Efficient Row Level Security in Databases

Ezat Karimi , Sr Solutions Architect, Amazon
Shailesh Doshi, Sr Solutions Architect, Amazon

Agenda

- What is Row level security (RLS)?
- RLS in different database platforms
- RLS Architecture
- RLS & multi-tenancy
- RLS & fine-grained access control (FGAC)
- Pros/Cons of RLS
- RLS Optimization & Best Practices
- RLS & Gen/AI
- Take-aways

What is RLS?

What is RLS ?

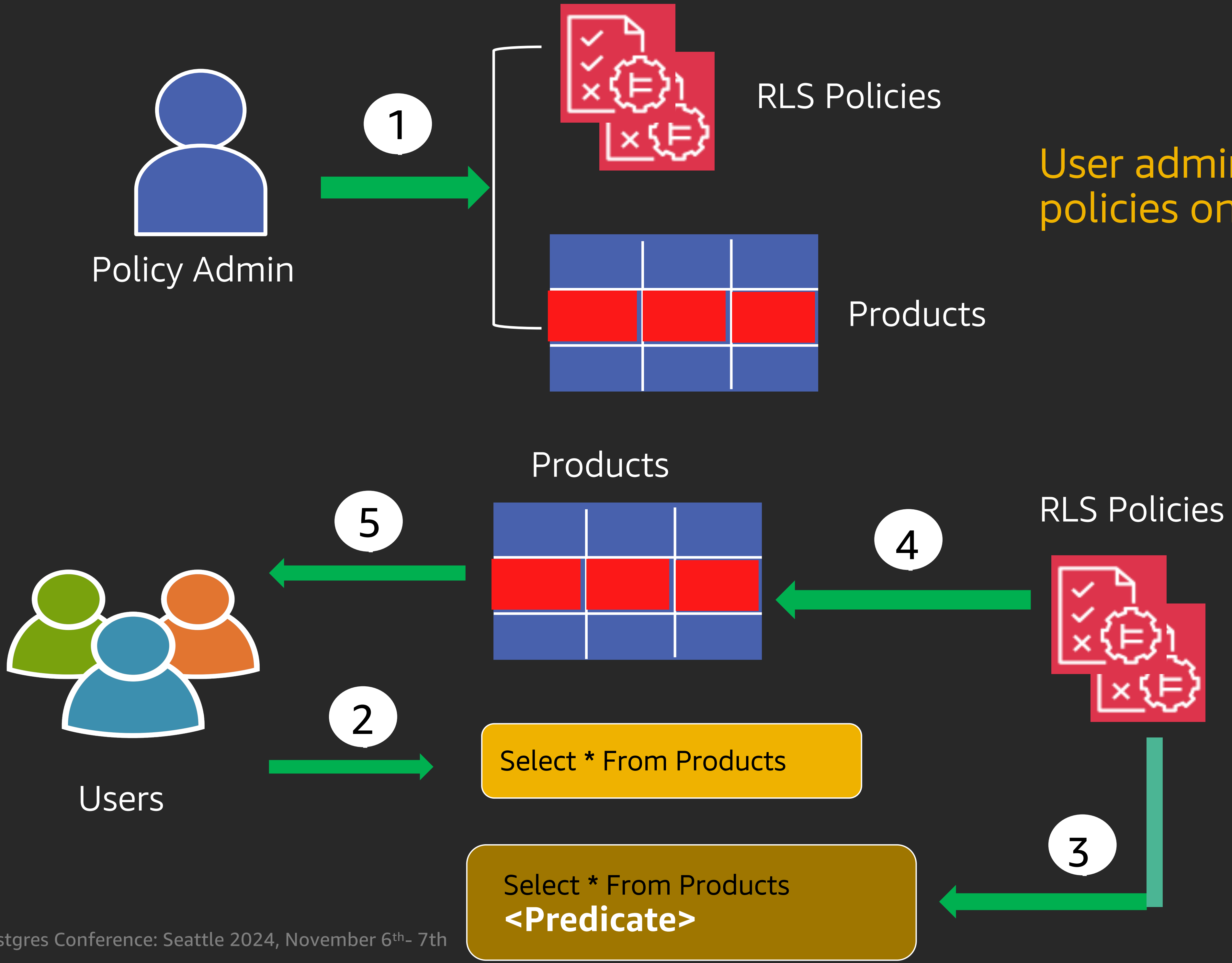
RLS is used to restrict access to specific rows in a table based on a user's identity, role, or other factors.

