Database Security

An Introduction

PRESENTATION BY:
João Luís (PDMFC)

Table of Contents (1/3)

Table of Contents (2/3)

Table of Contents (3/3)

Intended Audience

- Application Developers and Architects
- Database, System, and Network administrators
- (Data) Analysts

"To learn how to find, one must first learn how to hide!"

Spoken in “Fahrenheit 451” (1966), directed by François Truffaut

High-Level Objectives

Principles

Description of objectives in human language, written for humans!

(Based on social science's studies)

Why do we care about "security" ?

Privacy, Data Confidentiality, and Trust!

“Customer trust functions directly as a direct function of their privacy rights. Anxiety about data handling by collectors leads customers to lose trust which might reduce their business involvement. For increasing customer trust businesses should establish proactive precautions to protect their data by providing clear collection policies and strong security protocols. These security practices help build consumer trust which generates better transaction willingness and business accomplishments.“

“The Relationship between Data Privacy and Consumer Trust”,
July 2025 SHS Web of Conferences 218
Authors: Yijia Luo

G.D.P.R. in a nutshell (1/4)

General Data Protection Regulation (**GDPR**) is an EU law that establishes rules for how organizations collect, use, and store the personal data of people in the EU and EEA.

It requires companies to get explicit consent for data use, gives individuals rights over their data (like the right to access or delete it), and imposes strict security and transparency obligations. The goal is to protect individuals' privacy by making companies accountable for how they handle personal information.

G.D.P.R. in a nutshell (2/4)

For individuals (data subjects):

Right to be informed: You must be told how your data is being used.

Right to access: You can request to see what data a company has collected about you.

Right to erasure: You can request that a company delete your data.

Right to rectification: You can ask for inaccurate data to be corrected.

G.D.P.R. in a nutshell (3/4)

For companies (data controllers and processors):

Consent: Must obtain clear consent to collect and process data, with a few exceptions.

Transparency: Must state (openly) about what data it collects and why.

Data minimization: Should only collect the data needed for a specific purpose.

Security: Must implement strong security measures to protect the data.

Accountability: Must be able to demonstrate compliance with the rules.

G.D.P.R. in a nutshell (4/4)

What is **personal data**?

Any information that can directly or indirectly identify a living person.
This includes names, phone numbers, and addresses, but also data like interests, purchase history, and online behavior.

Techniques to enforce data confidentiality?

Access controls: Implement Role-Based Access Control (RBAC) to ensure employees only have access to the data necessary for their job functions.

Encryption: Use encryption to scramble data, making it unreadable to anyone without the correct decryption key. This applies to data both at rest (stored) and in transit (being sent).

Transparency and consent: Inform individuals about how their data will be used at the point of collection and obtain their explicit consent when possible.

Data anonymization and masking: Remove or obscure personally identifiable information from datasets for analytics or testing purposes.

Employee training: Regularly train employees on the importance of data confidentiality, security policies, and how to handle sensitive information responsibly.

Privacy-by-design: Integrate privacy and security measures into the design of new systems, services, and products from the outset, rather than as an afterthought.

The Role of Database Management Systems on Business Developments

“Databases are essential tools for businesses, serving as centralized repositories for data management. They enable organizations to store, retrieve, and manipulate data efficiently, which is crucial for driving business development initiatives. In a competitive landscape, leveraging data effectively can lead to significant growth opportunities Design/methodology/approach One of the primary roles of a system database is to organize data systematically. Businesses generate vast amounts of information, and without a structured database, this data can become overwhelming. A well-designed database ensures that data is categorized, indexed, and easily accessible, facilitating efficient data management and retrieval. “

December 2024 International Journal of Management Technology and Social Sciences

Volume 8(Issue 2):2581-6012

Authors: Mustaf Abdulle ; Abdisalan Muse Osoble

What is a database ? DB, DBMS, RDBMS

Q: What is a **Database** (DB) ?

A: An organized collection of data.

Q: What is an **Database Management System** (DBMS) ?

A: A system (software) that interacts with users or applications to provide the required access to data.

Q: What is a **Relational Database Management System** (RDBMS) ?

A: A DBMS where the data is organized into **relations** (or tables).

Popular RDBMS (SQL)

MySQL / MariaDB

PostgreSQL

SQLite

Apache DB (Java DB) / HSQL / H2

Firebird

Microsoft SQL Server

Oracle

Sybase

IBM DB2

SAP HANA

Popular DBMS (NoSQL)

MongoDB

Redis

Cassandra

Amazon DynamoDB

Couchbase

Apache HBase

Neo4j

Desirable functionality in an RDBMS for O.L.T.P. ?

(O.L.T.P.) = Online Transaction Processing

ACID Transactions (Atomicity, Consistency, Isolation, Durability)

Referential Integrity

Fine-grained locking

Unicode support

(All SQL databases shown before are OLTP oriented)

Desirable characteristics of O.L.T.P. ?

High-Availability … Business interruptions can be costly.

Fault tolerance … one of several ways to achieve high availability.

Scalability …

- How large business grow ?
- Keeping up with the demand ?
- Quick response => Happy customers ?

“If you don’t drive your business, you will be driven out of business.”

B.C. Forbes, Financial Journalist and Author Forbes Magazine

Desirable functionality in a DBMS for O.L.A.P. ?

(O.L.A.P.) = Online Analytical Processing

Optimized for analysis: Designed for reading large volumes of data for analytical queries, rather than processing daily transactions.

