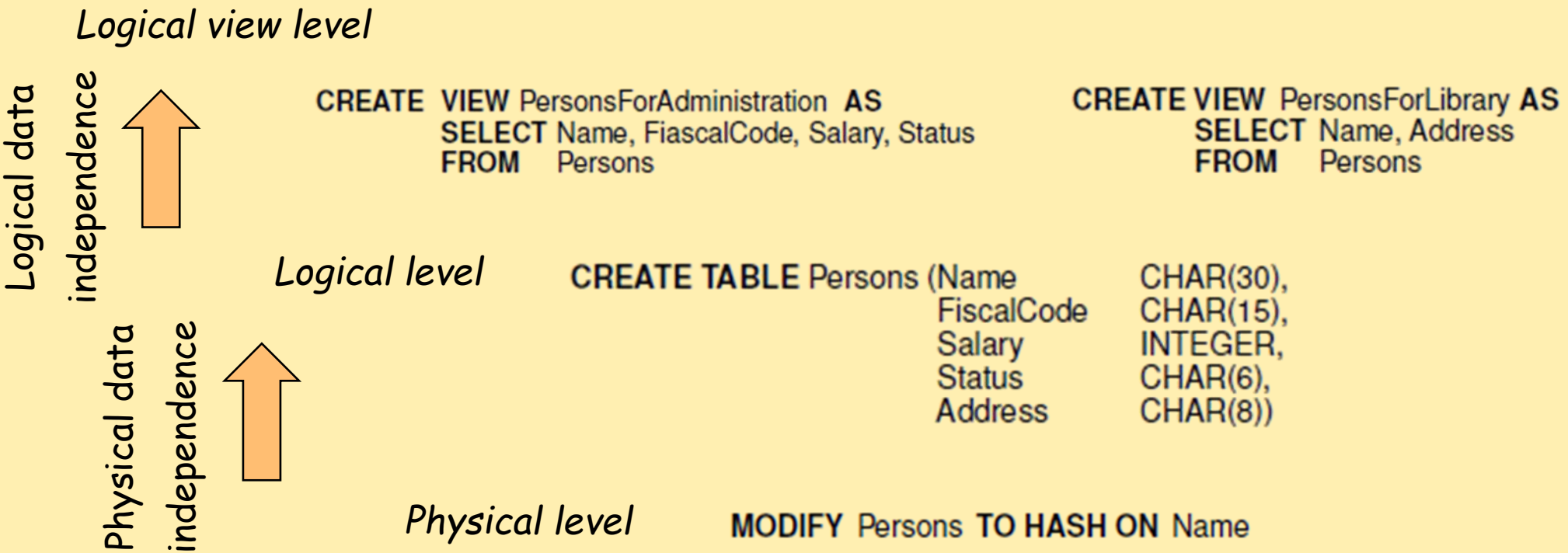
DB and DBMS

- A **database** is a collection of persistent data:
  - The **schema** (or **meta-data**), a collection of time-invariant definitions which describe the structure of admissible data, as well as constraints on legal data values, i.e. integrity constraints *(abstract knowledge)*
    - E.g., relation schemes in the relational data model
  - The **data**, a time-variant representation of specific facts *(concrete knowledge)*
    - E.g., a relation in the relational data model
- A **Data Base Management System (DBMS)** is a centralized or distributed software system, which provides the tools to:
  - define the database schema, and add/modify/delete data,
  - to select the data structures needed to store and retrieve data easily,
  - and to access the data, interactively using a query language or by means of a programming language.

Functions of a DBMS

- Data Description Language (DDL)
- Data Manipulation Language (DML)
- Data Query Language (DQL)
- Database administrator (DBA)



Functions of a DBMS

- A user-accessible system catalog

Table	Type of Information
SYSTABLES	Information about the relational tables
SYSCOLUMNS	Information about the columns in tables and views
SYSVIEWS	Information about views
SYSINDEXES	Information about the indexes on tables
SYSKEYS	Information about the keys on tables

- Data control
  - Access control
  - Integrity control
  - Concurrency control
  - Data recovery

