

# **Lecture 1.**

# **Introduction to**

# **Databases**

**Kuralbayev Aibek**

- Kuralbayev Aibek
- Telegram: @Aibek21
- aibekkuralbaev@gmail.com

# Database Applications Examples

- Enterprise Information
  - Sales: customers, products, purchases
  - Accounting: payments, receipts, assets
  - Human Resources: Information about employees, salaries, payroll taxes.
- Manufacturing: management of production, inventory, orders, supply chain.
- Banking and finance
  - customer information, accounts, loans, and banking transactions.
  - Credit card transactions
  - Finance: sales and purchases of financial instruments (e.g., stocks and bonds; storing real-time market data
- Universities: registration, grades

# Database Applications Examples (Cont.)

- Airlines: reservations, schedules
- Telecommunication: records of calls, texts, and data usage, generating monthly bills, maintaining balances on prepaid calling cards
- Web-based services
  - Online retailers: order tracking, customized recommendations
  - Online advertisements
- Document databases
- Navigation systems: For maintaining the locations of various places of interest along with the exact routes of roads, train systems, buses, etc.

# Purpose of Database Systems

In the early days, database applications were built directly on top of file systems, which leads to:

- Data redundancy and inconsistency: data is stored in multiple file formats resulting in duplication of information in different files
- Difficulty in accessing data
  - Need to write a new program to carry out each new task
- Data isolation
  - Multiple files and formats
- Document databases
- Integrity problems
  - Integrity constraints (e.g.,  $\text{account balance} > 0$ ) become “buried” in program code rather than being stated explicitly
  - Hard to add new constraints or change existing ones

# Purpose of Database Systems (Cont.)

- Atomicity of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - Example: Transfer of funds from one account to another should either complete or not happen at all
- Concurrent access by multiple users
  - Concurrent access needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - Ex: Two people reading a balance (say 100) and updating it by withdrawing money (say 50 each) at the same time
- Security problems
  - Hard to provide user access to some, but not all, data

**Database systems offer solutions to all the above problems**

# University Database Example

- In this text we will be using a university database to illustrate all the concepts
- Data consists of information about:
  - Students
  - Instructors
  - Classes
- Application program examples:
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts

A **database** is nothing more than a **set of related information**





# View of Data

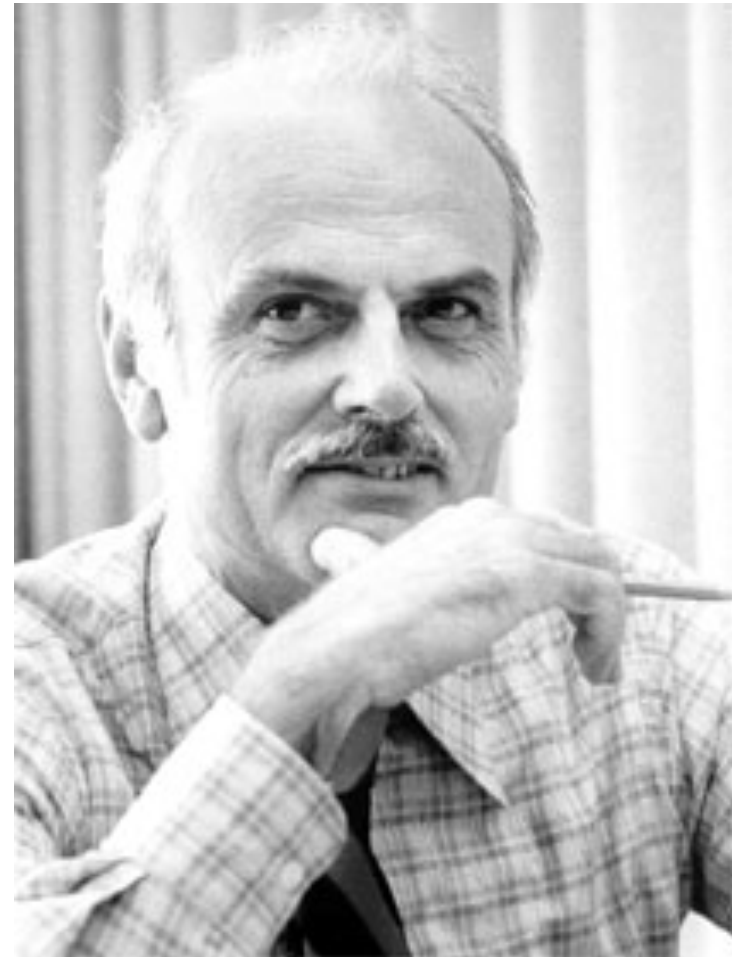
- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.
- A major purpose of a database system is to provide users with an abstract view of the data.
  - Data models
    - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.
  - Data abstraction
    - Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.

# Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

# The Relational Model

- Dr. E. F. Codd
- 1970
- «A Relational Model of Data for Large Shared Data Banks»

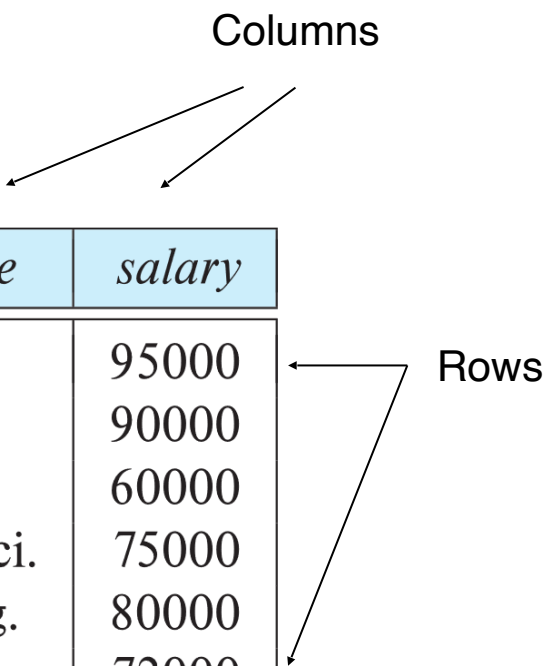


# The Relational Model

- Data can be represented as sets of **tables**.
- Rather than using pointers to navigate between related entities, **redundant data** is used to link records in different tables.

# Relational Model

- All the data is stored in various tables.
- Example of tabular data in the relational model



The diagram shows a table with four columns and ten rows. Two arrows labeled 'Columns' point to the first two columns, and two arrows labeled 'Rows' point to the first two rows.

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

# Relational Model

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

<i>dept_name</i>	<i>building</i>	<i>budget</i>
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

# Levels of Abstraction

- **Physical level:** describes how a record (e.g., instructor) is stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
type instructor = record
```