Integrated/Unified data (from multiple sources/other databases).

Historical data.

Structured for queries.

Data cleansing.

Popular DBMS for O.L.A.P.

Snowflake

Google BigQuery

Amazon Redshift

Azure Synapse Analytics

IBM Db2 Warehouse

Glossary: Network Protocol

A network protocol **is a set of formal rules and conventions** that govern how data is transmitted, communicated, and received **across a network**. It defines the procedures for devices to format, transmit, and interpret data, essentially acting as a common language for computer programs to understand each other.

Key aspects of network protocols include:

Syntax: Defines the structure or format of the data.

Semantics: Specifies the meaning of the sections of information.

Timing: Covers when and how fast data should be sent.

Error Detection and Correction: Includes mechanisms to ensure data integrity and reliable delivery.

Glossary: Common Internet Protocols

Internet Protocol (IP): The fundamental protocol for addressing and routing data packets to ensure they reach the correct destination. Data packets may be lost, and no delivery order is guaranteed.

User Datagram Protocol (UDP): A protocol for computer programs to interact with the IP protocol and have full control over sending and receiving individual data packets.

Transmission Control Protocol (TCP): Works with IP to create a reliable, ordered connection, checking for errors and retransmitting lost packets. Built on top of the IP protocol.

HyperText Transfer Protocol (HTTP/HTTPS): The protocol used to transfer web pages and other files on the World Wide Web. HTTPS adds a layer of security. Built on top of the TCP protocol (for versions 1 and 2). (Version 3 is built on top of UDP).

Domain Name System (DNS): Translates human-readable domain names (like google.com) into IP addresses. Built on top of UDP.

Simple Mail Transfer Protocol (SMTP): Used for sending email. Built on top of the TCP protocol.

File Transfer Protocol (FTP): Used for transferring files between computers. Built on top of the TCP protocol.

Secure Shell (SSH): Used for remote text terminal access and/or transmitting/receiving files (with better security than FTP). Built on top of the TCP protocol.

Glossary: Popular Internet Applications

Online services: Banking, Rental, Bookings, Learning, Shopping, Public Services, etc…

Multimedia communication and services: audio+video, chat, email, file transfer x one-to-one (also called point-to-point), one-to-many (ex: TV streaming/broadcast), many-to-many (ex: conferences).

… and many others.

(On the side of the service provider) All of these applications need to store content and management information in one or more:

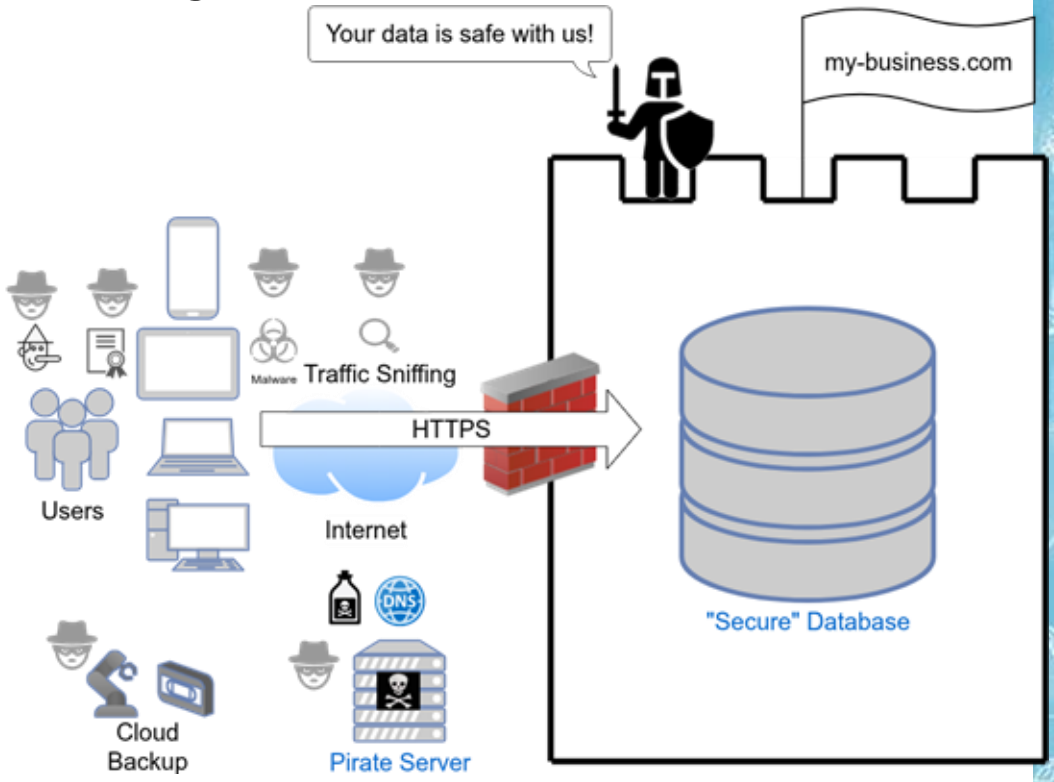
Databases

People, Data, Privacy, and Organizations

Typical Scenario

A view, **external** to the organization.

- Identify menaces to data privacy outside an organization that store and manage private user data ?