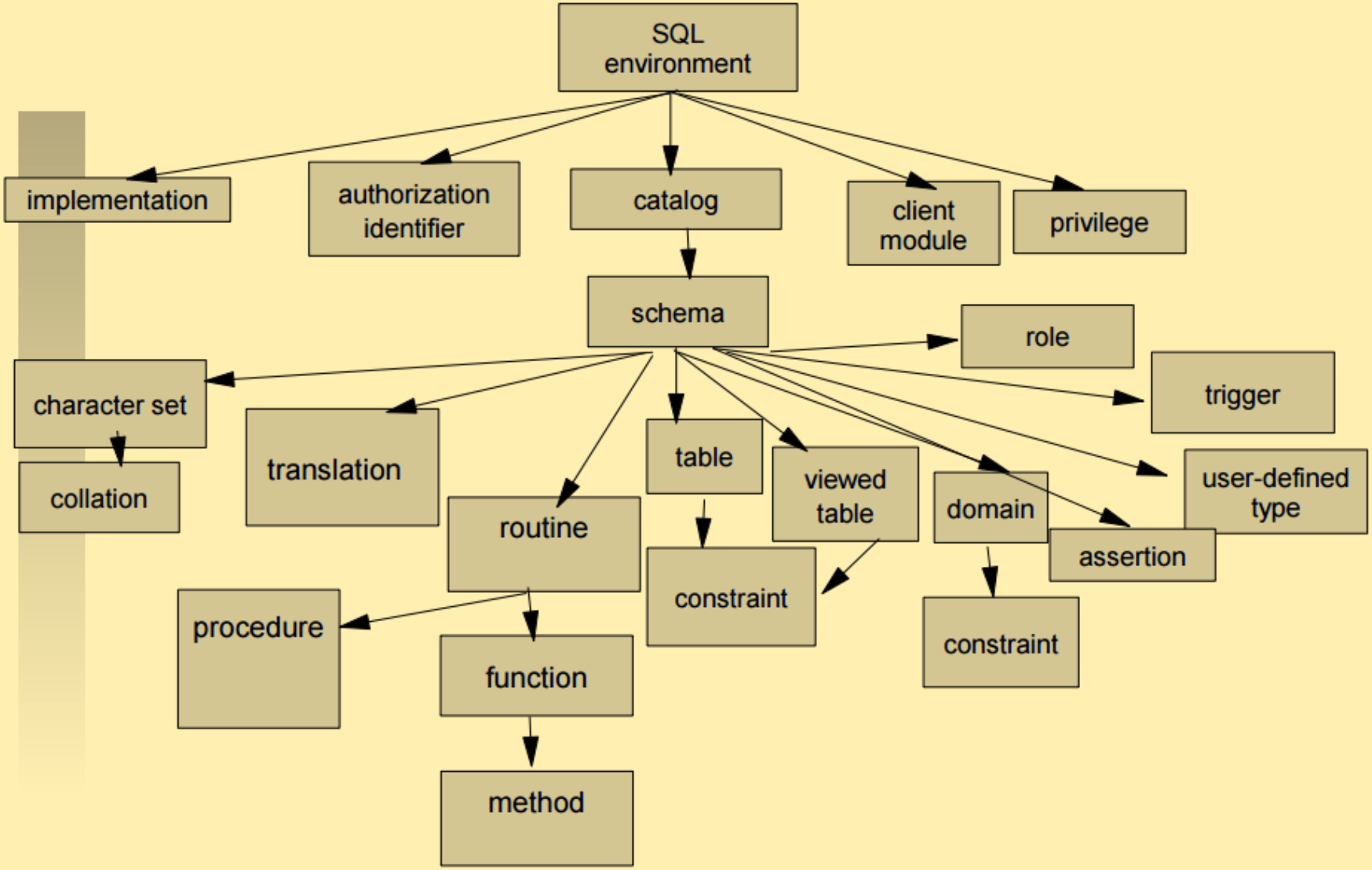
```
GRANT    ALL PRIVILEGES
ON       MyTable
TO       MyFriend WITH GRANT OPTION;

GRANT    SELECT, UPDATE(Grade)
ON       Exams
TO       Albano;

GRANT    SELECT
ON       Students
TO       PUBLIC;

REVOKE   SELECT
ON       Students
FROM     PUBLIC;
```

- Facilities for the DBA

SQL (Structured Query Language)

- First defined in 1974
- Standard (ANSI/ISO): SQL-84, SQL-89, SQL-92 (SQL2), **SQL:1999 (SQL3), SQL:2003 (4),** SQL:2006 (5), SQL:2008 (6), SQL:2011 (7), SQL:2016 (8)
- SQL-92: entry, intermediate and full SQL.
- SQL:1999: include GROUP BY ROLLUP, CUBE,
- SQL:2003: include analytic functions and windowing