It is used to define fine-grained access control.

Usage:

1. Restrict access to sensitive data
2. Implement row-level access control
3. Support multi-tenancy

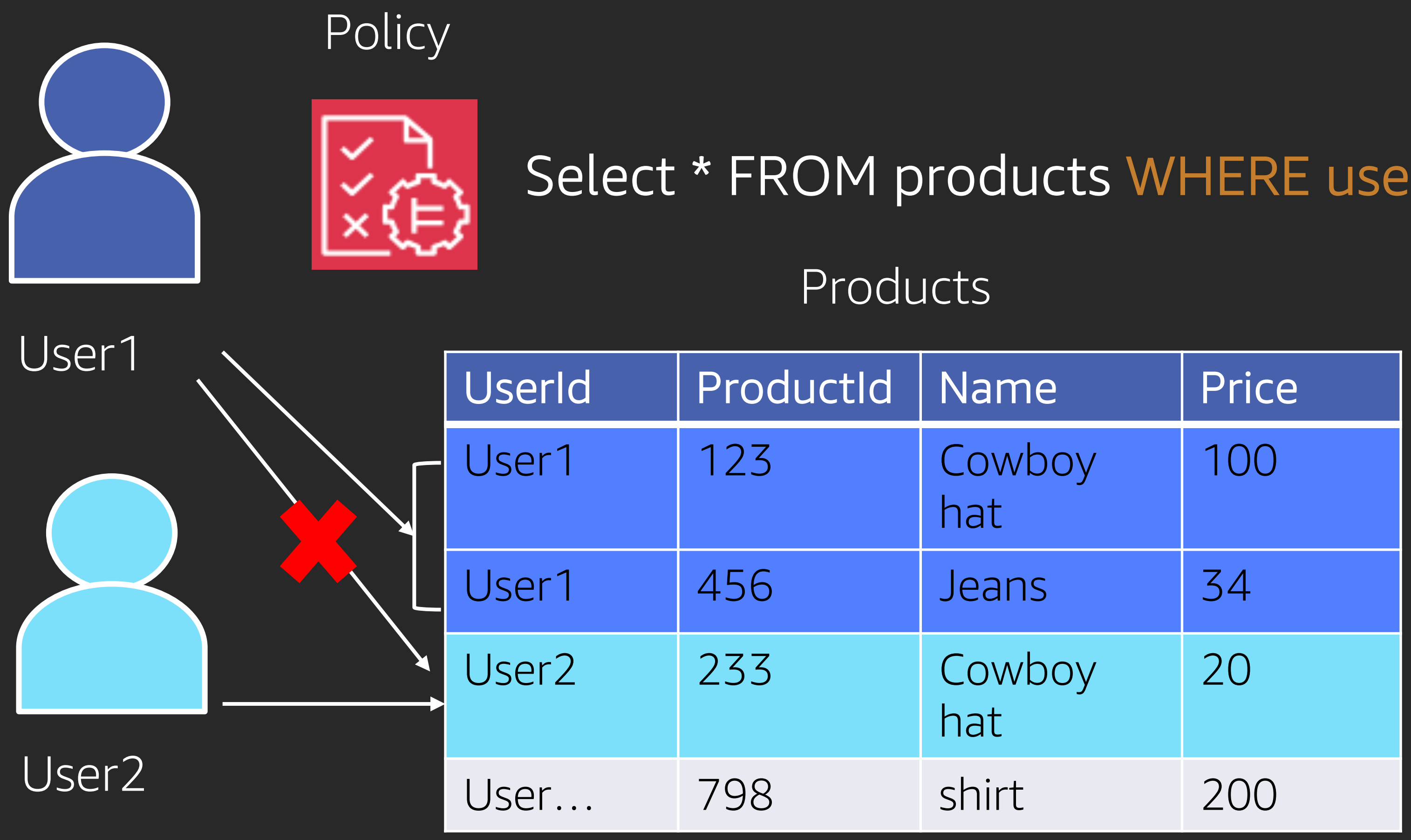
How Does RLS Work?



