Topics:

- Object-relational DBMS
- SQL Injection
- NoSQL Databases (Intro)

Christos Sardianos
Dec. 2017

Object-relational DBMS (ORDBMS)

ORDBMSs: Definition & Examples

Definition: An ORDBMS is a hybrid between the object-oriented model (OODBMS) and the relational model (RDBMS) database management system.

Examples of ORDBMSs include:

- Oracle Database
- PostgreSQL
- SQL Server
- Informix (IBM)

What is missing in relational model ??

- Handling of complex objects and complex relationships
- Handling of complex data types
- Code is not coupled with data
- No inherence, encapsulation, etc.

PL/SQL Block Types

Anonymous

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]


END;
```

Procedure

```
PROCEDURE name
IS


BEGIN
  --statements

[EXCEPTION]


END;
```

Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]


END;
```

User Defined Types Definition

```
CREATE TYPE <typename> AS
(
    <list of attribute-type pairs>
);
```

Oracle syntax:

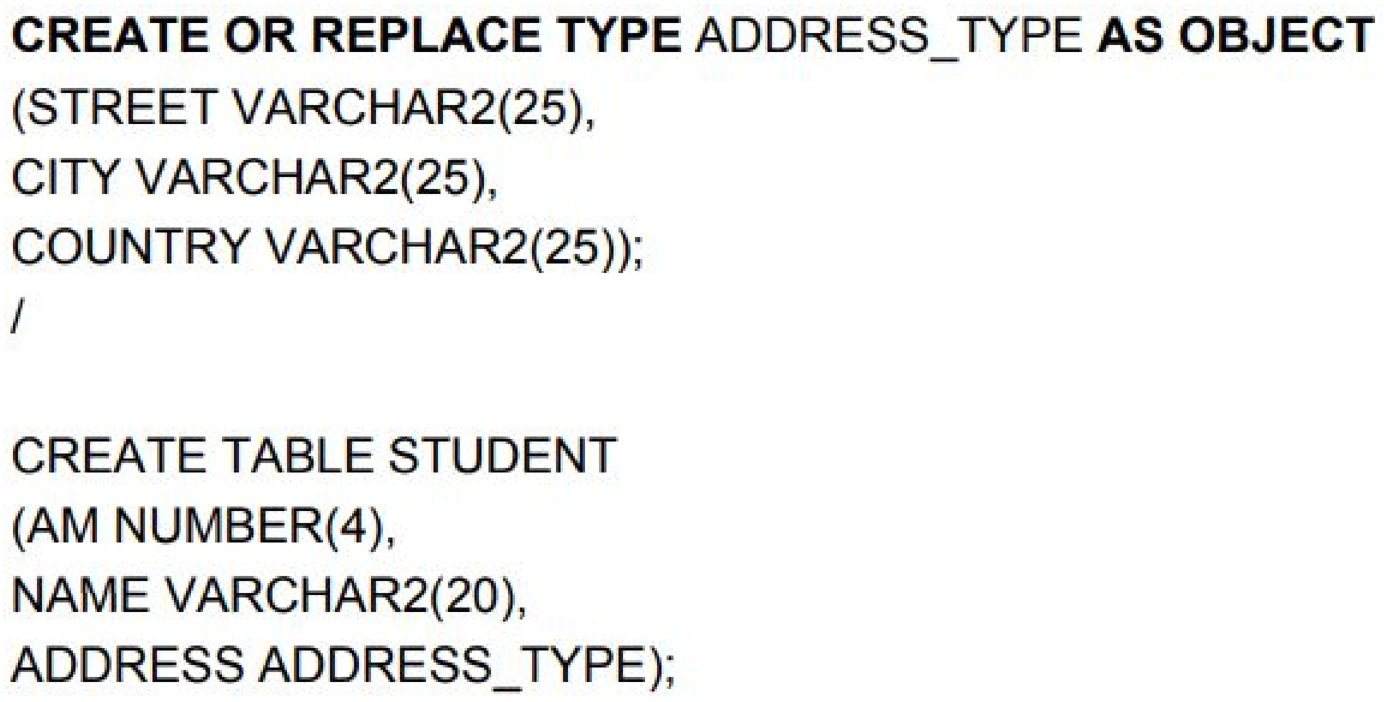
1. Add "OBJECT" in CREATE … AS OBJECT

2. Follow with / to have the type stored

User Defined Types (Example 1)

```
CREATE TYPE BarType AS (
    name CHAR(20),
    addr CHAR(20)
);


CREATE TYPE BeerType AS (
    name CHAR(20),
    manf CHAR(20)
);
```

