CS5322 Database Security

Last Lecture

- Discretionary Access Control (DAC)
- Example:
  - Bob:
    - GRANT select, insert ON Employee TO Ann WITH GRANT OPTION
    - REVOKE select ON Employee FROM Ann
- Recursive revocation
- Non-cascading revocation
- DAC with views

Coming Next

- Fine-Grained Access Control
- Oracle Virtual Private Database

Fine-Grained Access Control

- Suppose that we have the following table that stores CS5322 grades
  - Grades(student, grade)
- We want to
  - Allow each student to check her own grade but not those of others'
  - Allow each lecturer to check all grades
- If we are to implement such *fine-grained* access control using standard SQL, we have to use views
  - Create one view for each student about her grade
- Problem: way too many views to manage

Possible Solution 1: Let the applications handle it

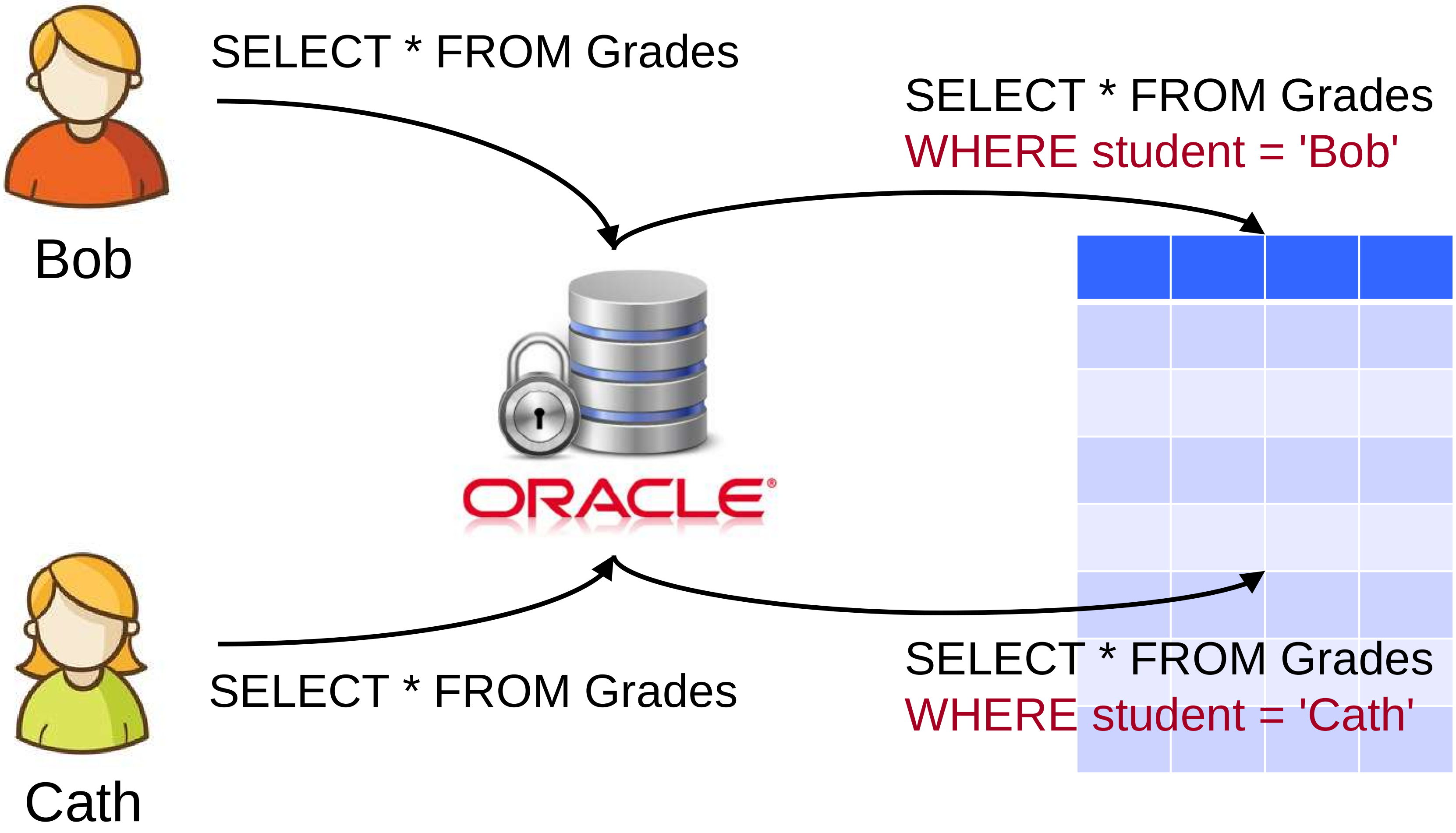
- Idea:
  - Do not create views for students
  - Instead, let the applications (e.g., a grade viewing service) check user identities and issue queries like SELECT * FROM Grades WHERE student = `Bob'
- Problem:
  - Application code can see everything on Grades
    - If the application is hijacked, the whole table can be compromised
- Sometimes it could be better to have access controls inside the database instead of relying on the applications

