Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Subject Summary

Jordi Ricard Onrubia Palacios

Departament d'Informàtica i Enginyeria Industrial Universitat de Lleida

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Data

- ➢ Data
- ❑ Databases
- ❑ Relational Databases
- ❑ SQL
- ❑ SQL Scripts

What Is a Data?

Data is a collection of facts, numbers, words, observations or other useful information. Through data processing and data analysis, organizations transform raw data points into valuable insights that improve decision-making and drive better business outcomes.

- Raw and unprocessed: Data is often in its original form, without any context or interpretation. For example, the number "25" by itself is data. It could represent age, temperature, or something else entirely.
- Organized: Data can be structured in various ways, such as tables, lists, or files, to make it easier to store and use.
- Expanded: When data is processed, organized, and given context, it becomes information. For example, if we know that "25" is the age of a person named "John," then we have information.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Databases

- Data
- Databases
- Relational Databases
- SQL
- SQL Scripts

What Is a Database?

A database is any collection of data items. More technically we can describe it as a collection of integrated records.

A record is a representation of some physical or conceptual object. For example, A company storing the data of they employees. Each record will have multiple attributes, following the previous example, Name, Telephone, Directions, etc.

These records can be stored in different ways depending on the type of database we are using, if we use Relational Databases, specifically SQL Databases, we will use tables and create relations between them.

Types of Databases

- Relational Databases: These are the most common type and organize data into tables with rows (records) and columns (fields). Relationships between tables are established using keys. They are excellent for structured data and applications requiring data integrity and consistency.
- NoSQL Databases: These databases are designed to handle large volumes of unstructured or semi-structured data. They offer more flexibility in data modeling compared to relational databases and are well-suited for applications with high scalability and availability requirements. There are several types of NoSQL databases:
    - Document Databases: Store data in JSON-like documents, making them flexible and easy to use for web applications.
    - Key-Value Stores: Store data as key-value pairs, providing fast read and write operations.
    - Column-Family Stores: Organize data into columns grouped into families, offering high performance for analytical queries.
    - Graph Databases: Use graph structures with nodes and edges to represent and store data, making them ideal for social networks, recommendation engines, and other applications with complex relationships.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: Relational Databases

- ❑ Data
- ❑ Databases
- ➢ Relational Databases
- ❑ SQL
- ❑ SQL Scripts

Relational Databases

A relational database is a type of database that organizes data into one or more tables (or "relations") of rows and columns, with relationships established between these tables. This structure makes it easy to understand how different pieces of data connect.

- Tables: These are the fundamental units of a relational database. Think of them as grids with rows and columns. Each table stores data about a specific type of entity (e.g., customers, products, orders).
- Rows (Records): Each row in a table represents a single instance of that entity. For example, in a "Customers" table, each row would represent a different customer.
- Columns (Fields or Attributes): Each column in a table represents a specific attribute or characteristic of the entity. For example, in a "Customers" table, columns might include "CustomerID," "Name," "Address," and "Phone Number."
- Relationships: These are the connections between tables. Relationships are established using "keys":
    - Primary Key: A unique identifier for each row in a table. For example, "CustomerID" could be the primary key in the "Customers" table.
    - Foreign Key: A field in one table that refers to the primary key of another table. This establishes a link between the two tables. For example, an "Orders" table might have a "CustomerID" column (foreign key) that links each order to the customer who placed it.

Relational Databases: Advantages

1. Data Integrity and Consistency:

- ACID Properties: SQL databases adhere to ACID properties (Atomicity, Consistency, Isolation, Durability) which guarantee reliable transaction processing. This ensures that data remains consistent and accurate even in the event of system failures or concurrent access.
    - Atomicity: Each transaction is treated as a single, indivisible unit of work. Either all changes within the transaction are successfully committed (applied to the database), or none of them are. There's no in-between state.
    - Consistency: A transaction must maintain the database's integrity constraints. It ensures that the database starts in a consistent state and ends in a consistent state after the transaction is completed. This means that any rules or constraints defined in the database (e.g., data types, unique keys, referential integrity) must be upheld.
    - Isolation: Concurrent transactions do not interfere with each other. Each transaction operates as if it were the only transaction running on the database, even if there are multiple transactions happening simultaneously.
    - Durability: Once a transaction is committed, the changes are permanent and will survive even system failures. The data is typically written to persistent storage to ensure this.
- Constraints: SQL allows you to define constraints (e.g., primary keys, foreign keys, unique constraints, check constraints) that enforce rules about the data stored in the database. This helps prevent data inconsistencies and errors.

Relational Databases: Advantages

2. Structured Data and Relationships:

- Organized Data: Data is stored in a structured format within tables, making it easy to understand, manage, and query.
- Relationships between Tables: The ability to define relationships between tables using keys allows you to efficiently store and retrieve related data without redundancy.

