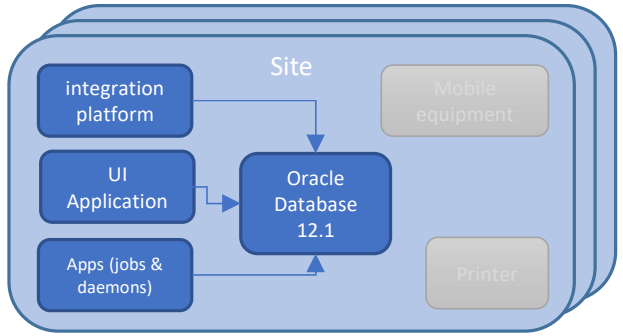
AMIS | CONCLUSION

Triple-C Consolidate Centralize & Cloudify

Martijn Pronk, AMIS

Het verhaal

- 100+ vestigingen
- Iedere vestiging heeft een lokale server met
  - 4CPU met Hyperthreading, 32GB or 64GB; 4 VMs
  - Oracle Database 12c R1 Enterprise Edition
    - 12-15 GB data
    - 30-100K transacties per dag, 60K logons p/dag
  - Java-based Integration platform
  - 100s background jobs & daemons
    - Shell script, Perl, C++ (binary), PL/SQL, SQL
  - UI platform (Oracle Forms & modern low code, Java based)
- Database gebruik begonnen in de Jaren ‘90

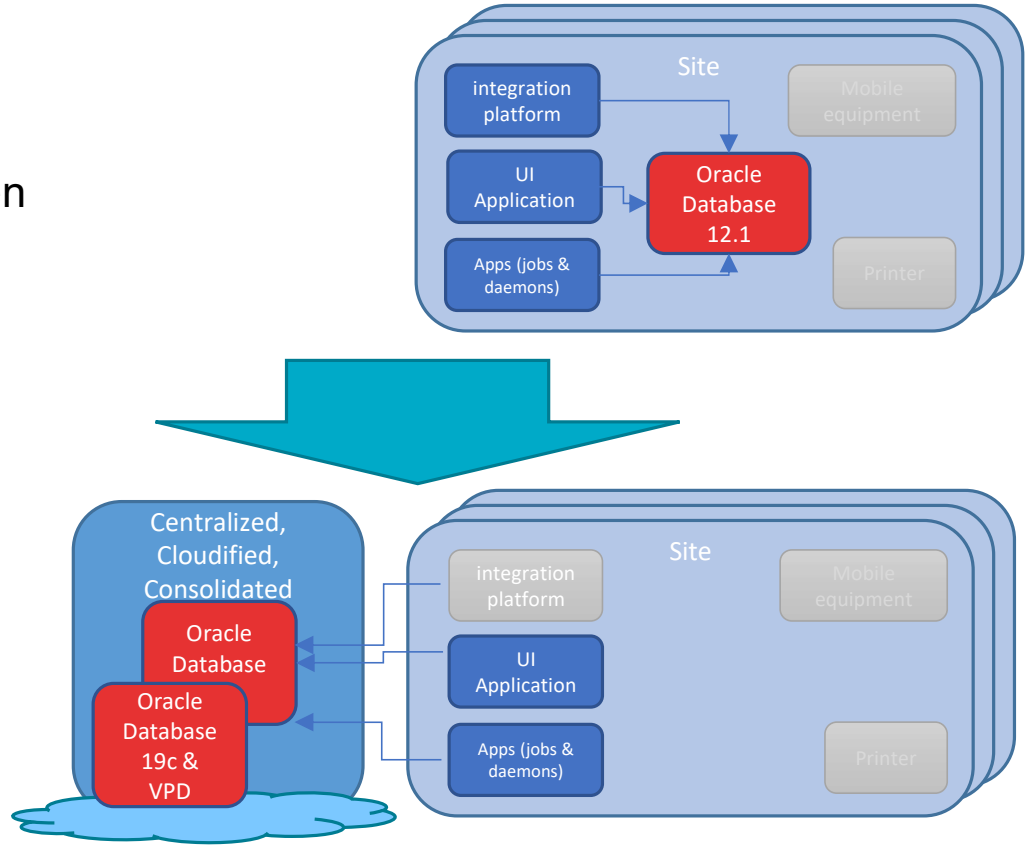


Consolideren!

