ORACLE
DevLive

The Future of Data and AI

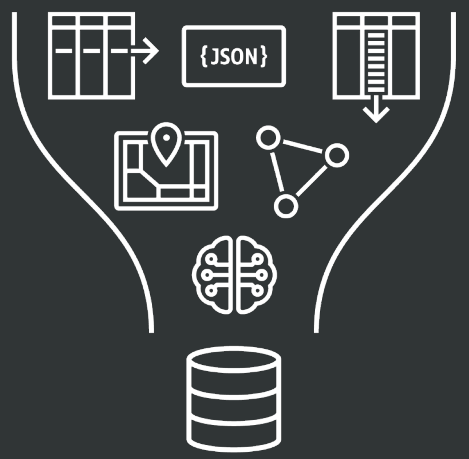
Gerald Venzl
Lead Product Manager of Developer Initiatives
Oracle Database Development

Oracle Database Vision

Make modern apps and analytics
easy to develop, run, and leverage AI
for all use cases at any scale

How we deliver the Vision

Complete and Simple Platform for All Data Management Needs



Converged Database

Complete: all modern data types, workloads, and development styles

Simple: Adds SQL statements, not another database, to support requirements of modern applications

Running on

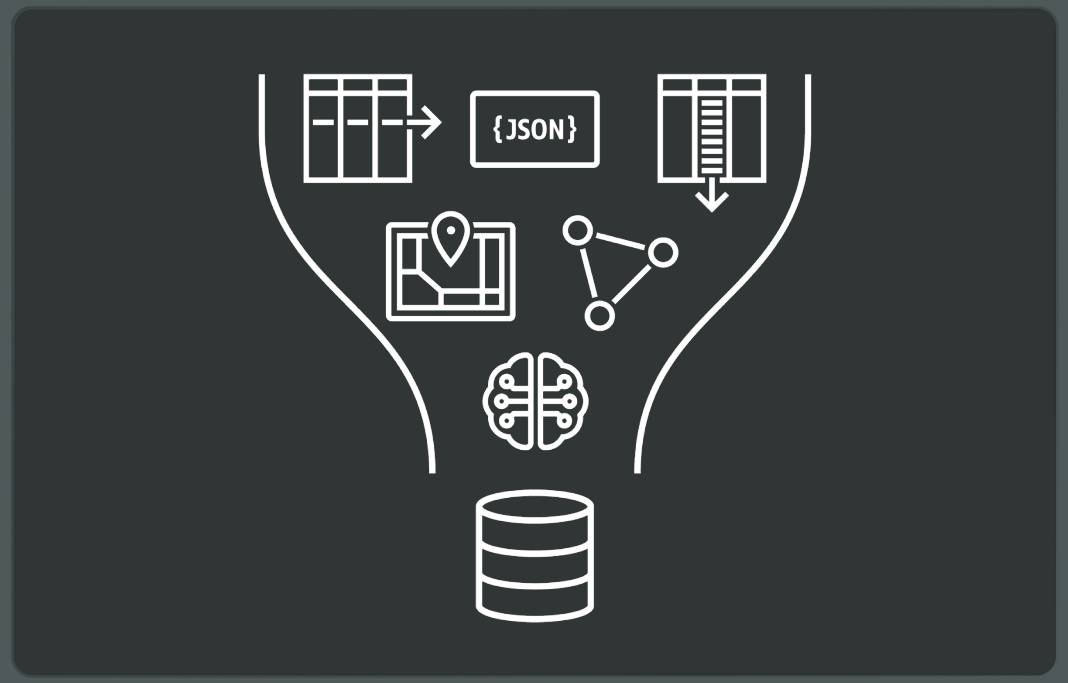
Autonomous Database

Fully Automated: end-to-end automation across the entire database lifecycle

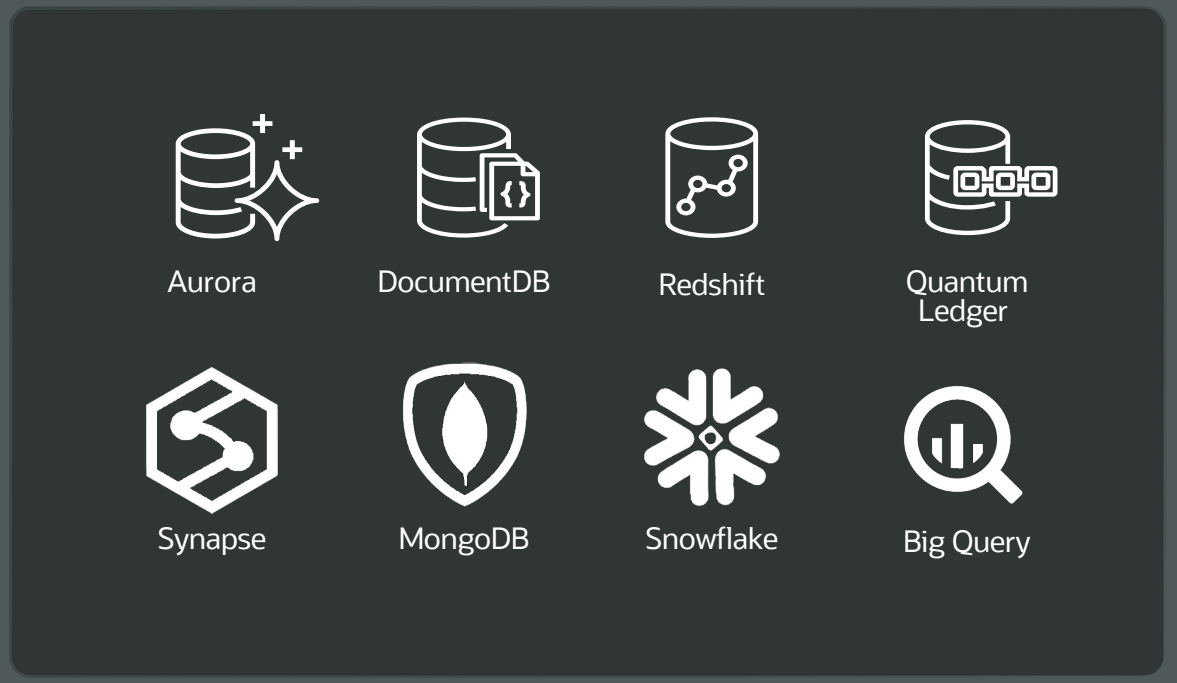
Cloud-Native: empowering organizations to build and run scalable apps in a modern, dynamic environment

Comparing Database Strategies

Run a converged, standards-based DB



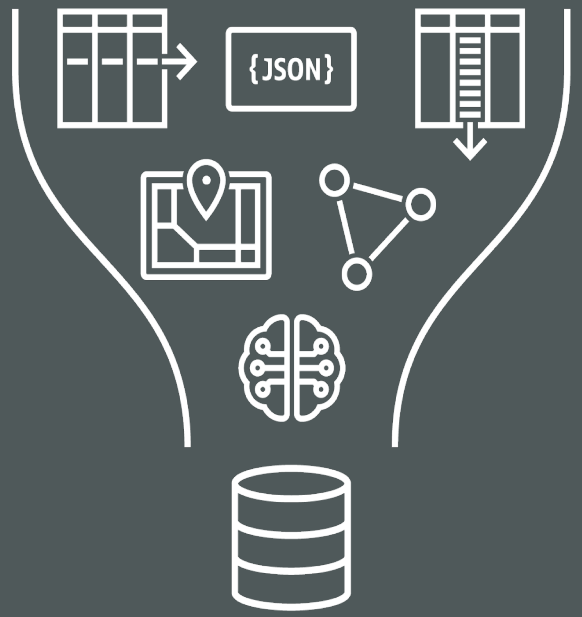
Instead of single-use proprietary databases



Developers and IT focus on Innovation

Developers and IT focus on Integration

Next Generation Converged Database – Oracle Database 23ai

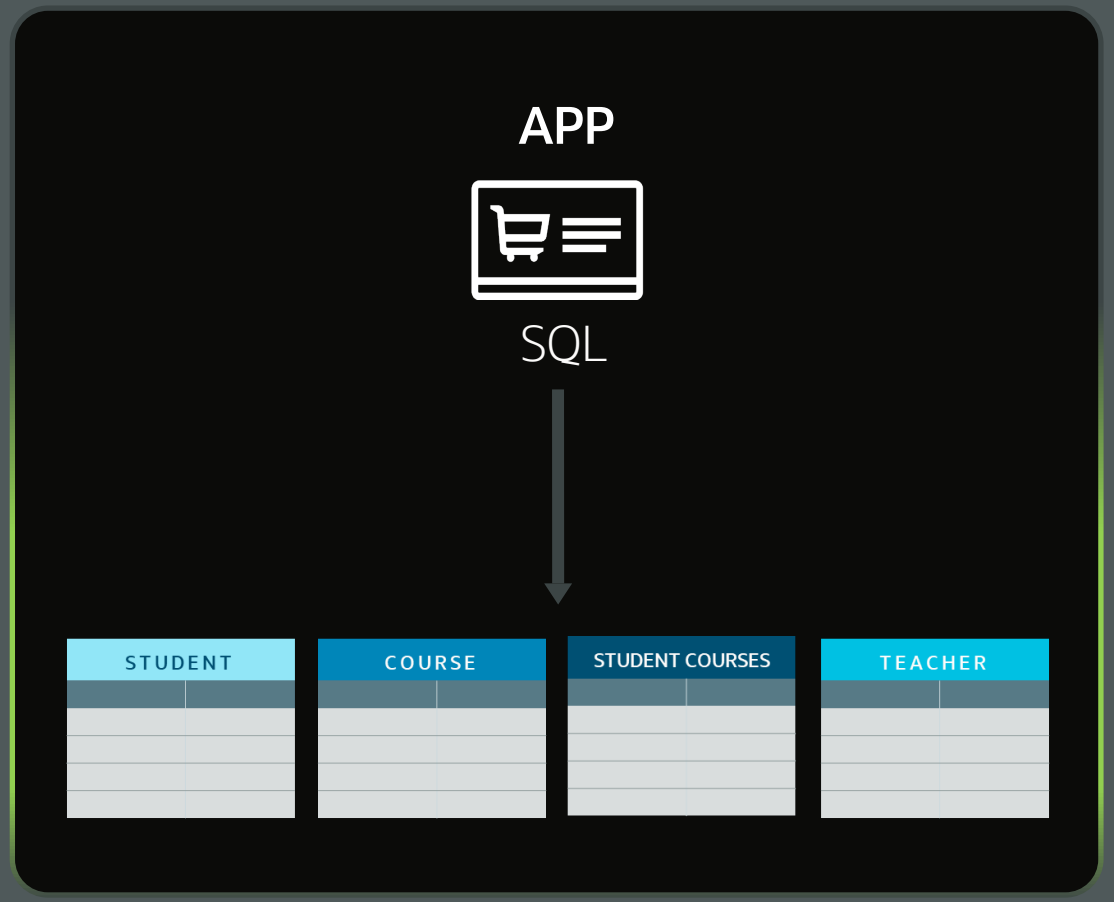


The Converged Database approach has delivered incredible value to businesses

- Further advances in the industry continue to push the boundaries of data management