SQL: Objects


SQL environment
implementation
authorization identifier
catalog
client module
privilege
schema
role
character set
trigger
translation
table
viewed table
user-defined type
collation
routine
domain
assertion
constraint
procedure
constraint
function
method

SQL: Data Definition Language

- Create/Alter/Drop Table/View

```
CREATE TABLE Students (
    Name            CHAR(20) NOT NULL,
    StudentCode     CHAR(8) NOT NULL,
    City            CHAR(20),
    BirthYear       INTEGER NOT NULL,
PRIMARY KEY         (StudentCode),
UNIQUE              (Name, BirthYear)
CHECK               (BirthYear > 1900));
```

```
CREATE TABLE ExamResults (
    Subject         CHAR(20) NOT NULL,
    Candidate       CHAR(8) NOT NULL,
    Date            CHAR(8) NOT NULL,
    Grade           INTEGER NOT NULL,
PRIMARY KEY         (Subject, Candidate),
FOREIGN KEY         (Candidate)
    REFERENCES      Students
    ON DELETE NO ACTION);
```

```
CREATE VIEW PisaStudents AS
        SELECT  Name, StudentCode, BirthYear
        FROM    Students
        WHERE   City = 'Pisa';
```

SQL: Data Manipulation Language

- Insert/Update/Delete

```
INSERT INTO   Students (Name, StudentCode, City, BirthYear)
              VALUES ('Rossi', '01234', 'Pisa', 1990);
```

```
UPDATE        Students
   SET        City = 'Florence'
   WHERE      StudentCode = '01234';
```

```
DELETE  FROM      Students
        WHERE     City = 'Pisa';
```

SQL: Data Query Language

SELECT **DISTINCT** *Attributes*
FROM *Tables*
WHERE *Condition*
ORDER BY *Attributes*;

where

Attributes ::= ∗ | *Attribute* {, *Attribute* }
Tables ::= *Table* [*Ide*] {, *Table* [*Ide*]}

(<subquery>)**UNION** [**ALL**] (<subquery>)
(<subquery>) **INTERSECT** [**ALL**](<subquery>)
(<subquery>) **EXCEPT** [**ALL**] (<subquery>)

FROM SQL (WHAT) TO ALGEBRA (HOW)

In SQL the tables of a database **may be without keys** and so they are not **sets** (**{T}**) but **multisets (bags)** (**{{T}}**). **To understand the semantics of an SQL query in terms of a relational algebra expression, the relational algebra is extended on multisets** using the following operators.

Project with duplicates. The result is a **multiset**. $\pi^b_{A_1,A_2,\ldots,A_n}(R)$

Duplicate elimination. The result is a **set**. $\delta(R)$

Sort. The result is a **list (seq T)**. $\tau_{A_1,A_2,\ldots,A_n}(R)$

The other operators of relational algebra extends naturally to multisets

Multiset union, intersection and difference. The result is **multiset**.

$$(R \cup^b S), (R \cap^b S), (R -^b S)$$

Multiset union, intersection and difference

If an element t appears n times in R and m times in S, then

t appears n + m times in the multiset **union** of R and S:

$$\{1,1,2,3\} \cup^b \{2,2,3,4\} = \{1,1,2,3,2,2,3,4\}$$

t appears **min**(n, m) times in the multiset **intersection** of R and S:

$$\{1,1,2,3\} \cap^b \{2,2,3,4\} = \{2,3\}$$

t appears **max**(0, n – m) times in the multiset **difference** of R and S:

$$\{1,1,2,3\} -^b \{1,2,3,4\} = \{1\}$$

FROM SQL (WHAT) TO ALGEBRA (HOW)

SELECT **DISTINCT** S_A, S_{AF}
FROM T
WHERE W_C
GROUP BY G_A
HAVING H_C
ORDER BY O_A;

Some clauses are optional

The clauses **HAVING** and **SELECT** use only:

- expr on grouping attributes

  i.e., $(S_A \subseteq G_A)$

- aggregation functions S_{AF} and H_{AF} (used in H_C) over non- grouping attributes.

