INSERT INTO *TableName*(*c*, *c'*, *c''*, ...) VALUES (*v*, *v'*, *v''*, ...);

- The number of columns and values must be same.

INSERT INTO Director(Id, FirstName, LastName) VALUES (1, 'Alfred', 'Hitchcock');

INSERT INTO Director(FirstName, LastName) VALUES ('Alfred', 'Hitchcock');

SQL × DML × INSERT

INSERT INTO *TableName*(*c*, *c'*, *c''*, ...) VALUES (*v*, *v'*, *v''*, ...);

- The number of columns and values must be same.

INSERT INTO Director(Id, FirstName, LastName) VALUES (1, 'Alfred', 'Hitchcock');

INSERT INTO Director(FirstName, LastName) VALUES ('Alfred', 'Hitchcock');

- Column list can be omitted. If so, columns with order in original table is assumed.

INSERT INTO Director VALUES (1, 'Alfred', 'Hitchcock');

SQL × DML × INSERT

INSERT INTO *TableName*(*c*, *c*', *c*'', ...) VALUES (*v*, *v*', *v*'', ...);

- The number of columns and values must be same.

INSERT INTO Director(Id, FirstName, LastName) VALUES (1, 'Alfred', 'Hitchcock');

INSERT INTO Director(FirstName, LastName) VALUES ('Alfred', 'Hitchcock');

- Column list can be omitted. If so, columns with order in original table is assumed.

INSERT INTO Director VALUES (1, 'Alfred', 'Hitchcock');

- The data type of value must be compatible with the data type of the corresponding column.

INSERT INTO Director(FirstName, LastName) VALUES ('Alfred', 'Hitchcock');

SQL × DML × INSERT

The following insert would fail. Why?

```
INSERT INTO Director(Id, FirstName, LastName) VALUES ('Alfred', 1, 'Hitchcock');
```

SQL × DML × INSERT

How about this one?

```
INSERT INTO Director(FirstName, LastName) VALUES ('Hitchcock', 'Alfred');
```

SQL × DML

Data Manipulation Language to

INSERT
UPDATE
DELETE
SELECT

from tables.

SQL × DML × UPDATE

UPDATE *TableName* SET *c* = *v*, *c'* = *v'*, *c''* = *v''*, … [WHERE θ];

The data type of value must be compatible with the data type of the corresponding column.
Only rows which satisfy the θ condition will be updated.
If there is no θ, all rows will be updated!

UPDATE Director SET LastName='Hitchkok' WHERE Id = 1;

1 rows affected

0

Id = 1000

SQL × DML × UPDATE

UPDATE *TableName* SET $c = v$, $c' = v'$, $c'' = v''$, … [WHERE θ];

The data type of value must be compatible with the data type of the corresponding column.
Only rows which satisfy the θ condition will be updated.
If there is no θ, all rows will be updated!

```
UPDATE Director SET LastName='Hitchkok' WHERE LastName = 'Hitchcock';
3 rows affected
```

SQL × DML × UPDATE

UPDATE *TableName* SET $c = v$, $c' = v'$, $c'' = v''$, … [WHERE θ];

The data type of value must be compatible with the data type of the corresponding column.
Only rows which satisfy the θ condition will be updated.
If there is no θ, all rows will be updated!

UPDATE Director SET LastName='Hitchkok'
36 rows affected

where 1=1

Be Careful! For safety include WHERE clause.

SQL × DML

Data Manipulation Language to

INSERT
UPDATE
DELETE
SELECT

from tables.

SQL × DML × DELETE

DELETE FROM *TableName* [WHERE θ];

Only rows which satisfy the θ condition will be deleted.
If there is no θ, all rows will be deleted.

```
DELETE FROM Director WHERE Id = 1;
1 rows affected
```

SQL × DML × DELETE

DELETE FROM *TableName* [WHERE θ];

Only rows which satisfy the θ condition will be deleted.
If there is no θ, all rows will be deleted.

```
DELETE FROM Director WHERE LastName = 'Hitchcock';
3 rows affected
```

SQL × DML × DELETE

DELETE FROM *TableName* [WHERE θ];

Only rows which satisfy the θ condition will be deleted.
If there is no θ, all rows will be deleted.

DELETE FROM Director
36 rows affected

Drop Table Dr

Be Extremely Careful! For safety always include WHERE clause.

SQL × DML × Data Integrity

Data Integrity | Integrity Constraints MUST always be assured by DBMS.
ACID Properties (Atomicity, Consistency, Isolation, Durability)

INSERT, UPDATE, DELETE will fail and their effect will be rolled backed if they violate (conflict with) any integrity constraints!

SQL × DML × Data Integrity

I) Domain Integrity
II) Entity Integrity
III) Referential Integrity
IV) User-defined Integrity

SQL × DML × Domain Integrity

Data Type: Specifies that all columns in a relational database must be declared upon a defined domain (datatype). Values in a column MUST comply with domain (datatype) of the column. This includes NULL or NOT NULL.

Varchar ← 1

INT ← 'a'

NOT NULL ← NULL

SQL × DML × Domain Integrity

Data Type: Specifies that all columns in a relational database must be declared upon a defined domain (datatype). Values in a column MUST comply with domain (datatype) of the column. This includes NULL or NOT NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES ('Alfred', 1, 'Hitchcock');
```

datatype mismatch!

SQL × DML × Domain Integrity

Data Type: Specifies that all columns in a relational database must be declared upon a defined domain (datatype). Values in a column MUST comply with domain (datatype) of the column. This includes NULL or NOT NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES (1, NULL, 'Hitchcock');
NOT NULL constraint failed: Director.FirstName
```

SQL × DML × Domain Integrity

Data Type: Specifies that all columns in a relational database must be declared upon a defined domain (datatype). Values in a column MUST comply with domain (datatype) of the column. This includes NULL or NOT NULL.

```
UPDATE Director SET LastName=12.25 WHERE Id = 1;
```

datatype mismatch!