Hence Oracle is introducing revolutionary technologies that further unify data management requirements at a more fundamental level

Relational Databases were created with the mission to – ensure data consistency and enable declarative access to any data

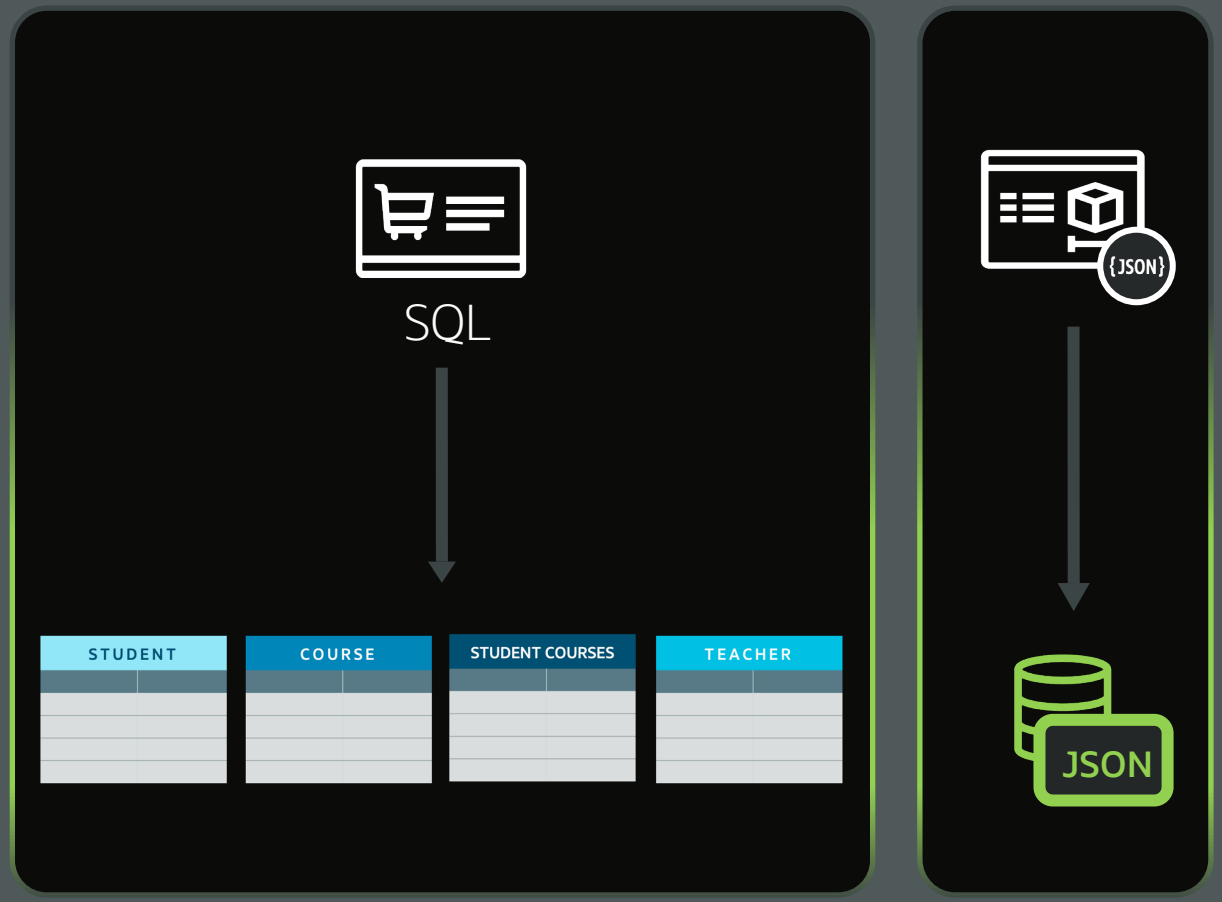


Simple Data Organization

Powerful Declarative Language

Optimizations Transparent to Apps

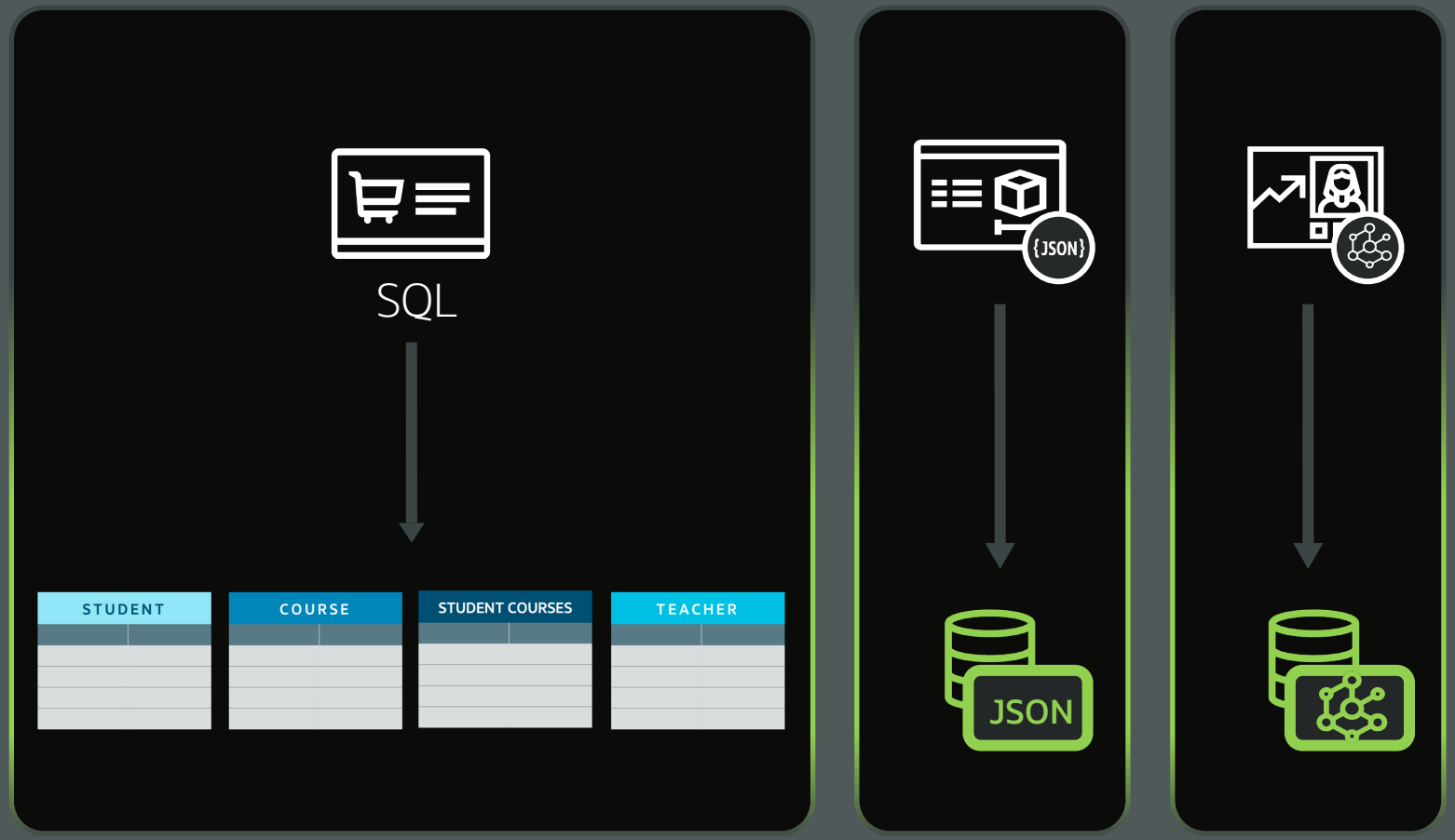
Then JSON Databases were introduced with the mission to – make data management simpler for application developers



Easy to map application objects to JSON documents

Forced choice between simplicity of JSON versus the power of relational

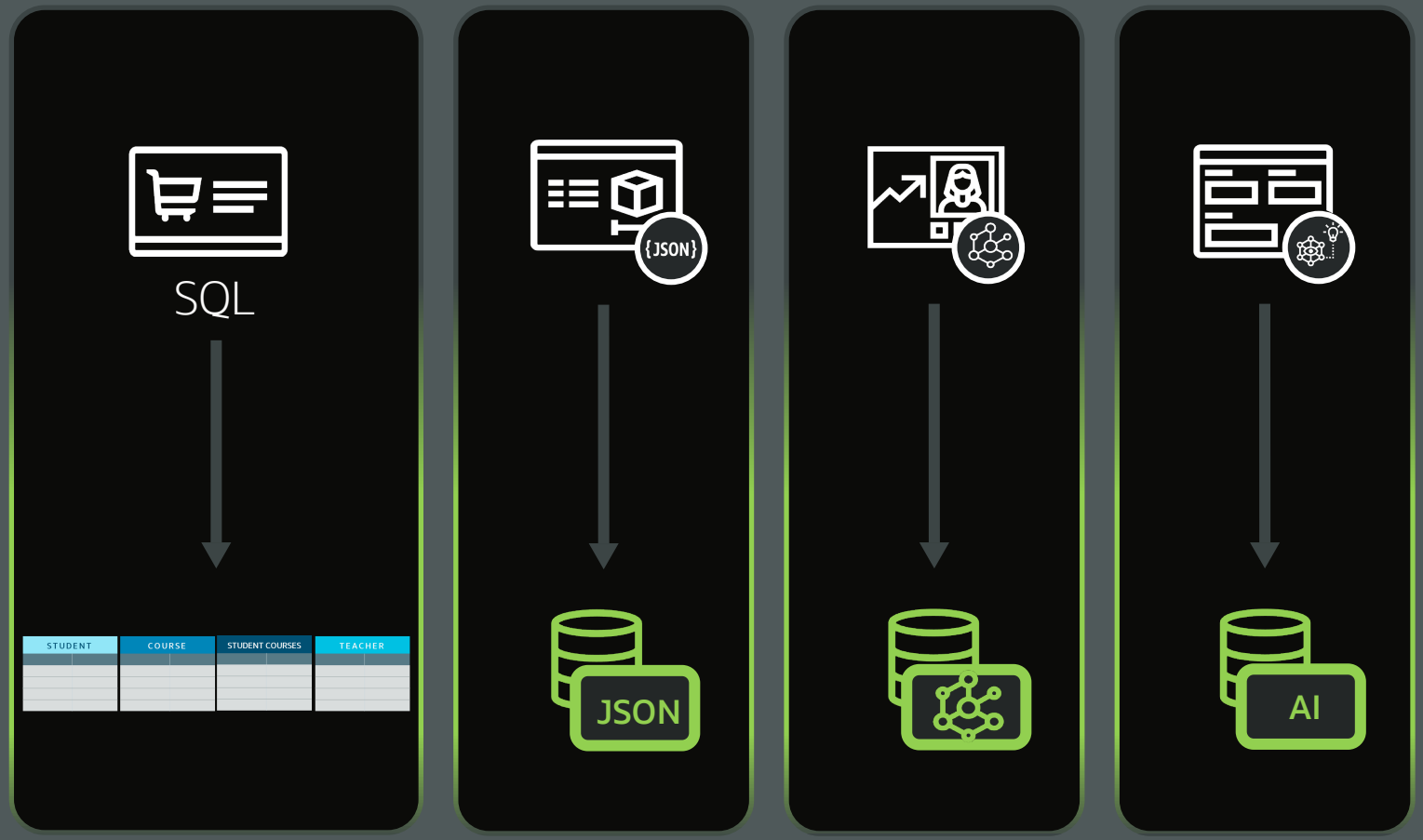
Then Graph Databases were introduced with the mission to – make it simple to navigate connections between data