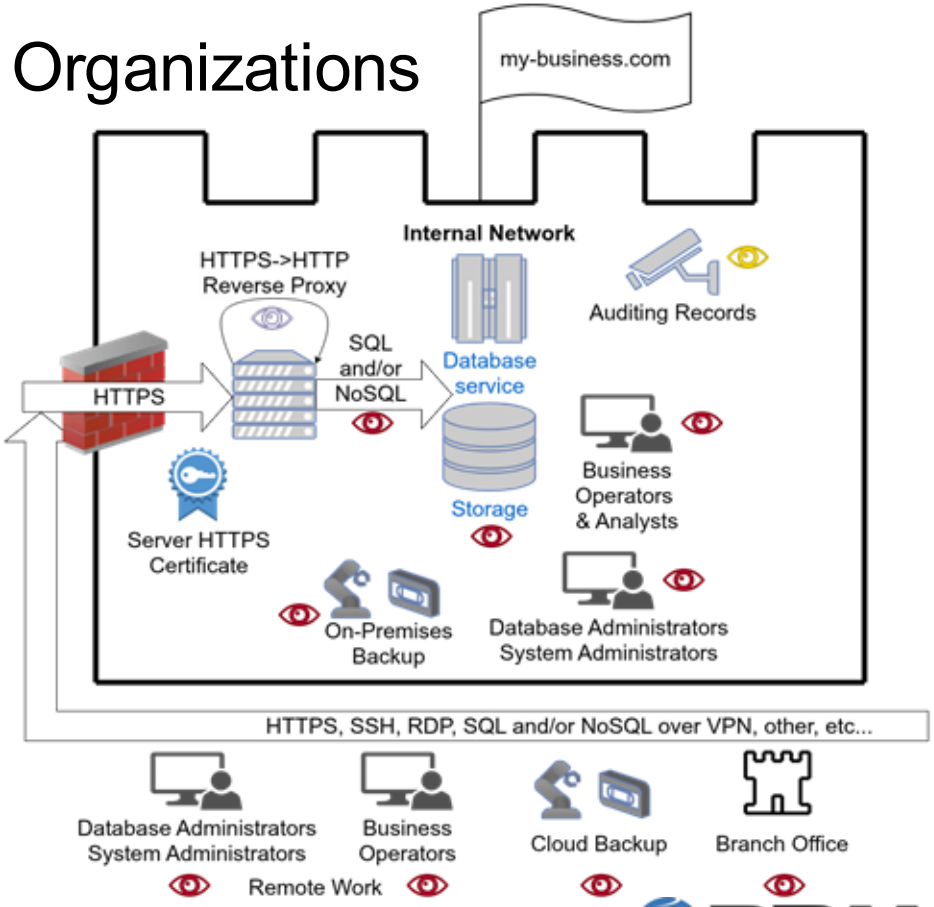


People, Data, Privacy, and Organizations

Typical Scenario

A view, **internal** to the organization.

- Identify menaces to data privacy inside an organization that store and manage private user data ?

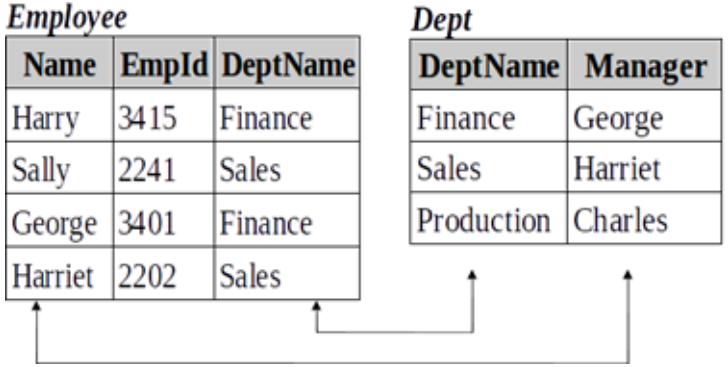


Relational Database Data Structure

In relational databases, data is organized in tables (also called “relations” in relational algebra terminology).

- Each table has a unique name.
- Tables are structured into columns.
- Each column has a name (unique within that table), and a data type.
- Column values may refer to rows in other tables.

The definition of all the tables and its structure is called “the database **schema**”.



Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

SQL – Structured Query Language

... is a database computer language designed for:

- the retrieval and management of data in relational database management systems (RDBMS),
- database schema creation and modification,
- and database object access control management.

... is pronounced “ess-kwe-el” or “sequel”.

... is an ANSI/ISO standard (SQL-89, …, SQL:2016, etc...).

SQL Statements Classification

SQL is a declarative language, not an imperative (procedural) one.

Language statements are classified as:

- DML – Data Manipulation Language statements
  - SELECT – extract data.
  - INSERT – insert data.
  - UPDATE – update data.
  - DELETE – delete data.

- DDL – Data Definition Language statements
  - CREATE TABLE – create and define a table.
  - DROP TABLE – delete a table, and its whole data.