User admin defines RLS access policies on products table

RLS policy is enforced each time user queries products table

RLS - Example



Select * FROM products WHERE userid = current_user

Products

UserId	ProductId	Name	Price
User1	123	Cowboy hat	100
User1	456	Jeans	34
User2	233	Cowboy hat	20
User…	798	shirt	200

RLS Definition & Implementation

Components

- ❖ Security Policy
- ❖ Row-Level Security Function
- ❖ Row-Level Security Predicate
- ❖ Table Access
- ❖ Row Filtering

Implementation Techniques

- ❖ Row-Level Security Functions
- ❖ Views
- ❖ Stored Procedures
- ❖ Triggers
- ❖ ABAC
- ❖ RBAC

RLS Policies (PostgreSQL)

```
-- Enable RLS
ALTER TABLE products ENABLE ROW LEVEL SECRURITY;
```

-- RLS Policy Definition

- Policy name
- Table name: the table the policy is applied to
- PERMISSIVE|RESTRICTIVE : policy type
- Command (CRUD): ALL, SELECT, DELETE, INSERT, UPDATE. ALL is the default.
- Role: the role the policy applies to; the default is PUBLIC.
- Using_expression: Each row is checked against this expression; if it returns false, it is silently suppressed and cannot be viewed or modified by the user.
- check_expression: a SQL expression returning a boolean, used when INSERT or UPDATE operations are performed on the table. Rows are allowed if the policy expression is true, and if it returns false, an error is returned.

```
--Define a RLS Policy that allow a sr-manager to do all CRUDs.

CREATE POLICY procurement_products ON products TO managers
USING ('sr-manager' = current_user);
```

RLS Policy Types (PostgreSQL)

Permissive & Restrictive

- RLS policies are permissive by default.
- Permissive:
  - Used to allow access to rows.
  - Applied using a boolean "OR"
- Restrictive:
  - Used to prevent access to rows.
  - Applied using a boolean "AND"
- When RLS is enabled, by default no one can access the table, unless BYPASSRLS attribute is specified.
- There has to be at least one permissive policy for anything to work

Define Multiple RLS Policies

Below one policy enables all rows to be viewed by all roles, and the other only allows each user to modify (CRUD – SELECT) on its own rows.

```
CREATE POLICY products_select_policy
            ON products FOR SELECT
            USING  (true);

CREATE POLICY products _mod_policy
        ON products
        USING (user = current_user);
```

