COMP9120

Week 7: Introduction to Database Application Development

Semester 1, 2025

Today's content will **not** be covered in exam or quiz. It is only for assignment2.

Professor Athman Bouguettaya
School of Computer Science

Warming up

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. We are currently on the land of the Gadigal people of the Eora nation and pay our respects to their Elders, past, present and emerging.

> **Introduction to Database Application Development**

> **DB Application Development in Java and Python**

> **Error Handling in Java and Python**

> **Security and SQL Injection**

> **Stored Procedure**

› Two ways:

- SQL used in other functions/procedures of a programming language

  - Example, Java/Python using SQL and database accessed through JDBC/Psycopg – called host languages

- SQL using procedures/functions from a programming language

  - Example: PL/pgSQL – called stored procedures.

› Main Advantages:

- Elimination of additional round trips between client and server

- Intermediate results that the client does not need, do not need transferred between server and client

› *Interactive SQL*: SQL statements input from terminal; DBMS outputs to screen

- Inadequate for many uses

  - It may be necessary to process the data before output

  - Amount of data returned not known in advance

› *Non-interactive SQL*: SQL statements are included in a database application program written in a host language, like C++, Java, Python, PHP

› *Client-side* vs. *Server-side* application development

- Client-side: database application program

- Server-side: stored procedures and triggers

centralized system

| presentation services | application services | API | DBMS |

user module

› *Presentation Services* - displays forms, handles flow of information to/from screen

› *Application Services* - implements user request, interacts with DBMS

› *DBMS* handles user requests – retrieves and returns results of queries or handles user requests for inserts/updates/deletes of data.

› Examples: Any application with integrated DB

- MS Access systems

- SQLite, esp. Smartphone apps

client machines

database server machine

presentation services | application services

presentation services | application services

DBMS

communication / network

client machines

application / web server machine

database server machine

presentation server

• • •

presentation server

application server

DBMS

communication (IPC or network)

Presentation Tier

Middle Tier

Data Management Tier
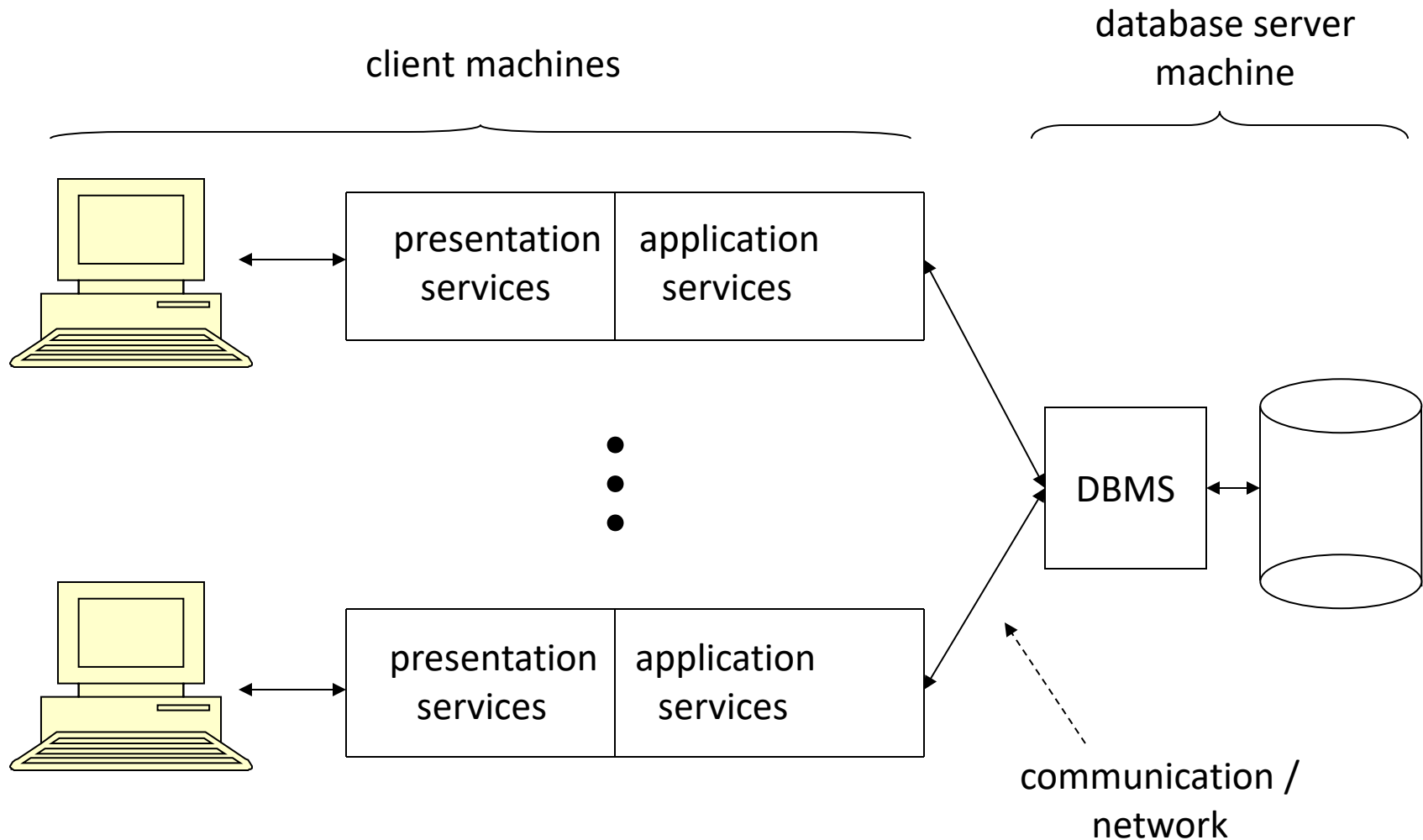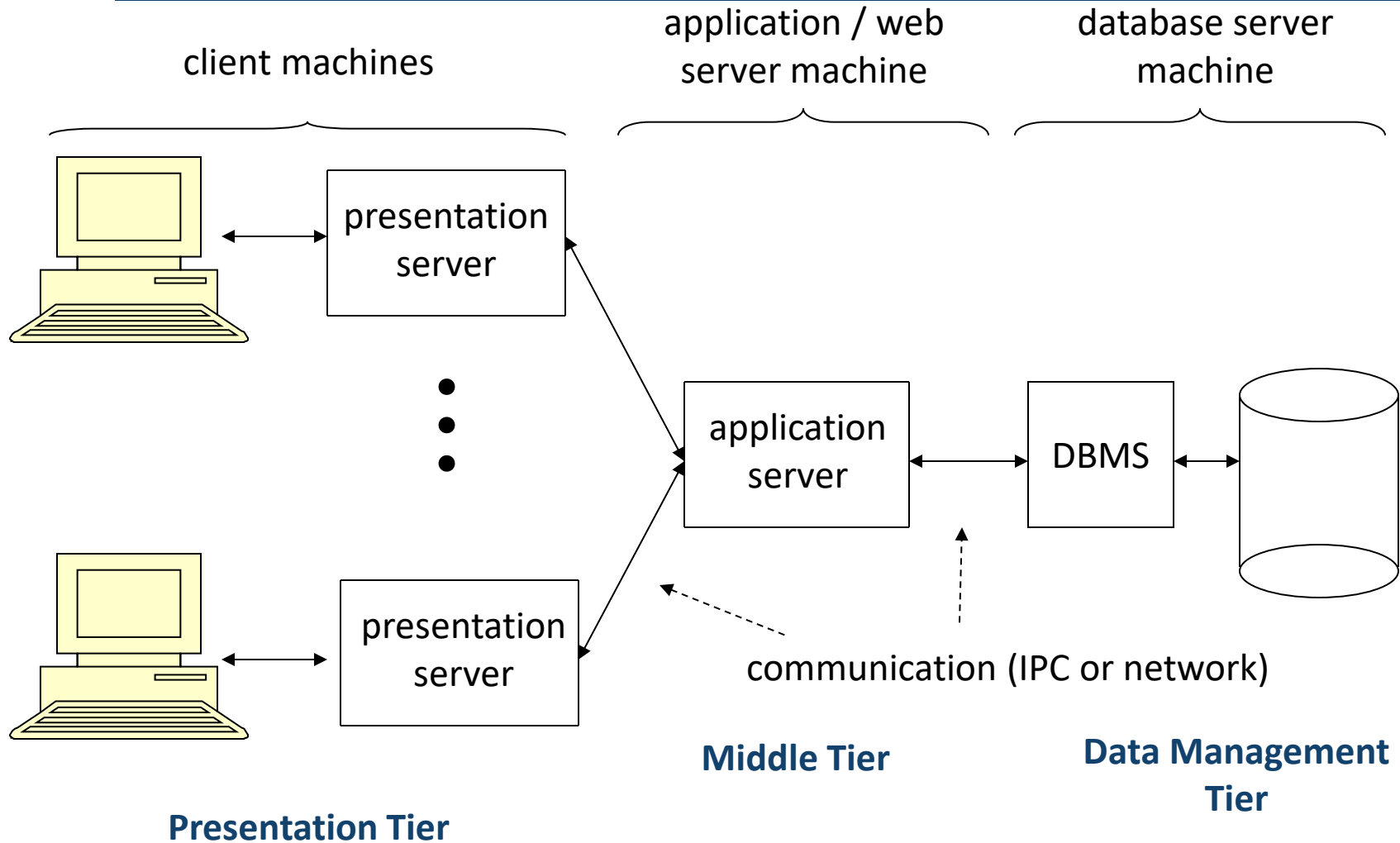
› SQL commands can be called from within a *host language* (e.g., C++, Java, PHP, Python) program