```
    ID : string;
```

```
    name : string;
```

```
    dept_name : string;
```

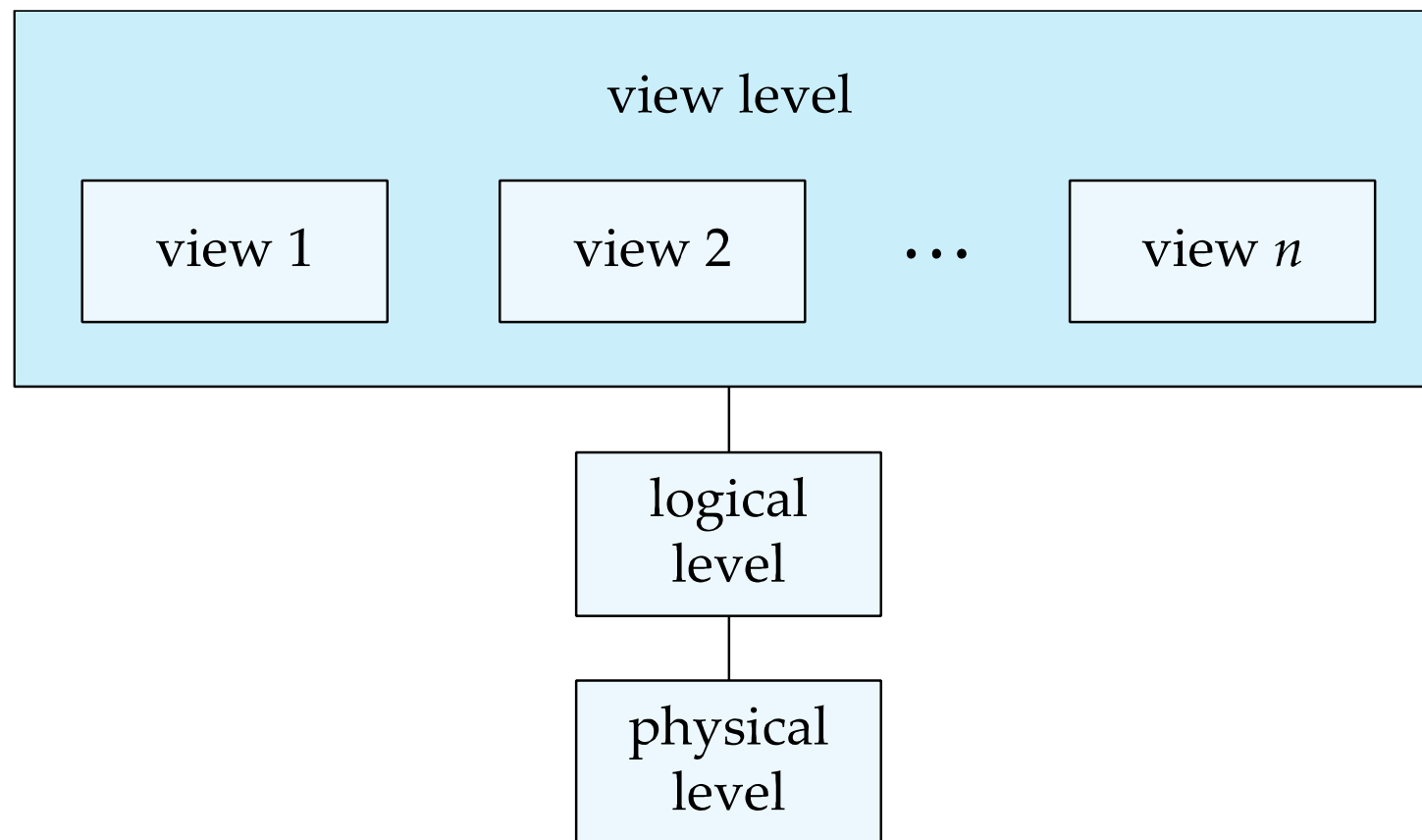
```
    salary : integer;
```

```
end;
```

- **View level:** application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes.

# Levels of Abstraction

An architecture for a database system





# Instances and Schemas

- Similar to types and variables in programming languages
- **Logical Schema** – the overall logical structure of the database
  - Example: The database consists of information about a set of customers and accounts in a bank and the relationship between them
    - Analogous to type information of a variable in a program
- **Physical schema** – the overall physical structure of the database
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable

# Physical Data Independence

- **Physical Data Independence** – the ability to modify the physical schema without changing the logical schema
  - Applications depend on the logical schema
  - In general, the interfaces between the various levels and components should be well defined so that changes in some parts do not seriously influence others.

# Data Definition Language (DDL)

- Specification notation for defining the database schema

Example:       **create table** *instructor* (  
                              *ID*              **char**(5),  
                              *name*          **varchar**(20),  
                              *dept\_name* **varchar**(20),  
                              *salary*      **numeric**(8,2))

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

# Data Definition Language (DDL)

```
CREATE TABLE corporation  
  (corp_id SMALLINT,  
   name VARCHAR(30),  
   CONSTRAINT pk_corporation PRIMARY KEY (corp_id)  
  );
```

# Data Manipulation Language (DML)

- Language for accessing and updating the data organized by the appropriate data model
  - DML also known as query language
- There are basically two types of data-manipulation language
  - **Procedural DML** -- require a user to specify what data are needed and how to get those data.
  - **Declarative DML** -- require a user to specify what data are needed without specifying how to get those data.
- Declarative DMLs are usually easier to learn and use than are procedural DMLs.
- Declarative DMLs are also referred to as non-procedural DMLs
- The portion of a DML that involves information retrieval is called a **query** language.

# Data Manipulation Language (DML)

```
INSERT INTO corporation (corp_id, name)  
VALUES (27, 'Acme Paper Corporation');
```

# Retrieve inserted row