(a) *SQL query* (b) *Logical query plan*

FROM SQL (WHAT) TO ALGEBRA (HOW)

SELECT **DISTINCT** S_A, S_{AF}
FROM T
WHERE W_C
GROUP BY G_A
HAVING H_C
ORDER BY O_A;

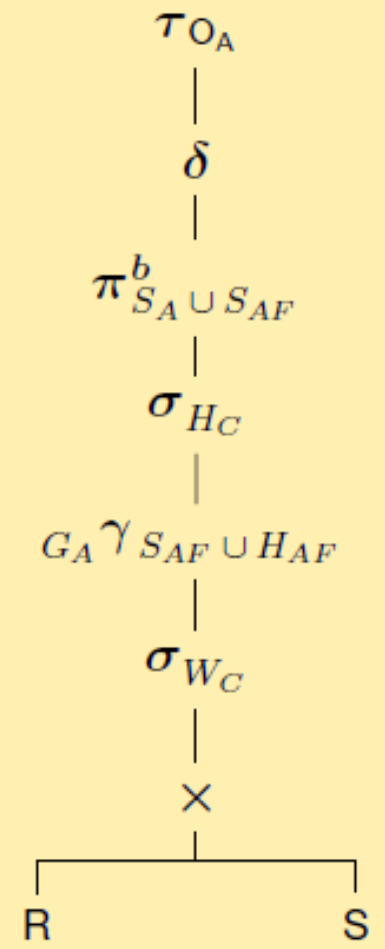
Some clauses are optional

The clauses **HAVING** and **SELECT** use only:

- expr on grouping attributes
  i.e., $(S_A \subseteq G_A)$
- aggregation functions S_{AF} and H_{AF} (used in H_C) over non- grouping attributes.

ORDER BY O_A
DISTINCT
SELECT S_A, S_{AF}
HAVING H_C
GROUP BY G_A
WHERE W_C
FROM R, S

(a) *SQL query*



(b) *Logical query plan*

The COUNT bug of SQL: without GROUP BY vs GROUP BY ()

```
SELECT Count(*)
FROM R
```

vs

```
SELECT Count(*)
FROM R
GROUP BY ()
```

$$\varnothing\gamma_{\text{Count}(*)}$$

|

R

Same result when R is non-empty.

What is the result if R is empty?

SQL: WITH Clause (subquery factoring)

- Simplify complex SQL queries, prevent using temporary views/table
  - WITH subquery_name AS
    (
    SQL query defining subquery
)
    SQL query using subquery_name as a table name

- Exercise: Average number of students per year that passed 'BSD'