- SQL statements can refer to host variables

- Must include a statement to *connect* to the right database

```
int userInput = takeUserInput();

SELECT *
FROM student
WHERE sid = userInput
```

› Two main integration approaches

- Statement-level interface (SLI)

  - Embed SQL in the host language (Embedded SQL in C, SQLJ)

- Call-level interface (CLI)

  - Create special API to call SQL commands (ODBC, JDBC, PHP-PDO, etc.)

  - SQL statements are passed as arguments to host language (library) procedures / APIs

› **Introduction to Database Application Development**

› **DB Application Development in Java and Python**

› **Error Handling in Java and Python**

› **Security and SQL Injection**

› **Stored Procedure**

› JDBC is a **Java API** for communicating with database systems supporting SQL

› JDBC supports a variety of features for querying and updating data, and for retrieving query results

› Model for communicating with the database:

- Acquire a connection

- Create a "Statement" object

- Execute queries using the Statement object to send queries and fetch results

- Exception mechanism to handle errors
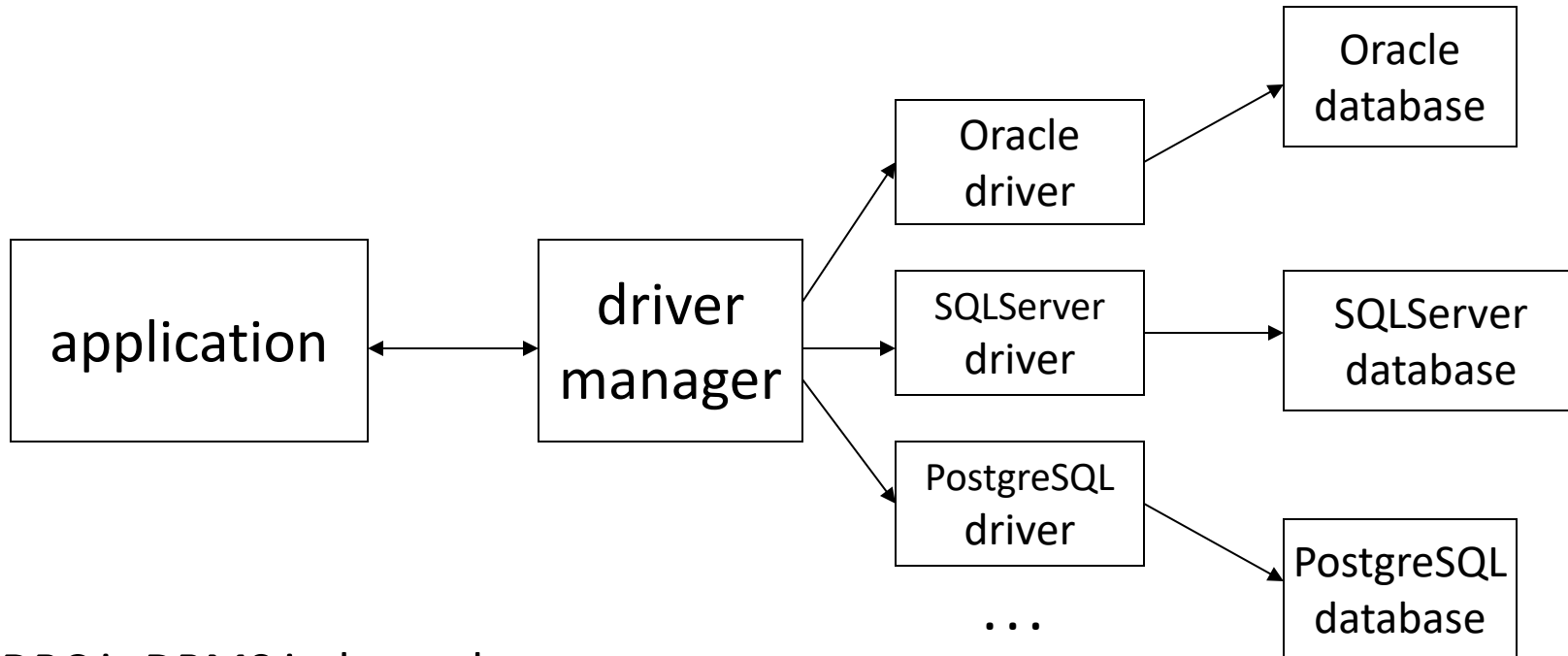
> JDBC is DBMS independent