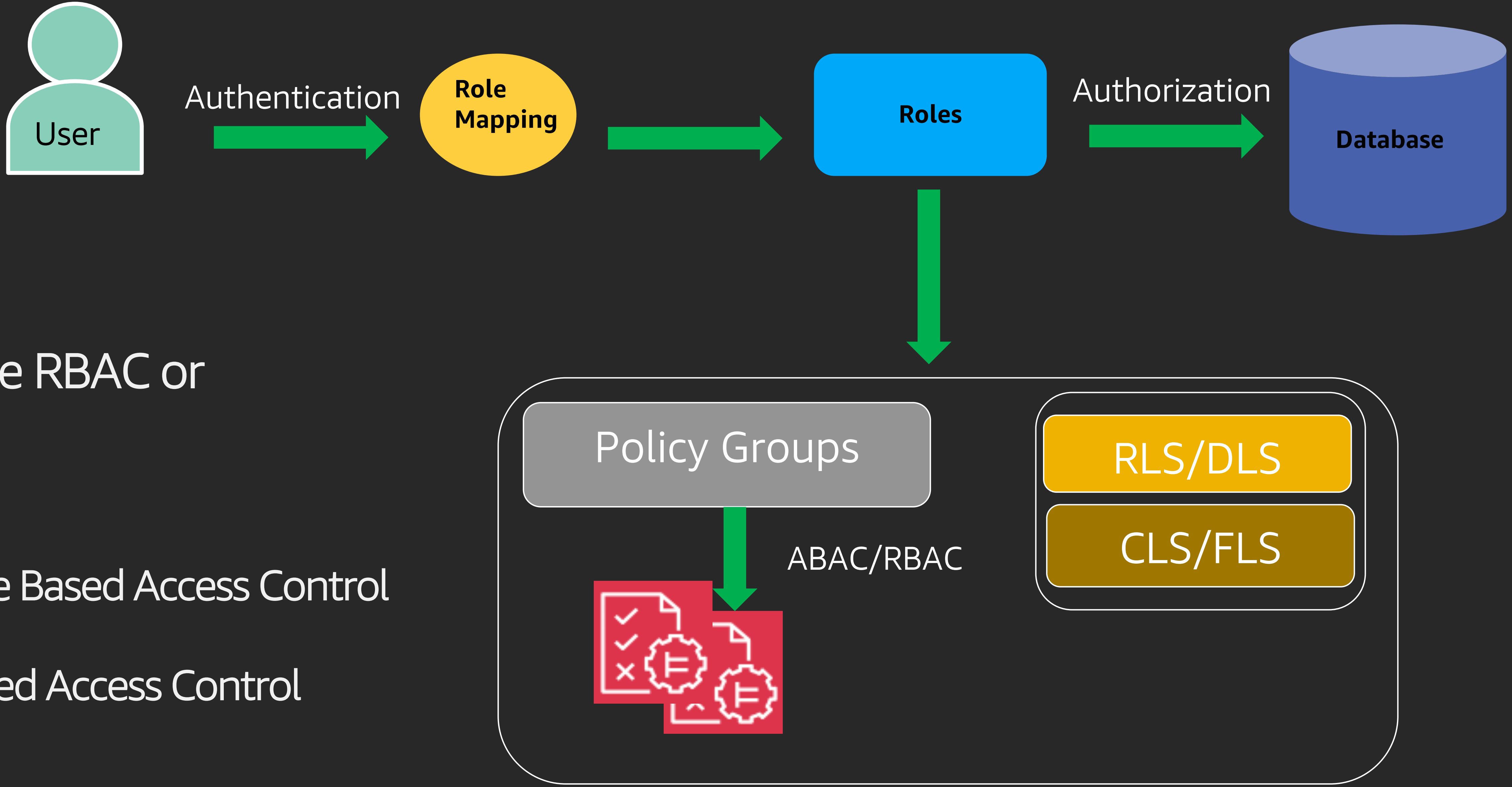
RLS in Different Database Platforms

RLS in Different Database Platforms

Feature	PostgreSQL	SQL Server	Oracle	OpenSearch
Row/Column Level Security	RLS/CLS	RL/SCLS	RLS/CLS	DLS/FLS
RLS implementation	Built-in	Built-in	Built-in	Built-in
RLS Policy Enablement	ALTER TABLE <table-name> ENABLE ROW LEVEL SECURITY	ALTER DATABASE <db-name> SET ENABLE_ROW_LEVEL_SECURITY ON	DBMS_RLS.ENABLE_POLICY	N/A
RLS policies	CREATE POLICY	CREATE SECURITY POLICY <pol-name> ADD FILTER PREDICATE <pred-name> ON <table-name>	DBMS_RLS.ADD_POLICY(object_schema => <schema>, object_name => <table_name> policy_name => <pol_name> policy_function =><func_name>	"dls": "[.. some DLS here ..]", "allowed_actions": ["indices:data/read/search"]
