Databases

Software Development & Python

Albina Rukhadze, 2022

girafe
ai

girafe ai

SQL basics

SQL Data Types

Integer Types

- **`smallint`** – signed two-byte integer
- **`integer`** – signed four-byte integer
- **`bigint`** – signed eight-byte integer

Arbitrary Precision Numbers

- **`numeric[(p, s)] (= decimal[(p, s)])`**
  – exact numeric of selectable precision

Floating-Point Types

- **`real (= float4)`** – single precision floating-point number (4 bytes)
- **`double precision (= float8)`** – double precision floating-point number (8 bytes)

SQL Data Types

Character Types

- **`character varying(n), varchar(n)`** – variable-length with limit
- **`character(n), char(n)`** – fixed-length, blank padded
- **`text`** – variable unlimited length

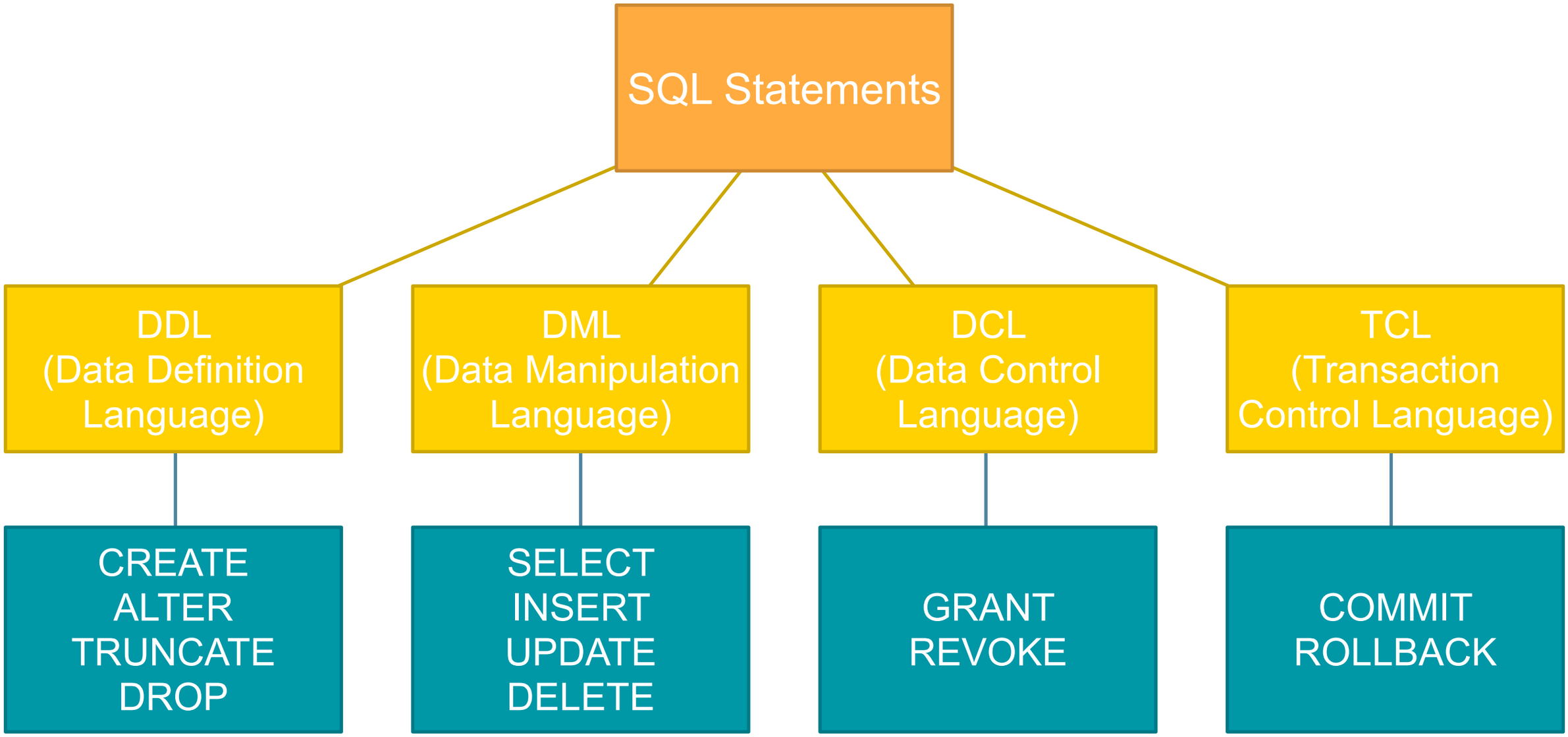
Date/Time Types

- **`date`** – date (no time of day)
- **`time`** – time of day (no date)
- **`timestamp`** – both date and time
- **`time with timezone`** – time of day (no date), with time zone
- **`timestamp with timezone`** – both date and time, with time zone
- **`interval`** – time interval

Boolean Types

- **`boolean`** – state of true or false

SQL Operators



Data Definition Language

Database objects creation, modification
and deleting

girafe
ai

CREATE

– define a new table

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name(
    col_name_1   datatype_1,
    col_name_2   datatype_2,
    …
    col_name_N   datatype_N
);
```

```
CREATE TABLE films (
    code          char(5) PRIMARY KEY,
    title         varchar(40) NOT NULL,
    did           integer NOT NULL,
    date_prod     date,
    kind          varchar(10),
    len           interval hour to minute
);
```

ALTER

– change the definition of a table

```
ALTER TABLE table_name ADD column_name datatype;
ALTER TABLE table_name DROP column_name;
ALTER TABLE table_name RENAME column_name TO new_name;
ALTER TABLE table_name ALTER column_name TYPE datatype;
…
```

```
ALTER TABLE distributors ADD COLUMN address varchar(30);