- JDBC functions are generic

- DriverManager allows to connect to specific driver

  - Even to different databases from the same program

› A session with a data source is started through the creation of a **Connection** object

- example with PostgreSQL

- Always release resource with conn.**close**();

› SQL operations are conducted using **java.sql.Statement**

  - Constructed from a Connection object:

    Statement stmt = conn.**createStatement**();

  - Execute a SQL query:

    stmt.**executeQuery**("**SELECT** ... ");

  - Execute a DML statement (INSERT/UPDATE/DELETE)

    stmt.**executeUpdate**("**INSERT INTO** ... ");

  - Release resource with stmt.**close**();

› Two other ways (to be covered later)

  - *PreparedStatement* (semi-static SQL statements)

  - *CallableStatement* (stored procedures)

› stmt.**executeQuery** returns data, encapsulated in a **ResultSet** object (a cursor)

```
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()) {

    //Iterate through records
   // process the data

     rs.getString("name");
}
rs.close() // Release resources
```

› A **ResultSet** can be a very powerful cursor:

- **previous**(): moves one row back

- **absolute**(**int** num): moves to the row with the specified number

- **relative**(**int** num): moves forward or backward

- **first**() and **last**(): jump to ends

- **wasNull**(): dealing with NULL values

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

For python matching example: http://initd.org/psycopg/docs/usage.html#adaptation-of-python-values-to-sql-types

› **Python's API (DB-API)** for communicating with database systems supporting SQL

- Specific *modules* for each db engine (eg: Oracle, Postgres, IBM DB2, etc) provide an implementation for common DB-API functionality

› Model for communicating with the database:

- Acquire a connection

- Create a "cursor" object

- Execute queries using the cursor object to send queries and fetch results
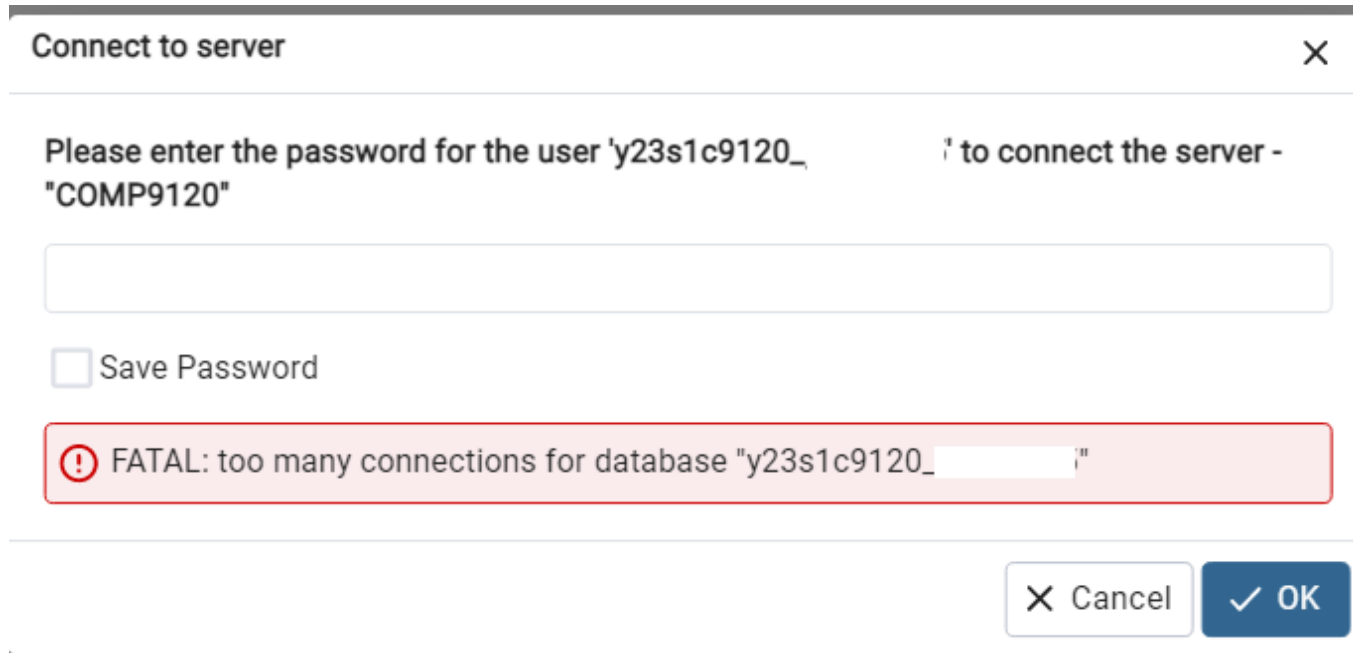
- Exception mechanism to handle errors

```python
import psycopg2

try:
    # fetch connection object to connect to the database
    conn = psycopg2.connect(database="postgres",user="test",password="secret", host="host")

    # fetch cursor prepare to query the database
    curs = conn.cursor()

    # execute a SQL query
    curs.execute("SELECT name
                  FROM Student NATURAL JOIN Enrolled
                  WHERE uos_code = 'COMP9120'")

    # can loop through the resultset
    for result in curs:
        print (" student: " + result[0])

    # for illustrating close methods – calling close() on cursor and connection
    # objects will release their associated resources.
    curs.close()
    conn.close()

except Exception as e: # error handling
    print("SQL error: unable to connect to database or execute query")
    print(e)
```

› **execute**(operation[,parameters])

- Execute a query or a command with the sql provided as the operation argument;

- the sql query parameter values provided in the parameters argument.