User Defined Types (Example 2)



```
CREATE OR REPLACE TYPE ADDRESS_TYPE AS OBJECT
(STREET VARCHAR2(25),
CITY VARCHAR2(25),
COUNTRY VARCHAR2(25));
/

CREATE TABLE STUDENT
(AM NUMBER(4),
NAME VARCHAR2(20),
ADDRESS ADDRESS_TYPE);
```

User Defined Types (Inserting Data)

INSERT INTO STUDENT
VALUES (1234, ‘George Georgiou’,
ADDRESS_TYPE(‘Ermou 10’, ‘Athina’, ‘Hellas’));

Objects & Methods

```
CREATE OR REPLACE TYPE empType AS OBJECT (
    fname VARCHAR2(20),
    lname VARCHAR2(20),
    salary REAL,
    hiredate DATE,

    MEMBER FUNCTION getBonus RETURN REAL,
    MEMBER FUNCTION getYearsInService RETURN INT,
    MEMBER FUNCTION isOlderThan(x empType) RETURN INT
);
/
```

Object Body

```
CREATE OR REPLACE TYPE BODY empType AS
    MEMBER FUNCTION getBonus RETURN REAL IS
    BEGIN
        RETURN salary*(getYearsInService()/35);
    END getBonus;
    MEMBER FUNCTION getYearsInService RETURN INT IS
    BEGIN
        RETURN months_between(sysdate,hiredate)/12;
    END getYearsInService;
    MEMBER FUNCTION isOlderThan(x empType) RETURN INT IS
    BEGIN
        IF (hireDate<x.hireDate) THEN RETURN 1;
        ELSE RETURN -1;
        END IF;
    END isOlderThan;
END;
/
```

Examples

Create Table:

```
CREATE TABLE project(
	name VARCHAR2(30) name VARCHAR2(30),
	place VARCHAR2(20),
	contactPerson empType
);
```

Insert Data:

```
INSERT INTO project VALUES ( 'Gipedo', ‘Botanikos’, empType('George','Adams',1800,'13/1/2007'));
INSERT INTO project VALUES ('Opera', 'Marousi', empType('Bill','Brown',1500,'1/6/2008'));
INSERT INTO project VALUES ('Metro', 'Ag.Paraskeyi', empType('Bob','Thorton',2000,'22/4/2003'));
```

Examples

Select Data:

```
SELECT name,place
FROM pj p ro ect p
WHERE p.contactPerson.lname='Adams';
```

Method call:

```
SELECT p.contactPerson.getYearsInService(),
p.contactPerson.getBonus()
FROM p j p; ro ect p;
```

Update Data:

```
UPDATE project p UPDATE project p
SET p.contactPerson=empType('James','Ferry',2000,'1/6/2007')
WHERE name='Opera';
```

How to use these example blocks.

Find the last name of the employee, who is older than the employee named “Adams”, and print it.

Usage Example

```
DECLARE
    x empType;
    name VARCHAR2(30);
BEGIN
    SELECT p.contactPerson INTO x
    FROM project p
    WHERE p.contactPerson.lname='Adams';
    SELECT p.contactPerson.lname INTO name
    FROM project p
    WHERE p.contactPerson.isOlderThan(x) >0;
    DBMS_OUTPUT.PUT_LINE(name);
END;
/
```