3. Powerful Querying Capabilities:

- SQL Language: SQL is a powerful and standardized language specifically designed for interacting with relational databases. It allows you to perform complex queries to retrieve, filter, sort, and aggregate data with ease.
- Optimized Performance: SQL databases are often optimized for query performance, allowing for efficient retrieval of large datasets.

Relational Databases: Advantages

4. Mature Technology and Wide Support:

- Long History: Relational databases have been around for decades, making them a mature and well-understood technology.
- Large Community and Ecosystem: There is a large community of developers and database administrators with expertise in SQL and relational databases. This translates to ample resources, tools, and support.
- Wide Availability: Many different SQL database systems are available, both commercial (e.g., Oracle, Microsoft SQL Server) and open-source (e.g., MySQL, PostgreSQL), offering a range of options to suit different needs and budgets.

5. Security:

- Access Control: SQL databases provide mechanisms for controlling user access and permissions, ensuring that sensitive data is protected.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: SQL

- ❏ Data
- ❏ Databases
- ❏ Relational Databases
- ➢ SQL
- ❏ SQL Scripts

What is SQL?

SQL stands for Structured Query Language. It's a powerful and widely used programming language designed for managing and manipulating data in relational database management systems (RDBMS).

It provides instructions for:

- Data Definition Language (DDL), creating databases and tables, modifying database structures, deleting databases and tables
- Data Manipulation Language (DML), retrieving, inserting and updating data
- Data Control Language (DCL), managing user access.

SQL: Advantages

- Standardized: Although there are some variations between different database systems, SQL is largely standardized, meaning that the basic syntax is the same across most systems.
- Declarative: SQL is a declarative language, meaning you specify what data you want to retrieve, not how to retrieve it. The database system is responsible for optimizing the query execution.
- Powerful: SQL provides a wide range of functions and operators that allow you to perform complex data manipulations and analysis.

SQL: Data Definition Language

Constraints are rules we apply to a table to define what types we can insert in a column or a table, if can be missing values, if its a primary or a foreign key…

- Not null: The specified column must not contain empty values
- Primary key: The specified column/s act as an identifier for the table
- Foreign key: The specified column/s act as an identifier for another table
- Check: Define a range of values for a column
- Default: Specifies a default value in case none is applied

SQL: Data Definition Language

Databases can store a multiple types of values, the most common ones are the following:

- Numerics: Integer, Float, Double, Numeric, Decimal (The 2 latest allows us to specify the length of the number to store)
- Alphanumerics: Char (to specify strings of length n), Varchar (to specify strings of max length n), Text (to store strings without caring about the length)
- Dates: Date (day, month and year), Datetime (same as date but with hour, minutes and seconds too)
- Logics: Bit, and Boolean (1,0, True, False)

SQL: Data Manipulation Language

Data Manipulation Language (DMA) is in charge of the data manipulation, select it, insert it, modify it and remove it.

- SELECT - extracts data from a database
- UPDATE - updates data in a database
- DELETE - deletes data from a database
- INSERT INTO - inserts new data into a database

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Programming and Communications III: SQL Scripts

- ❑ Data
- ❑ Databases
- ❑ Relational Databases
- ❑ SQL
- ➢ SQL Scripts

SQL: Table Creation

- To create tables the CREATE TABLE instruction is used with the desired name for the table
- As a parameters the instruction expect coma separated arguments were each argument is the name of the column, the type of the column and a constraint if required

```
CREATE TABLE "TABLE_NAME" (
    "COL1" type contstraint,
    "COL2" type contstraint,
    "COL3" type contstraint,
);
```

SQL: Table Creation

```sql
CREATE TABLE "employee" (
    "id" serial primary key not null,
    "first_name" varchar(255) not null,
    "last_name" varchar(255) not null
);
```

```sql
CREATE TABLE "call" (
    "id" serial primary key not null,
    "employee_id" int REFERENCES "employee"("id"),
    "customer_id" int REFERENCES "customer"("id"),
    "start_time" date,
    "end_time" date,
    "call_outcome_id" int
);
```

```sql
CREATE TABLE "customer" (
    "id" serial primary key not null,
    "customer_name" varchar(255) not null,
    "city_id" int not null,
    "customer_address" varchar(255) not null,
    "next_call_date" date,
    "ts_inserted" date
);
```

SQL: Table Modification

- To modify tables the ALTER TABLE instruction is used with the desired name for the table to modify
- The next instruction must be the type of alteration that must be applied to the table

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name constraint_conditions;
```

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

```
ALTER TABLE table_name
ADD column_name datatype;
```

```
ALTER TABLE table_name
RENAME COLUMN old_name to new_name;
```

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

SQL: Table Modification

```sql
CREATE TABLE "city" (
    "id" serial primary key not null,
    "name" varchar(255) not null
);
```