› **executemany**(operation[,parameters])

- Can execute the same query (operation) multiple times, each time with a different parameter set

› **fetchone**()

- Fetch the next row of a query result set, or returns None if no more data is available

› **fetchmany**([size=cursor.arraysize])

- Return multiple rows from the result set for a given query, where one can specify the desired number of rows to return (where they are available)

› **fetchall**()

- fetch all remaining rows

› **close**()

Reference: https://www.psycopg.org/docs/cursor.html

- Release cursor resources

› **Introduction to Database Application Development**

› **DB Application Development in Java and Python**

› **Error Handling in Java and Python**

› **Security and SQL Injection**

› **Stored Procedure**

THE UNIVERSITY OF SYDNEY

Bine ati venit in magazinul eSIMM'S.

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

INVALID SQL: 1016 : Can't op
SQL QUERY FAILURE: SELEC

Association for Computing Machinery
Advancing Computing as a Science & Profession

ACM Order Rectification

The web site you are accessing has experienced an unexpected error. Please contact the website administrator.

The following information is meant for the website developer for debugging purposes.

Error Occurred While Processing Request

Element ORDERID is undefined in URL.

The error occurred in **D:\wwwroot\Public\rectifyCC\rectifyCC.cfm: line 463**

```
461 :      WHERE a.order_id = b.order_id
462 :        AND a.order_id = c.order_id
463 :        AND a.order_id = '#URL.orderID#'
464 :    </CFQUERY>
465 :
```

Resources:

- Check the ColdFusion documentation to verify that you are using the correct syntax.
- Search the Knowledge Base to find a solution to your problem.

Browser   Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_8; en-us) AppleWebKit/531.9 (KHTML, like Gecko) Version/4.0.3 Safari/531.9

Remote Address   129.78.220.7

Cauta:

24

› Most of java.sql can throw an **SQLException** if an error occurs.

- Catch and process with **catch**(**SQLException** e) { **…** }

- Use **getMessage**()or **getSQLState**() or **getErrorCode**() to identify problem

Table 12. Class Code 23: Constraint Violation

SQLSTATE Value	Meaning
23001	The update or delete of a parent key is prevented by a RESTRICT update or delete rule.
23502	An insert or update value is null, but the column cannot contain null values.
23503	The insert or update value of a foreign key is invalid.
23504	The update or delete of a parent key is prevented by a NO ACTION update or delete rule.
23505	A violation of the constraint imposed by a unique index or a unique constraint occurred.
23510	A violation of a constraint on the use of the command imposed by the RLST table occurred.
23511	A parent row cannot be deleted, because the check constraint restricts the deletion.
23512	The check constraint cannot be added, because the table contains rows that do not satisfy the constraint definition.
23513	The resulting row of the INSERT or UPDATE does not conform to the check constraint definition.
23514	Check data processing has found constraint violations.
23515	The unique index could not be created or unique constraint added, because the table contains duplicate values of the specified key.
23520	The foreign key cannot be defined, because all of its values are not equal to a parent key of the parent table.
23521	The update of a catalog table violates an internal constraint.

› Most of java.sql can throw an **SQLException** if an error occurs.

- Catch and process with **catch**(**SQLException** e) { **…** }

- Use **getMessage**()or **getSQLState**() or **getErrorCode**() to identify problem

› Sub-classes available to catch specific types e.g.:

- **SQLTimeoutException**

- **SQLIntegrityConstraintViolationException**

› **SQLWarning** is a subclass of **SQLException**; not as severe

- conn.**getWarnings**()**;**

- conn.**getNextWarning**();

- conn.**clearWarnings**();

The following function may fail to work for a number of reasons.

```
void exampleEnrolment() {
    PGSimpleDataSource source = new PGSimpleDataSource();
    source.setServerName(myHost);
    source.setDatabaseName(myDB);
    source.setUser(userid);
    source.setPassword(passwd);
    Connection conn = source.getConnection();

    Statement stmt = conn.createStatement();
    stmt.executeUpdate("INSERT INTO Transcript VALUES
      (123,'COMP9120', 'S1', 2024,'HD')");

    conn.close();
}
```

May fail due to server/connection problems

› Query could time out
› Primary key violation
› Foreign key violation

28

```java
void exampleEnrolment() {
 Connection conn = null;
 try {
   conn = openConnection();
   Statement stmt = conn.createStatement();
   stmt.executeUpdate("INSERT INTO Transcript VALUES (123,'COMP9120', 'S1', 2024,'HD')");
 }
 catch (SQLIntegrityConstraintViolationException e) {
         System.err.println("Violated a constraint!");
 }
 catch (SQLTimeoutException e) {
         System.err.println("Operation timed out");
 }
 catch (SQLException e) {
         System.err.println("Other problem");
 }
finally {
   if (conn != null)
     try{conn.close();} catch(SQLException e) {//handle exception}
 }
}
```

› Error handling via normal exception mechanism of Python

- Errors and warnings are made available as Python exceptions

  - **Warning**   raised for warnings such as data truncation on insert, etc

  - **Error**   exception raised for various db-related errors

› psycopg API extension:

- Exception attributes for detailed SQL error codes and messages

  - **pgerror**   string of the error message returned by backend

  - **pgcode**   string with the **SQLSTATE** error code returned by backend

› Example:

```
try:
   psycopg2.connect(…)
except psycopg2.Error as e:
   print("Problem connecting to database:")
   print(e.pgerror)
   print(e.pgcode)
```

Demo purpose only.
please do not directly print SQL exceptions :)

```
StandardError
|__ Warning
|__ Error
     |__ InterfaceError
     |__ DatabaseError
          |__ DataError
          |__ OperationalError
          |    |__ psycopg2.extensions.QueryCanceledError
          |    |__ psycopg2.extensions.TransactionRollbackError
          |__ IntegrityError
          |__ InternalError
          |__ ProgrammingError
          |__ NotSupportedError
```

http://initd.org/psycopg/docs/module.html#exceptions

Also see: https://www.python.org/dev/peps/pep-0249

Lightning talk

by

Neeraj Shetkar

Short break

Let us have some fun again…

THE UNIVERSITY OF
SYDNEY

› **Introduction to Database Application Development**

› **DB Application Development in Java and Python**

› **Error Handling in Java and Python**

› **Security and SQL Injection**

› **Stored Procedure**

User Name:	' or '1'='1
Password:	' or '1'='1

Details:

http://www.open.edu/openlearn/ocw/mod/oucontent/view.php?id=48319&section=1.3

Statement stmt = conn.**createStatement**();
ResultSet rs = stmt.**executeQuery**(
 "SELECT * **FROM** Student **WHERE**
 Name = '" + uName + "' **AND** Pass ='" + uPass + "'");

SELECT * **FROM** Student **WHERE**
Name ='John' **AND** Pass ='myPassword'

SELECT * **FROM** Student **WHERE**
Name ='' **or** '1'='1' **AND** Pass ='' **or** '1'='1'

Will return all students' details!!