DML Queries - SELECT

Syntax of the SELECT statement:

```
SELECT column_expression [,...]
FROM table_expression [,...]
[WHERE condition]
[GROUP BY expression [,...]
   [HAVING condition]]
[ORDER BY expression [ASC|DESC][,...]]
```

SELECT * Example

```
SELECT * FROM Employee;

Name       EmpId   DeptName
----       -----   --------
Harry    3415    Finance
Sally    2241    Sales
George   3401    Finance
Harriet  2202    Sales

4 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT … WHERE condition

Conditions evaluate to TRUE or FALSE. Can use comparison operators, predicates, logical operators, mathematical expressions, functions, etc...

Table 14.5 Logical Operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
OR, ||	Logical OR
XOR	Logical XOR

Table 14.4 Comparison Operators

Name	Description
>	Greater than operator
>=	Greater than or equal operator
<	Less than operator
<>, !=	Not equal operator
<=	Less than or equal operator
<=>	NULL-safe equal to operator
=	Equal operator
BETWEEN ... AND ...	Whether a value is within a range of values
COALESCE()	Return the first non-NULL argument
EXISTS()	Whether the result of a query contains any rows
GREATEST()	Return the largest argument
IN()	Whether a value is within a set of values
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
ISNULL()	Test whether the argument is NULL
LEAST()	Return the smallest argument
LIKE	Simple pattern matching
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT EXISTS()	Whether the result of a query contains no rows
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

SELECT+WHERE = Example

```
SELECT Name
FROM Employee
WHERE EmpId = 2241;

Name
----
Sally

1 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT+WHERE >= Example

```
SELECT Name
FROM Employee
WHERE EmpId >= 3000;

Name
----
Harry
George

2 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT+WHERE >= Example

```
SELECT Name
FROM Employee
WHERE EmpId >= 2000
  AND EmpId < 3000;

Name
----
Sally
Harriet

2 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT+ORDER BY Example

```
SELECT Name
FROM Employee
WHERE EmpId >= 3000
ORDER BY Name ASC;

Name
----
George
Harry

2 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

DML Inserting Data - INSERT

Syntax of the INSERT statement (to insert a single new row of data):

```
INSERT INTO table_name
[( colum_name1, ... )]
VALUES ( value1, ...)
```

Syntax of the INSERT statement (to insert a multiple new rows of data):

```
INSERT INTO table_name
[( colum_name1, ... )]
SELECT value1, ...
```

INSERT Example

```
INSERT INTO Employee
(Name, EmpId, DeptName)
VALUES (
'John',3602,'Finance'
);
1 rows inserted
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
John	3602	Finance

Depending on the software, a COMMIT statement may be required, so that the data is persisted to storage, and visible to other client applications.

DML Updating Data - UPDATE

Syntax of the UPDATE statement :

```
UPDATE table_name
SET column_name1 = expression1
[, column_name2 = expression2
   [,...]]
[WHERE condition]
```

On the absence of a WHERE condition, all the rows in the *table_name* are updated.

UPDATE Example

```
UPDATE Employee
SET name=CONCAT('Mr.',name)
WHERE EmpId=3602;

1 row updated
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Mr. John	3602	Finance

Depending on the software, a COMMIT statement may be required, so that the data is persisted to storage, and visible to other client applications.

DML Deleting Data - DELETE

Syntax of the DELETE statement :

```
DELETE FROM table_name [WHERE condition]
```

Deletes all rows where *condition* is true.

If the `WHERE condition` is absent, **all rows are deleted**.

The effects of the DELETE are (usually) irreversible!

Optional SQL Introductory Practice

Unzip `sqlsecurity.zip` and open `sqlsecurity/practices/index.html` and perform the “Setup” section.

Or visit https://homes.pdmfc.com/jpsl/sqlsecurity/practice/

Or https://shorturl.at/yEGUo

Then go the SQL Playground, click Examples, then select “Employees”, and then press “Execute”.

You now have a database setup, and are free to try out any SQL statements you wish on the right “SQL Editor” pane. Press “Execute” to run them. Multiple statements must be separated by ;

Optional Introductory SQL Exercises

Translate from english to SQL, and verify by executing in the “SQL Playground”:

1. List all employee names sorted by alphabetical ascending order. (Tip: Terminate the select with “`ORDER BY name ASC`”).
2. List all employee names sorted by alphabetical descending order.
3. What is the employee name and department of the employee with number 2241 ?
4. Insert a new employee, name “Bart”, number 2991, department “Sales”.
5. Rename employee number 2991 to “Simpson”.
6. Delete employee number 2991.

Solutions on the next slide.

Optional Introductory SQL Exercises

Translate from english to SQL, and verify by executing in the “SQL Playground”:

1. `SELECT name FROM Employee ORDER BY name ASC;`
2. `SELECT name FROM Employee ORDER BY name DESC;`
3. `SELECT name, deptName FROM Employee WHERE empId=2241;`
4. `INSERT INTO Employee (name, empId, deptName)`
   `VALUES ('Bart',2291,'Sales');`
5. `UPDATE Employee SET name='Simpson' WHERE empId=2291;`
6. `DELETE FROM Employee WHERE empId=2291;`

More SQL Possibilities (optional)

It is not going to be needed in the exercises, but

SQL can do much, much more…

SELECT COUNT(*) Example

```
SELECT COUNT(*)
FROM Employee
WHERE EmpId >= 3000;

COUNT(*)
----
2

1 row selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT+AVG Example

```
SELECT AVG(EmpID)
FROM Employee
WHERE EmpId >= 3000;

AVG(EmpID)
----------
3408

1 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

SELECT+GROUP BY Example

```
SELECT DeptName, SUM(EmpID)
FROM Employee
GROUP BY DeptName;

DeptName   SUM(EmpID)
--------   ----------
Finance    6816
Sales      4443

2 rows selected
```

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Cartesian Product

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee X Dept

Employee			Dept	
Name	EmpId	DeptName	DeptName	Manager
Harry	3415	Finance	Finance	George
Sally	2241	Sales	Finance	George
George	3401	Finance	Finance	George
Harriet	2202	Sales	Finance	George
Harry	3415	Finance	Sales	Harriet
Sally	2241	Sales	Sales	Harriet
George	3401	Finance	Sales	Harriet

… 4x3 = 12 rows

SQL SELECT for Cartesian Product