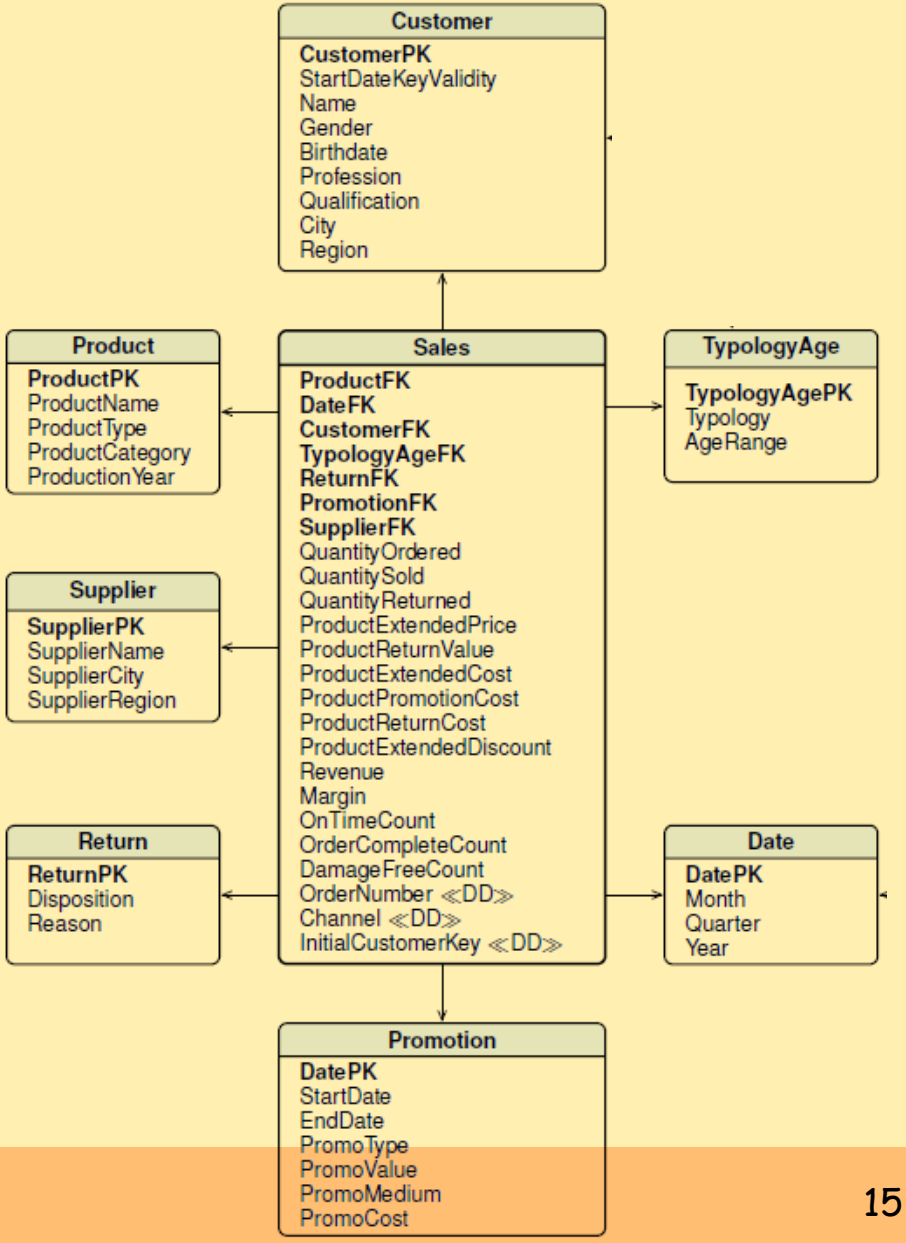
```
WITH agg AS
      (SELECT Count(*) As N
      FROM ExamResults
      WHERE Subject='BSD'
      GROUP BY Year(Date))
SELECT Avg(N)
FROM agg
```

```
CREATE TABLE ExamResults (
    Subject          CHAR(20) NOT NULL,
    Candidate        CHAR(8) NOT NULL,
    Date             CHAR(8) NOT NULL,
    Grade            INTEGER NOT NULL,
PRIMARY KEY          (Subject, Candidate),
FOREIGN KEY          (Candidate)
    REFERENCES       Students
    ON DELETE NO ACTION);
```

EXERCISE AT HOME FROM A PREVIOUS LESSON

- Write a SQL query that returns all constant customers

- **Constant**: with at least two orders per month for at least three months in the last four months.



EXERCISE AT HOME – SOLUTION

Assuming that a month is a number in the format MM (hence **Month -> Year NOT holds**)

```
WITH NOrders AS (
    SELECT InitialCustomerKey, COUNT(DISTINCT OrderNumber) AS norders
    FROM Sales, Date
    WHERE DateFK = DataPK
    WHERE Year*12+Month BETWEEN f_lastMonth-3 AND f_lastMonth
    GROUP BY InitialCustomerKey, Month, Year
)
SELECT InitialCustomerKey
FROM NOrders
WHERE norders > 1
GROUP BY InitialCustomerKey
HAVING COUNT(*) > 2
```

EXERCISE AT HOME – SOLUTION

Assuming that a month is a number in the format YYYYMM (hence **Month -> Year holds**)

Let f(YYYYMM) = YYYY*12 + MM E.g., f(n) = (n / 100)*12 + n % 100

Let f_lastMonth = f(lastMonth) E.g., f_lastMonth = f(202410) = 24298

```
WITH NOrders AS (
    SELECT InitialCustomerKey, COUNT(DISTINCT OrderNumber) AS norders
    FROM Sales, Date
    WHERE DateFK = DataPK
    WHERE (Month/100)*12+Month%100 BETWEEN f_lastMonth-3 AND f_lastMonth
    GROUP BY InitialCustomerKey, Month
)
SELECT InitialCustomerKey
FROM NOrders
WHERE norders > 1
GROUP BY InitialCustomerKey
HAVING COUNT(*) > 2
```

SQL: Nested Queries

- Student code and name who passed at least one exam with grade 'A'

```
SELECT StudentCode, Name
FROM Students
WHERE StudentCode IN (SELECT Candidate
                      FROM ExamResults
                      WHERE Grade='A')
```

```
CREATE TABLE Students (
    Name          CHAR(20) NOT NULL,
    StudentCode   CHAR(8) NOT NULL,
    City          CHAR(20),
    BirthYear     INTEGER NOT NULL,
PRIMARY KEY       (StudentCode),
UNIQUE            (Name, BirthYear)
CHECK             (BirthYear > 1900));
```