35

User Name:	' or '1'='1
Password:	' or '1'='1

Details:

› How do we limit risks?

- Hide error messages that expose the internals

- Use *salted hash passwords*

- Prevent SQL injection with:

  - *Prepared Statements* in Java

  - *Anonymous or Named Parameters* in Python

› **java.sql.PreparedStatement** allows a statement to be executed with host variables after the query plan has been evaluated

› Example:

credit(**int** amount, **int** accountno) {
  stmt = conn.**prepareStatement**("**UPDATE** account
                              SET balance = balance + ?
                              WHERE account_number = ?");

  stmt.**setFloat**(1, amount);

  stmt.**setInt**(2, accountno);

  result = stmt.**executeQuery**();
}

› More flexible and <u>secure</u>

› As a rule, always use in preference to Statements

› What security issues can you identify in this function? How could they be fixed?

```java
void getStudentAddress(int studID, String password) {
  Connection conn = openConnection();

  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery(
          "SELECT address FROM Student WHERE studId='"
          + studID + "' AND password = '" + password + "'");

  // Process results
  while (rs.next()){
    System.out.println(rs.getString("address"));
  }
  conn.close();
}
```

Try:
getStudentAddress(307088592, "' OR 1=1 -- ");

SELECT address FROM Student WHERE studId='307088592' AND password = '' OR 1=1 –-

```java
void getStudentAddress(int studID, String password) {
 Connection conn = openConnection();

 PreparedStatement stmt = conn.prepareStatement(
  "SELECT address FROM Student WHERE studId=? AND password=?");

 stmt.setInt(1, studID);
 stmt.setString(2, password);

 ResultSet rs = stmt.executeQuery();

 // Process results
 while (rs.next()){
  System.out.println(rs.getString("address"));
 }
 conn.close();
}
```

NEVER ever use Python string concatenation (+) or string parameter interpolation (%) to pass variables to a SQL query string!
=> otherwise your program is vulnerable to SQL Injection attacks

```
query = """SELECT E.studId FROM Enrolled E
           WHERE E.uosCode = """ + uosCode +
           " AND E.semester = " + semester

cursor.execute( query )
```

Two (safe) approaches for passing query parameters:

1. Anonymous Parameters

```
studid = 12345
cursor.execute(
    "SELECT name FROM Student WHERE sid=%s",
    (studid) )
```

2. Named Parameters

```
studid = 12345
cursor.execute(
    "SELECT name FROM Student WHERE sid=%(sid)s",
    {'sid': studid} )
```

Resource: 1. https://realpython.com/prevent-python-sql-injection/
2.https://www.datacamp.com/tutorial/tutorial-postgresql-python
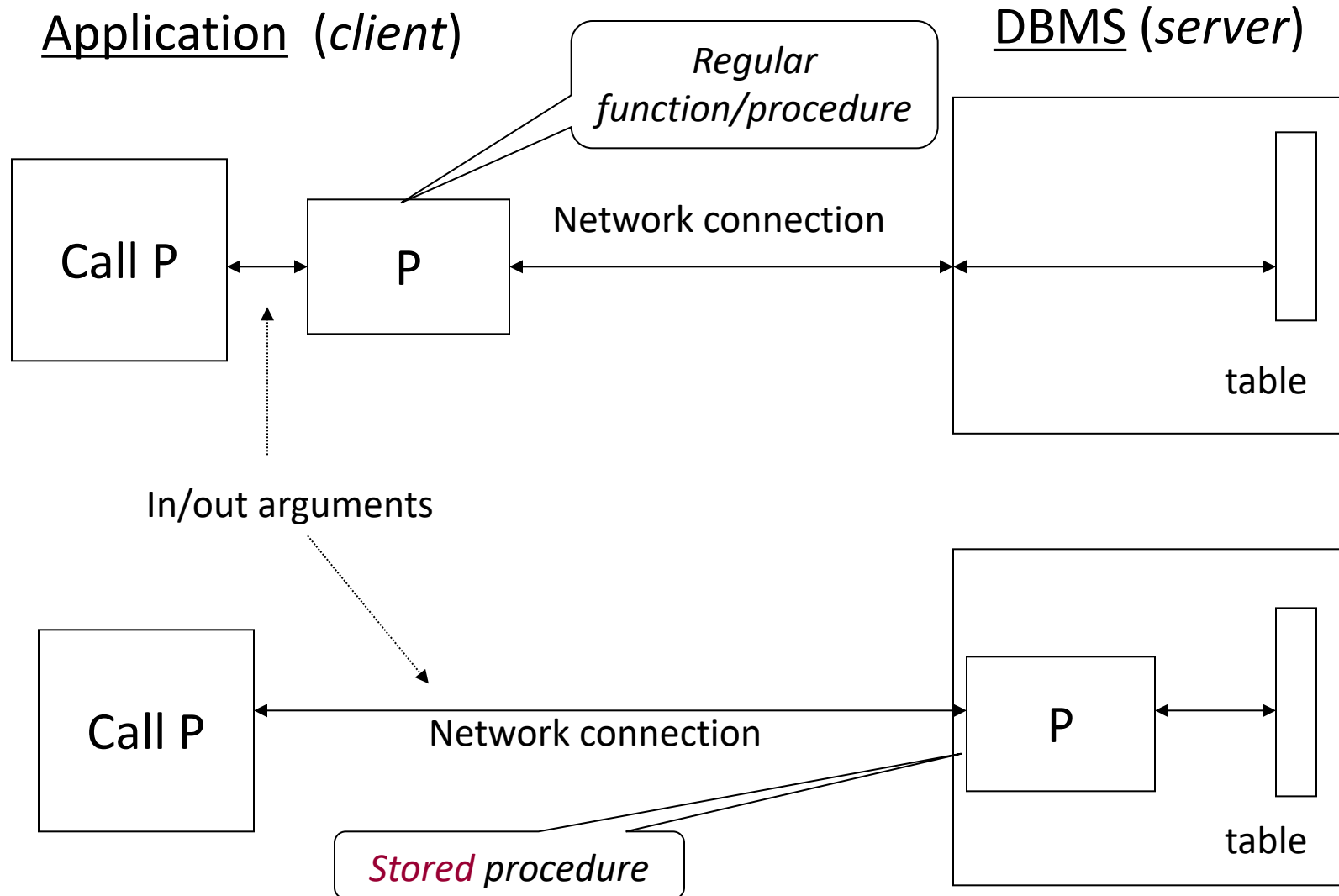3. https://www.psycopg.org/psycopg3/docs/basic/params.html