ALTER TABLE transactions
  ADD COLUMN status varchar(30) DEFAULT 'old',
  ALTER COLUMN status SET default 'current';
```

DROP

– remove a table

```
DROP TABLE [IF EXISTS] table_name;
```

```
DROP TABLE films, distributors;
```

TRUNCATE

– empty a table or set of tables (remove all rows from a set of tables)

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE bigtable, fattable;
```

Data Manipulation Language

Inserting, deleting and modifying data
in a database

girafe
ai

SELECT

– retrieve rows from a table or view

```
SELECT [DISTINCT] select_item_comma_list  -- columns output list
FROM table_reference_comma_list           -- table list
-- filter conditions, AND/OR/NOT may be used
[WHERE conditional_expression]
[GROUP BY column_name_comma_list]     -- grouping condition
-- filter conditions after grouping
[HAVING conditional_expression]
[ORDER BY order_item_comma_list];     -- output sorting columns
```

```
SELECT *
FROM films
ORDER BY title;
```

Query execution order

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

INSERT

– create new rows in a table

```
INSERT INTO table_name [(comma_separated_column_names)]
     VALUES (comma_separated_values);
```

```
INSERT INTO films VALUES
    ('UA502', 'Bananas', 105, DEFAULT,
     'Comedy', '82 minutes');
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama');
```

UPDATE

– update rows of a table

```
UPDATE table_name
    SET update_assignment_comma_list
WHERE conditional_expression;
```

```
UPDATE films
    SET kind = 'Dramatic'
WHERE kind = 'Drama';
```

DELETE

– delete rows of a table

```
DELETE
     FROM table_name
[WHERE conditional_expression];
```

```
DELETE
     FROM films
WHERE kind <> 'Musical';
```

girafe ai

Profound SQL

Aggregate Functions

Calculating statistics on a set of values

girafe
ai

Standard Aggregate Functions

– when grouping, the **`SELECT`** block may contain either attributes by which grouping occurs, or attributes that are input to aggregating functions.