Performance impact	Moderate	Moderate	Moderate	Modetate
Support for multi-tenancy	Yes	Yes	Yes	Yes
Integration with IAM	Yes	Yes (Azure AD)	Yes (Oracle IAM)	Yes
CLS Implementation	GRANT/REVOKE <access>(<column-list>) ON <table-name> TO <User>	GRANT/DENY <access> ON TABLE (<Column-list>) TO <User>	Oracle Advanced Security's data redaction capability	Include or exclude fields in search query

RLS & Fine-Grained Access Control (FGAC)

RLS/CLS FGAC



RLS/CLS can be RBAC or ABAC driven

ABAC – Attribute Based Access Control

RBAC – Role Based Access Control

RLS & FGAC

RLS & RBAC

Limit the access based on user role

WHERE <current_role> = 'ADMIN'
AND salary > 10000

RLS & ABAC

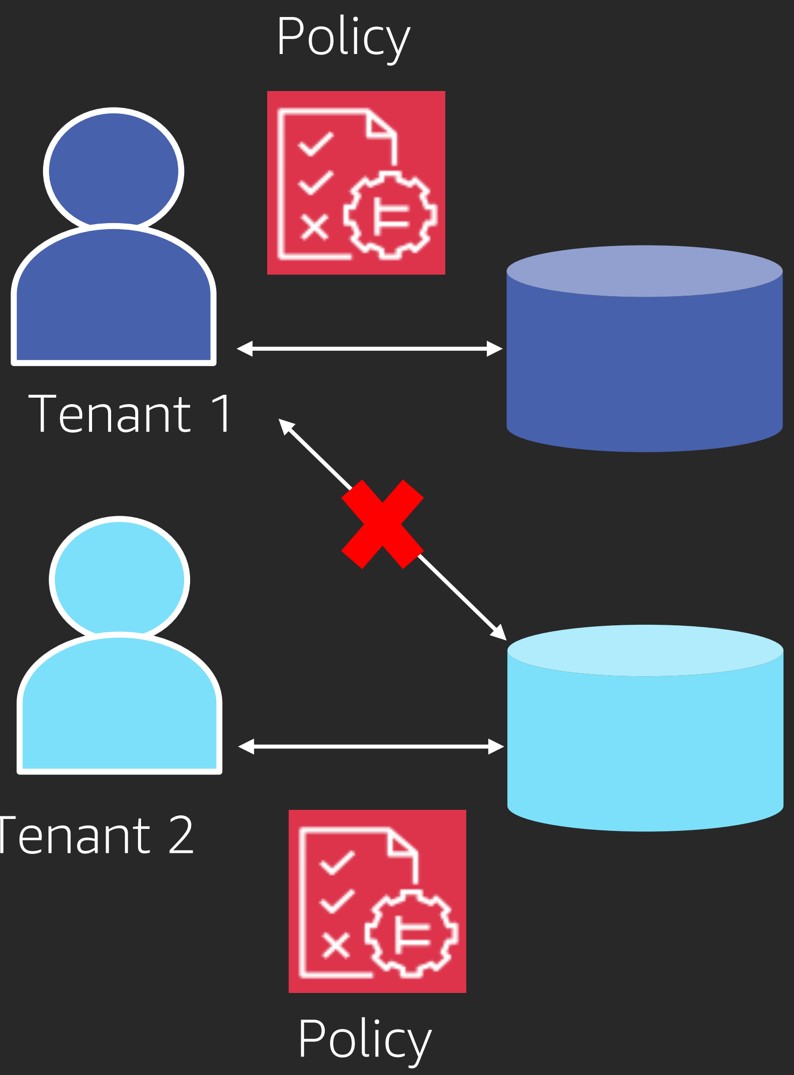
Limit the access based on location attribute of the user