Great for querying social networks, supply chains, sequences of events, etc.

Forced choice between simplicity of Graph versus the power of relational

Now Vector Databases have been introduced with the mission to - use AI to search data based on its semantic content

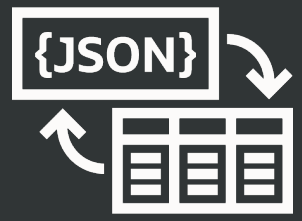


The worlds of data and app dev have further fractured

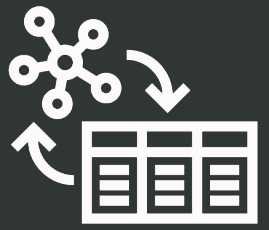
Cannot search for data using a combination of SQL, JSON Documents, Graphs and AI Vector Search

Revolutionary new Oracle technologies unify these worlds, eliminating the need for limited and costly fractured solutions

Unification of JSON and Relational



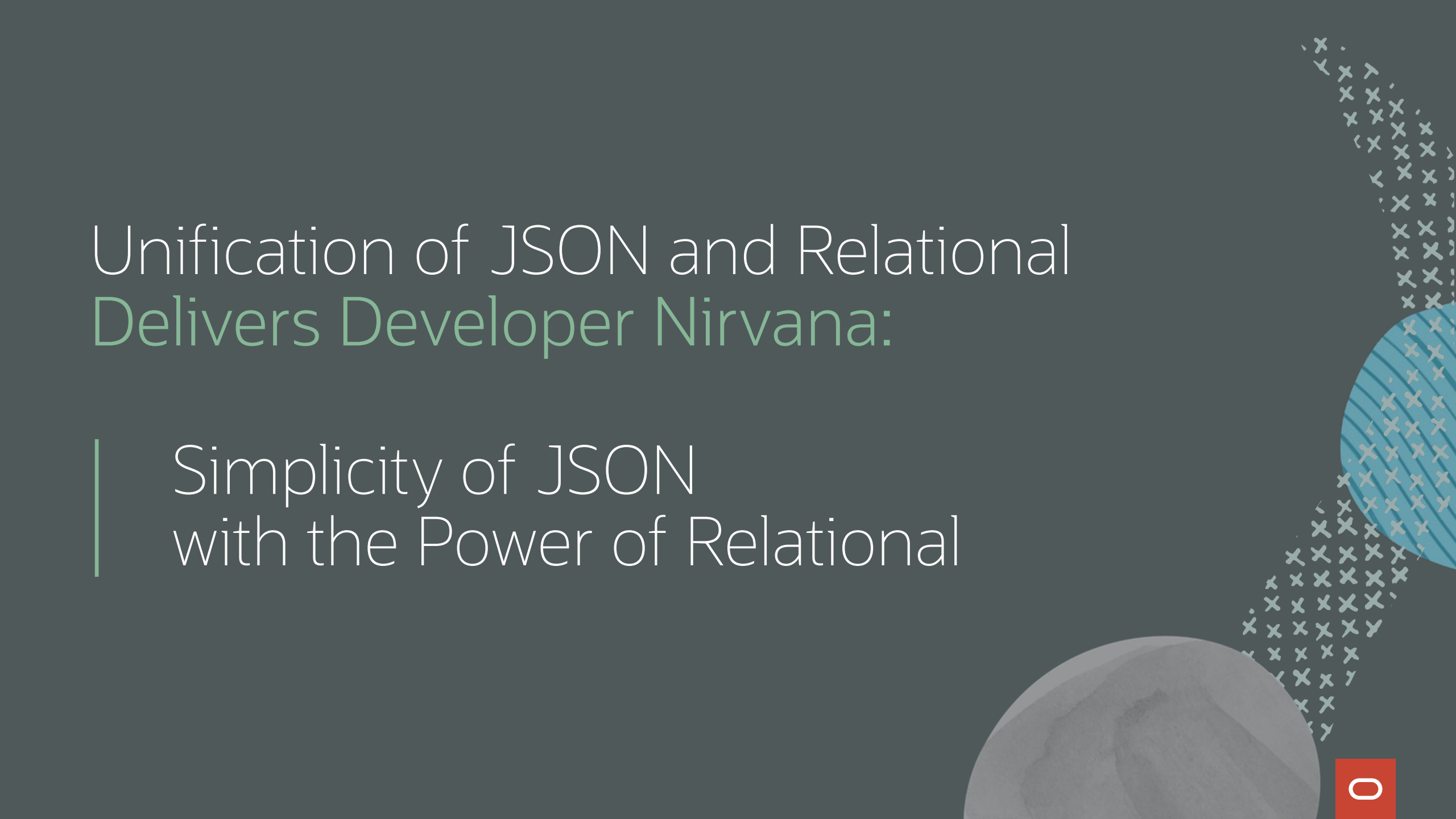
Unification of Graph and Relational



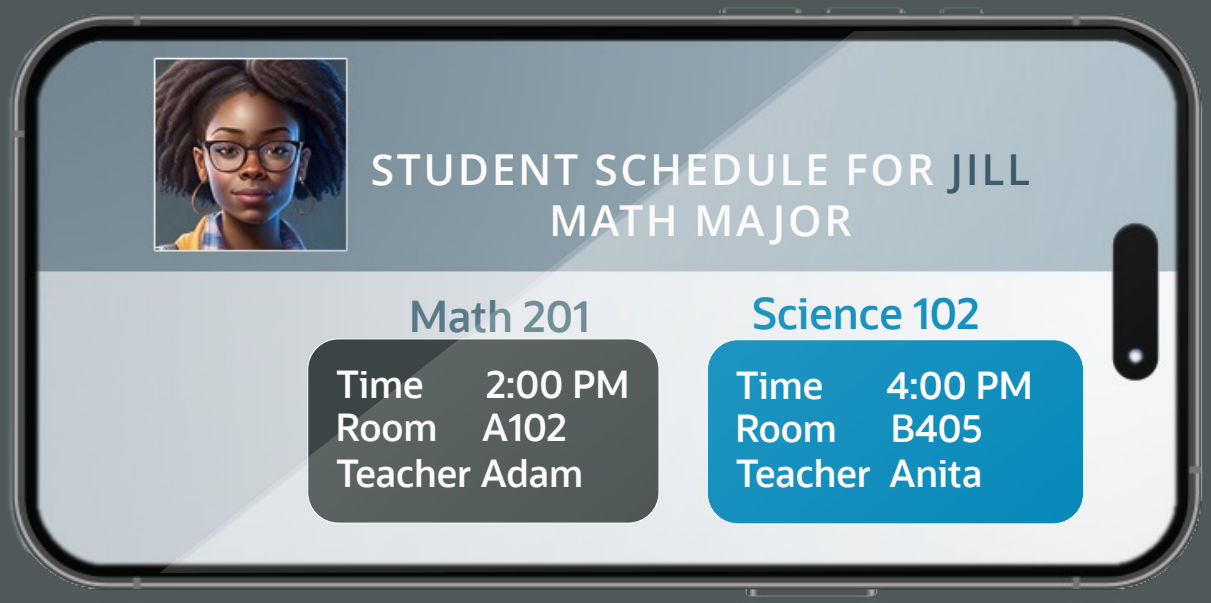
Unification of AI and Databases

Unification of JSON and Relational Delivers Developer Nirvana:

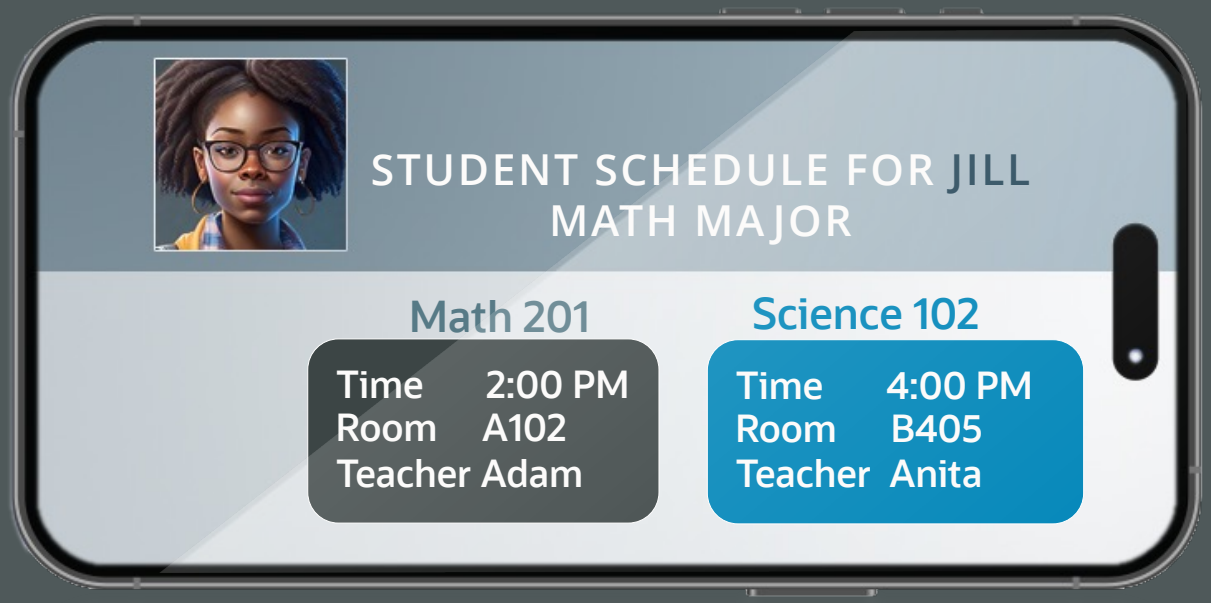
Simplicity of JSON with the Power of Relational