```
SELECT * FROM Employee, Dept;

Name EmpId DeptName DeptName Manager
---- ----- -------- -------- -------
Harry 3415 Finance  Finance  George
Sally 2241 Sales    Finance  George
...
12 rows selected.
```

Natural Join

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee INNER JOIN Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

SQL SELECT for Natural Join

```
SELECT * FROM Employee A, Dept B
WHERE A.DeptName = B.DeptName;
```

or

```
SELECT * FROM Employee AS A
INNER JOIN Dept AS B
ON A.DeptName = B.DeptName;
```

```
A.Name A.EmpId A.DeptName B.DeptName B.Manager
------ ------- ---------- ---------- ---------
Harry  3415    Finance    Finance          George
Sally  2241    Sales      Sales            Harriet
...
4 rows selected.
```

SQL has a much bigger set of functions…

And much more…

- Single Row Functions: return a single result row for each row of the queried table or view. These include Numeric Functions ; Character Functions ; Data Mining Function ; Datetime Functions ; Conversion Functions ; Collection Function ; XML Functions ; JSON Functions ; 2D functions ; N-Dim functions ; etc…
- Aggregate Functions: return a single value from a group of rows. AVG ; SUM ; COUNT ; MIN/MAX ; STDDEV ; etc…
- Analytic Function: calculations based on the rows before and after the current row.
- User Defined Functions: defined by the user/programmer.

SQL has a bigger set of (non-standard) capabilities…

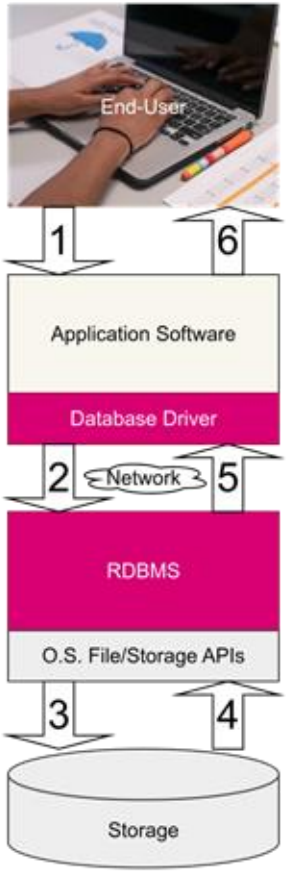
- Data constraints: Validate that data obeys specific constraints. Includes referential integrity constraints (values reference rows in other tables), and user defined constraints.
- In-database programming (code that runs in the database engine, with “low-impedance” data handling).
- Triggers: Events of data being inserted, updated, or deleted, trigger the occurrence of user-defined actions.

Application Layers and SQL Queries (0/2)

Security scopes discussed:

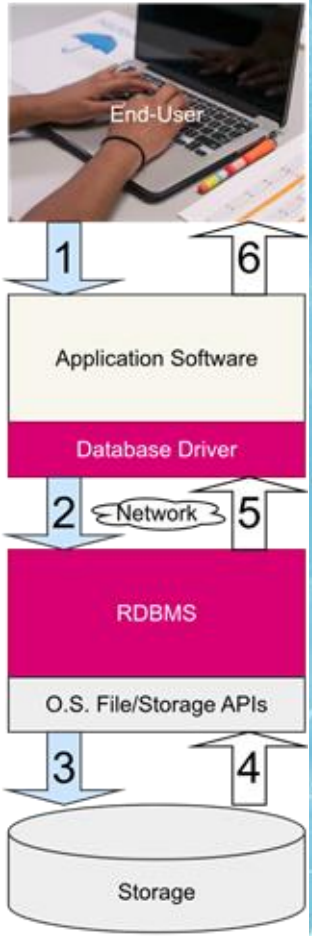
- ✔ Main focus.
- ✔ Related to DB only!

- ✔ Application Security
- ✔ Network Security
- ✔ Database Security
- ✔ O.S. Security
- ✔ Storage Security



Application Layers and SQL Queries (1/2)

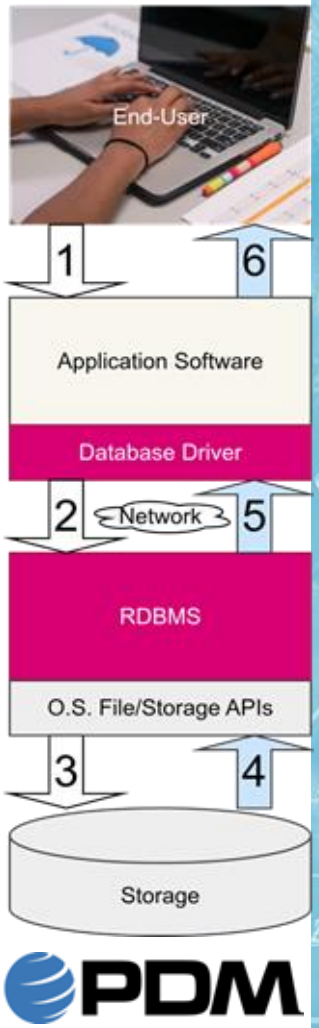
1. The user interacts with an application, and requests to view some specific data.
2. The application software translates the request into an SQL query (typically a SELECT statement), and makes use of the RDBMS driver to send the request to the database. (The request typically includes database access credentials).
3. The RDBMS receives the request, (validates the access credentials,) parses the SQL query, and makes a series of O.S. API calls to read some specific files (or specific disk blocks directly).



Application Layers and SQL Queries (2/2)

4. The disk hardware + firmware fetches the requested disk data blocks and sends them back to the O.S.
5. The RDBMS receives the data from the O.S., processes it (filter, sort, aggregate, compute, etc), and sends them back to the application, according to the specific SQL network protocol.
6. The application receives the DB API call results, with the requested data, and displays it (in proper format) to the end-user.

The network protocol used for steps 2 and 5 is database-specific and (in most cases) TCP based.



Database Protocols ?

*"**Database protocols are the unseen bridges connecting applications with databases.** Choosing the right database protocol impacts performance, scalability, and compatibility. Understanding these protocols and their networking layers helps developers make better decisions when designing database-driven applications and migrating between different database systems."*

https://designvault.medium.com/understanding-database-protocols-how-databases-communicate-c1ab61e21a40 Ganesh Sahu

6 examples for popular DBMSs

Database Protocols: Generic Security Characteristics

Authentication: Most DBMS allow the administrator/developer to configure a set of database users and privileges that can be used for **Role Based Access Control (RBAC)** of data. Some can share the O.S. user database and authentication mechanisms. All support the classic username/password authentication method, with security variations (including client X.509 certificates).

Transport: All examples shown here are TCP based, and most support SSL/TLS TCP transport (using certificates), with specific configuration possibilities.

Simple TCP transport is (normally) vulnerable to **network traffic sniffing!**

=> Simple TCP transport should only be used in **"secure" networks.**

Keep in mind that opening a SSL/TLS connection is more slow (and resource consuming) that a simple TCP connection. As such, the use of connection pools (on the application side) is recommended to avoid the connection overhead.

Database Protocols 1/6: PostgreSQL Wire Protocol

1. PostgreSQL Wire Protocol: Used by: PostgreSQL, CockroachDB (wire-compatible), YugabyteDB (wire-compatible), Amazon Aurora PostgreSQL (full compatibility advertised).

Security: Supports SSL/TLS for encrypted communication, role-based access control (RBAC), and strong authentication methods like SCRAM.

Performance: Optimized for complex queries, ACID compliance, and transaction integrity. May have slightly higher latency compared to NoSQL databases.

Java Library: JDBC PostgreSQL Driver

Python Library: psycopg2

Database Protocols 2/6: MySQL Protocol

2. MySQL Protocol

Used by: MySQL, MariaDB, TiDB (wire-compatible), Amazon Aurora MySQL (full compatibility advertised).

Security: Supports SSL/TLS encryption, native password authentication, and fine-grained access control.

Performance: Efficient for read-heavy workloads with replication support. Query execution may slow down under high concurrency.

Java Library: JDBC MySQL Connector

Python Library: mysql-connector-python

Database Protocols 3/6: TDS (Tabular Data Stream)

3. TDS (Tabular Data Stream) for MSSQL

Used by: Microsoft SQL Server (MS-SQL), Sybase

Security: Uses SSL/TLS encryption, Windows authentication, and Kerberos support for secure communication.

Performance: Efficient for enterprise workloads, supports connection pooling but may have higher overhead due to security enforcement.

Java Library: JDBC SQL Server Driver

Python Library: pyodbc

Database Protocols 4/6: MongoDB Wire Protocol

4. MongoDB Wire Protocol

Used by: MongoDB