WHERE <location> = 'London'
AND salary > 10000

name	job	salary
John	painter	10000
Mary	waiter	20000
Betty	CEO	500000
Pete	writer	90000

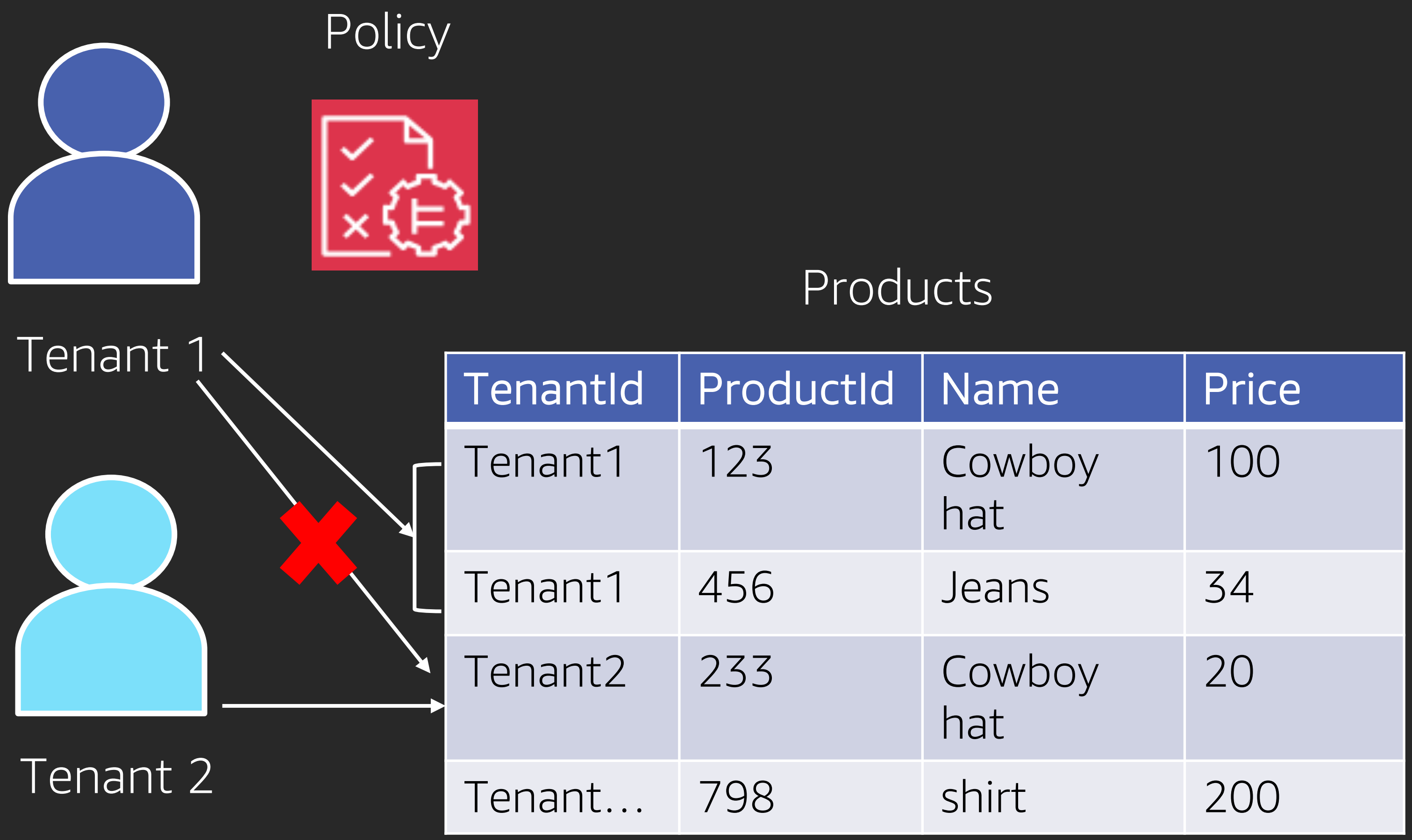
RLS & Multi-Tenancy

Multi-Tenant Database Isolation Models



Silo isolation model

Separate resource per tenant



TenantId	ProductId	Name	Price
Tenant1	123	Cowboy hat	100
Tenant1	456	Jeans	34
Tenant2	233	Cowboy hat	20
Tenant…	798	shirt	200

Pool isolation model

Multiple tenants sharing a resource (here a table)

RLS with Pool Isolation Model (PostgreSQL)

```
-- Enable RLS
ALTER TABLE products ENABLE ROW LEVEL SECURITY;

-- Define RLS policy
CREATE POLICY products_select_policy ON products
USING (tenantid::TEXT = current_user);
```

```
-- Tenant1 queries
SELECT * FROM products;

-- RLS enforced
SELECT * FROM products WHERE tenantid = 'tenant1'
```

Pros & Cons of RLS

Benefits of RLS

- Data security by default
- Compliance with regulations
- Reduced risk of data breaches and security related mistakes
- Fine-grained access control
- Dynamic access control
- Simplified security management
- Application ORM agnostic
- Enhanced multi-tenancy support
- User and data segregation
- Improved data governance
- Improved data quality
- Improved incident response
- Reduced cost due to simplified and central security management

RLS Challenges

- Complexities
  - Schema: joins, able inheritance
  - Policy: too many rules, too many policies
- User management
- Maintenance
- Auditing and compliance
- Integration with existing systems
- Performance
- Scalability

When to avoid RLS?

- High-performance database requirements
- Simple security requirements
- Static data
- Legacy systems
- Auditing and logging
- Complex security policies
- Over-engineering
- Database vendor limitations
- Other security mechanisms, such as data encryption is in place