› **Introduction to Database Application Development**

› **DB Application Development in Java and Python**

› **Error Handling in Java and Python**

› **Security and SQL Injection**

› **Stored Procedure**

› Run logic within the database server

- Included as schema element (stored in DBMS)

- Invoked by the application

› Pros:

- Additional abstraction layer
  (programmers do not need to know the schema)

- Reduced data transfer

› Cons:

- What if you wanted to switch DBMS?

  - rewrite all logic?

Application (*client*)

DBMS (*server*)

Regular function/procedure

Call P

P

Network connection

table

In/out arguments

Call P

Network connection

P

table

Stored procedure

› All major database systems provide extensions of SQL to a simple, general-purpose language

- PostgreSQL: PL/pgSQL, Oracle: PL/SQL (syntax differs!!!)

› Procedure Declarations

CREATE PROCEDURE name (parameter1,…, parameterN)
local variable declarations
procedure code;

› Stored Procedures can have parameters

- of a valid SQL type (parameter types must match)

- three different modes

- **IN** arguments to procedure   Insert into students values(123, 'Adam', 'M', 'AUS');

- **OUT** return values   Select * from students;

- **INOUT** combination of **IN** and **OUT**   Select * from students where sid = 1005;

› Stored Procedures have full access to SQL, plus extensions:

- Local variables, loops, if-then-else conditions

```
CREATE FUNCTION RateStudent(studId INTEGER, uos VARCHAR) RETURNS CHAR AS $$
DECLARE
        grade       CHAR;
        marks       INTEGER;
BEGIN

        SELECT SUM(mark) INTO marks
        FROM Assessment
        WHERE sid=studId AND uosCode=uos;

        IF          marks>=85  THEN grade := 'H';
        ELSIF       marks>=75  THEN grade := 'D';
        ELSIF       marks>=65  THEN grade := 'C';
        ELSIF       marks>=50  THEN grade := 'P';
        ELSE                        grade := 'F';
        END IF;

        RETURN grade;
END; $$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION RateStudent_INOUT(IN studId INTEGER, IN uos VARCHAR, OUT result CHAR) AS $$
DECLARE
        grade       CHAR;
        marks       INTEGER;
BEGIN

        SELECT SUM(mark) INTO marks
        FROM Assessment
        WHERE sid=studId AND uosCode=uos;

        IF          marks>=85   THEN grade := 'H';
        ELSIF       marks>=75   THEN grade := 'D';
        ELSIF       marks>=65   THEN grade := 'C';
        ELSIF       marks>=50   THEN grade := 'P';
        ELSE                        grade := 'F';
        END IF;


        result := grade;
END; $$ LANGUAGE plpgsql;
```

› Use **java.sql.CallableStatement** subclass of Statement

› Calling a stored procedure with return value:

```
CallableStatement call = conn.prepareCall("{? = call RateStudent(?, ?)}");
call.registerOutParameter(1, Types.CHAR);
call.setInt(2, 101);
call.setString(3, "COMP9120");
call.execute();

String result = call.getString(1);
```

› Calling a stored procedure with IN/OUT parameter :

```
CallableStatement call = conn.prepareCall("{call RateStudent_INOUT(?, ?, ?)}");
call.setInt(1, 101);
call.setString(2, "COMP9120");
call.registerOutParameter(3, Types.CHAR);
call.execute();

String result = call.getString(3);
```

› **Cursor** objects have an explicit **callproc**()method

  - **cursor.callproc**() makes the OUT parameters available as resultset

› Calling a stored procedure with return value:

```
curs.callproc("RateStudent", [101, "comp9120"])
output = curs.fetchone()


result = output[0];
```

› Calling a stored procedure with IN/OUT parameter :

```
curs.callproc("RateStudent_INOUT", [101, "comp9120"])
output = curs.fetchone()


result = output[0];
```

› The following function performs several queries and updates, incurring several network round trips. Rewrite as a stored procedure and change the function to call this.

```
void enrolStudent(Connection conn, int studID, String uos, String sem, int year)
{
  PreparedStatement stmt = conn.prepareStatement("INSERT INTO Transcript
                                                  VALUES (?, ?, ?, ?, null)");
  stmt.setInt(1, studID);
  stmt.setString(2, uos);
  stmt.setString(3, sem);
  stmt.setInt(4, year);
  stmt.executeUpdate();
  stmt.close();

  stmt = conn.prepareStatement("UPDATE UoSOffering SET enrollment=enrollment+1
                                WHERE UoSCode=? AND Semester=? AND Year=?");
  stmt.setString(1, uos);
  stmt.setString(2, sem);
  stmt.setInt(3, year);
  stmt.executeUpdate();
  stmt.close();
}
```

```
CREATE OR REPLACE FUNCTION ENROLSTUDENT ( sid INTEGER, uos VARCHAR,
                                          sem VARCHAR,  yr  INTEGER) AS $$
BEGIN
 INSERT INTO Transcript VALUES (sid, uos, sem, yr, null);
 UPDATE UoSOffering SET enrollment=enrollment+1
                 WHERE uoSCode=uos AND semester=sem AND year=yr;
END; $$ LANGUAGE plpgsql;
```

```java
void enrolStudent(Connection conn, int studID, String uos,
                                   String sem, int year) {
 CallableStatement stmt = conn.prepareCall("{call ENROLSTUDENT (?, ?, ?, ?)}");
 stmt.setInt(1, studID);
 stmt.setString(2, uos);
 stmt.setString(3, sem);
 stmt.setInt(4, year);
 stmt.executeUpdate();
 stmt.close();
}
```

