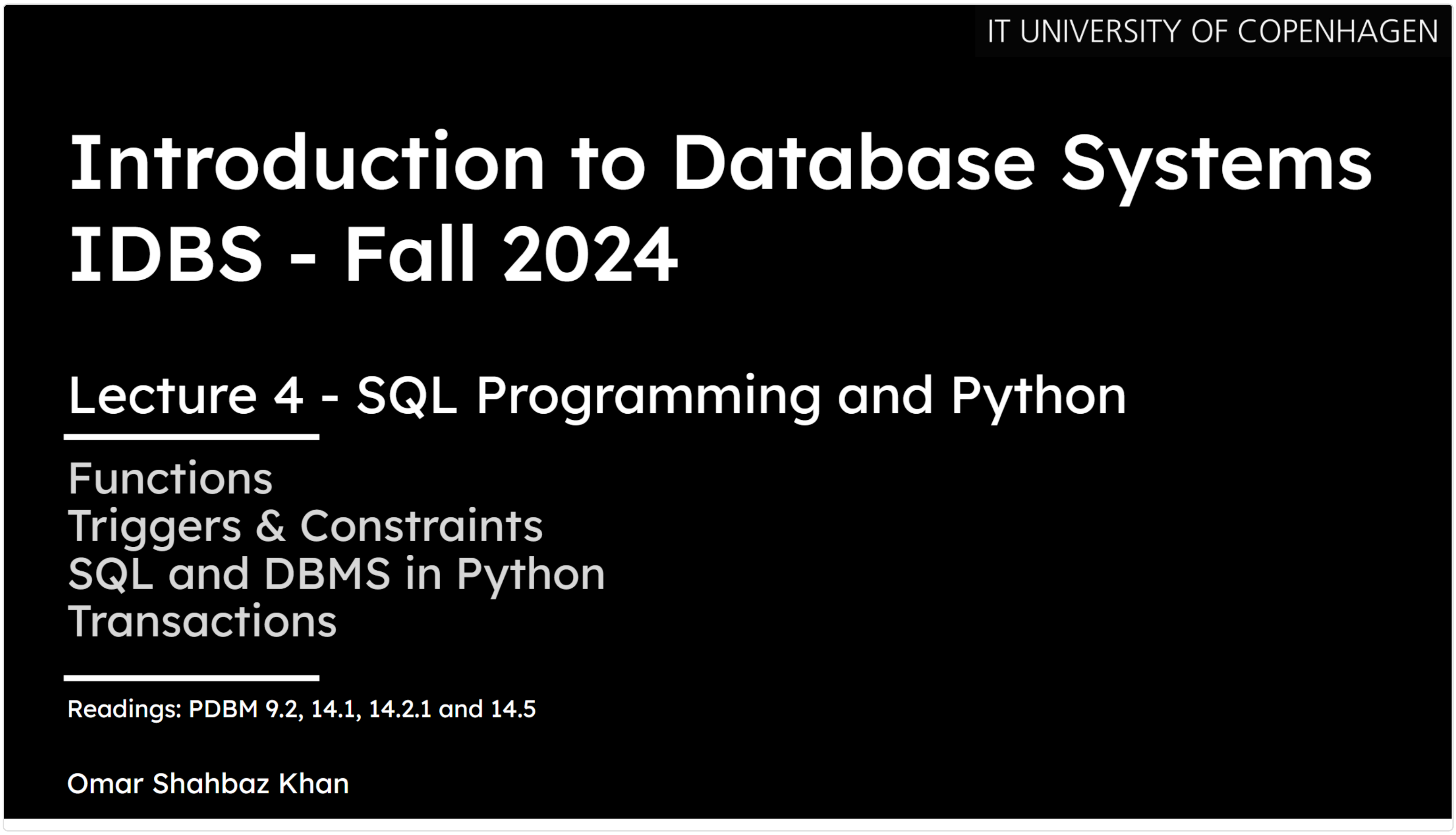
Relational Database Management Systems



IT UNIVERSITY OF COPENHAGEN

Introduction to Database Systems
IDBS - Fall 2024

Lecture 4 - SQL Programming and Python

Functions
Triggers & Constraints
SQL and DBMS in Python
Transactions

Readings: PDBM 9.2, 14.1, 14.2.1 and 14.5

Omar Shahbaz Khan

Database Definition

> A **database** is a collection of related data items within a specific business process or problem setting
> A **database system** provides a way to systematically organize, store, retrieve, and query a database

IDBS - Lecture 1, page 22

Relational Database

> A **relational database** is a type of database that is based on the The Relational Model
> - stores data in a set of tables with rows and columns (a.k.a. Relations)
> - uses relationships between these tables to manage the data.
> A relational database system (RDBMS) implements and manages relational databases

IDBS - Lecture 1, page 23

Three-Layer Architecture

- Presentation Tier
- Logic tier
- Data tier

> Traditional RDBMSs are passive, in the sense that they only execute transactions explicitly invoked by users and/or applications. Most modern-day RDBMSs are active, since they can autonomously take initiative for action if specific situations occur. Two key components of active RDBMSs are triggers and stored procedures.