Imagine we've been asked to build an app that creates a student course schedule



Storing each student's schedule as a single JSON document is simplest for development



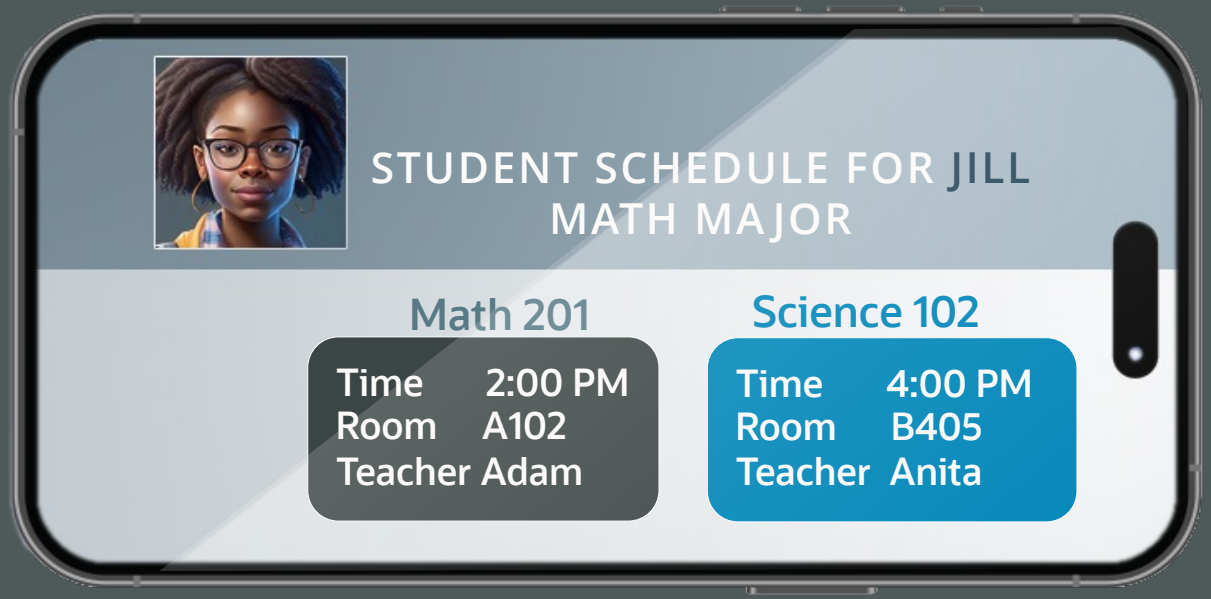
SCHEDULE FOR: JILL

```
"student"     : "S3245",
"name"        : "Jill",
"major"       : "Math",
"schedule"    :
  [  {
       "time"    : "14:00",
       "course"  : "Math 201",
       "room"    : "A102",
       "teacher" : "Adam"
     },
     {
       "time"    : "16:00",
       "course"  : "Science 102",
       "room"    : "B405",
       "teacher" : "Anita"
     }
  ]
}
```

Reading and writing the schedule as a JSON document is easy

One GET operation retrieves a JSON document containing all the schedule data in a simple hierarchical format

Any changes can be stored in one PUT of the document



GET

PUT

SCHEDULE FOR: JILL

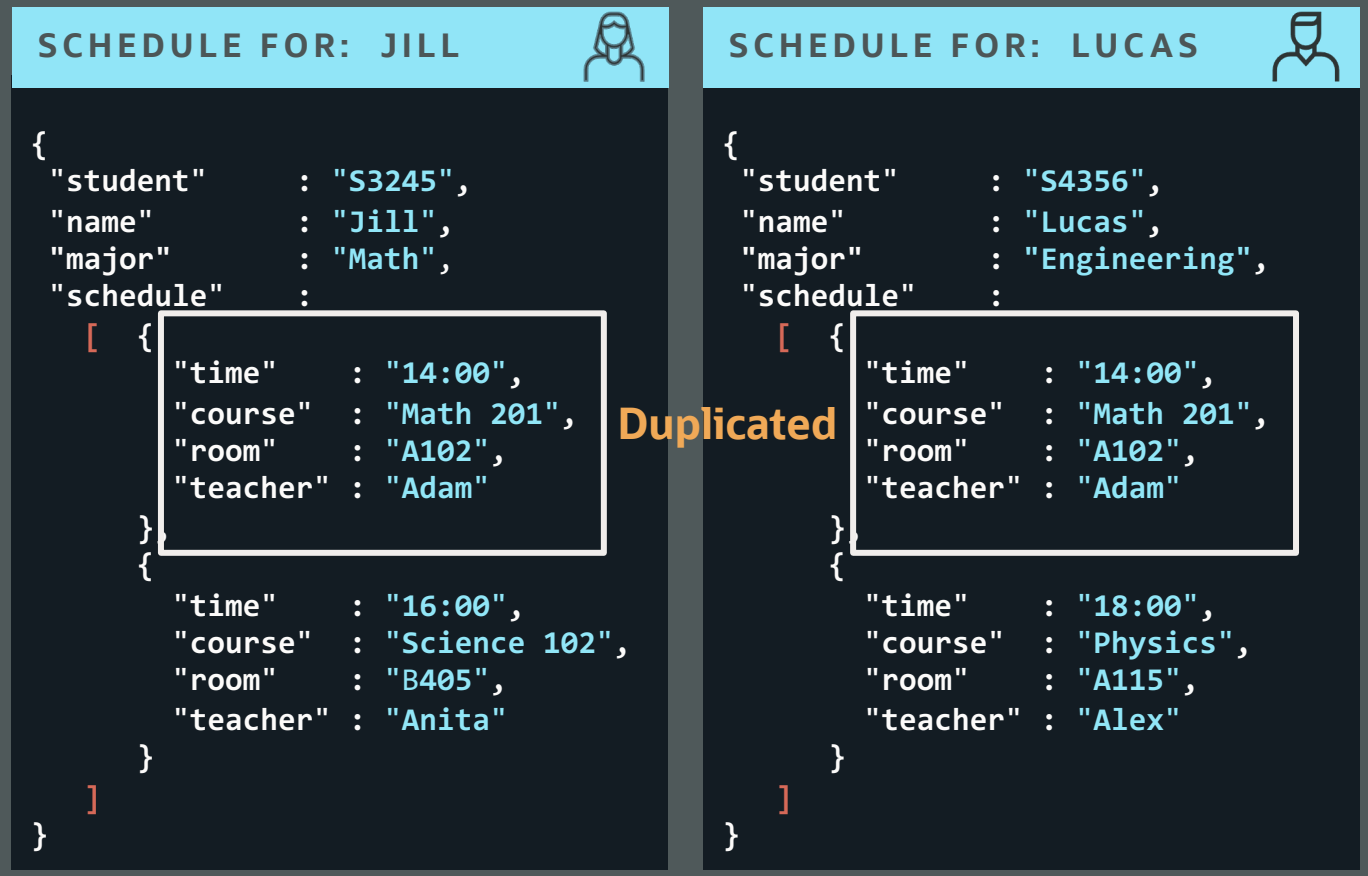
```
"student"     : "S3245",
"name"        : "Jill",
"major"       : "Math",
"schedule"    :
  [  {
       "time"    : "14:00",
       "course"  : "Math 201",
       "room"    : "A102",
       "teacher" : "Adam"
     },
     {
       "time"    : "16:00",
       "course"  : "Science 102",
       "room"    : "B405",
       "teacher" : "Anita"
     }
  ]
}
```

However, there are downsides to storing data as JSON documents

Document storage models suffer from data consistency issues

For example, all students taking a course have a copy of the course schedule and teacher information in their document, making updates expensive and risky

Also, declarative SQL is far more powerful than the queries that JSON databases provide



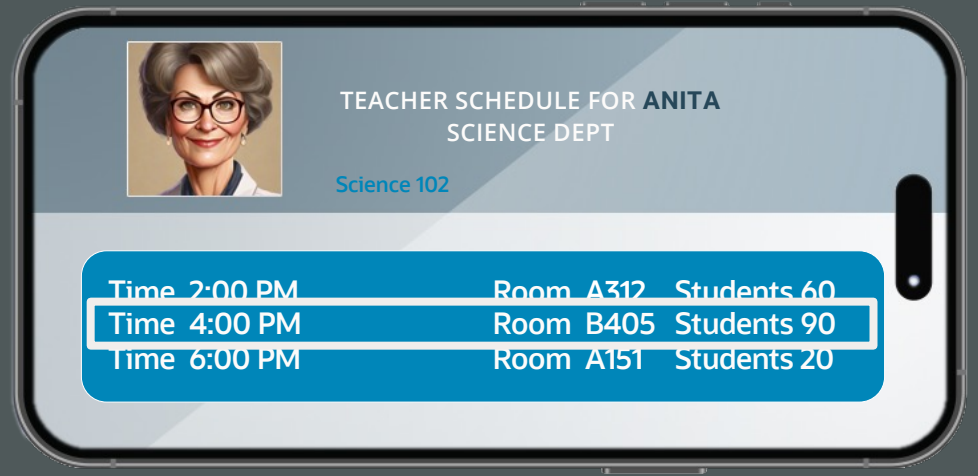
JSON creates severe issues when data is used for multiple use cases