Possible Solution 2: Parameterized Views

- Conventional views:
  - CREATE VIEW BobGrades AS
    SELECT * FROM Grades WHERE student = `Bob'
- Parameterized views:
  - CREATE VIEW StudentGrades AS
    SELECT * FROM Grades WHERE student = $username
  - Here $username is a variable that is given at runtime
- Problem:
  - Applications need different queries for students and lecturers
    - For students, need to query StudentGrades
    - For lecturers, need to query Grades instead
- It could be more convenient to have an *authorization-transparent* method that avoids this

Possible Solution 3: View Replacement

- Idea: Instead of creating views for users, modify users' queries “behind the scene” to exercise access control
- Bob's query:
  - SELECT * FROM Grades
- The database system modifies the query into:
  - SELECT * FROM Grades WHERE student = 'Bob'
- This modification is *transparent* to the users
- This approach is used in Oracle's Virtual Private Database (VPD)



Bob

SELECT * FROM Grades

SELECT * FROM Grades
WHERE student = 'Bob'

Cath

SELECT * FROM Grades

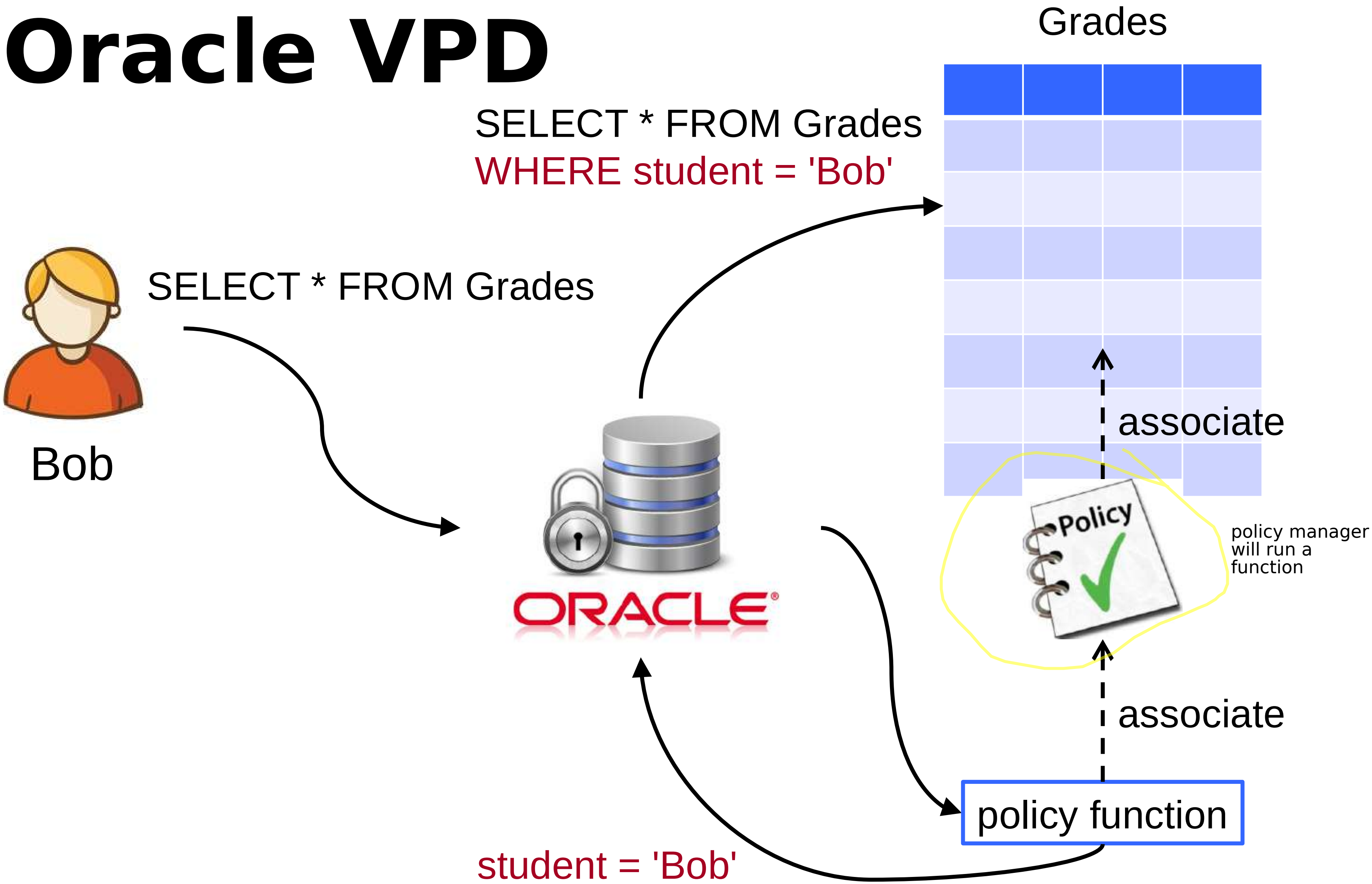
SELECT * FROM Grades
WHERE student = 'Cath'