```sql
ALTER TABLE "customer"
ADD CONSTRAINT "FK_CityCustomer"
FOREIGN KEY ("city_id") REFERENCES "city"("id");
```

```sql
ALTER TABLE "city"
ADD COLUMN "country_code" VARCHAR(5) DEFAULT "ES";
```

SQL: Table Deletion

- To delete tables the DROP TABLE instruction is used with the desired name for the table to delete
- The DROP TABLE statement can include CASCADE at the end, this is used when the table has dependencies with other tables like the FOREIGN KEY we have defined previously

```
DROP TABLE table_name CASCADE;
```

```
DROP TABLE "city" CASCADE;
```

SQL: Insert Data

- To insert data the statement INSERT INTO can be used followed by the name of the table
- This function expects two things
  - the first one is the name of the columns we will add every value, if all the values are in the same order of the columns at the time of the creation we can avoid this step
  - The second one are the values always starting with the keyword VALUES and each value we want to insert on each column

```
INSERT INTO table_name (column1, column2, column3, ... )
VALUES (value1, value2, value3, ... );
```

SQL: Insert Data

```sql
INSERT INTO "employee" ("first_name", "last_name") VALUES ('Victor', 'Altés Gaspar');
INSERT INTO "employee" ("first_name", "last_name") VALUES ('Salma', 'Assiad Sebai');

INSERT INTO "customer" ("customer_name", "city_id", "customer_address", "next_call_date", "ts_inserted")
VALUES ('Abraham Castro Criado', 1, 'Calle de la Piruleta', '2022-12-01','2022-12-01');
INSERT INTO "customer" ("customer_name", "city_id", "customer_address", "next_call_date", "ts_inserted")
VALUES ('Pelayo Cobos Rodriguez', 1, 'Salchichon', '2022-05-01', '2022-05-01');


INSERT INTO "call" ("employee_id", "customer_id", "start_time", "end_time", "call_outcome_id")
VALUES (2, 4, '2022-08-01 11:43:06', '2022-08-01 11:55:06', 0);
INSERT INTO "call" ("employee_id", "customer_id", "start_time", "end_time", "call_outcome_id")
VALUES (1, 5, '2022-07-01 04:22:06', '2022-07-01 05:22:06', 0);

INSERT INTO "city" ("name") VALUES ('Lleida');
INSERT INTO "city" ("name") VALUES ('Murcia');
INSERT INTO "city" ("name") VALUES ('Barcelona');
```

SQL: Delete Data

- To delete data the statement DELETE FROM can be used followed by the name of the table followed by a WHERE statement, if the WHERE statement is not applied it will remove all the data from that table

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

```
DELETE FROM "call" WHERE "employee_id" = 1;
```

SQL: Update Data

- To update data the statement UPDATE table_name
- The SET statement follows to modify the value for each column in the form of colum_name = new_value
- A WHERE statement should be added to modify the value only for the required records

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE "customer"
SET "customer_address" = 'modified_address'
WHERE id=1;
```

SQL: Select Data

- To select the data we have already stored the SELECT statement can be used followed by the columns we want to retrieve or * to retrieve all the columns inside the table
- The statement FROM must be added followed by the required table
- A WHERE statement can be added to retrieve data following an specific condition
- Moreover, some other statements can be applied to queries, ORDER BY to order the output by the value of a column in descending or ascending order, GROUP BY to group data, nested queries, different types of JOINS …
- For time reasons will keep it simple in this subject and we will only see ORDER BY and INNER JOINS

SQL: Select Data

```sql
SELECT *
FROM "call"
WHERE employee_id=1
ORDER BY id;
```

```sql
SELECT col_name1, col_name2, ...
FROM table_name
WHERE condition
ORDER BY col_name
```

SQL: Select Data Joining Tables

- INNER JOIN in SQL is used to combine rows from two or more tables based on a related column between them. It returns only the rows where the join condition is met in both tables. It finds the intersection of the data in the tables being joined.
- Requires a join condition, which specifies how the tables are related. This condition typically compares a column in one table with a column in another table using an equality operator (=).
- The join condition is started by the statement INNER JOIN (also JOIN in almost all implementations will use INNER JOIN although the INNER is not specified) followed by the table_name of the table required
- The next required statement is the ON followed by the the condition table1.column = table2.column
- This makes total sense with the concept of FOREIGN KEY and PRIMARY KEY, those are usually used for INNER JOINS

SQL: Select Data Joining Tables

```sql
SELECT *
FROM "call"
INNER JOIN "customer"
ON "call".customer_id = "customer".id
WHERE "call".customer_id = 1;
```

```sql
SELECT column_list
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

SQL Scripts: Links

Create Tables: Link
Modify Tables: Link
Drop Tables: Link
Insert Data: Link
Delete Tables: Link
Update Tables: Link
Select Tables: Link