- Minder operationeel werk, minder licenties

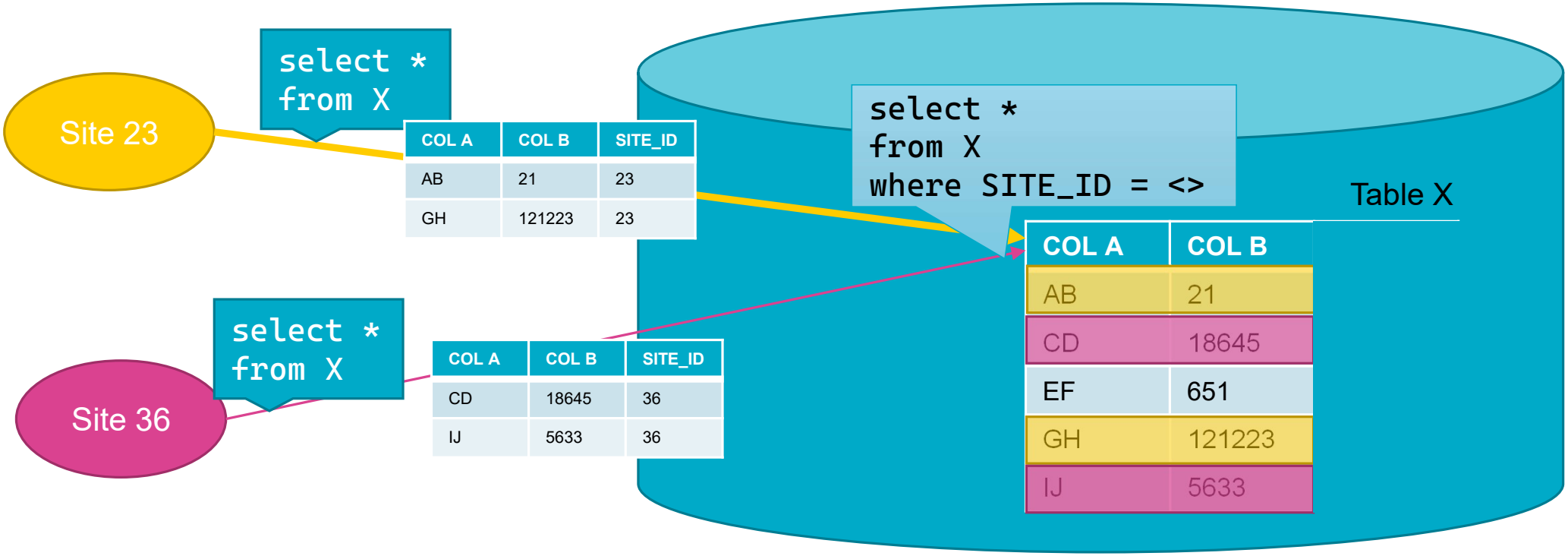
- Maar:
  - Elke vestiging mag alleen eigen data zien
  - Bestaande applicaties zo min mogelijk aanpassen

- Doel: **Azure Cloud**
- Performance moet gelijkwaardig zijn
- Migratie met zo min mogelijk impact (as-is bijna)