- VPD attaches a policy to the Grades table
- Predicates in red are added in runtime based on the policy

Oracle VPD

- Sometimes referred to as Oracle Row-Level Security (RLS) or Fine Grained Access Control (FGAC)
- Idea:
  - Associate security policies to database objects
  - Transparently add predicates to the WHERE clause of queries/updates
  - The predicates are generated by user-defined functions given in the policies
    - Can be in Oracle's PL/SQL, or even C or Java, etc
    - Can access *session parameters*, e.g., user name

Oracle VPD



Grades

SELECT * FROM Grades
WHERE student = 'Bob'

SELECT * FROM Grades

Bob

associate

policy manager
will run a
function

associate

policy function

student = 'Bob'

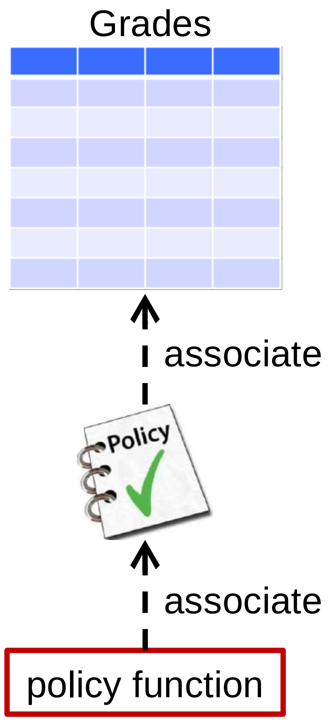
Oracle VPD: Example 1

- Suppose that the 'CredicDB' database has the following table
  - Grades(student varchar2(30), grade varchar2(2));
- Suppose that we want to implement the following policy:
  - Everyone can only see the rows with A+ grades
- We will first create a policy function, *check_grade*