SQL × DML × Domain Integrity

Data Type: Specifies that all columns in a relational database must be declared upon a defined domain (datatype). Values in a column MUST comply with domain (datatype) of the column. This includes NULL or NOT NULL.

'12.25'

```
UPDATE Director SET LastName=12.25 WHERE Id = 1;
SQLite> 1 rows affected.
```

Stupid

SQL × DML × Entity Integrity

Primary Key: Every table MUST have a primary key. The column or columns chosen to be the primary key MUST be UNIQUE and NOT NULL.

CK

SQL × DML × Entity Integrity

Primary Key: Every table MUST have a primary key. The column or columns chosen to be the primary key MUST be UNIQUE and NOT NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES (1,'Alfred', 'Hitchcock');
INSERT INTO Director(Id, FirstName, LastName) VALUES (1,'Alfred', 'Hitchcock');
UNIQUE constraint failed: Director.Id
```

SQL × DML × Entity Integrity

Primary Key: Every table MUST have a primary key. The column or columns chosen to be the primary key MUST be UNIQUE and NOT NULL.

Based on our course, we never face such problem. Why?

SQL × DML × Referential Integrity

Foreign Key: Any foreign key value can only be in one of two states, either

a) primary key value of some table.
b) NULL.

SQL × DML × Referential Integrity

Foreign Key: Any foreign key value can only be in one of two states, either

a) primary key value of some table.
b) NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES (1,'Alfred', 'Hitchcock');
INSERT INTO Movie(Id, Title) VALUES (1, 'The Birds');

INSERT INTO MovieDirector(MovieId, DirectorId) VALUES (1, 1);
```

SQL × DML × Referential Integrity

Foreign Key: Any foreign key value can only be in one of two states, either

a) primary key value of some table.
b) NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES (1,'Alfred', 'Hitchcock');
INSERT INTO Movie(Id, Title) VALUES (1, 'The Birds');

INSERT INTO MovieDirector(MovieId, DirectorId) VALUES (10, 1);
FOREIGN KEY constraint failed, FK_MovieDirector_MovieId_2_Movie_Id
```

SQL × DML × Referential Integrity

Foreign Key: Any foreign key value can only be in one of two states, either

a) primary key value of some table.
b) NULL.

```
INSERT INTO Director(Id, FirstName, LastName) VALUES (1,'Alfred', 'Hitchcock');
INSERT INTO Movie(Id, Title) VALUES (1, 'The Birds');
INSERT INTO MovieDirector(MovieId, DirectorId) VALUES (1, 1);

DELETE FROM Director;
FOREIGN KEY constraint failed
```

Cascade

SQL × DML × User-defined Integrity

Any other constraints specified by a database designer such as candidate keys (UNIQUE), CHECK, …

?

```
INSERT INTO Movie(Title, ReleaseDate) VALUES ('The Birds', 1963);
INSERT INTO Movie(Title, ReleaseDate) VALUES ('The Birds', 1964);
UNIQUE constraint failed: Movie.Title
```

SQL × DML

Data Manipulation Language to

INSERT
UPDATE
DELETE
SELECT

from tables.

SQL × DML × SELECT (Relational Algebra) 65

Operations in relational algebra , i.e.,

- Project(π)
- Select(σ)
- Rename(ρ)
- Union($\cup$)
- Set Difference($\setminus$)
- Cartesian Product($\times$)
- Intersection($\cap$), Division, Joins, …

Only by one statement!

SQL × DML × SELECT × Project(π)

2 SELECT * | *ColumnName1*,
ColumnName2,
...
ColumnNameN

1 FROM *TableName*

$\pi_{ColumnName1,\ ColumnName2,\ \ldots,\ ColumnNameN}(TableName)$

SQL × DML × SELECT × Project(π)

Director						
Id	FirstName	LastName	DateOfBirth	PlaceOfBirth	BestMovieId	MovieCount
1	*Stanley*	*Kubrick*	*Jul. 26, 1928*	*USA*	*1*	*13*
2	*Alfred*	*Hitchcock*	*Aug. 13, 1899*	*England*	*203*	*47*
3	*Clint*	*Eastwood*	*May 31, 1930*	*USA*	*803*	*35*

What are directors' name?

$\pi_{\text{FirstName, LastName}}(\text{Director})$

SELECT FirstName, LastName FROM Director

SQL × DML × SELECT × Project(π)

Director						
Id	FirstName	LastName	DateOfBirth	PlaceOfBirth	BestMovieId	MovieCount
1	*Stanley*	*Kubrick*	*Jul. 26, 1928*	*USA*	*1*	*13*
2	*Alfred*	*Hitchcock*	*Aug. 13, 1899*	*England*	*203*	*47*
3	*Clint*	*Eastwood*	*May 31, 1930*	*USA*	*803*	*35*

How many movies each director made?

$\pi_{\text{FirstName, LastName, MovieCount}}(\text{Director})$

SELECT FirstName, LastName, MovieCount FROM Director

SQL × DML × SELECT × Project(π)

Director						
Id	FirstName	LastName	DateOfBirth	PlaceOfBirth	BestMovieId	MovieCount
1	*Stanley*	*Kubrick*	*Jul. 26, 1928*	*USA*	*1*	*13*
2	*Alfred*	*Hitchcock*	*Aug. 13, 1899*	*England*	*203*	*47*
3	*Clint*	*Eastwood*	*May 31, 1930*	*USA*	*803*	*35*

All information about directors?

$\pi_{\text{Id, FirstName, LastName, ..., MovieCount}}(\text{Director})$

SELECT * FROM Director