Security: Supports SSL/TLS encryption, authentication via SCRAM and X.509 certificates, and role-based access control.

Performance: High throughput for unstructured data, optimized for horizontal scaling but weaker consistency guarantees than SQL databases.

Java Library: MongoDB Java Driver

Python Library: pymongo

Database Protocols 5/6: Cassandra Binary Protocol

5. Cassandra Binary Protocol

Used by: Apache Cassandra, ScyllaDB (wire-compatible)

Security: Supports SSL/TLS encryption, password authentication, and access control lists (ACLs).

Performance: Designed for scalability and fault tolerance, optimized for fast writes but can have higher read latency compared to relational databases.

Java Library: Datastax Java Driver

Python Library: cassandra-driver

Database Protocols 6/6: Redis Serialization Protocol

6. Redis Serialization Protocol (RESP)

Used by: Redis

Security: Lacks built-in authentication by default, but supports password authentication and SSL/TLS encryption in enterprise editions.

Performance: Extremely low latency, high throughput, and optimized for in-memory operations. Best suited for caching rather than transactional workloads.

Java Library: Jedis

Python Library: redis-py

Principle of Least Privilege 1/2: Role Based Access Control

Most databases support “Principle of Least Privilege” by organizing the database users into groups, also called roles. One user can belong to more than one group (have more than one role).

Each group/role has a set of privileges. A user in multiple groups/with multiple roles inherits all the privileges of each group/role.

A privilege is the ability to perform a SELECT and/or INSERT and/or UPDATE and/or DELETE, … on a particular database object (table, view, sequence, etc).

Principle of Least Privilege 2/2: RBAC

Examples of role names and privileges:

dashboard - Can SELECT from a particular VIEW that gives the number of customer enrolled this month. No other operation allowed.

analytics - Can SELECT from all the tables, except for the table with each customer's password. Cannot perform INSERT UPDATE or DELETE on any table.

customer - Can SELECT, INSERT or UPDATE on any table with customer data. Cannot perform DELETEs, and is only allowed SELECTs on system tables. The application must ensure that the SQL statements only affect rows related to one specific customer.

owner - Can do anything, including new table creation, modification, deletion, inside one specific schema.

superuser - Can do anything on all schemas.

Data Segregation 1/5: Row-Level Security

On some situations, it is convenient to have a single database (a single instance or a single cluster), a single schema, the same tables, but some of the data in those table rows must be accessible only by some users, and not by others.

These **rules** are normally **coded** in the **applications** themselves, but if one user (with data visibility restrictions) needs to access the database directly using a SQL tool (like a Data Analytics tool, a Data Visualizer tool, or even a spreadsheet with an SQL client, etc), a new mechanism is needed.

Data Segregation 2/5: Row-Level Security

Example 1: A government/regulator service has business/fiscal/customer/etc data from **distinct business competitors**. Extending the “Right to access” for business, each business owner can view the data that the government has about its own business (ie. through a web application), but, never the data from its competitors.

The schema is the same, data from distinct competitors is stored in the same table(s), but distinct table rows have data from distinct competitors. Users from one competitor are never allowed access data from other competitor.

Data Segregation 3/5: Row-Level Security

Example 2. A **commercial service** with personal customer data allows some entities to become **resellers of the service**. For example, business franchise, where the owner of the franchise offers the same services (same applications, same database) to all franchisers.

Each reseller/franchise is only allowed to view customer data from its own sales, and never data from other resellers. A few tables store business data (from all resellers), but the visibility of the data on each row obeys this access restriction.

Data Segregation 4/5: Row-Level Security

Each row that a SELECT statement outputs is subject to extra filtering by special coded rules - coded in the database themselves - not on an external application. The rules decide if the row is visible or not.

Some databases provide embedded procedural languages for this (and other) purposes: Oracle's PL/SQL, MS-SQL Server's Transact-SQL procedures, PostgreSQL's PL/pgSQL, etc.

Data Segregation 5/5: Row-Level Security

Oracle calls this feature “Virtual Private Databases”, but the majority of database vendors call this feature **“Row Level Security” (RLS)**.

If data is segregated at the database level, it is one extra layer of security, in addition to the security features provided by the applications that access the database.

Data Backups - Single Snapshot

Database backups are essential for business recovery, in situations of **catastrophic** failure of the RDBMS (hardware and/or software).

No backup feature is configured (by default) out-of-the-box in any of the SQL databases mentioned, so, it is up to the system administrator or database administrator (DBA) to set up such a system (preferably, in an automated way).

The simplest database backup tools create a consistent snapshot of the data in the database (or of a single schema) at a given point in time.

(Backups performed at the filesystem level with the RDBMS running are usually inconsistent - unrestorable - unless specific precautions - database dependent - are taken).

Data Backups - validation

Data Backups are to be considered non existent until validation, by performing an actual backup restore. (Restoration can be performed on a secondary, but, identically configured RDBMS for validation purposes).

Treat access to the backup files (or to restored databases) with the same level of security as access to the original database.

Data Backups - Continuous Backup

Single data snapshot backups loose all new data / new transactions, since the moment in time the backup snapshot started. Depending on the periodicity of the backup and the frequency of new transaction, it may be called a high Recovery Point Objective.