```
SELECT name FROM corporation
WHERE corp_id = 27;
```

```
+ - - - - - - - - - - - - +
| name                               |
+ - - - - - - - - - - - - +
| Acme Paper Corporation           |
+ - - - - - - - - - - - - +
```

# SQL Query Language

- SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table.
- Example to find all instructors in Comp. Sci. dept

```
select name  
from instructor  
where dept_name = 'Comp. Sci.'
```

- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database



# Database Access from Application Program

- SQL does not support actions such as input from users, output to displays, or communication over the network.
- Such computations and actions must be written in a **host language**, such as C/C++, Java or Python, with embedded SQL queries that access the data in the database.
- **Application programs** -- are programs that are used to interact with the database in this fashion.

# Database Design

The process of designing the general structure of the database:

- Logical Design – Deciding on the database schema. Database design requires that we find a “good” collection of relation schemas.
  - Business decision – What attributes should we record in the database?
  - Computer Science decision – What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

# Database Engine

- A database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of a database system can be divided into
  - The storage manager,
  - The query processor component,
  - The transaction management component.

# Storage Manager

- A program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- The storage manager components include:
  - Authorization and integrity manager
  - Transaction manager
  - File manager
  - Buffer manager

# Storage Manager (Cont.)

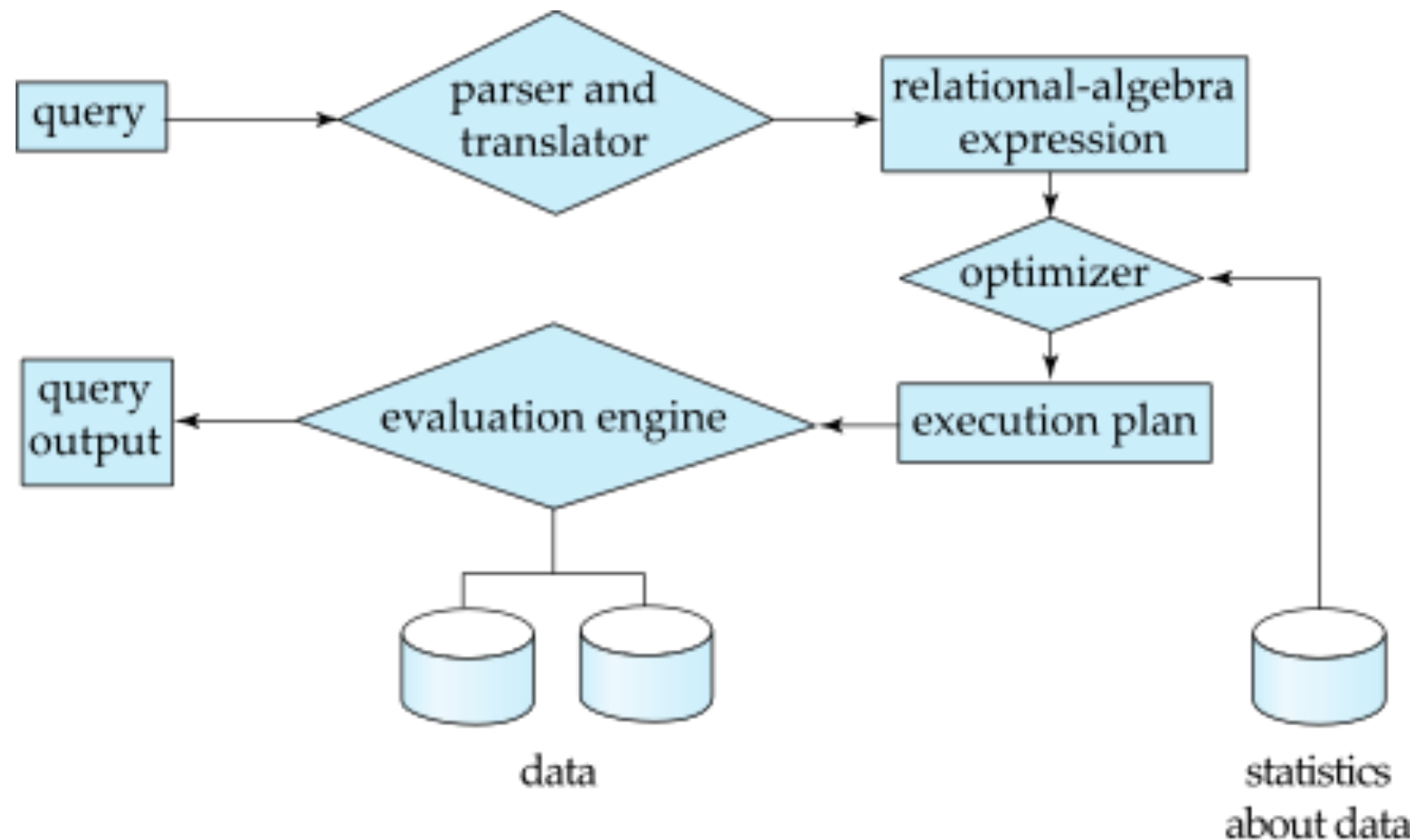
- The storage manager implements several data structures as part of the physical system implementation:
  - Data files -- store the database itself
  - Data dictionary -- stores metadata about the structure of the database, in particular the schema of the database.
  - Indices -- can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.

# Query Processor

- The query processor components include:
  - DDL interpreter -- interprets DDL statements and records the definitions in the data dictionary.
  - DML compiler -- translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
    - The DML compiler performs query optimization; that is, it picks the lowest cost evaluation plan from among the various alternatives.
  - Query evaluation engine -- executes low-level instructions generated by the DML compiler.

# Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



# Transaction Management

- A **transaction** is a collection of operations that performs a single logical function in a database application
- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database.



# SQL transaction statements