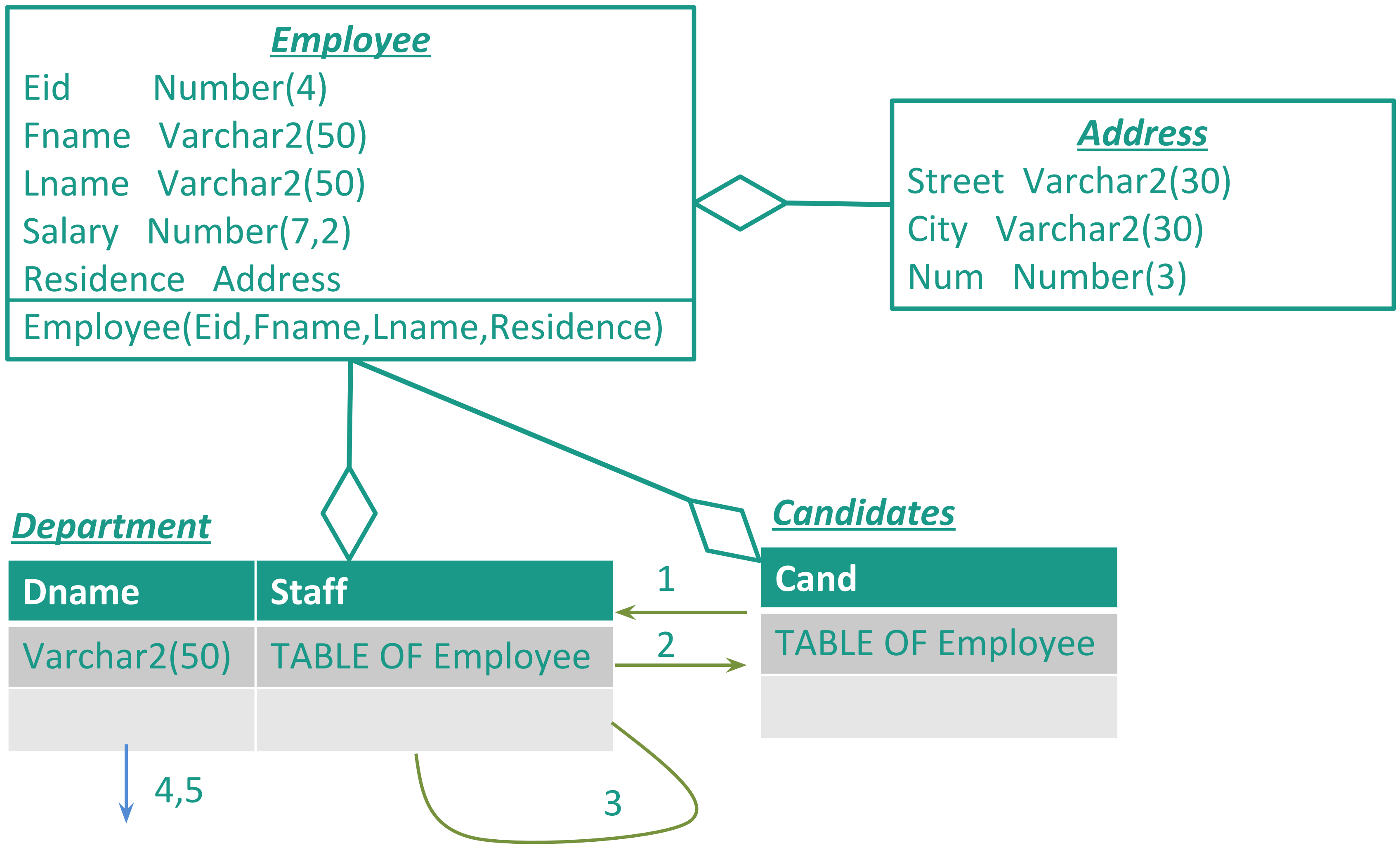
Tables of Objects

```
CREATE OR REPLACE TYPE people_type AS OBJECT (
	last_name VARCHAR2(25),
	department_id NUMBER(4),
	salary REAL
);
/

CREATE TABLE people_demo1 OF people_type;

INSERT INTO people_demo1 VALUES (people_type('Morgan', 10, 1000));
```

Declaring a Nested Table

```
CREATE TYPE DeptList AS TABLE OF VARCHAR2(10)
/

CREATE TYPE empl AS OBJECT (
        id_num  INTEGER(4),
        name    VARCHAR2(25),
        address VARCHAR2(35),
        status  CHAR(2),
        departments DeptList
);
/
```

- We can create nested tables inside other tables.



1) hire(deptname Varchar2, emp Employee, salary NUMBER)
2) fire(Employee e, deptname Varchar2)
3) raiseSalaries(dname Varchar2, raise NUMBER)
4) salaryCap(deptname Varchar2)→number(8,2)
5) getsize(deptname Varchar2)→number(5)

SQL Injection

SQL Injection stages

- Research: Attacker tries submitting various unexpected values for the argument, observes how the application responds, and determines an attack to attempt.
- Attack: Attacker provides a carefully-crafted input value that, when used as an argument to a SQL query, will be interpreted as part of a SQL command rather than merely data; the database then executes the SQL command as modified by the attacker.

Attack types & examples

SQL Injections can do more harm than just by passing the login algorithms. Some of the attacks include:

- Deleting data
- Updating data
- Inserting data
- Executing commands on the server that can download and install malicious programs such as Trojans
- Exporting valuable data such as credit card details, email, and passwords to the attacker's remote server
- Getting user login details etc