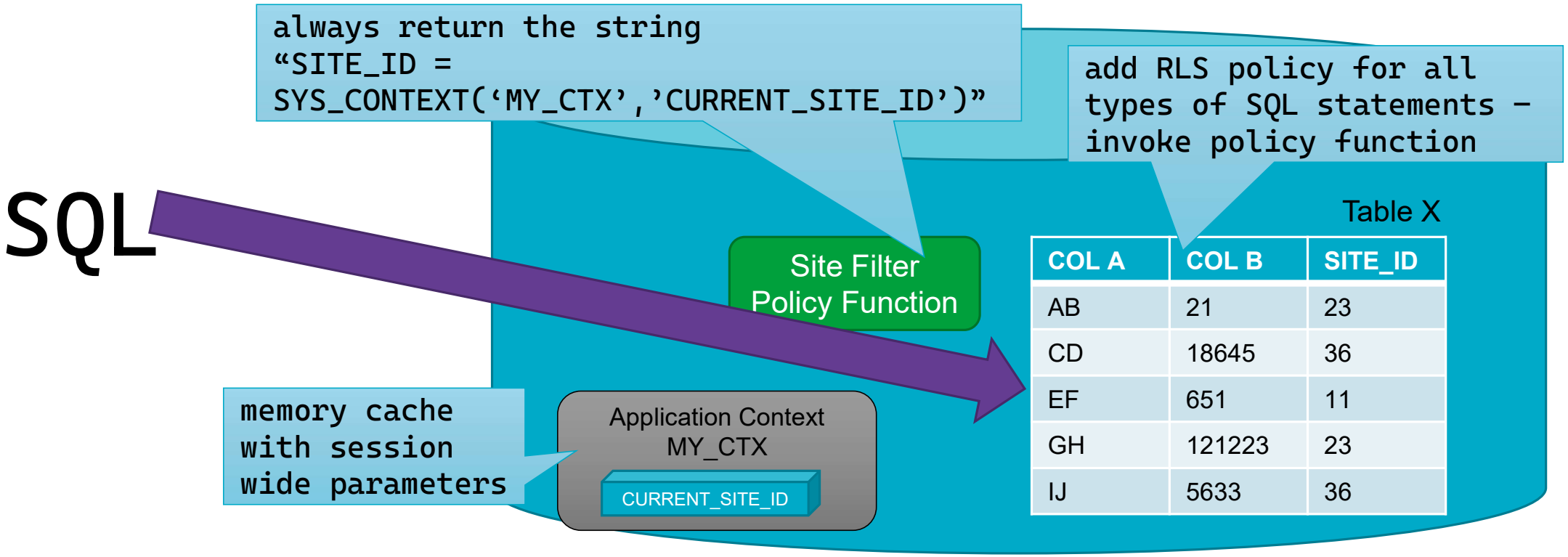
Virtual Private Database

- Iedere table krijgt een extra kolom SITE_ID
- Ieder SQL statement krijgt een extra WHERE conditie:
  - WHERE SITE_ID = <current site's identifier>



Virtual Private Database, Fine Grained Access or Row Level Security Access Policy

- Policy toegepast op een tabel of view
- Wijst naar een functie die een extra conditie aan de SQL statements (SELECT & DML) toevoegt



Proof of Concept

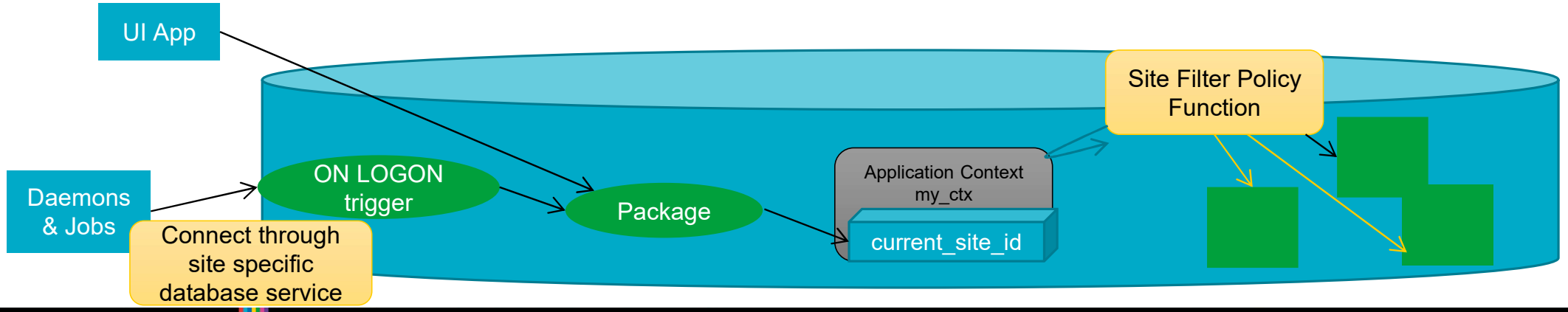
AMIS | CONCLUSION

- Centraliseer en Consolideer de database(s) in de cloud
  - Eerst een enkele database: werkt alles nog met een DB in de cloud
  - Consolideer database: werkt het nog als we VPD policies toevoegen
  - Hoeveel databases passen er eigenlijk in één geconsolideerde DB?