```
BEGIN;  
UPDATE accounts SET balance = balance - 100.00  
WHERE name = 'Alice';  
-- etc etc  
ROLLBACK;  
COMMIT;
```

# DBMS (Database Management System)

- MySQL
- MS SQL Server (Microsoft)
- Oracle DB (Oracle)
- DB2 Universal Database (IBM)
- PostgreSQL (PostgreSQL GDG)

# Why PostgreSQL?

PostgreSQL is an **enterprise-class** relational database management system, on par with the very best proprietary database systems



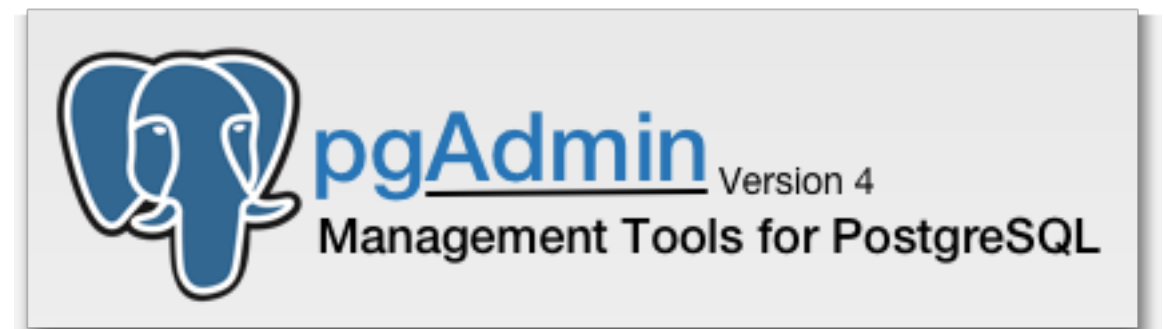
PostgreSQL

# Why PostgreSQL?

- Open source (Large community)
- Cross platform
- Fast
- Stored procedures and functions in numerous programming languages (C, SQL, Python, JS, Ruby etc.)