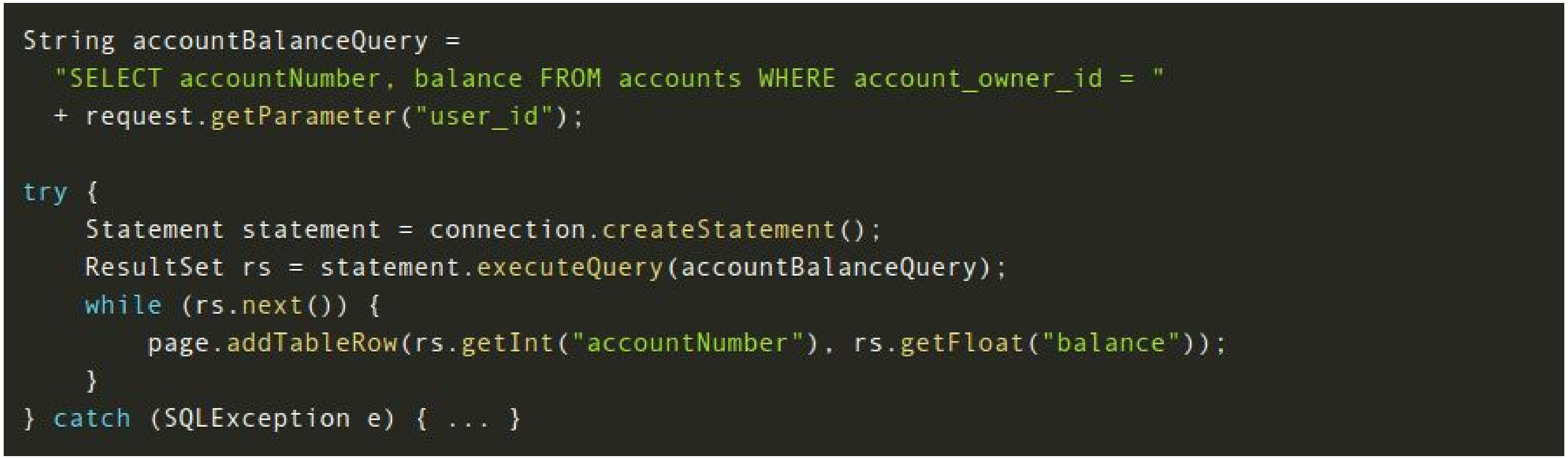
Attack types & examples

There are plenty methods exploiting for performing SQL injection attacks such as:

- ❏ Making your user into an administrator
- ❏ Error-based SQLi (Returning error messages thrown by the DB server)
- ❏ Time-based Blind SQLi (Time-based Blind SQLi)
- ❏ Returning more data than expected

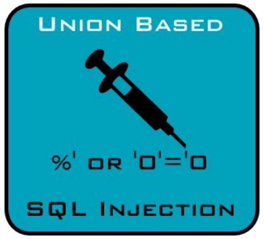
Returning more data than expected

Imagine a developer needs to show the account numbers and balances for the current user's id as provided in a URL. They might write (in Java):



```
String accountBalanceQuery =
  "SELECT accountNumber, balance FROM accounts WHERE account_owner_id = "
  + request.getParameter("user_id");

try {
    Statement statement = connection.createStatement();
    ResultSet rs = statement.executeQuery(accountBalanceQuery);
    while (rs.next()) {
        page.addTableRow(rs.getInt("accountNumber"), rs.getFloat("balance"));
    }
} catch (SQLException e) { ... }
```

Source: https://www.veracode.com/security/sql-injection



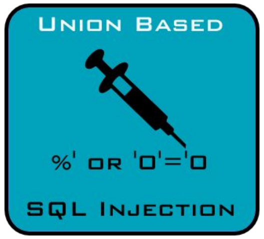
Returning more data than expected

Under normal operation, the user with ID 984 might be logged in, and visit the URL:
https://bankingwebsite/show_balances?user_id=984

So the resulting query would be something like:
SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 984

How could someone possibly only exploit this query to perform an SQLi?

Source: https://www.veracode.com/security/sql-injection



Returning more data than expected

Under normal operation, the user with ID 984 might be logged in, and visit the URL:
https://bankingwebsite/show_balances?user_id=984

So the resulting query would be something like:
SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 984

A common practice used by the attackers in SQL injection attacks is the change of the input parameter to be interpreted as:
0 OR 1=1

So the query that would be sent to the database will end up being:
SELECT accountNumber, balance FROM accounts WHERE account_owner_id = 0 OR 1=1

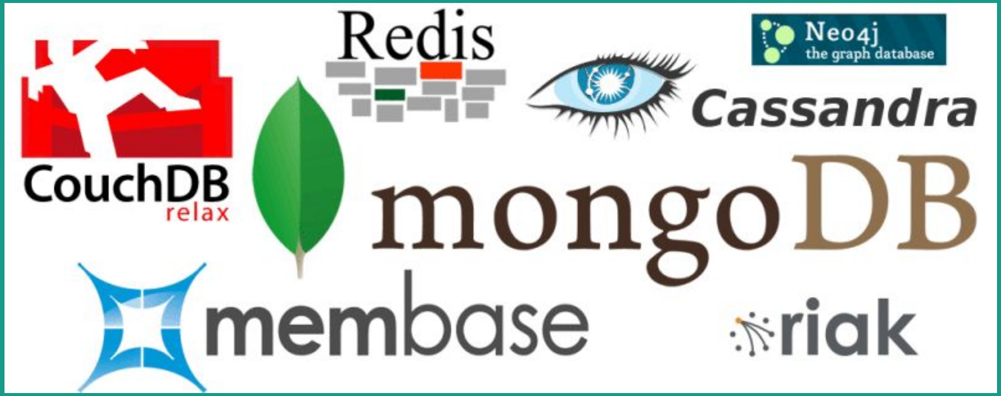
Source: https://www.veracode.com/security/sql-injection