- **`count()`** – the number of records with a known value, **`count(DISTINCT field_nm)`** may be used to count the number of unique values;
- **`max()`** – the largest of all selected field values;
- **`min()`** – the least of all selected field values;
- **`sum()`** – sum of all selected field values;
- **`avg()`** – average of all selected field values.

Note: **`NULL`**-values are ignored

Joining Tables

Merge tables on a given condition

girafe
ai

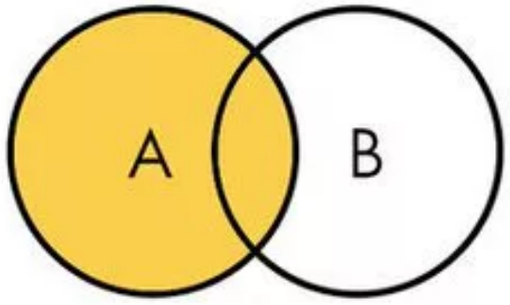
Join Operators

Join operators are divided into 3 groups:

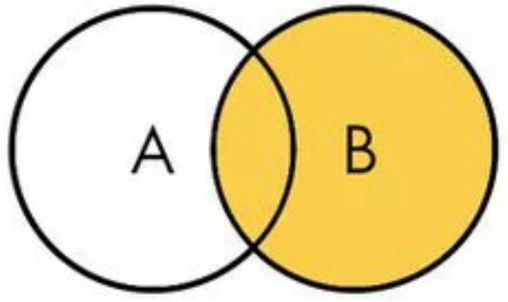
- **`CROSS JOIN`** – Cartesian product of 2 tables;
- **`INNER JOIN`** – joining 2 tables by condition. Query result will include only records that satisfy the connection condition;
- **`OUTER JOIN`** – joining 2 tables by condition. Query result may include records that do not satisfy the connection condition:
  - **`LEFT (OUTER) JOIN`** – result contains all rows of the "left" table;
  - **`RIGHT (OUTER) JOIN`** – result contains all rows of the "right" table;
  - **`FULL (OUTER) JOIN`** – result contains all rows of both tables.

Join Operators

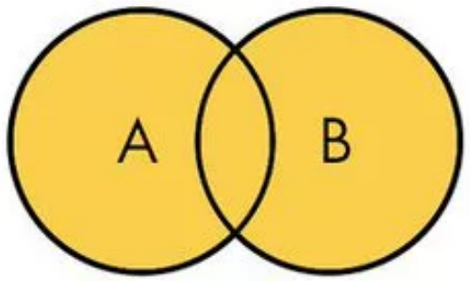
- **CROSS JOIN**
- **INNER JOIN**
- **OUTER JOIN**
  - **LEFT (OUTER) JOIN**
  - **RIGHT (OUTER) JOIN**
  - **FULL (OUTER) JOIN**



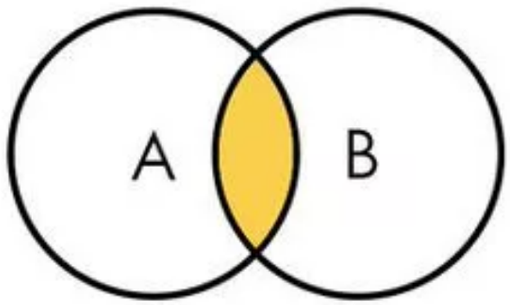

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
On A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

Join Operators

```
SELECT *
FROM weather INNER JOIN cities
     ON city = name;
```

```
     city      | temp_lo | temp_hi | prcp |    date    |     name      | location
---------------+---------+---------+------+------------+---------------+-----------
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San Francisco | (-194,53)
(2 rows)
```

Join Operators

```
SELECT *
FROM weather LEFT OUTER JOIN cities
    ON weather.city = cities.name;
```

```
     city      | temp_lo | temp_hi | prcp |    date    |     name      | location
---------------+---------+---------+------+------------+---------------+----------
 Hayward       |      37 |      54 |      | 1994-11-29 |               |
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San Francisco | (-194,53)
(3 rows)
```