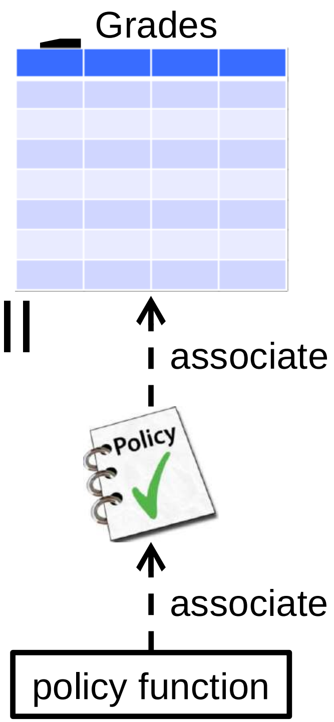
Oracle VPD: Example 1

- CREATE FUNCTION check_grade(
  v_schema IN VARCHAR2, v_obj IN VARCHAR2)
- RETURN VARCHAR2 AS condition VARCHAR2 (200);
- BEGIN
- condition := 'grade = ''A+''';
- RETURN condition;
- END check_grade;

- v_schema would be the database name
- v_obj would be the table name
- condition would be the predicate to add



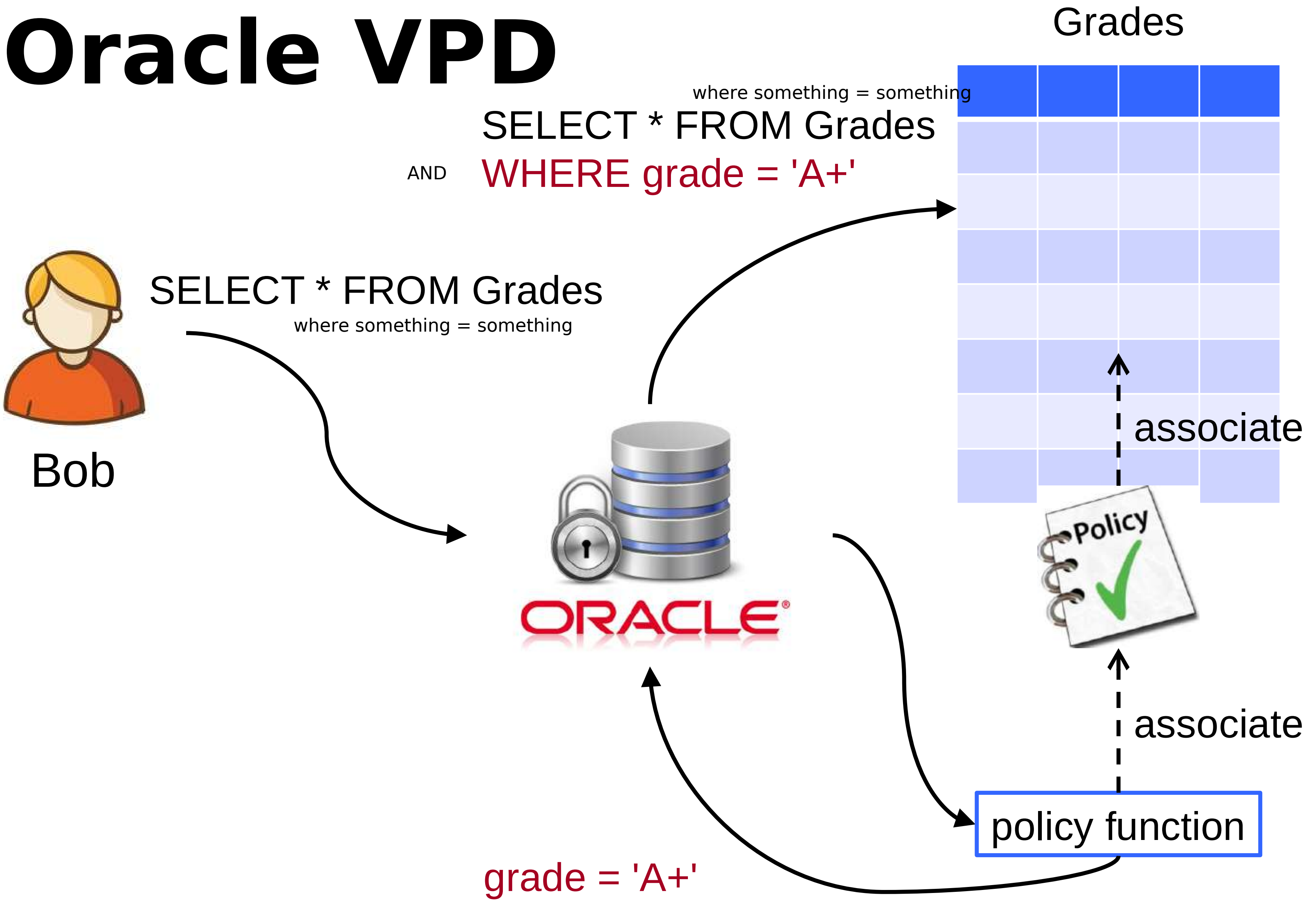
Oracle VPD: Example 1



- After creating the policy function, we will add a policy based on the function:

- BEGIN
-    DBMS_RLS.ADD_POLICY (
-    object_schema => 'CedricDB',
-    object_name => 'Grades',
-    policy_name => 'check_grade_policy',
-    policy_function => 'check_grade');
- END;

Oracle VPD



Grades

where something = something

SELECT * FROM Grades

AND WHERE grade = 'A+'

Bob

SELECT * FROM Grades

where something = something

associate

associate

policy function

grade = 'A+'

Oracle VPD: Example 2

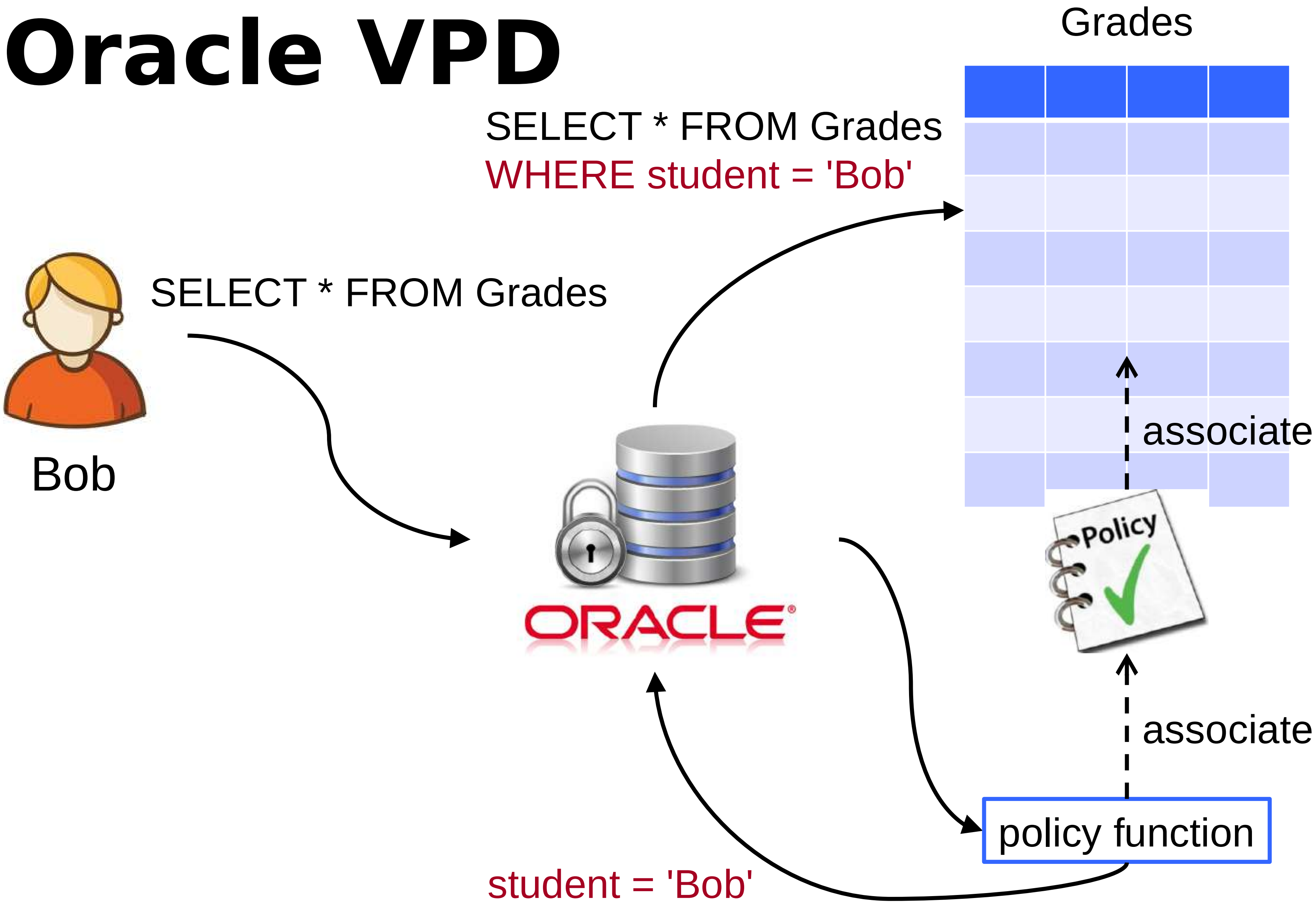
- Consider again the Grades table in the 'CredicDB' database:
- Suppose that we want to change the check_grade_policy to the following:
  - Every student can only see her own grade
- We will change the check_grade function