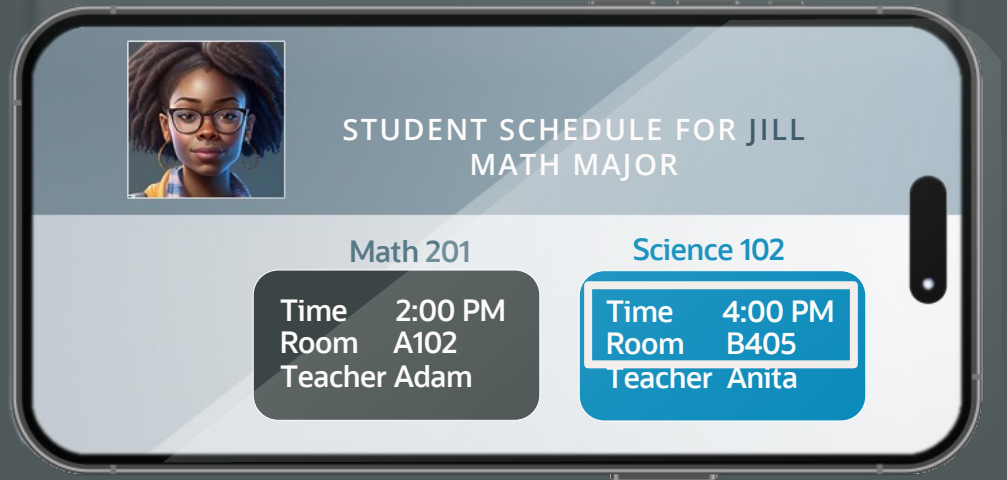
For example, adding a teacher schedule use case requires a new document shape with the teacher as the root

- The teacher document duplicates course data that is also in the student documents

Changing the classroom for a course now requires atomically updating

- Many student schedule documents
- AND many teacher schedule documents

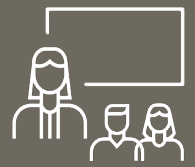




Storing data in normalized Relational format enables consistency and declarative SQL

STUDENT

STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...



COURSE

CID	CLASS	ROOM	TIME	TCHID
C123	MATH 201	A102	14:00	T543
C345	SCIENCE 102	B405	16:00	T789
...	...	...	...	...
...	...	...	...	...

STUDENT COURSES

STUID	CID
S3245	C123
...	...
S3245	C345
...	...

TEACHER

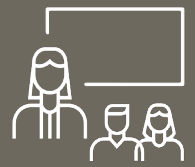
TCHID	TEACHER	TINFO
...	...	...
T543	Adam	...
T789	Anita	...
...	...	...

Normalized rows are the single source of truth for the data they store

However, relational apps must join multiple tables to retrieve a student schedule

STUDENT

STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...



COURSE

CID	CLASS	ROOM	TIME	TCHID
C123	MATH 201	A102	14:00	T543
C345	SCIENCE 102	B405	16:00	T789
...	...	...	...	...
...	...	...	...	...

STUDENT COURSES

STUID	CID
S3245	C123
...	...
S3245	C345
...	...

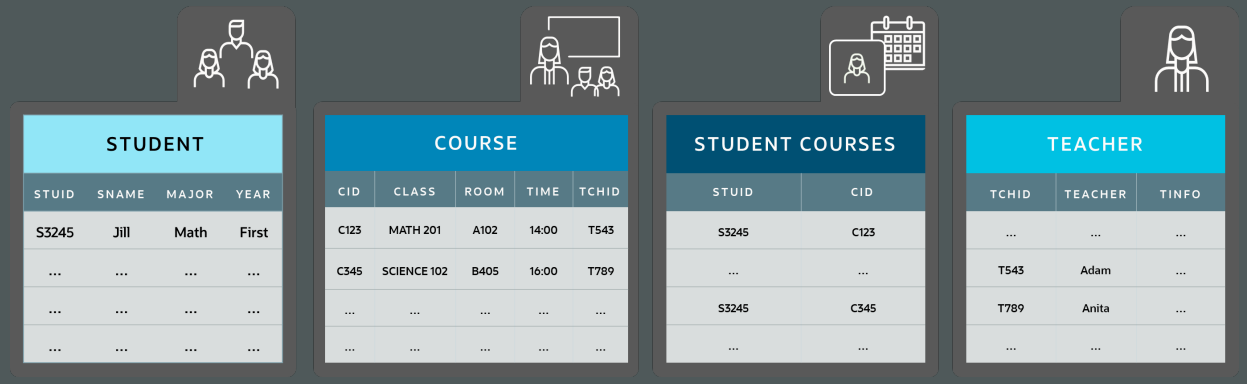
TEACHER

TCHID	TEACHER	TINFO
...	...	...
T543	Adam	...
T789	Anita	...
...	...	...

And construct application objects of the resulting data (ORM mapping)

JSON Relational Duality eliminates these tradeoffs

Delivers the simplicity of JSON for developers, with the power of relational data



STUDENT

STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...

COURSE

CID	CLASS	ROOM	TIME	TCHID
C123	MATH 201	A102	14:00	T543
C345	SCIENCE 102	B405	16:00	T789
...	...	...	...	...
...	...	...	...	...

STUDENT COURSES

STUID	CID
S3245	C123
...	...
S3245	C345
...	...

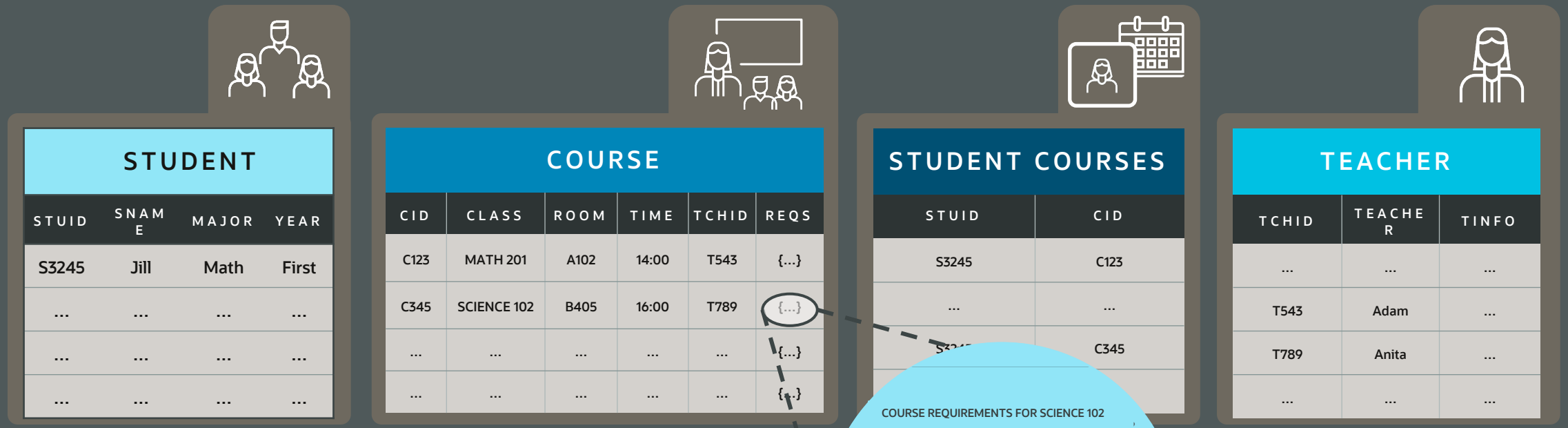
TEACHER

TCHID	TEACHER	TINFO
...	...	...
T543	Adam	...
T789	Anita	...
...	...	...

SCHEDULE FOR: JILL

```
{
 "student"      : "S3245",
 "name"         : "Jill",
 "major"        : "Math",
 "schedule"     :
   [  {
        "time"     : "14:00",
        "course"   : "Math 201",
        "room"     : "A102",
        "teacher"  : "Adam"
      },
      {
        "time"     : "16:00",
        "course"   : "Science 102",
        "room"     : "B405",
        "teacher"  : "Anita"
      }
   ]
}
```

Data is stored as rows in tables to provide the consistency and declarative query benefits of relational



STUDENT

STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...

COURSE

CID	CLASS	ROOM	TIME	TCHID	REQS
C123	MATH 201	A102	14:00	T543	{...}
C345	SCIENCE 102	B405	16:00	T789	{...}
...	...	...	...	...	{...}
...	...	...	...	...	{...}

STUDENT COURSES

STUID	CID
S3245	C123
...	...
	C345

TEACHER

TCHID	TEACHER	TINFO
...	...	...
T543	Adam	...
T789	Anita	...
...	...	...

COURSE REQUIREMENTS FOR SCIENCE 102

```
{
  "course"   : "Science 102",
  "preqs"    :
  [  {
       "course"     : "MATH 101",
       "minGrade"   : "C",
     },
     {
       "course"     : "Science 101",
       "minGrade"   : "B",
     },
     …
  ]
}
```

Tables can include JSON columns to store data whose schema is dynamic or evolving

New JSON Duality Views declare the mapping between rows and JSON

The view definition mirrors the structure of the desired JSON

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{
   student    : stuid
   name       : sname
   major      : major
   schedule   : student_courses
   [ {
    course
      {
         time      : time
         course    : cname
         courseId  : cid
         room      : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```

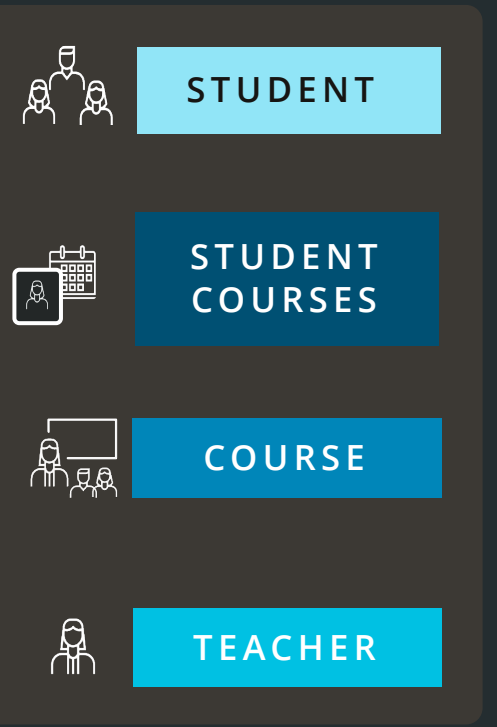
Uses familiar GraphQL syntax

STUDENT SCHEDULE FOR: JILL

```
{
 "student"     : "S3245",
 "name"        : "Jill",
 "major"       : "Math",
 "schedule"    :
   [  {
        "time"    : "14:00",
        "course"  : "Math 201",
        "room"    : "A102",
        "teacher" : "Adam"
      },
     …
   ]
}
```