› Some Keywords

  - Cursors

  - Prepared Statement

  - Stored Procedure

  - SQL Injection

› Understanding (within a DB application API)

  - Error handling

› Skills

  - Write application code (e.g. Java functions with JDBC) to interact with a database

  - Identify and avoid major security flaws in client code

  - Write stored procedures and call from client code

› Ramakrishnan/Gehrke (3rd edition - the 'Cow' book)

- Chapter 6; 7.5

› Kifer/Bernstein/Lewis (2nd edition)

- Chapter 8

› Ullman/Widom (3rd edition of 'First Course in Database Systems')

- Chapter 9 (*covers Stored Procedures, ESQL, CLI, JDBC and PHP*)

Further Documentation:

- Java JDBC reference: http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html

- **PostgreSQL JDBC Documentation:** https://jdbc.postgresql.org/documentation/head/index.html

- Python DB-API: https://www.python.org/dev/peps/pep-0249/

- A Postgres Python DB-API Adapter: http://initd.org/psycopg/docs/

  - Connection pooling: http://initd.org/psycopg/docs/pool.html

- Database PL/pgSQL Language Reference: https://www.postgresql.org/docs/9.5/static/plpgsql.html

› Java & Python Language References:

- Java

  - https://www.ibm.com/developerworks/learn/java/intro-to-java-course/index.html

  - https://www.udemy.com/java-tutorial/

- Python

  - https://www.learnpython.org/

  - https://docs.python.org/2/tutorial/

› Schema Normalization Functional Dependencies

› Normal Forms

› Readings:**Ramakrishnan/Gehrke, Chapter 19**

› Kifer/Bernstein/Lewis book, Chapter 6

› Ullman/Widom, Chapter 3 (up-to 3.5)

SQL Database
PostgreSQL

SQL databases, also known as relational databases, are systems that store collections of tables and organize structured sets of data in a tabular columns-and-rows format, like a spreadsheet.

Why use it?

01
Structured Data

SQL excels with structured data, where relationships between entities are well-defined, making it ideal for applications like banking or inventory systems.

02
ACID Compliance

SQL databases ensure Atomicity, Consistency, Isolation, and Durability, guaranteeing data integrity during transactions—critical for financial systems.
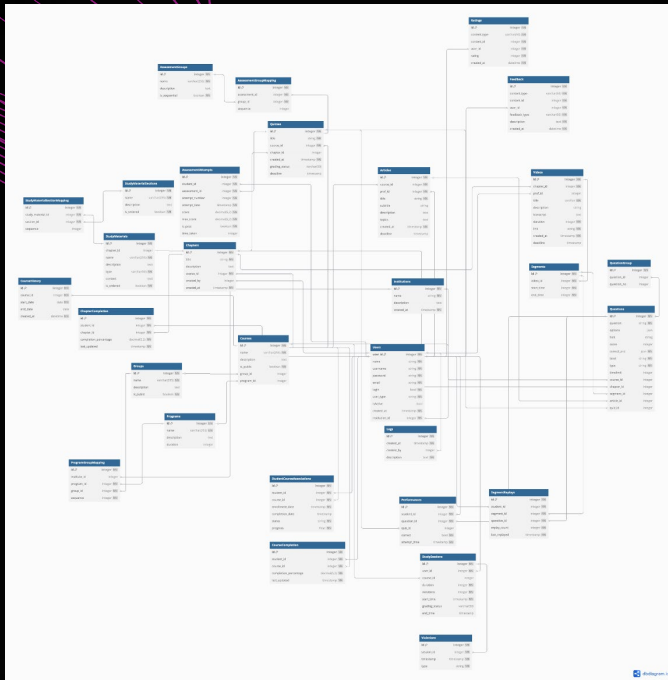
Why not to use it?

01
Rigid Schema

Modifying the schema can be challenging, often requiring downtime or complex migrations.

02
Performance with Unstructured Data

AI fuels groundbreaking discoveries, leading to new inventions, solutions, and advancements in various fields.

NoSQL
MongoDB
cluster

MongoDB is a document database. It stores data in a type of JSON format called BSON. A record in MongoDB is a document, which is a data structure composed of key value pairs similar to the structure of JSON objects.

```
_id: ObjectId('67e692d4b1484bda9067b03f')
prompt : "propane"
▾ response : Object
    name : "Propane"
    properties : "A three-carbon alkane, gas at standard temperature and pressure."
    description : "Propane (C3H8) is a saturated hydrocarbon with three carbon atoms link…"
    formula : "C3H8"
  ▸ atoms : Object
  ▾ bonds : Array (10)
    ▸ 0: Object
    ▸ 1: Object
    ▸ 2: Object
    ▸ 3: Object
    ▸ 4: Object
    ▸ 5: Object
    ▸ 6: Object
    ▸ 7: Object
    ▸ 8: Object
    ▸ 9: Object
  ▸ functional_groups : Array (1)
  ▸ molecular_geometry : Object
```
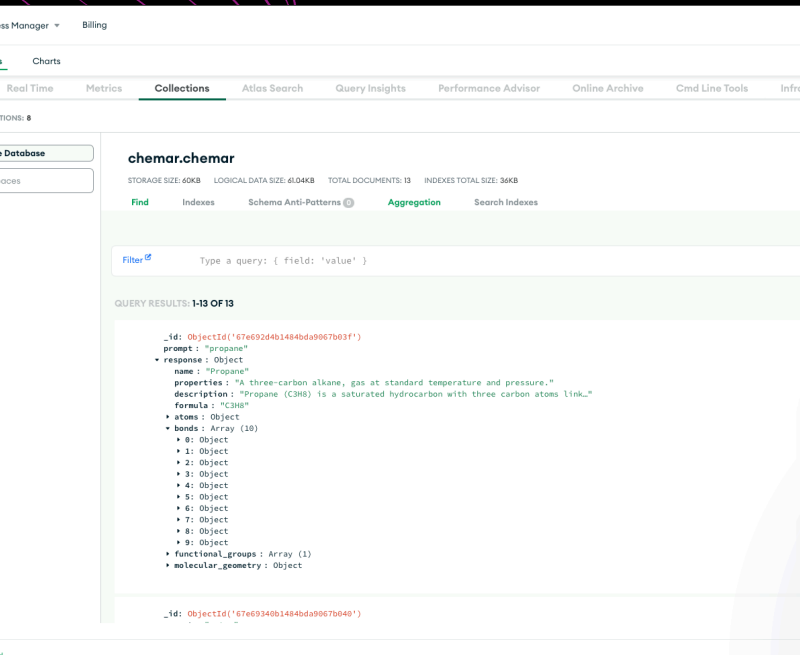

mongoDB

Pros

- **Flexible Schema**: NoSQL databases, like MongoDB, allow dynamic schemas, making them perfect for unstructured data or applications with evolving requirements.
- **Horizontal Scalability**: They scale out across multiple servers, efficiently managing large datasets and high traffic.
- **Variety of Data Models**: NoSQL supports multiple models (e.g., key-value, document, graph), offering flexibility for different use cases.