- Techniek
  - Oracle 19c op IaaS VM op Azure
    - Multi-tenant DB gemaakt met
      - 1 PDB voor de as-is database
      - 1 PDB voor de geconsolideerde database met data van 3 vestigingen
      - 1 PDB voor testen (RAT)
  - VPD op basis van database service naam

VPD in de database

- Defineer VPD policy op alle applicatie tabellen met een functie die een filter retourneert
- Creeer een applicatie context en bijbehorende package(s) voor het managen van de sessie
- De meeste applicaties (daemons, jobs, scripts) kunnen gebruik maken van de service bedoeld voor de betreffende vestiging.
  - Via een logon trigger wordt de service naam uitgelezen, en de juiste SITE_ID in de context gezet
- Applicaties (zoals Outsystems/Mendix/…) die centraal zijn, kunnen ook zelf de context laten zetten
  - Hiermee hoeft het betreffende platform niet honderden connection pools te onderhouden naar net zo veel services maar kan het af met één pool.



Hoe VPD opzetten

AMIS | CONCLUSION

- Alles gescript: 10 schema's met 600+ tabellen die aangepast worden
- Stappen:
  - VPD ID kolom toevoegen
  - Primary keys, Foreign keys, (unique) indexen aanpassen
    - Uitdaging: unique index met meerdere nullable kolommen

- Creeer VPD objecten
  - Aparte eigenaar voor deze objecten
    - CONTEXT, Package voor deze context, policy function, logon trigger
    - Service -> VPD SITE ID mapping tabel
- VPD policy activeren
  - Zet op alle tabellen de policy

- Creeer en manage services: PL/SQL API of srvctl

Code!

```
create or replace context site_ctx using vpd_admin.site_ctx;
CREATE OR REPLACE PACKAGE BODY vpd_admin.site_ctx IS

  PROCEDURE set_site_id
  AS
    st_name   vpd_sites.site_service_name%TYPE;
    st_vpd_id vpd_sites.site_vpd_id%TYPE;
  BEGIN
    select store_service_name, store_vpd_id into st_name, st_vpd_id
      from vpd_sites
      where store_service_name = upper(SYS_CONTEXT('USERENV', 'SERVICE_NAME'))
    DBMS_SESSION.SET_CONTEXT('site_ctx', 'site_vpd_id', st_vpd_id);
    end if;
  EXCEPTION
   WHEN NO_DATA_FOUND THEN
      log_connection('NO STORE FOUND', NULL,'No store matched');
  END set_site_id;
END;
/

create or replace trigger set_site_ctx after logon on database
begin
  vpd_admin.site_ctx.set_site_id;
end;
/
```

```
CREATE OR REPLACE FUNCTION vpd_admin.site_vpd_policy_by_id (
  schema_p    IN VARCHAR2,
  table_p     IN VARCHAR2)
 RETURN VARCHAR2
 AS
  store_pred VARCHAR2 (400);
 BEGIN
  store_pred := 'site_vpd_id = SYS_CONTEXT(''site_ctx'', ''store_vpd_id'')';
  RETURN store_pred;
END;
/

alter table my_schema.sample_table add (
  site_vpd_id number invisible default SYS_CONTEXT('site_ctx', 'site_vpd_id') not null);

BEGIN
  dbms_rls.add_policy(
    object_schema   => 'MY_SCHEMA',
    object_name     => 'SAMPLE_TABLE',
    policy_name     => 'MY_SCHEMA_SAMPLE_TABLE',
    function_schema => 'VPD_ADMIN',
    policy_function => 'site_vpd_policy_by_id',
    policy_type     => DBMS_RLS.CONTEXT_SENSITIVE,
    namespace       => 'site_ctx',
    attibute        => 'site_vpd_id'
  );
END;
/
```

Consolideren!