Oracle VPD: Example 2

- CREATE OR REPLACE FUNCTION check_grade(
  v_schema IN VARCHAR2, v_obj IN VARCHAR2)
- RETURN VARCHAR2 AS condition VARCHAR2 (200);
- BEGIN
- condition := 'student = SYS_CONTEXT(''grade_app'', ''stu_name'')';
- RETURN condition;
- END check_grade;

- SYS_CONTEXT('grade_app', 'stu_name') returns the value of the 'stu_name' parameter in the 'grade_app' context
- Here we assume that the 'grade_app' context has been created, and the 'stu_name' parameter equals the name of the student whose grade is being queried

Oracle VPD



Grades

SELECT * FROM Grades
WHERE student = 'Bob'

SELECT * FROM Grades

Bob

associate

associate

policy function

student = 'Bob'

SYS_CONTEXT

- In Oracle, the SYS_CONTEXT function is used to retrieve parameters in the Oracle environment.
- The syntax for the function is:
  SYS_CONTEXT(*namespace*, *parameter*, [*length*])
  - *namespace* is a context either built-in or created by the user
    - Oracle provides a built-in namespace called USERENV, which provides information about the current Oracle session
  - *parameter* is an attribute in the context, and it must be set in advance using the DBMS_SESSION.set_context procedure.
  - *length* is optional; it specifies the length of the parameter value in bytes

USERENV: The built-in namespace

- Some sample parameters in USERENV
  - SESSION_USER :   name of the database user at logon
  - SESSION_USERID :   Id of the database user at logon
  - DB_NAME: name of the database
  - ISDBA: Returns TRUE if the user has been authenticated as having DBA privileges
  -

Oracle VPD: Example 3

- Consider again the Grades table in the 'CredicDB' database:
- Suppose that we want to change the check_grade_policy to the following:
  - Every student can only see her own grade
  - The DBA can see all grades
- We will change the check_grade function

Oracle VPD: Example 3

- CREATE OR REPLACE FUNCTION check_grade(
  v_schema IN VARCHAR2, v_obj IN VARCHAR2)
- RETURN VARCHAR2 AS condition VARCHAR2 (200);
- BEGIN
- IF (SYS_CONTEXT('USERENV', 'ISDBA')) THEN
- RETURN ' ';
- ELSE
- RETURN 'student = SYS_CONTEXT(
  ''grade_app'', '' stu_name'')';
  END IF;
- END check_grade;

Oracle VPD: Example 3

- Bob accesses the Grades table
  - SELECT * FROM Grades
- Result?
  - If Bob is a DBA, return everything in Grades
  - Otherwise, if Bob is a student, return his grade in Grades

Oracle VPD: Example 3

- What if Bob also issues the following?
  - UPDATE Grades SET grade = 'F' WHERE student = 'Alice'
- Our previous policy does not prevent this, since the policy does not restrict update and insert
  - Solution: Don't grant Bob update rights on Grades
- Now assume (for the sake of argument) that we allows Bob to update his own grade (but not others' grades)
  - We can change our policy for this purpose