Cons

- **Eventual Consistency**: Many NoSQL systems prioritize availability over immediate consistency, which may lead to temporary data inconsistencies.
- **Limited Query Capabilities**: They often lack the advanced querying features of SQL, such as complex joins or transactions across records.
- **Less Mature Tools**: Compared to SQL, NoSQL ecosystems may have fewer established tools and less community support.

Why MongoDB?

MongoDB is a leading NoSQL database that stores data in flexible, JSON-like documents (specifically BSON, or Binary JSON). Here's why it's a popular choice:

01 Document-Oriented

MongoDB organizes data into documents that can contain nested structures and arrays, making it ideal for hierarchical or complex data.

02 Schema Flexibility

You can modify document structures without disrupting existing data, which suits projects with changing requirements or agile development.

03 Scalability

MongoDB supports horizontal scaling via sharding, distributing data across servers to handle large volumes and high traffic.

04 Use Cases

MongoDB shines in content management systems, real-time analytics, IoT applications, and scenarios where data structures evolve over time.

Caching
Redis

Redis is an in-memory key–value database, used as a distributed cache and message broker, with optional durability. Because it holds all data in memory and because of its design, Redis offers low-latency reads and writes, making it particularly suitable for use cases that require a cache. Redis is the most popular NoSQL database, and one of the most popular databases overall.

01 Speed

By storing data in memory, Redis delivers extremely fast read and write operations, reducing latency in applications.

02 Caching

It stores frequently accessed data (e.g., user sessions, API responses) temporarily, minimizing queries to slower, persistent databases.

03 Expiration

You can set time-to-live (TTL) for cached data, ensuring it expires when no longer relevant.

04 When use Redis?

Redis is ideal for session management, real-time analytics, leaderboards, and reducing database load in high-traffic systems.

What is chemAR?

chemAR is an iOS app that transforms chemistry education by overlaying interactive 3D molecular and reaction models onto the real world. Built for the WWDC Swift Student Challenge, chemAR helps students visualize and manipulate chemical structures and processes in augmented reality, making abstract concepts tangible and engaging.
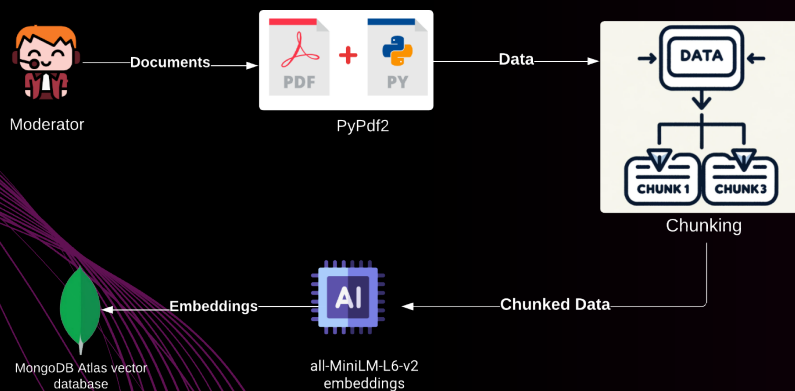
What are Embedding Models?

Embedding models are a type of machine learning model designed to represent data (such as text, images, or other forms of information) in a continuous, low-dimensional vector space. These embeddings capture semantic or contextual similarities between pieces of data, enabling machines to perform tasks like comparison, clustering, or classification more effectively.

Objects as vectors
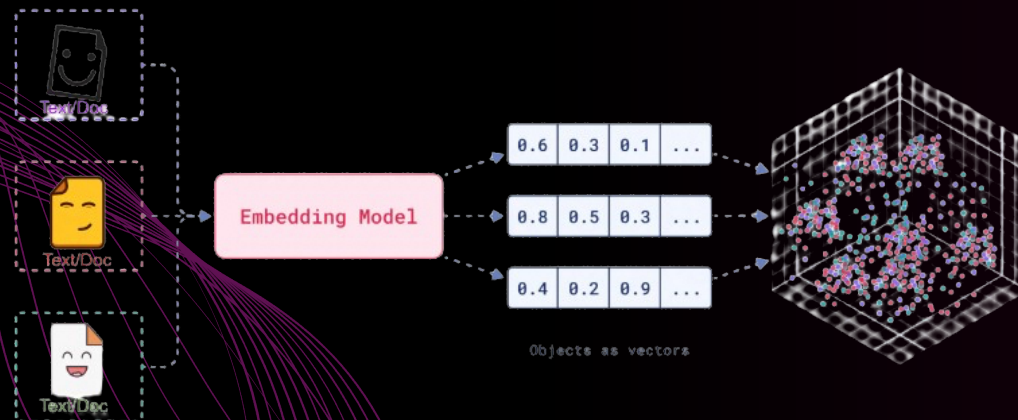
MongoDB Atlas Search

Vector search is a method for finding similar items based on their numerical representations, commonly used in AI-driven applications like search engines or recommendation systems. Here's a step-by-step explanation:

Embeddings

Data (e.g., text, images, or chemical compounds) is transformed into vectors—arrays of numbers that capture the data's meaning—using techniques like word embeddings or neural networks.

Storing Vectors

These vectors are stored in a specialized database (e.g., MongoDB Atlas Vector Search or Pinecone) optimized for similarity searches.

Querying

When a user submits a query, it's converted into a vector. The database then compares this query vector to stored vectors using metrics like cosine similarity or Euclidean distance to find the "closest" matches.

Retrieval

The most similar vectors (and their associated data) are returned, enabling tasks like recommendations or relevant search results.

Thank you

Feel free to reach me to discuss great ideas...

nshe0491@uni.Sydney.edu.au

Thank you!