Join Operators

```
SELECT *
FROM weather RIGHT OUTER JOIN cities
    ON weather.city = cities.name;
```

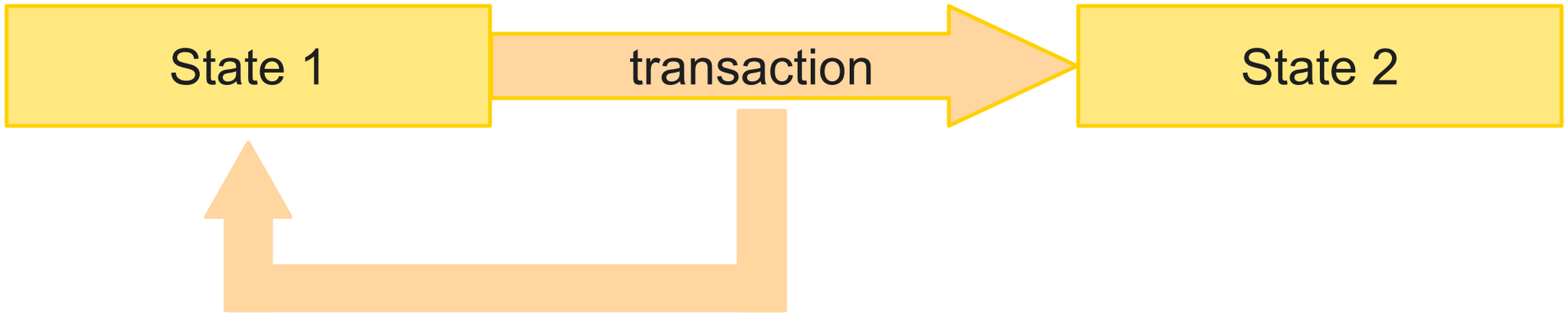
```
     city      | temp_lo | temp_hi | prcp |    date    |     name      | location
---------------+---------+---------+------+------------+---------------+----------
               |         |         |      |            | Tokyo         | (139,35)
 San Francisco |      46 |      50 | 0.25 | 1994-11-27 | San Francisco | (-194,53)
 San Francisco |      43 |      57 |    0 | 1994-11-29 | San Francisco | (-194,53)
(3 rows)
```

girafe ai

Transactions

Transaction

– an object that groups a sequence of operations that must be performed as a whole. Ensures the transition of the database from one consistent state to another.



Transactions ensure the integrity of the database under the conditions:

- Parallel data processing;
- Physical disk failures;
- Emergency power failure.

ACID

A set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps

girafe ai

Atomicity (ACID)

- A transaction must be an atomic (indivisible) unit of work
- Either all operations included in the transaction must be performed, or none of them
- Therefore, if it is impossible to perform all operations, all the changes made must be cancelled

Consistency (ACID)

- Upon completion of the transaction, all data should remain in a consistent state
- When executing a transaction, all the rules of a relational DBMS must be followed:
  - Checks for compliance with restrictions (domains, uniqueness indexes, foreign keys, checks, rules, etc.)
  - Updating indexes;
  - Executing triggers
  - etc.