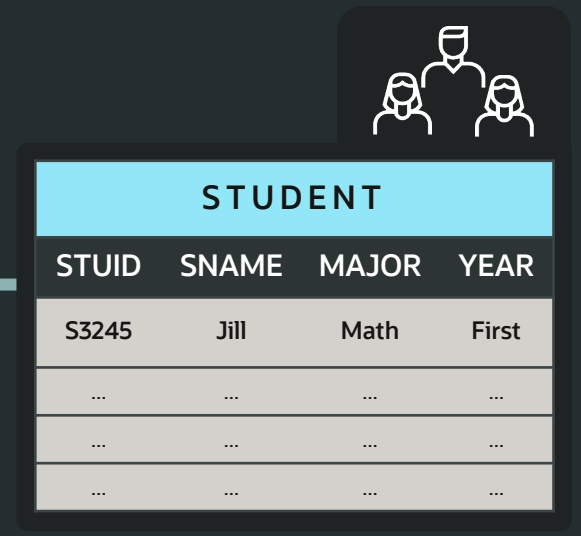
The view simply specifies the tables that contain the data to include in the JSON document

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{
   student    : stuid
   name       : sname
   major      : major
   schedule   : student_courses
   [ {
    course
      {
         time      : time
         course    : cname
         courseId  : cid
         room      : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```



And specifies the table columns that hold the values

```
CREATE JSON DUALITY VIEW student_schedule
AS student
{
   student    : stuid
   name       : sname
   major      : major
   schedule   : student_courses
   [ {
    course
      {
         time      : time
         course    : cname
         courseId  : cid
         room      : room
         teacher @unnest
         {
           teacher : tname
         }
      }
    } ]
};
```



STUDENT

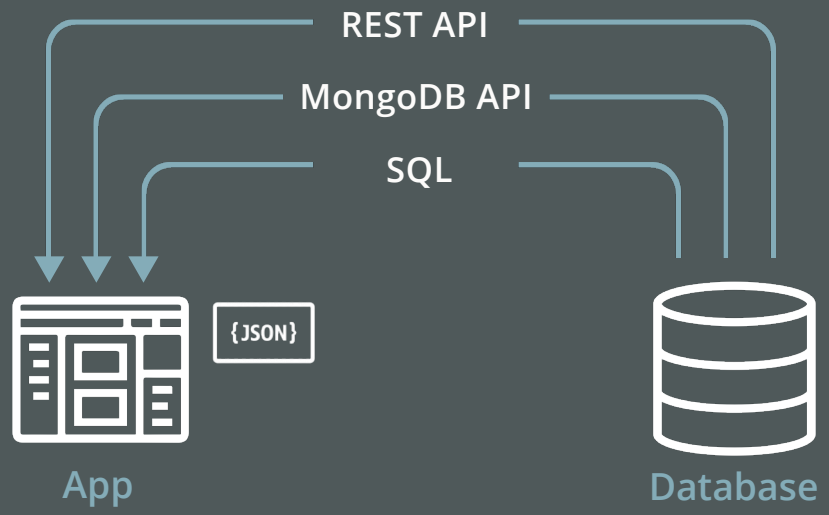
STUID	SNAME	MAJOR	YEAR
S3245	Jill	Math	First
...	...	...	...
...	...	...	...
...	...	...	...

JSON Duality Views are simple to query using document APIs

Apps use standard REST APIs to GET a document from the View

```
GET school.edu/student_schedule?q={"student":{"$eq":"Jill"}}
```

Views can also be accessed using
a MongoDB compatible API or SQL



JSON Duality Views are also simple to update

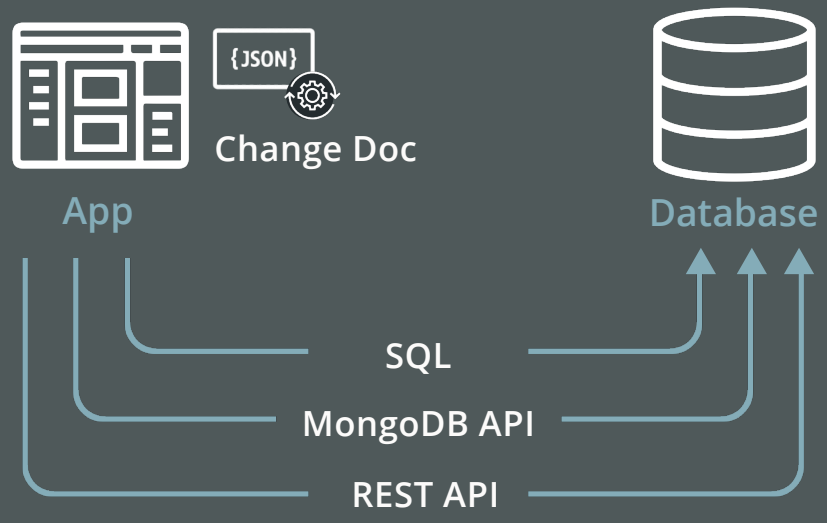
Apps edit the document they previously got

Then simply PUT the document back into the View

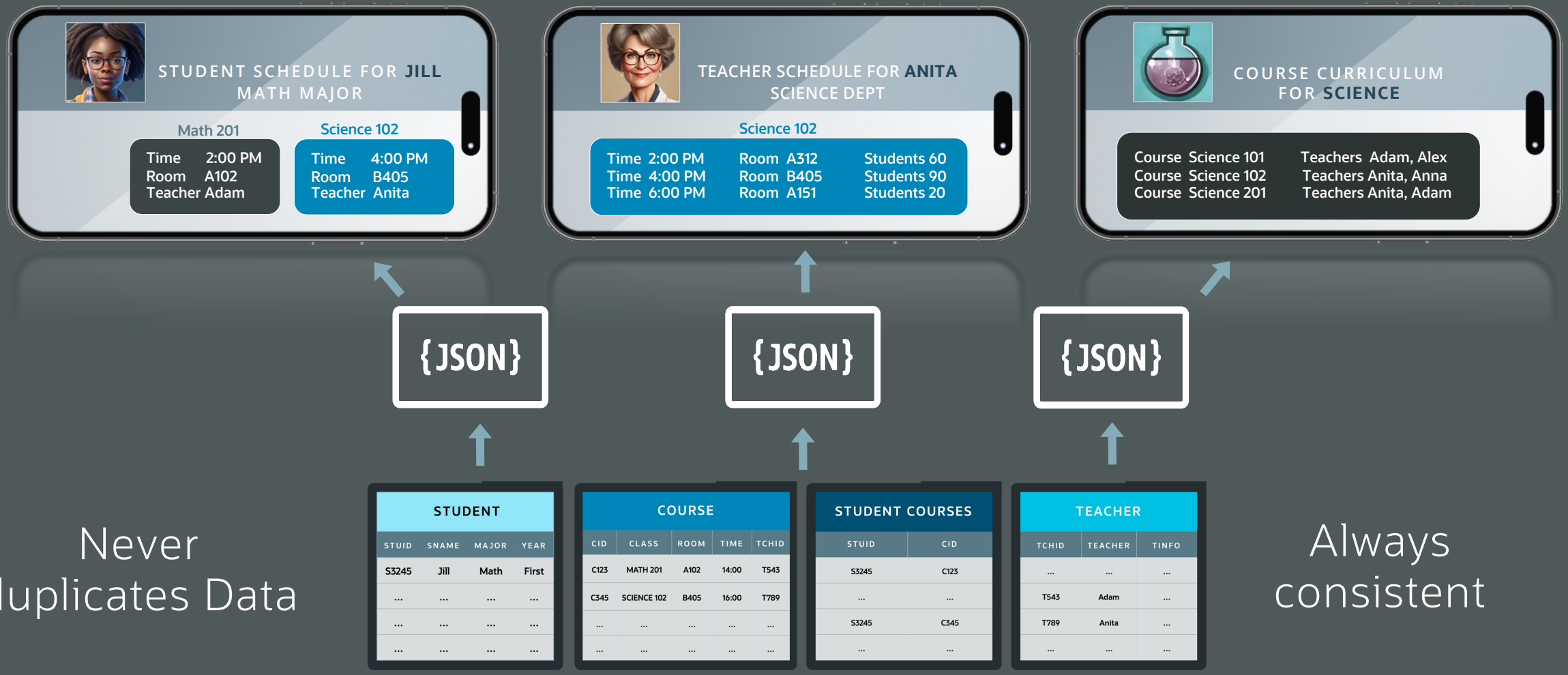
- Or write it with the MongoDB API or SQL

```
PUT school.edu/student_schedule/:stuid
```

As part of the update, the database detects the changes made to the document and only modifies the underlying table rows that have changed



JSON Duality views allow the same underlying data to be customized to match the needs of each app use case

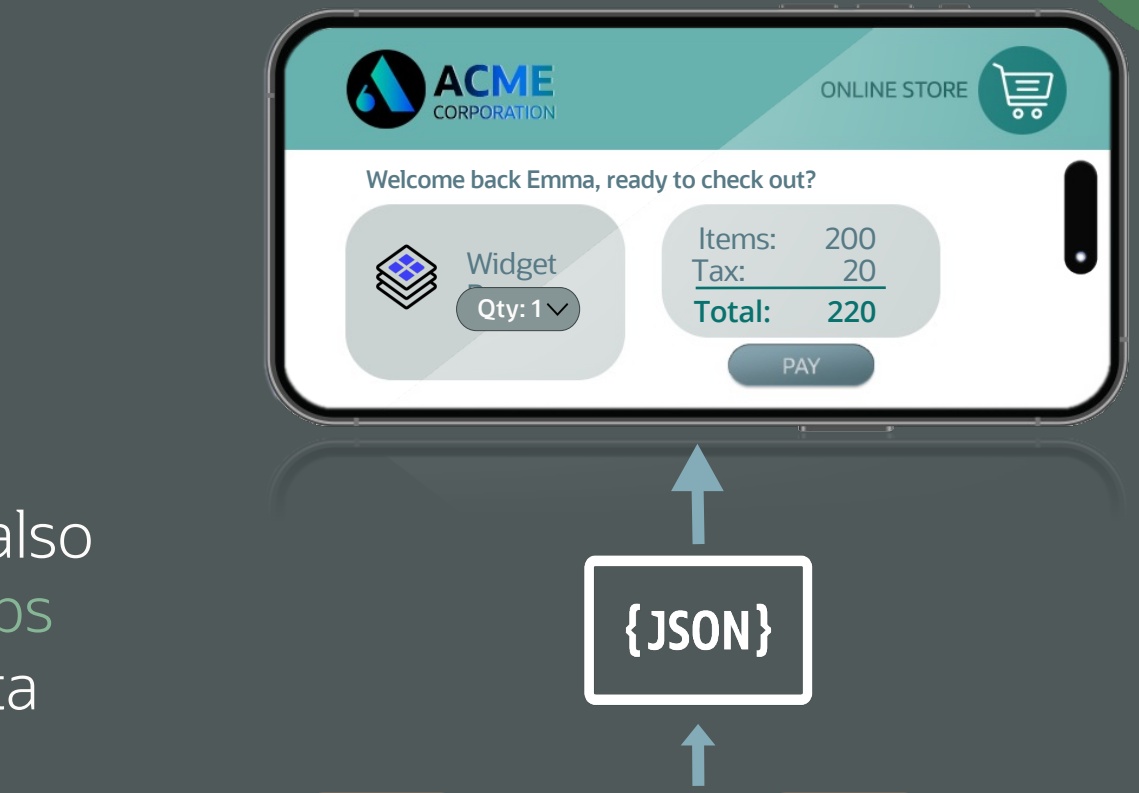


Never duplicates Data

Always consistent

Much better for App Dev than JSON Databases!

