Curs 1+2

## The Relational Model

Relational Database = a collection of relations and logical links between them

Relation(TABLE) = SET of records

- → schema : name, attributes (columns/fields), attributes types
  - student (sid: String, name: String, year: Integer)
- → relational instance : physical table (ex: on disk)
- → grade = nr. of columns - fixed nr. of ordered attributes
- → cardinality = nr. of rows - variable nr. of unordered distinct records

Design

① Requirements analysis : data identification

② Result : a semantic model of data

- → abstract model describing the semantic of data (orthogonal to implementation)
- → used to understand data role and constraints

③ Modeling languages

- → Entity Relationship (ER) - graphical language describing entities and relationships (logical model)
- → UML - graphical language, more general than ER

Levels of abstraction
- → physical - how the data is stored and where it is stored in database
- → conceptual = describes the model of data → data application
- → external = simplified domain-specific views

A database instance that satisfies all integrity constraints is called legal ("in a consistent state"). Verification of consistency is performed by DBMS

! A set of attributes forms a key for a relation if

- → there is no pair of tuples that have equal values for all attributes in the set (ensures uniqueness)
- → any subset of it breaks the above property (must be minimal)

Ensuring referential integrity

① for INSERT → block the addition of a record having a wrong reference (FK without corresponding PK)

② for DELETE
- → cascade delete related records
- → avoiding operation that breaks constraints
- → voiding the reference

③ for UPDATE
- → cascade update related records
- → avoiding operation that breaks constraints

## Functions

- abs(n) - absolute value
- concat(str1, str2, ..) - concatenates string
- round(n)
- trunc(n)
- lower(str) -> the argument in lowercase
- substr ( substr, str, [s_pos] ) - position of the first occurrence of substring
- trim(str) - removes leading and trailing spaces
- length(str)
- CURRENT_DATE

  SELECT CURRENT-DATE FROM Dual
- EXTRACT (spec, d) -> extracts a part of a date

  spec ∈ { year, month, day, hour, minute, second }

## Ordering the result

```
SELECT col1, col2, ..
FROM table
ORDER BY coli, colj ASC/DESC
```

ex:
```
SELECT * FROM Sailor
ORDER BY rank, age DESC;
```

## Cartesian product

```
SELECT t1.*, t2.*
FROM table1 t1, table2 t2;
```

ex:
```
SELECT s.*, r.*
FROM Sailor s, Reserves r;
```

## SQL - Join

```
SELECT t1.col1, ..., t2.col1, ...
FROM table t1, table t2, ..
WHERE t1.columni = t2.columnj
```

- inner join => returns all rows where there is at least one match in both tables
- left join => returns all rows from the left table and the matched rows from the right table and NULL for missing corresponding fields
- right join => returns all rows from the right table and the matched rows from the left table and NULL for missing corresponding fields
- full join => returns all rows from both tables (LEFT ∪ RIGHT)

```
SELECT t.name, f.*
from teacher t JOIN faculty f ON t.fid = f.fid
```

```
SELECT s.name, s.yos
from student s LEFT JOIN contract c ON s.sid = c.sid
WHERE c.cno is NULL
ORDER BY s.yos
```

## Subqueries

→ can't manipulate their results internally ⇒ a subquery can NOT include the ORDER BY clause

Ex: select s.name from sailor s
where s.sid NOT IN (select r.sid from reserves r where r.bid=103)



## Aggregate functions

→ operates on a single column or expression from a group of rows
↳ return a single value for the group of rows
↳ used ONLY in the projection list, subqueries and in the having (but NOT in WHERE)

**Count:** select count(distinct CustomerID) as NoOfCst
from Invoice

count(*) → counts all the rows regardless of whether Nulls or duplicates occur

**Sum:** select sum(Salary) as SalaryBudget
from Employee
where department_id=101

**Group by** select dep-id as Department,
MAX(salary) as MaxDepSalary
from Employee
group by dep-id

**Having** select rank, COUNT(*) as NrSail
from Sailor
group by rank having rank>3

Part II
curs 8

DB developments steps:
① Requirement analysis
② Conceptual design – ER model
③ Logical DB design – relational model
④ DB Schema refinement – normalization
⑤ Physical DB design – indexes etc
⑥ Client application design
⑦ Application implementation
⑧ DB deployment

Data base related models
→ conceptual models: used to analyse and understand application data (ER standard)
→ data models: used to describe data (ex: a schema)
→ physical models: representation of a data design which considers the facilities and constraints of a given DBMS

Conceptual model = a high level description of a business informational needs
→ identifies the general relationships between the different entities
Characteristics → includes info. of all important entities and the relationships amongst them
→ no data organisation is specified
→ just some constraints are specified

The ER Model
Entity → real world entity (object) distinguishable from other entities
→ described using a set of attributes

Entity set = a collection of similar data types
• representation: a rectangle (for entity)
• all entities in an entity set have the same set of attributes
• each entity set has a key
Relationship = association among two or more entities
Relationship set = collection of similar relationships
representation of a relation: a diamond