Isolation (ACID)

- Data changes performed within a transaction must be isolated from all changes performed in other transactions until the transaction is committed
- There are different levels of isolation – to achieve a compromise between the degree of parallelization of work with the database and the rigor of the principle of consistency:
  - The higher the isolation level, the higher the degree of data consistency
  - The higher the isolation level, the lower the degree of parallelization and the lower the degree of data availability

Durability (ACID)

- If a transaction has been completed, its result should be saved in the system, despite the system failure
- If the transaction has not been completed, its result must be completely canceled following a system failure

Transaction Control Language

Transactions managing

girafe
ai

BEGIN

– set a transaction block

```
BEGIN TRANSACTION transaction_mode [, ...]
    ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ |
                      READ COMMITTED | READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [NOT] DEFERRABLE
```

COMMIT

– commit the current transaction

```
COMMIT [ WORK | TRANSACTION ] [ AND [ NO ] CHAIN ]
```

ROLLBACK

– abort the current transaction

```
ROLLBACK [ WORK | TRANSACTION ] [ AND [ NO ] CHAIN ]
```

SAVEPOINT

– define a new savepoint within the current transaction

```
SAVEPOINT savepoint_name
```

Transaction Operators

```
BEGIN;
    INSERT INTO table_name VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table_name VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table_name VALUES (3);
COMMIT;
```

Transaction Isolation

Classification of problems and isolation levels

girafe
ai

Problem Classification

girafe ai

Dirty read

– a transaction reads data written by a concurrent uncommitted transaction.

Transaction 1	Transaction 2
`UPDATE table_1 SET attr_2 = attr_2 + 20 WHERE attr_1 = 1;`	
	`SELECT attr_2 FROM table_1 WHERE attr_1 = 1;`
`ROLLBACK WORK;`	

Nonrepeatable read

– a transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

Transaction 1	Transaction 2

	`SELECT attr_2`
	`FROM table_1`
	`WHERE attr_1 = 1;`

`UPDATE table_1`
`SET attr_2 = attr_2 + 20`
`WHERE attr_1 = 1;`	

`COMMIT;`	

	`SELECT attr_2`
	`FROM table_1`
	`WHERE attr_1 = 1;`

Phantom read

– a transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.

Transaction 1	Transaction 2
	`SELECT sum(attr_2) FROM table_1;`
`INSERT INTO table_1 (attr_1, attr_2) VALUES (15, 20);`	
`COMMIT;`	
	`SELECT sum(attr_2) FROM table_1;`

Serialization anomaly

– the result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

Transaction 1	Transaction 2
`UPDATE table_1 SET attr_2 = attr_2 + 20 WHERE attr_1 = 1;`	`UPDATE table_1 SET attr_2 = attr_2 + 25 WHERE attr_1 = 1;`

Isolation Level

girafe
ai

Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
0. Read Uncommitted	possible	possible	possible	possible
1. Read Committed	-	possible	possible	possible
2. Repeatable Read	-	-	possible	possible
3. Serializable	-	-	-	-

girafe ai

Databases in Python

DB types, ORM libraries

Basic Concepts

Python DB API – the standard of interfaces for packages working with the database. A set of rules that individual modules implementing work with specific databases must comply with.

Database	Interface
SQLite	**sqlite3**
PostgreSQL	**psycopg2**
MySQL	**PyMySQL**
Oracle	**cx_Oracle**

Basic Concepts

Benefits of ORM usage:

- no need to write queries in pure SQL
- developer may abstract from thoughts about the internal structure of the database

Possible difficulties:

- ORM may work much slower than direct interaction with the database
- Writing queries in ORM syntax can be technically much more difficult

ORM (Object Relational Mapping)

Python DB API – the standard of interfaces for packages working with the database. A set of rules that individual modules implementing work with specific databases must comply with.

Database	Interface
SQLite	**sqlite3**
PostgreSQL	**psycopg2**
MySQL	**PyMySQL**
Oracle	**cx_Oracle**

SQLite3

Main features:

- Lightweight solution to use a database
- All database is stored in one file
- Suitable for mobile apps
- Non-scalable solution

Peewee

Main features:

- Built-in support for SQLite, MySQL and PostgreSQL
- Lightweight, flexible and expressive
- Low entry threshold

SQLAlchemy

Advantages:

- SQL injection protection: query parameters are escaped
- Scalability and flexibility
- Ability to use both pure SQL and ORM syntax
- Large-scale solution

girafe
ai