Using Duality, developers can also add new document-centric apps on top of existing relational data



{JSON}

ORDERS

ORDER	NAME	PROD_ID	PRICE
345	EMMA	948	$200
...	...	...	...
.	...	...	...
...	...	...	...

CUSTOMERS

NAME	ADDRS	PHONE
....	...	...
EMMA	...	(415)...
...	...	...
...	...	...

PRODUCTS

PROD_ID	PROD_NAME
...	...
...	...
948	WIDGET B
...	...

“Oracle’s JSON Relational Duality, a truly revolutionary solution, is perhaps one of the most important innovations in information science in 20 years.”

CARL OLOFSON, RESEARCH VP,
DATA MANAGEMENT SOFTWARE, IDC

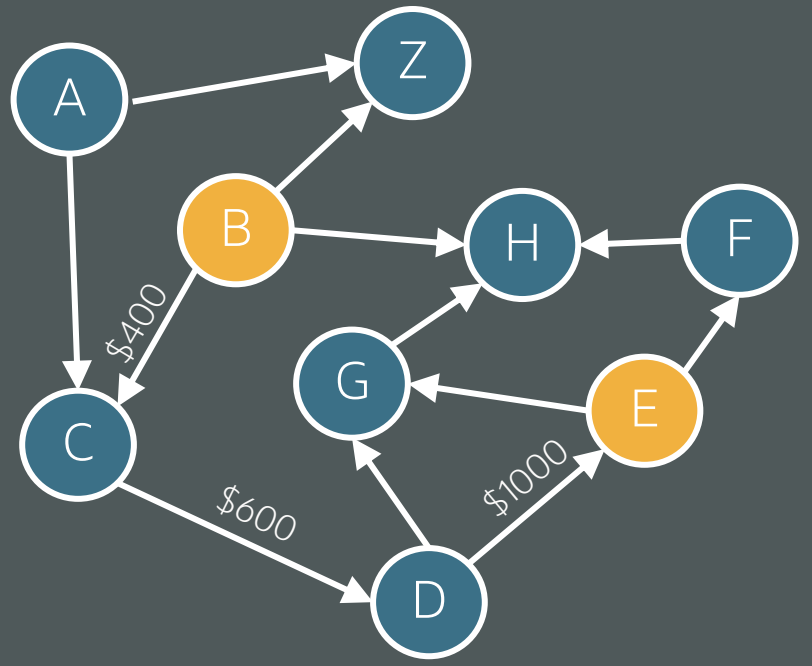
Unification of Graph and Relational Delivers Developer Nirvana:

Navigation simplicity of Graph with the Power of Relational

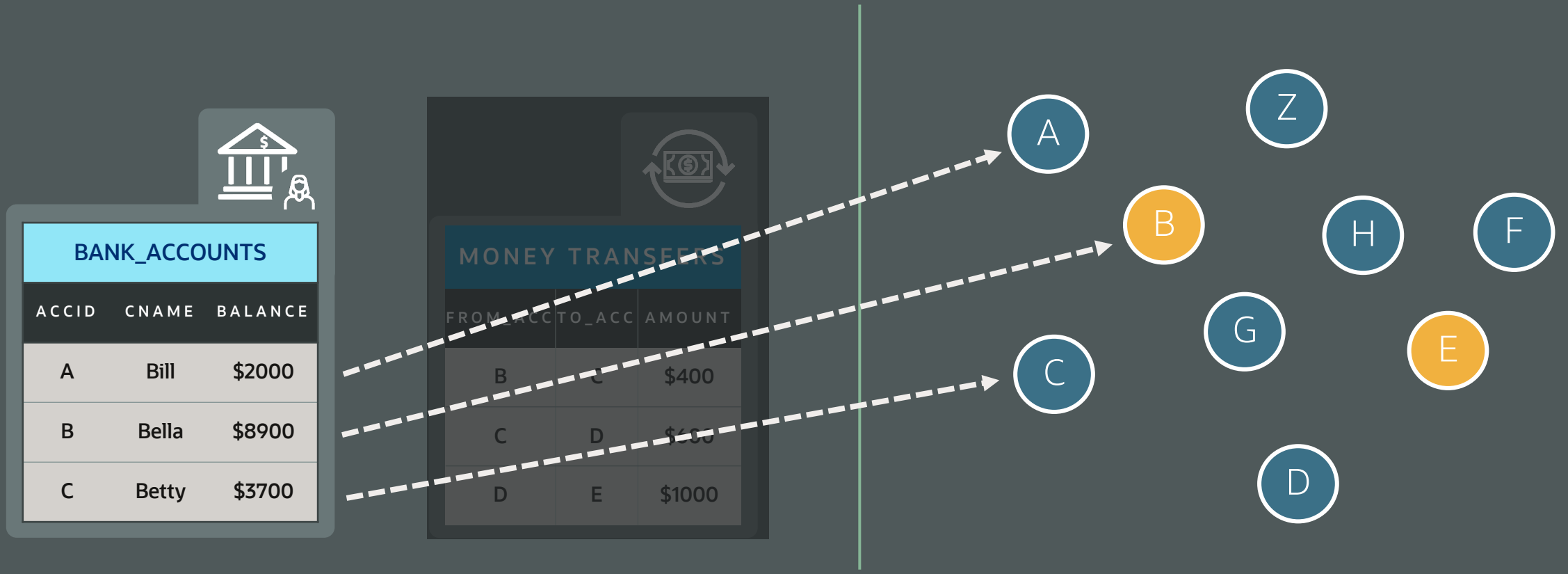
New Property Graph Views in Database 23ai enable developers to treat data as graph vertices or edges

Graphs are a powerful way to query connections and relationships between data

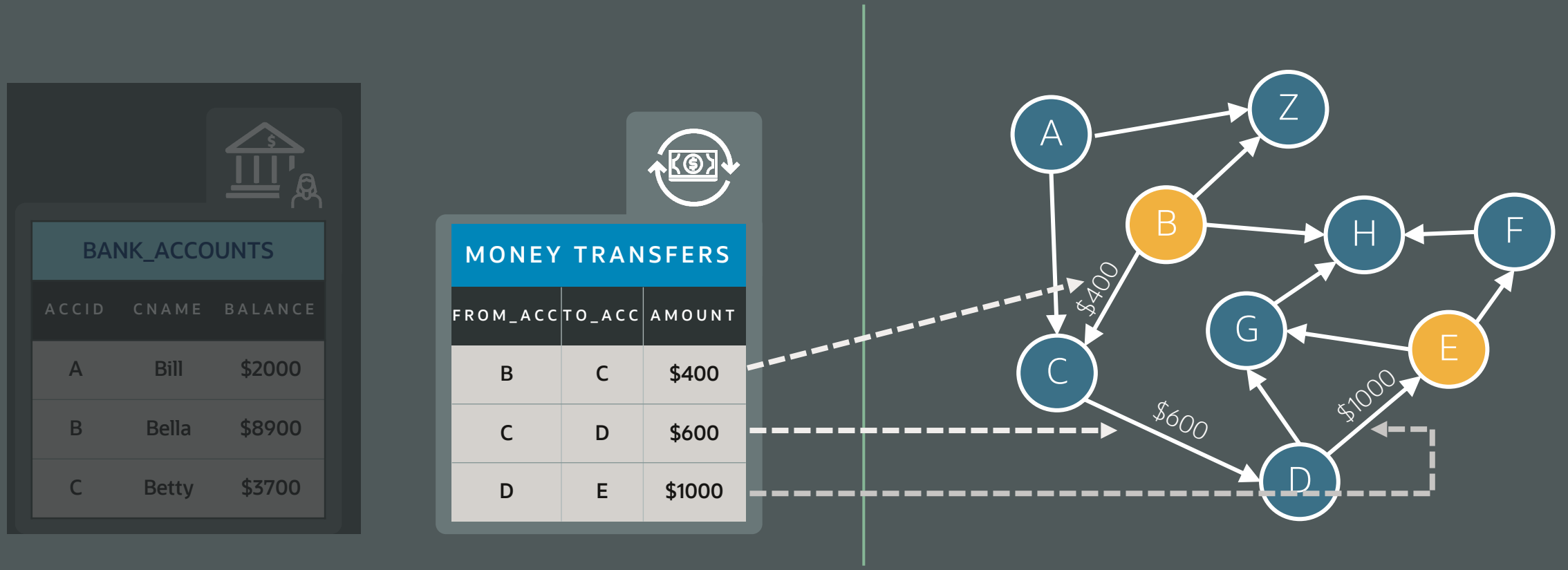
For example, to discover indirect money movements from bank account 'B' to bank account 'E'