How to Prevent against SQL Injection Attacks

- **User input should never be trusted** - It must always be sanitized before it is used in dynamic SQL statements.
- **Stored procedures** – these can encapsulate the SQL statements and treat all input as parameters.
- **Prepared statements** –prepared statements to work by creating the SQL statement first then treating all submitted user data as parameters.
- **Regular expressions** –these can be used to detect potential harmful code and remove it before executing the SQL statements.
- **Database connection user access rights** –only necessary access rights should be given to accounts used to connect to the database.
- **Error messages** –these should not reveal sensitive information and where exactly an error occurred. Simple custom error messages such as “Sorry, we are experiencing technical errors. The technical team has been contacted. Please try again later” can be used instead of display the SQL statements that caused the error.

Source: https://www.guru99.com/learn-sql-injection-with-practical-example.html

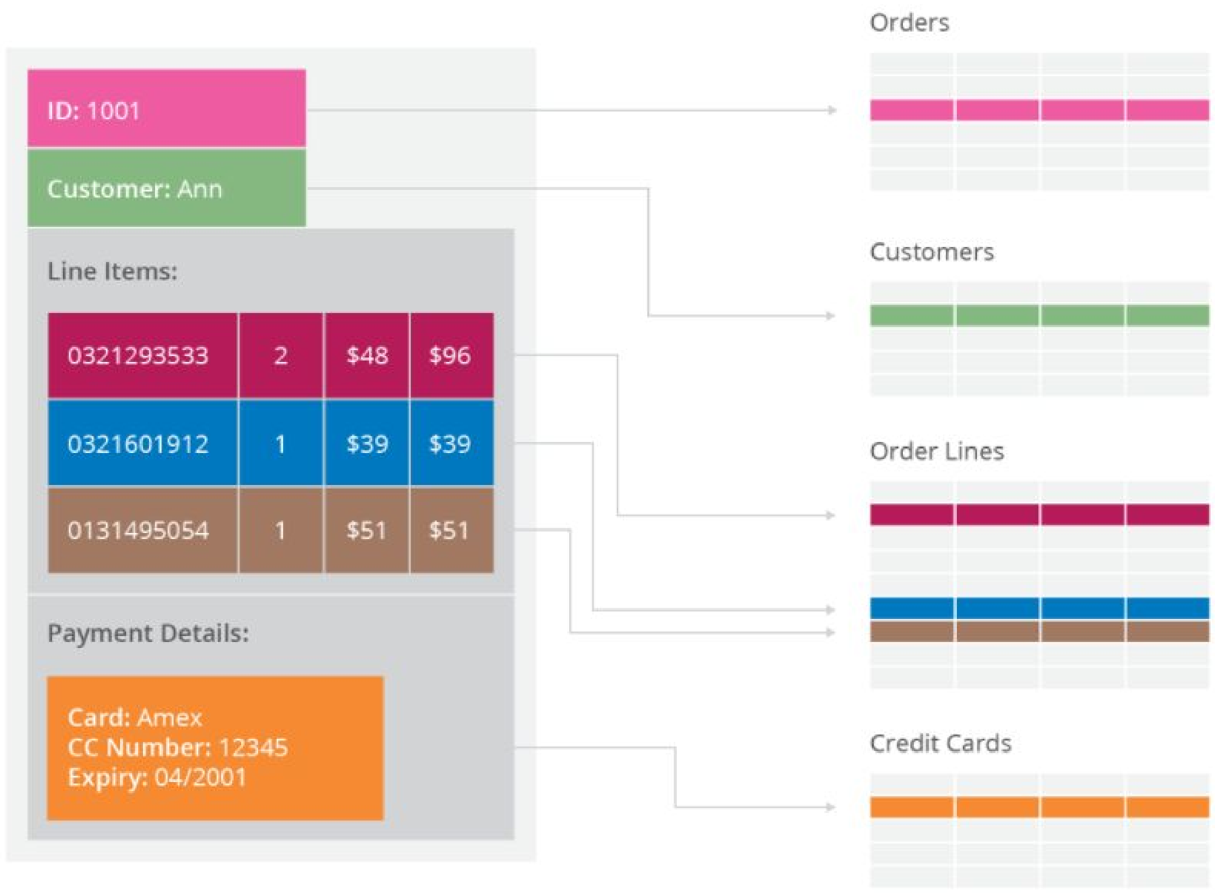
NoSQL Databases



Definition

A NoSQL database is a DBMS that provides a mechanism for storage and retrieval of data modeled in means other than the tabular relations used in relational databases.