AMIS | CONCLUSION

- Data van een vestiging in de VPD enabled database laden
  - INSERT INTO .. SELECT * FROM …
    - Bovenstaand statement werkt als:
      - De vpd site id kolom invisible is
      - Vpd site id heeft een default waarde (bv SYS_CONTEXT(ctx, siteid))
      - Er geen virtuele kolommen zijn
      - Er geen long (raw) typen zijn
        - Deze omgebouwd naar BLOB of CLOB typen
  - Deze insert statements uitvoeren in de context van de betreffende winkel
  - Voor de PoC een simpel lineair script gebruikt: tijdelijk FK en triggers disabled.

Meer uitdagingen

- Tegelijk een upgrade naar Oracle 19c (vanaf 12.1!)
  - Utl_file met de utl_file_dir parameter in gebruik
    - Dat werkt niet meer sinds 18c: directory object gebruiken!
    - Gepatched in code
- DBMS_ALERT, DBMS_PIPE, UTL_FILE
  - Werkt allemaal prima in één vestiging
  - Maar geconsolideerd? In de cloud?

Performance (hoeveel, hoe groot, hoeveel mag het kosten)

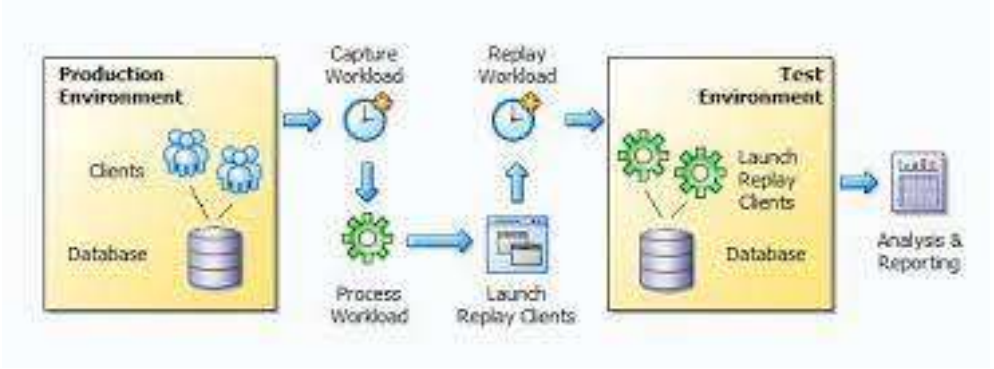
- Honderden vestigingen in één database:
  - Leuk plan, waarschijnlijk niet helemaal haalbaar (in de huidige setup)
  - Reken op meer dan één database
    - Maar hoeveel per database?
  - Geen stress testen bekend vanuit applicaties

  - Strategiën:
    - Scale Up 
    - Scale Out 
    - Tunen: Storage, redesing, refactor… indexen/MV/In Memory