- Student code and name who did not passed any exam

```
SELECT StudentCode, Name
FROM Students
WHERE StudentCode NOT IN (SELECT Candidate
                          FROM ExamResults)
```

```
CREATE TABLE ExamResults (
    Subject       CHAR(20) NOT NULL,
    Candidate     CHAR(8) NOT NULL,
    Date          CHAR(8) NOT NULL,
    Grade         INTEGER NOT NULL,
PRIMARY KEY       (Subject, Candidate),
FOREIGN KEY       (Candidate)
    REFERENCES    Students
    ON DELETE NO ACTION);
```

SQL: NULLs

- Missing or unknown values of attributes are modelled with the NULL value
- Problems introduced by the NULL value:
  - Test whether a value is NULL: WHERE age IS [NOT] NULL
  - Truth value of: age > 25 when age is NULL?
    - Three-valued logic

x	y	x AND y	x OR y	NOT x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Figure 1: Truth table for three-valued logic

  - Which tuples satisfy WHERE C? those where C evaluates to TRUE

SQL: NULLs

- Features introduced by the NULL value:
  - New join operator: $R(A_1, \ldots, A_n)$ $S(B_1, \ldots, B_m)$ relations!
    - R **LEFT OUTER JOIN** S ON $R.A_i = S.B_j$

Tuples with no match

$$(R \bowtie_{R.Ai=S.Bj} S) \cup [\ (\ R -^{b} \pi^{b}_{A1, \ldots, An}\ (\ R \bowtie_{R.Ai=S.Bj} S)) \times \{\ \{B_1{:}NULL, \ldots, B_m{:}NULL\}\ \}\ ]$$

R

Name	StudentCode
Mario	1
Lucia	2
Anna	3

Subject	Candidate	Grade
BSD	1	A
DM1	2	B
BSD	2	B

S

Name	StudentCode	Subject	Candidate	Grade
Mario	1	BSD	1	A
Lucia	2	DM1	2	B
Lucia	2	BSD	2	B
Anna	3	NULL	NULL	NULL

  - Others: RIGHT OUTER JOIN, FULL OUTER JOIN

SQL: CASE

R

Name	Gender	StudentCode
Mario	M	1
Lucia	F	2
Anna	F	3

Subject	Candidate	Grade
BSD	1	A
DM1	2	B
BSD	2	B

S

- SQL to compute

Subject	NExamsF	NExamsM
BSD	1	1
DM1	1	0

```
SELECT Subject, Gender, Count(*)
FROM R, S
WHERE StudentCode = Candidate
GROUP BY Subject, Gender
```

SQL: CASE

R

Name	Gender	StudentCode
Mario	M	1
Lucia	F	2
Anna	F	3

Subject	Candidate	Grade
BSD	1	A
DM1	2	B
BSD	2	B

S

- SQL to compute

Subject	NExamsF	NExamsM
BSD	1	1
DM1	1	0

```
SELECT Subject, SUM( CASE WHEN Gender='F' THEN 1 ELSE 0 END ) As NExamsF,
                SUM( CASE WHEN Gender='M' THEN 1 ELSE 0 END ) As NExamsM
FROM R, S
WHERE StudentCode = Candidate
GROUP BY Subject
```

TEST

1. **(5 points) (Mandatory)** Let us consider the following database, without null values:

Products

PkP	UnitPrice	...
10	5	...
20	10	...
30	20	...

Sales

FkP	Qty	...
10	50	...
20	10	...
30	20	...
10	30	...
20	100	...
30	10	...
10	30	...

```
SELECT FkP, SUM(Qty*UnitPrice)
FROM Sales, Products
WHERE FkP = PkP
GROUP BY FkP
```

```
SELECT FkP, SUM(Qty*UnitPrice)
FROM Sales, Products
WHERE FkP = PkP AND UnitPrice > 5
GROUP BY FkP
HAVING COUNT(*)>5
```

(a) Write an SQL query to find the total sales revenue by product.

(b) Give a logical query plan for the SQL query, the type and the value of the result.
Modify the logical query plan to consider only products with UnitPrice > 5 sold each of them more than 5 times.