Principles of database management the practical guide to storing, managing and analyzing big and small data - PDF Room, page 553

Active RDBMS

Functions

Functions in SQL allow encapsulating operations that can be reused across multiple queries and procedures. They help achieve data and functional independence by allowing applications to share and invoke functions stored within the DBMS environment. This reduces code redundancy and improves maintainability across applications.
Syntax

```
CREATE FUNCTION function_name(parameters)
RETURNS data_type AS $$
BEGIN
    -- function body
END;
$$ LANGUAGE plpgsql;
```

Example
Define a function to calculate employee bonus:

```
CREATE FUNCTION calculate_bonus(salary DECIMAL, bonus_rate DECIMAL)
RETURNS DECIMAL AS $$
BEGIN
    RETURN salary * bonus_rate;
END;
$$ LANGUAGE plpgsql;
```

Triggers

Triggers are automatically activated stored procedures that run in response to specific events on a table, such as `INSERT`, `UPDATE`, or `DELETE`. Unlike Functions, which are explicitly called, triggers fire autonomously based on defined conditions, allowing enforcement of complex integrity constraints across relational tables.

- Triggers run in alphabetical order
- To declare a trigger, the Functions has to return `TRIGGER`
  Syntax

```
CREATE TRIGGER trigger_name
AFTER | BEFORE trigger_event ON table_name
FOR EACH ROW
```

```
WHEN (condition)
EXECUTE FUNCTION function_name();
```

Example
Suppose an organization needs to automatically update a department’s total salary when a new employee is added. Here’s how the trigger might look:

```
CREATE TRIGGER update_department_salary
AFTER INSERT ON employee
FOR EACH ROW
WHEN (NEW.department_id IS NOT NULL)
EXECUTE FUNCTION update_salary();
```

When To Use Trigger Types

These triggers should be used in the following scenarios:

1. Are you only checking the newly inserted/updated entry? -> **BEFORE (or AFTER)**
   - Checking happens earlier with BEFORE, so less work
   - But if other triggers might modify the values, then prefer AFTER
2. Are you inserting a row to another table with a foreign key constraint to the NEW record? -> **AFTER**
   - Otherwise, the NEW record is NOT in the database, so your insertion will fail!
3. Are you modifying the NEW record? -> **BEFORE**
   - Otherwise, the record is already in the database and will not be changed
4. Are you doing both 2 and 3? -> **BEFORE and AFTER**
   - Two different triggers!

Trigger Variables

NEW and OLD are special variables that you can use with PL/SQL triggers without explicitly defining them.

- `NEW` - Refers to the new table row for insert and update operations in row-level triggers
  - When used in a before row-level trigger, the initial content of :NEW.column is the column value in the new row that is to be inserted or in the row that is to replace the old row.
  - When used in an after row-level trigger, the new column value has already been stored in the table.
  - When a trigger is activated by a DELETE operation, the :NEW.column used in that trigger is null.
- `OLD` - Refers to the old table row for update and delete operations in row-level triggers.
  - When used in a before row-level trigger, the initial content of :OLD.column is the column value in the row that is to be deleted or in the old row that is to be replaced by the new row.
  - When used in an after row-level trigger, the old column value is no longer stored in the table.
  - When a trigger is activated by an INSERT operation, the :OLD.column used in that trigger is null.

If a trigger `RETURN NEW` on af `AFTER` trigger; The return value is ignored.
A `AFTER` trigger is NOT valid if the `NEW` is getting changed.

Constraints

Types of Constraints

- **Primary Key**: Uniquely identifies each record.
- **Foreign Key**: Enforces a link between two tables.

- **Unique Constraint**: Ensures all values in a column are unique.
- **Check Constraint**: Ensures column values meet specified criteria.
  Example

```
CREATE TABLE employees (
    emp_id SERIAL PRIMARY KEY,
    emp_name VARCHAR(50) NOT NULL,
    emp_salary DECIMAL CHECK (emp_salary > 0),
    dept_id INT REFERENCES departments(dept_id)
);
```

Integrity and Validation

Constraints enforce data integrity rules and prevent invalid data entries, allowing databases to maintain consistency over time.

SQL w. Python

use the `psycopg` lib to conenct to a database