RLS Optimization & Best Practices

RLS Optimization

- ❖ Simplify security policies
- ❖ Use efficient predicate functions
- ❖ Index security predicate columns
- ❖ Use filtered indexes
- ❖ Apply RLS last (apply base query filters first)
- ❖ Avoid select *; specify the columns to retrieve
- ❖ Optimize join orders
- ❖ Use RLS with other security features, such FGACs, data encryption
- ❖ Optimize security policy evaluation (e.g., by caching previous results)
- ❖ Use database-specific optimization techniques, such as table partitioning
- ❖ Limit the number of security predicates
- ❖ Use materialized views to pre-compute and store the results of RLS
- ❖ Cache security metadata

RLS Best Practices

- Have a clear understanding of the business requirements
- Define a clear security model
- Use row-level security policies
- Leverage views and virtual tables
- Optimize database design
- Use indexing and caching
- Do not use RLS for to implement business logic
- Measure the impact of the RLS filters
- Regularly review and update security policies
- Test & validate thoroughly
- Instrument, monitor, audit and remediate
- Be prepared to deal with anomalies
- Keep it simple

RLS & Gen/AI

Gen AI & RLS

- ❖ Predictive Analytics
- ❖ Anomaly Detection
- ❖ Access Control Optimization
- ❖ Data Classification
- ❖ User Behavior Analysis
- ❖ RLS Policy Generation
- ❖ Incident Response
- ❖ Data Loss Prevention
- ❖ Compliance
- ❖ User Segmentation
- ❖ Audit Log Analysis

Take Aways

Take Aways

- Factor in scale and performance when designing for RLS
- Use RLS for enforcing access control only
- Keep it simple
- Test and measure the impact
- Follow the best practices
- Avoid RLS when it is not warranted
- Use Gen/AI to improve RLS

Q/A

Thank You!

ezatk@amazon.com
doshisk@amazon.com