Participation constrains

Partial participation in the relationship => thin line (just some)

Total participation => thick line (all)

Weak entity = entity that can be identified uniquely only by considering the primary key of another (owner) entity

Owner entity set – weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)

Weak entities MUST have total participation

is-A hierchies => triangle

A ternary relationship could always be replaced by an entity (through a "verb to noun" transformatio

The relational data model

Relational database = a set of relations

A relation is described by
- -> instance : a tables with rows and columns (rows: cardinality, fields/columns = degree)
- -> schema : specifies name of relation + name and type of each attribute

Foreign key = set of fields in one relation that is used to refer to a tuple in another relation

-> must correspond to primary key of the second relation

Transforming ER into relational model

- -> each entity will be converted directly to a relation
- -> attributes of the entity become the attributes of the relation
- -> identifier of the entity becomes a key in the relation
- -> relationship will be mapped on relations or as Foreign keys

Curs 9

Relational model: redundancy
- → waste of storage
- → insert/delete/update anomalies

Functional dependencies (FD) can be used to identify schemas with such problems and to suggest refinaments

Main refinament technique: decomposition (replacing ABCD with AB and BCD or ACD and ABD)

A FD holds over relationship R if for every allowable instance r of R
(X → Y)

$t_1 \in R, t_2 \in R$

$\Pi_X(t_1) = \Pi_X(t_2) \Rightarrow \Pi_Y(t_1) = \Pi_Y(t_2)$

<=> given 2 tuples in R, if the X values agree, then the Y values must also agree,
X, Y - sets of attributes

An FD is a statement about all allowable relations

Armstrong's Axioms:
- → reflexivity: $X \subseteq Y \Rightarrow Y \to X$
- → augmentation: if $X \to Y$ then $XZ \to YZ$ for any Z
- → transitivity: if $X \to Y$ and $Y \to Z$ then $X \to Z$

$F^+$ (closure of F) is the set of all FDs that are implied by F

Attribute closure:

We want to check if a given FD, $X \to Y$, is in the closure of a set of FDs F:
- → compute attributes closure of X (denoted $X^+$) = set of all attributes A s.t. $X \to A$ is in $F^+$
- → check if Y is in $X^+$

Relational scheme decomposition
- → each new relational scheme contains a subset of the attributes of R (fără alte atribute!)
- → every attribute of R appears as an attribute of one of the new relation

! It is essential that all decomposition used to deal with redundancy be lossless!

$(F_1 \cup F_2)^+ = F^+$

Tot curs 9

First Normal Form (1NF)

→ the domain of each attributes must contain ONLY atomic values (doar o valoare) de ex:

de ex: Adresa: Timisoara, Strada Ceva se sparge în Adresa_localitate: Timisoara și Adresa_strada: Strada Ceva

→ each attributes contains a single value for that domain (doar o coloană se repetă în cele 2 tabele)

Second Normal Form (2NF)

→ relation is already in 1NF

→ any non-prime attributes of R (not part of the primary key) must be fully functionally dependent on the primary key of R

(dacă avem o coloană care-și păstrează valoare neschimbată indiferent de PK spargem tabelul în 2 tabele, primul doar cu PK și valoarea neschimbată și celălalt cu PK și coloanele care depind de PK)

→ OR: there are no attributes that depend only on a part of the PK

Third Normal Form (3NF)

→ relation is already in 2NF

→ no transitive dependency is allowed

Fourth Normal Form (4NF)

→ relation is already in 3NF

→ there are no multivalued dependencies

||

If to each A corresponds many B and many C but B and C are independent of each other

Curs 10

Data on external storage
→ RAM
→ HDD (SSD)
→ Tape / DVD

Data file = a sequence of records
→ records are mapped on disk's sectors
→ Variable or fixed lenght records

Index classification
→ primary ⇒ if search key contains primary key
→ secundary

→ clustered ⇒ order of data records is the same as order of data entries
→ unclustered

Hash - based index
→ good for equality selections
→ index is a collection of buckets
→ bucket = primary page plus zero or more overflow pages
→ hashing function: $h(r)$ = bucket in which record $r$ belongs
→ $h$ directs the search for indexing key

$h(key) = a * key + b$   a,b - constants → Static hashing

$h(k) \bmod N$ = bucket to which data entry with key $k$ belongs
↓
nr. of buckets

} Static hashing

Extensible hashing
→ global depth of directory = max nr. of bits needed to tell which bucket an entry belongs to
→ local depth of a bucket = nr. of bits used to determine if an entry belongs to this bucket

Before insert local depth = global depth, after ⇒ local > global

A tree index
→ nu trebuie inițializați toți indecșii dar trebuie luați în ordine (nu poți spune $x_3 = 7$ înainte să zici ceva de $x_1$ și $x_2$)

Hash index
→ fiecare indice trebuie inițializat și fiecare trebuie să aibă o valoare distinctă

2 tabele sunt corelate dacă:
(cea de-a 2 se bazează pe prima)

select
from Student s1
where

select from Student s2 where s2.ceva = s1.ceva