Why NoSQL Databases



Source: https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Pros and Cons

Pros

- May support SQL-like query languages
- Simplicity of design
- Simpler "horizontal" scaling to clusters
- Finer control over availability
- Availability
- Partition tolerance
- Speed

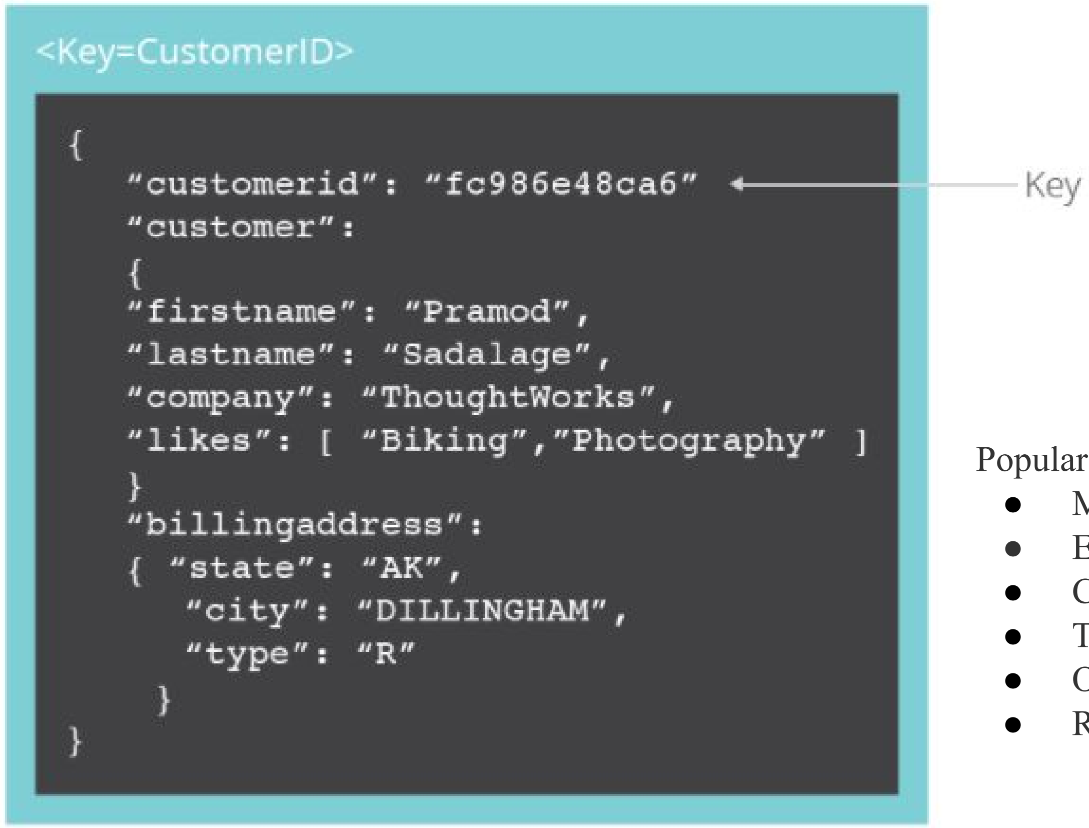
Cons

- Consistency
- Use of low-level query languages (i.e. the lack of ability to perform ad-hoc joins across tables)
- Lack of standardized interfaces
- Huge previous investments in existing relational databases
- Lack of ACID (Atomicity, Consistency, Isolation, Durability) transactions

NoSQL Database Types

- **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4J and Giraph.
- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Examples of key-value stores are Riak and Berkeley DB. Some key-value stores, such as Redis, allow each value to have a type, such as 'integer', which adds functionality.
- **Wide-column** stores such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