Oracle VPD: Example 3

everything in a begin and end will be executed atomically

- BEGIN
-    DBMS_RLS.DROP_POLICY(
-    object_schema => 'CedricDB',
-    object_name => 'Grades',
-    policy_name => 'check_grade_policy');
-    DBMS_RLS.ADD_POLICY (
-    object_schema => 'CedricDB',
-    object_name => 'Grades',
-    policy_name => 'check_grade_policy',
-    policy_function => 'check_grade',
-    update_check => TRUE);
- END;

- Bob:
  - UPDATE Grades SET grade = 'F' WHERE student = 'Alice' ✗
  - UPDATE Grades SET grade = 'F' WHERE student = 'Bob' ✓

Column-Level VPD

- Our previous policy only allows each student to see her own tuple
- But what if we want the following:
  - Each student can only see her own grade
  - Each student can see the names of all other students
- We can use a *column-level* policy

Column-Level VPD

- We can re-use the previous policy function
- CREATE OR REPLACE FUNCTION check_grade(
  v_schema IN VARCHAR2, v_obj IN VARCHAR2)
- RETURN VARCHAR2 AS condition VARCHAR2 (200);
- BEGIN
-     IF (SYS_CONTEXT('USERENV', 'ISDBA')) THEN
-     RETURN '';
-     ELSE
-     RETURN 'student = SYS_CONTEXT(''grade_app'', ''
  stu_name'')';
      END IF;
- END check_grade;

- But we will change the policy

Column-Level VPD

- BEGIN
-     DBMS_RLS.ADD_POLICY (
-     object_chema   => 'CedricDB',
-     object_name => 'Grades',
-     policy_name=>   'check_grade_policy',
-     policy_function => 'check_grade',
-     sec_relevant_cols => 'grade');
- END;

Column-Level VPD

Grades

student	grade
Alice	A+
Bob	B+
Cath	C+

- Bob:
  - SELECT student FROM Grades

student
Alice
Bob
Cath

- Bob:
  - SELECT * FROM Grades

student	grade
Bob	B+

What if we want this?

Grades

student	grade
Alice	A+
Bob	B+
Cath	C+

- Bob:
  - SELECT student FROM Grades

student
Alice
Bob
Cath

- Bob:
  - SELECT * FROM Grades

student	grade
Alice	NULL
Bob	B+
Cath	NULL

Column-level VPD

- BEGIN
-     DBMS_RLS.ADD_POLICY (
-     object_chema   => 'CedricDB',
-     object_name => 'Grades',
-     policy_name => 'check_grade_policy',
-     policy_function => 'check_grade',
-     sec_relevant_cols => 'grade',
-     sec_relevant_cols_opt => dbms_rls.ALL_ROWS);
- END;

Multiple Policies

- What happens if we add multiple policies on the same table?
  - The policies are enforced with AND syntax.
- Example: suppose that table T is associated with {P1, P2, P3}
  - Consider a query: SELECT A FROM T WHERE C.
  - It would be modified into:
    SELECT A FROM T WHERE C AND (c1 AND c2 AND c3)
  - c1, c2, and c3 are from P1, P2, and P3, respectively

Coming Next

- Issues with VPD

Issues with VPD: Inconsistencies

- Suppose that a policy authorizes each employee to see his/her own salary
- Alice issues the following query:
  - SELECT MIN(Salary) FROM Employee
- The query will be rewritten to
  - SELECT MIN(Salary) FROM Employee WHERE Name = 'Alice'
- The results could confuse users

Issues with VPD: Recursion

- Consider the following policy function on the Grades table

- CREATE check_grade(
    v_schema IN VARCHAR2, v_obj IN VARCHAR2)
- RETURN VARCHAR2 AS condition VARCHAR2 (200);
- BEGIN
-     SELECT 'grade = ' || MAX(grade) into condition
    FROM Grades
-     RETURN condition;
- END check_grade;

- This is not allowed
- In general, if we have a policy on a table T, then the policy function F cannot access T

Issues with VPD: Recursion

- In general, if we have a policy on a table T, then the policy function F cannot access T
- Why?
- Because of potential recursions
  - A query Q wants to access T
  - Oracle invokes the policy function F for Q
  - F executes a query Q' on T
  - Oracle invokes F again for Q'
  - F executes a query Q' again on T
  - ...

Summary

- FGAC is a powerful access control mechanism
- Oracle VPD implements FGAC using query rewriting mechanisms