Load Test met realistische workload

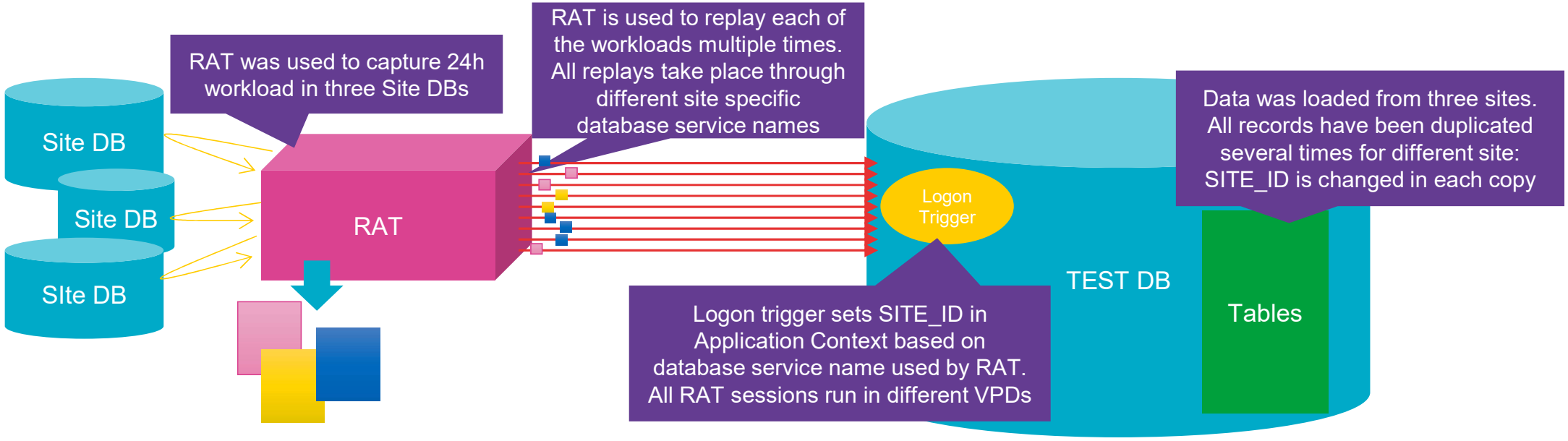
- Een realistische load krijgen op de consolidated database?
  - Zelfde queries en PL/SQL als op een lokale vestiging
  - Opschalen naar tientallen keren de workload

- Oracle Real Application Testing
  - Capture workload in live database (eventueel gefilterd)
  - Speel workload af op doel DB
    - Spelen met de tijd
    - Meer workloads (consolidated replay)
  - Analyseer de resultaten
    - Vanuit Azure gezien (resource gebruik), AWR, ASH, SQL Tuning sets



Hoe krijg je zoveel workloads? (met maar 3 datasets en 3 captures)

- Een losse PDB met VPD policy klaar
- Gebaseerd op de data van 3 vestigingen, deze is gedupliceerd tot de gewenste hoeveelheid
- RAT captures zijn gemaakt van die zelfde 3 vestigingen gedurende 24 uur
- Deze captures worden afgespeeld in de PDB, iedere losse capture wordt gericht tegen de service van één vestiging in de PDB zodat de VPD policy goed staat.



RAT deel 2

AMIS | CONCLUSION

- PL/SQL API is c00l!
  - Heb je meer dan 3 captures die je wil testen met RAT: gebruik de API
  - Enterprise Manager Cloud Control werkt goed zolang het aantal captures beperkt blijft.
- AWR reports, SQL tuning sets helpen bij het bepalen of je verbetering ziet
  - Consolidated replay reports zijn nauwelijks te lezen als je heel veel captures gebruikt.

- Testen tegen verschillende VM shapes: 8 vcpu, 32 vcpu, 64 vcpu
- Testen met verschillende workloads: 50 vestigingen, 100 vestigingen, 150+?

Uitdagingen

AMIS | CONCLUSION

- Azure VM:
  - In het begin te klein geschaald
  - Azure Premium SSD: aantal IOPS is beperkt, hoe groter hoe meer
- Testen uitvoeren
  - EMCC: niet handig, zeker bij meer workloads
  - RAT tooling neemt ook wat resources
  - CDB/PDB: wanneer is een SQL tuning set bruikbaar?
- Workload:
  - Veel PL/SQL, DBMS_PIPE, DBMS_ALERT, UTL_FILE…
  - Locking, DDL's, zeer kleine tabellen (block contention)
  - Missende indexen op Foreign keys
- Resultaten vissen uit een brij van cijfers:
  - AWR's, SQL tuning sets

Resultaten van RAT

AMIS | CONCLUSION

- We weten nu hoeveel vestigingen in één database passen (zonder aanpassingen)
- We weten waar de bottlenecks zitten
- We weten nu ook waar de verbeteringen in kunnen zitten
  - Tabel partitionering geeft enorme verbeteringen
  - Indexen scherp naar kijken
  - Meer generieke data delen (niet meer in de vpd policy plaatsen dus)
  - Opschonen van data, code (veel historische code, data)
  - Optimaliseren van SQL en PL/SQL code