Recovery Point Objective (RPO) is the the maximum acceptable amount of data loss, measured in time, that an organization can tolerate after an unexpected event.

Continuous Backup, also known as **Continuous Data Protection** (CDP), ensures that all live-transactions are being backed up (in addition to the initial snapshot backup) shortly after occurrence. This usually assures a much lower RPO.

Continuous Backups usually allow backup restoration to be done at any point-in-time between the last snapshot and the last transaction backed up. (Point-in-Time Recovery).

Data Backups - Backup Encryption

The backup files may be encrypted to prevent unauthorized access to the data in the backup. This is especially important if the backup storage is done outside the organization (motivated by better disaster recovery).

The backup encryption/decryption keys need to be secured in a way that only the legitimate data owners can decrypt the backup (and restore data).

Datasets for Analytics/Tests:Anonymization&Masking

If the organization is big enough to hire developers to work on the applications that manipulate the data, developers may request access to production data (for example, by restoring the backup on a development replica of the production database) so that the applications may be tested against real data.

Data **anonymization and masking** should be applied whenever possible: Remove or obscure personally identifiable information from datasets for analytics or testing purposes.

Data Fingerprinting (or Watermarking) 1/2

Real Data samples used in development/tests should be subject to **anonymization and masking** whenever possible. But there are situations where that may not be possible:

Example 1: An application fails to process some names or addresses (unknowingly caused by the presence of unexpected Unicode characters in the data). The developers might (justifiably) need access to the original data to be able to replicate and identify the problem. (Limit the dataset to the minimum set needed to reproduce the fault).

Example 2: Database performance tuning, where the database performance over masked data is not the same as over real data. Database execution plans are dependent on the data itself. (Anonymization preserving the same statistical data distribution characteristics as the real data may not be possible, or very hard).

Data Fingerprinting (or Watermarking) 2/2

Real Data samples used in analytics/tests should be subject to **anonymization and masking** whenever possible. If anonymization and masking are not possible, ensure that this data is accessed **by the least number of people needed**, and that those people are motivated - **professionally, contractually, and legally** - to uphold the required confidentiality levels.

If unsure of people's professional ethics, have those activities closely monitored by someone trustworthy.

Inserting specific data modifications to detect and track the origin of data leaks is a common practice in these situations. This is called **Data Fingerprinting** (or **Data Watermarking**).

Data Encryption 1/3

The database's datafiles are usually accessible to anyone with O.S. administration privileges. This is of particular importance when:

- The database hardware is operated by a 3rd party cloud provider.
- The sys.admin. services are provided by 3rd party contractors.
- Replacement of disk hardware (to upgrade capacity/performance or to replace faulty disks) is performed by 3rd party contractors.

Besides **contractual obligations**, having the data encrypted provides another layer of data protection.

Data on old replaced disks (or tapes) should be *shredded* (even if encrypted). Data overwrite is usually enough for magnetic disks or tapes, but SSD are more complex.

Data Encryption 2/3 - Implementation Levels

Database-level (Transparent Data Encryption - TDE): This is handled automatically by the database engine. It encrypts and decrypts data as it's written to and read from storage, without requiring changes to applications. It protects data on disk, and should include log files and backups.

Application-level: The application itself encrypts the data before it is sent to the database. This gives developers more control over the encryption algorithms and keys, but requires modifications to the application code. (SQL calculations/searches/analytics may not be possible).

Column/field-level: Instead of encrypting the entire database, only specific sensitive columns, like a "credit_card" number, are encrypted. This can be a good balance between security and performance.

Full disk encryption: This encrypts the entire hard drive, protecting all data at rest. It is a strong defense, but it does not protect data once the system is running and the drive is decrypted by the operating system.

Data Encryption 3/3 - TDE & Keys

Database-level (Transparent Data Encryption - TDE): The most transparent option, no need to change applications. This feature is usually provided by the RDBMS software itself. Without hardware support, this (usually) introduces a performance penalty.

In any level, data encryption requires keys, which are the most sensitive information, needed to operate the solution.

Encryption/decryption keys might need to be available automatically (so that the system may reboot - shutdown/startup - unattended). If decryption keys are available to automated processes, they are also available to hackers.

Data Encryption 3/3 - Key best practices

Human interaction required: A person needs to provide a credential in-place (password typing or time-based key or biometric key or hardware token). The (private) key is not stored in the system. Cons: The system cannot usually perform some operations (like a reboot or backup restore) unattended.

Key splitting techniques and Key Vault APIs: Several technologies/processes are needed to obtain the complete decryption key. Automated processes can work unattended, but just makes it harder for a hacker to do the same.

Preparing for Practices

Exercises on SQL Injection

A set of examples and exercises to demonstrate what is SQL Injection, and how to prevent it.

But before,

A quick mention of some common Web Technology Stacks:

Popular Web Tech Stacks: L.A.M.P.

(Linux, Apache, MySQL, PHP)

Example of a PHP snippet connecting and querying a MySQL database:

```
<?php
// Create connection
$conn = new mysqli("localhost", "dbusername", "dbpassword", "dbname");
// Check connection
if ($conn->connect_error) { die("Connection failed: " . $conn->connect_error); }

$sql = "SELECT empId, name, deptname FROM Employees";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "empId: " . $row["empId"]. " - Name: " . $row["name"] . "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
?>
```

Popular Web Tech Stacks: Python (possibly w/ Flask)

Example of a Python snippet connecting and querying a MySQL database:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="dbusername",
  password="dbpassword",
  database="dbname"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Popular Web Tech Stacks: NodeJS + MySQL

JS Example in NodeJS, connecting and querying MySQL:

```
let mysql = require('mysql');

let con = mysql.createConnection({
  host: "localhost",
  user: "dbusername",
  password: "dbpassword",
  database: "dbname"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

Tech Stack for Practice: SQLite

Tech Stack for Practice: sql.js.org

Tech Stack for Practice: sql.js exec() example

CODE

```
var db = new SQL.Database();
var res = db.exec(
    "DROP TABLE IF EXISTS test;\n"
    + "CREATE TABLE test (id INTEGER, age INTEGER, name TEXT);"
    + "INSERT INTO test VALUES ($id1, :age1, @name1);"
    + "INSERT INTO test VALUES ($id2, :age2, @name2);"
    + "SELECT id FROM test;"
    + "SELECT age,name FROM test WHERE id=$id1",
    {
        "$id1": 1, ":age1": 1, "@name1": "Ling",
        "$id2": 2, ":age2": 18, "@name2": "Paul"
    }
);
```

Practices

Exercises on SQL Injection

Unzip `sqlsecurity.zip` and open
`sqlsecurity/practices/index.html`
and perform the “Setup” section.

Or visit

https://homes.pdmfc.com/jpsl/sqlsecurity/practice/

https://shorturl.at/yEGUo

Practices - Lessons learned

After going through these exercises, a few obvious recommendations should stick in memory when writing code:

- Never, ever, let "unsanitized" user input be directly added to a SQL statement code.
- Make use of the existing API mechanisms to properly replace literal parameter values instead of attempting to build the complete SQL text yourself by concatenating/replacing strings.
- When executing statements on the database, always strive to use the least amount of privileges needed! (ie. If executing read queries, there should be no write privileges. Only the required tables and rows should be accessible.)
- Do not trust data sources, and neither blindly trust existing data. Always assume that the data can be compromised at any time by unexpected means, and try to write your code to minimize impacts of handling such corrupted data.

Other Resources

OWASP:

https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html

https://owasp.org/www-project-web-hacking-incident-database/

https://owasp.org/Top10/

OWASP Database Security Cheat Sheet: ToC

Table of contents