Document databases

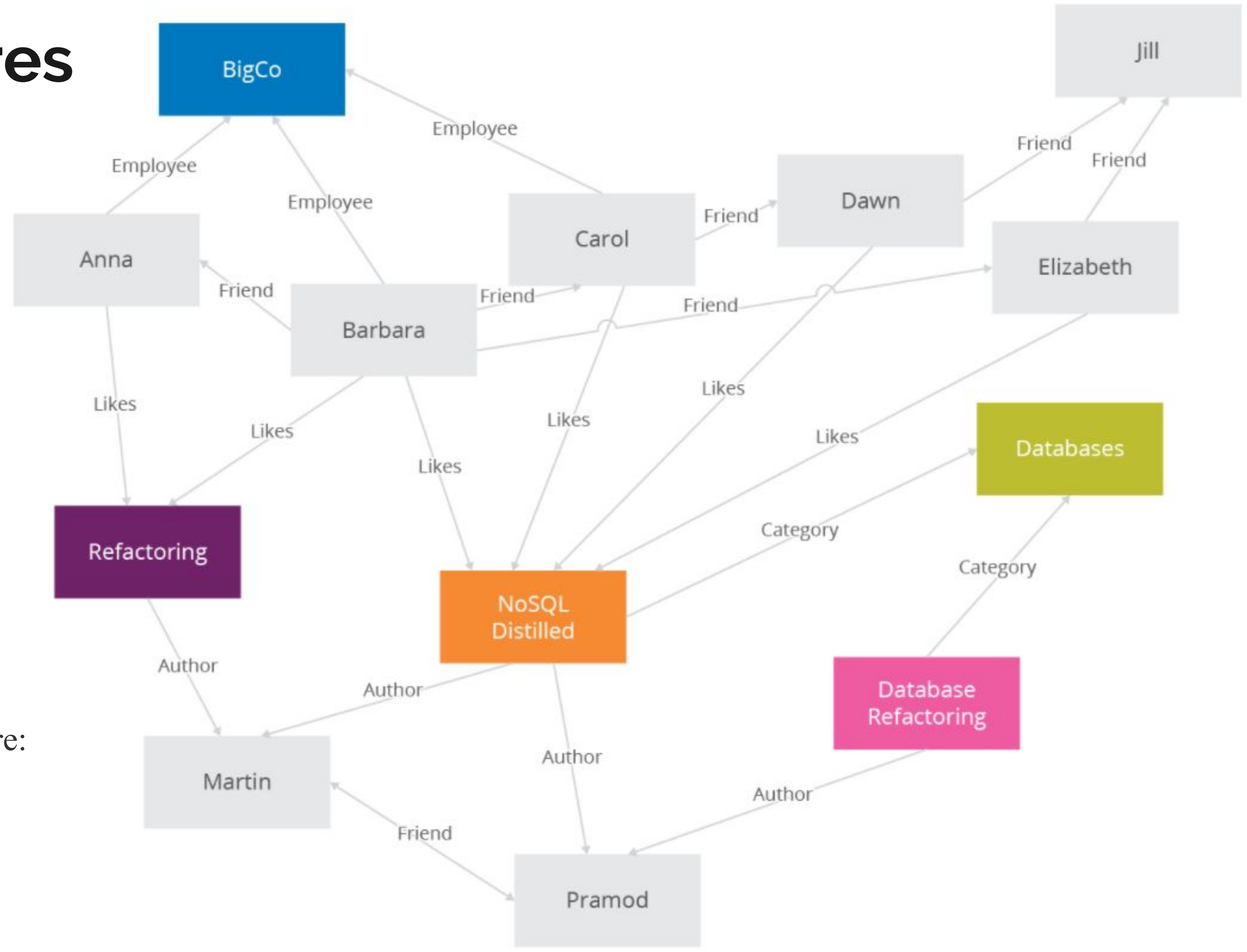


```
{
   "customerid": "fc986e48ca6"
   "customer":
   {
   "firstname": "Pramod",
   "lastname": "Sadalage",
   "company": "ThoughtWorks",
   "likes": [ "Biking","Photography" ]
   }
   "billingaddress":
   { "state": "AK",
     "city": "DILLINGHAM",
     "type": "R"
    }
}
```

Popular document databases are:

- MongoDB
- Elastic Search
- CouchDB
- Terrastore
- OrientDB
- RavenDB

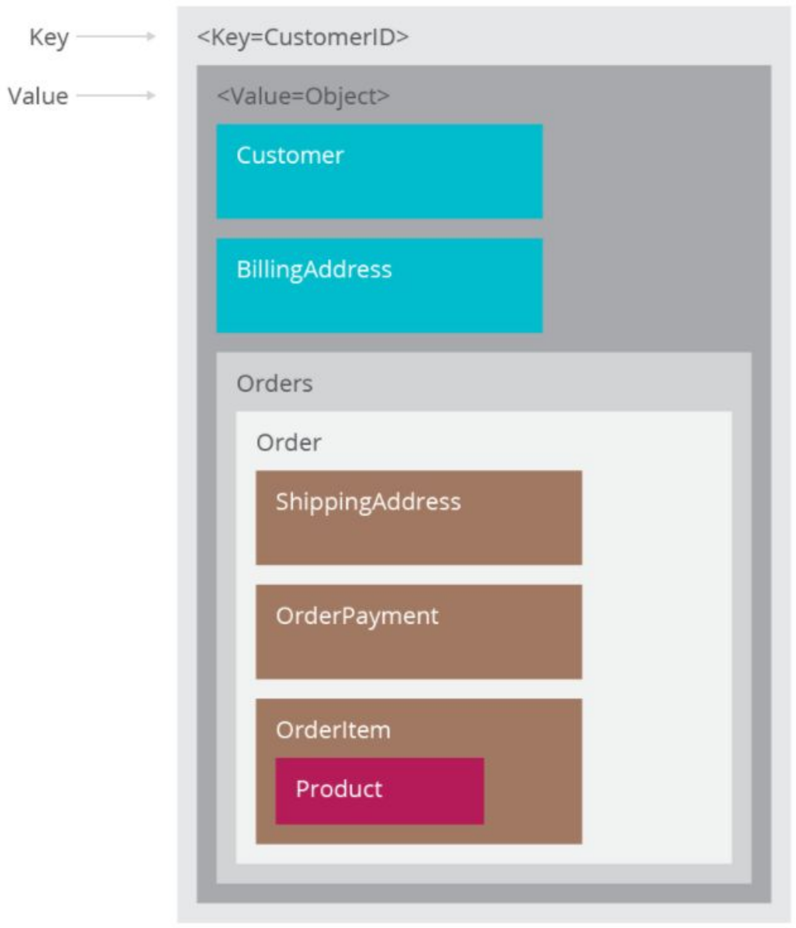
Graph stores



Popular key-value databases are:

- Neo4J
- Infinite Graph
- OrientDB
- FlockDB

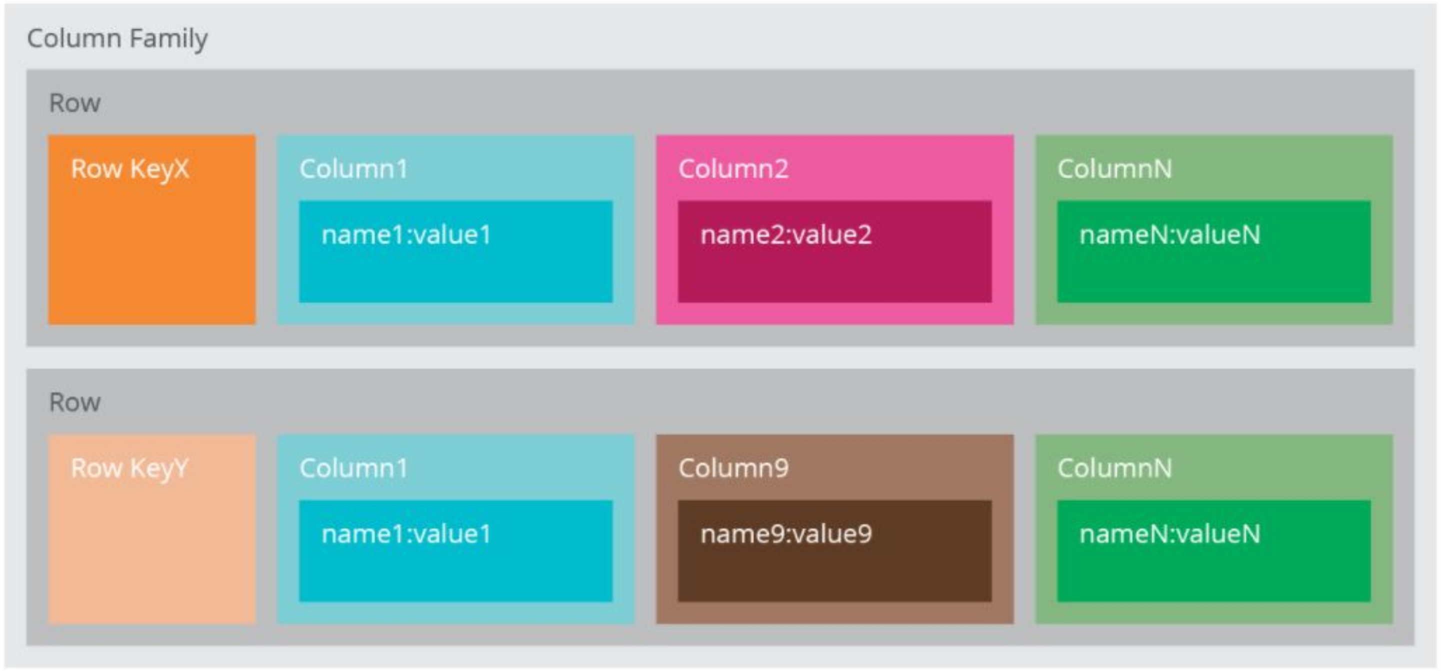
Key-value stores



Popular key-value databases are:

- Riak
- Redis
- Memcached
- Berkeley DB
- Amazon DynamoDB (not open-source)
- Project Voldemort and Couchbase.

Wide-column store / Column Families



Popular key-value databases are:

- Cassandra
- HBase
- Hypertable
- Amazon DynamoDB