Applicaties en releases

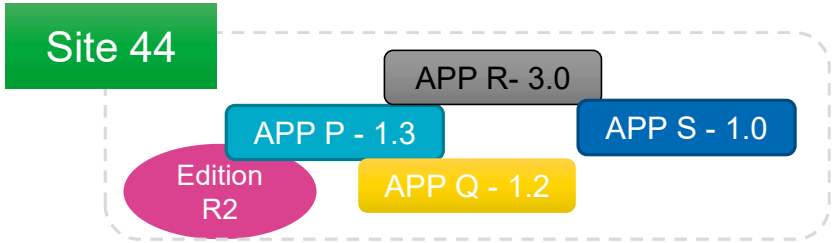
- Niet alle vestigingen draaien dezelfde software versies
- Centrale applicaties kunnen ook in verschillende versies bestaan

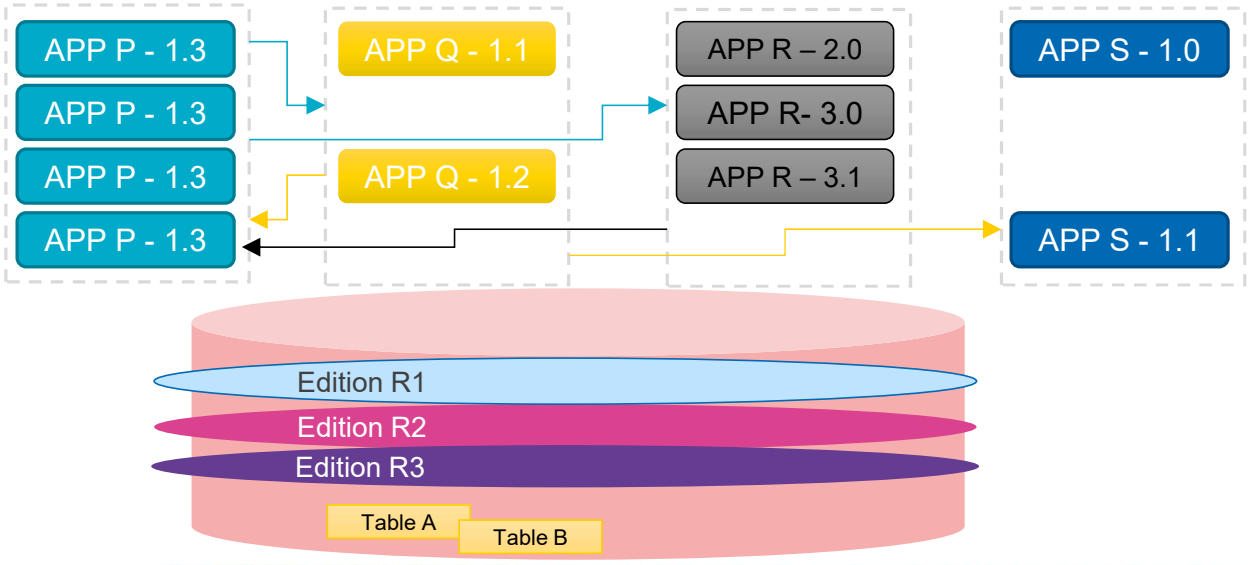
- Niet alle vestigingen gebruiken dezelfde versie van database objecten
  - Verschillende versies van Packages, Types en Views

- Hoe maken we het mogelijk dat binnen dezelfde database verschillende versies van objecten kunnen bestaan?
  - (zonder dat we vestigingen moeten verplaatsen naar een andere database)

Edition based redefinition

- Elke vestiging krijgt de juiste database Edition toegewezen
  - Dit kan bijvoorbeeld via de database service van de vestiging
- Een vestiging gebruikt altijd één versie van de applicatie(s)
  - Deze applicatie versie hoort dan ook bij de betreffende Edition





Application version compatibility matrix

PoC resultaat

Deze PoC bewees dat:

- Consolidatie middels VPD technisch werkt
- Een database in Azure met applicatieservers on-premise kan werken
- Performance is goed genoeg
- Veel legacy aanwezig, en nu eindelijk goed geinventariseerd
- Low code platform upgrade en gebruik van VPD gaat ook lukken
- Reductie van Oracle licenties en support kosten

Dank voor uw aandacht