- Large number of data types and define your own
- NoSQL

# Management Tools

- psql
- DataGrip
- phpPgAdmin
- pgAdmin



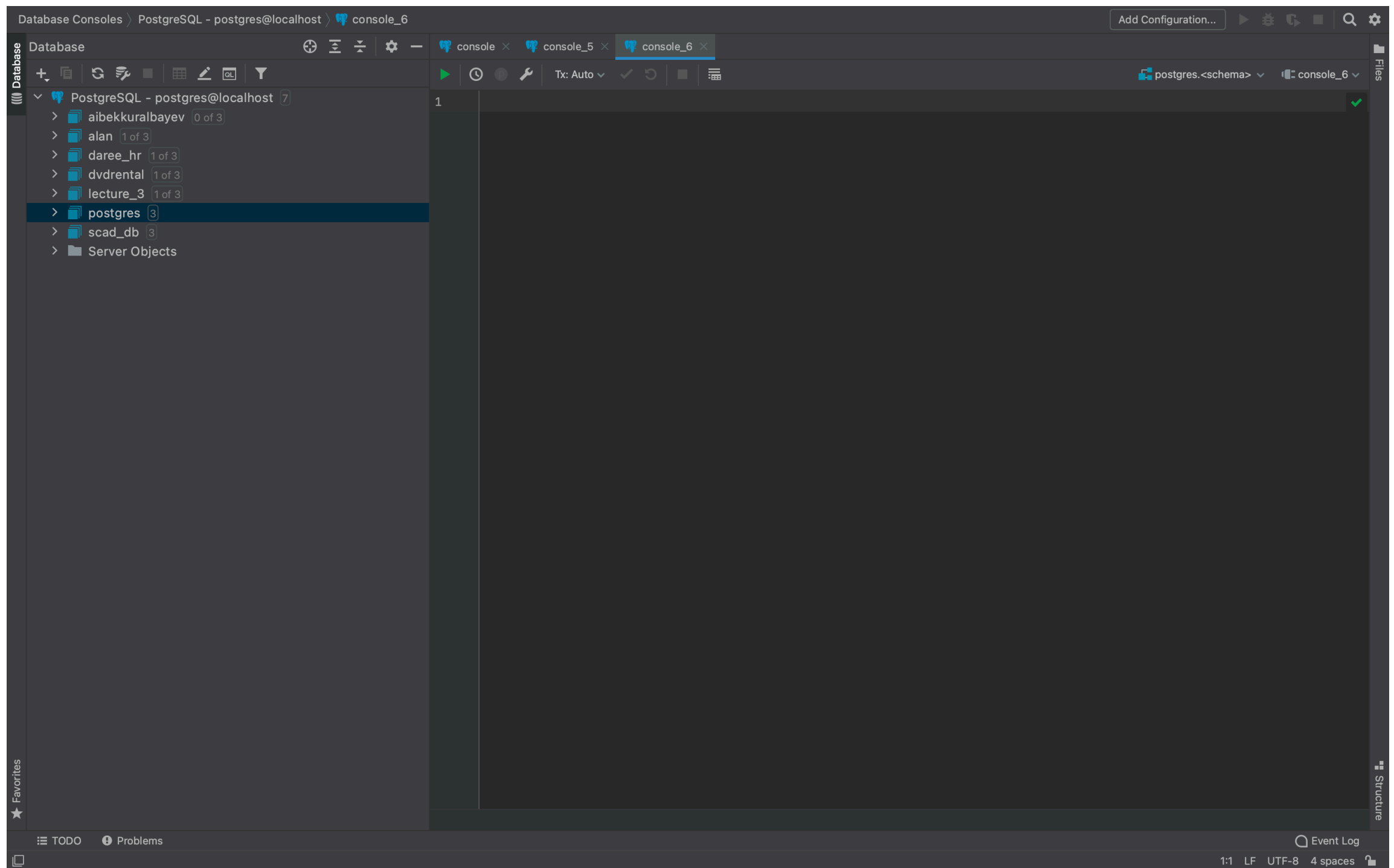
# psql

```
# aibekkuralbaev @ MacBook-Pro-Aibek in ~ [16:41:42]
$ psql
psql (9.6.4)
Type "help" for help.

aibekkuralbaev=# SELECT datname FROM pg_database;
      datname
-----
aibekkuralbaev
template1
template0
pillowzdb
mydb
exercises
(6 rows)

aibekkuralbaev=#
```

# Datagrip



# Where to Get PostgreSQL

- <https://www.postgresql.org/download/>
- <https://www.jetbrains.com/datagrip/download>
- <https://www.jetbrains.com/ru-ru/community/education/#students>