Graph views enable treating
bank account rows as graph vertices



And enable treating money transfers between accounts as graph edges



BANK_ACCOUNTS

ACCID	CNAME	BALANCE
A	Bill	$2000
B	Bella	$8900
C	Betty	$3700

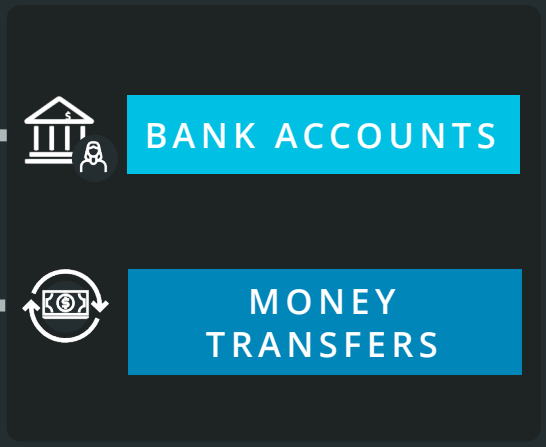
MONEY TRANSFERS

FROM_ACC	TO_ACC	AMOUNT
B	C	$400
C	D	$600
D	E	$1000

Defining a Property Graph view is simple

Just declare the tables whose rows represent vertices or edges in the graph

```
CREATE PROPERTY GRAPH bank_graph
       VERTEX TABLES (
         bank_accounts as accounts
         PROPERTIES (id, balance))
       EDGE TABLES  (
         bank_transfers
         SOURCE KEY      (from_acc) REFERENCES ACCOUNTS(ID)
         DESTINATION KEY (to_acc)   REFERENCES ACCOUNTS(ID)
         PROPERTIES (amount, to_acc))
;
```



Querying the Graph is simple

This query finds money flows from account 'B' to account 'E' via one intermediary bank account

```
SELECT graph.path
FROM   GRAPH_TABLE (
       bank_graph
        MATCH (v1)-[e is BANK_TRANSFERS]->{1,3} (v2)
         WHERE  v1.id = 'B'
         AND    v2.id = 'E'
        COLUMNS LISTAGG(e.to_acc, ',') AS path)
       ) graph
;
```

A pure relational query is much more complex

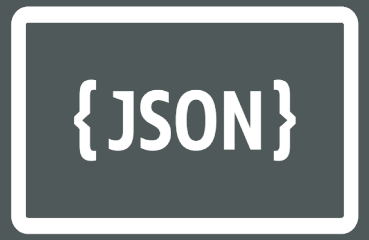
```
-- transfers indirectly from 'B' to 'E'

SELECT v1.id as account_id1 , v2.id as account_id2
FROM   bank_accounts  v1,
       money_transfers  btx,
       bank_accounts  v2
WHERE  (v1.id = btx.from_acc  AND v2.id = btx.to_acc)
AND     v1.id= 'B' AND v2.id= 'E'
UNION ALL
SELECT v1.id as account_id1 , v2.id as account_id2,
FROM   bank_accounts  v1,
       money_transfers btx,
       bank_accounts  bc2,
       money_transfers btx2,
       bank_accounts  v2
WHERE  (v1.id = btx.from_acc  AND  bc2.id = btx.to_acc  AND
        bc2.id = btx2.from_acc  AND v2.id = btx2.to_acc  )
AND     v1.id= 'B' AND v2.id= 'E'
UNION ALL
SELECT v1.id as account_id1 ,v2.id as account_id2
FROM   bank_accounts  v1,
       money_transfers btx,
       bank_accounts  bc2,
       money_transfers btx2,
       bank_accounts  bac4,
       money_transfers btx5,
       bank_accounts  v2
WHERE  (v1.id = btx.from_acc  AND bc2.id = btx.to_acc  AND
        bc2.id = btx2.from_acc  AND bac4.id = btx2.to_acc  AND
        bac4.id = btx5.from_acc  AND  v2.id = btx5.to_acc  )
AND     v1.id= 'B' AND v2.id= 'E'
;
```

Requires 12 joins and 3 unions to handle all combinations of intermediate accounts

“Now any enterprise running Oracle Database can benefit from simple declarative graph navigation on all their existing business data”

RON WESTFALL, RESEARCH DIRECTOR,
DIGITAL TRANSFORMATION, FUTURUM

With Oracle Database 23ai, one part of an app can treat the data as relational, while other parts treat the same data as a document, and others treat it as a graph



You get the best of all these worlds, at the same time
A huge benefit for developers, app dev, and data consistency

WIKIBON

“Developing new apps using a pure JSON or Graph model is now like committing to using a basic flip phone for the next 20 years”

MARC STAIMER
SENIOR ANALYST, WIKIBON

Unification of AI and Databases

AI uses a data representation called Vectors

Vectors represent the semantic content of images, documents, videos, etc.

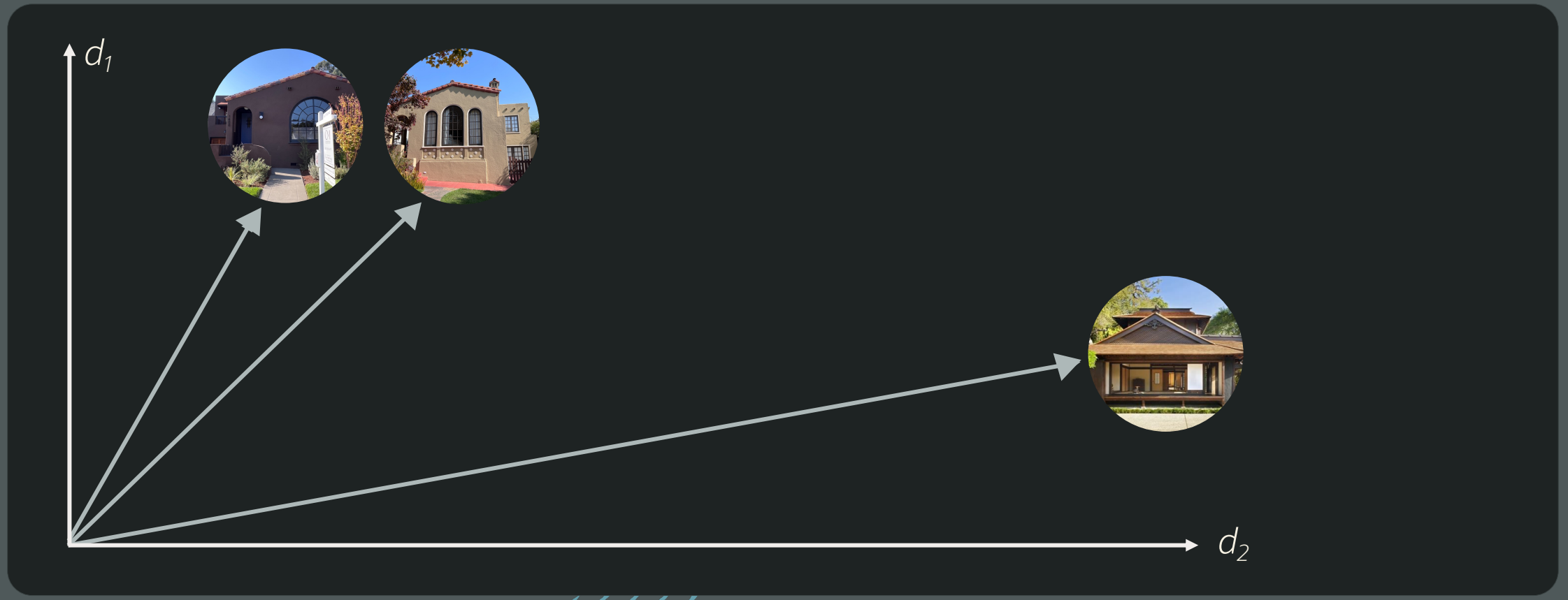
A vector is a sequence of numbers, called dimensions, used to capture the important “features” of the data

Example: the features for a house image could be

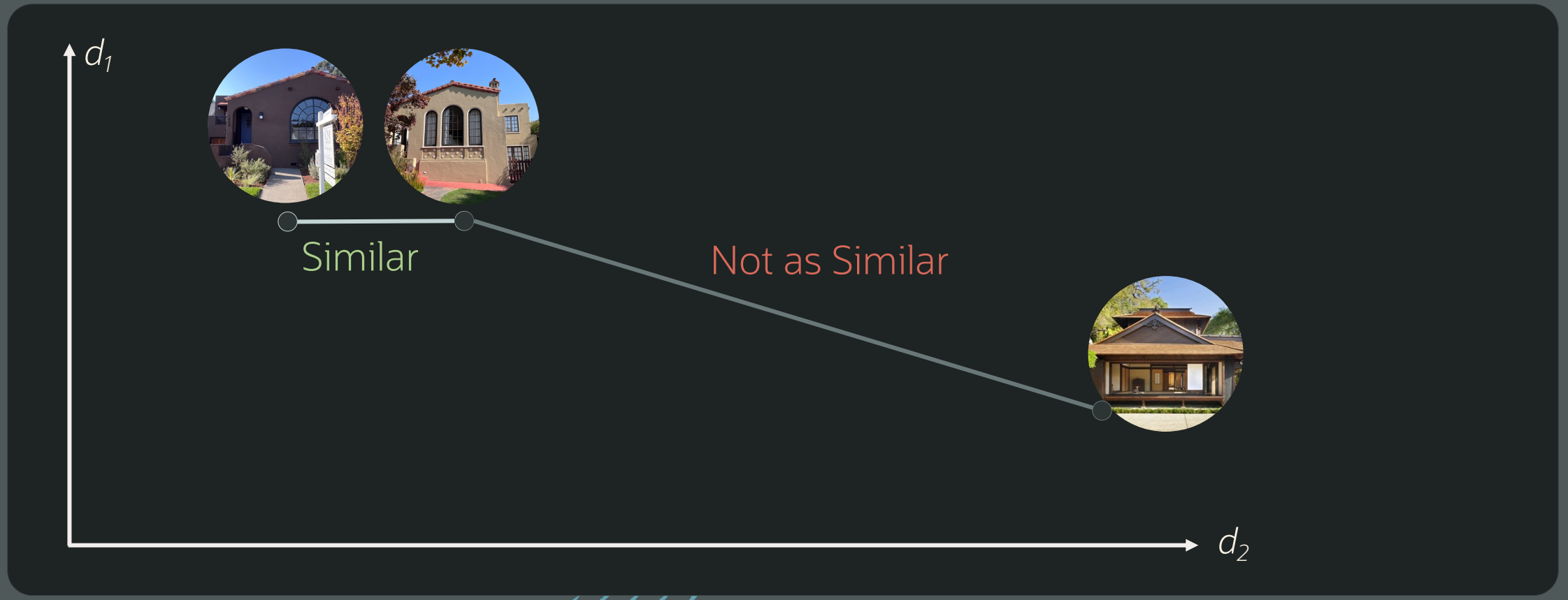
Each dimension (number), represents a different feature of the house

Note: Features are determined by ML algorithms so are not as simple as shown here

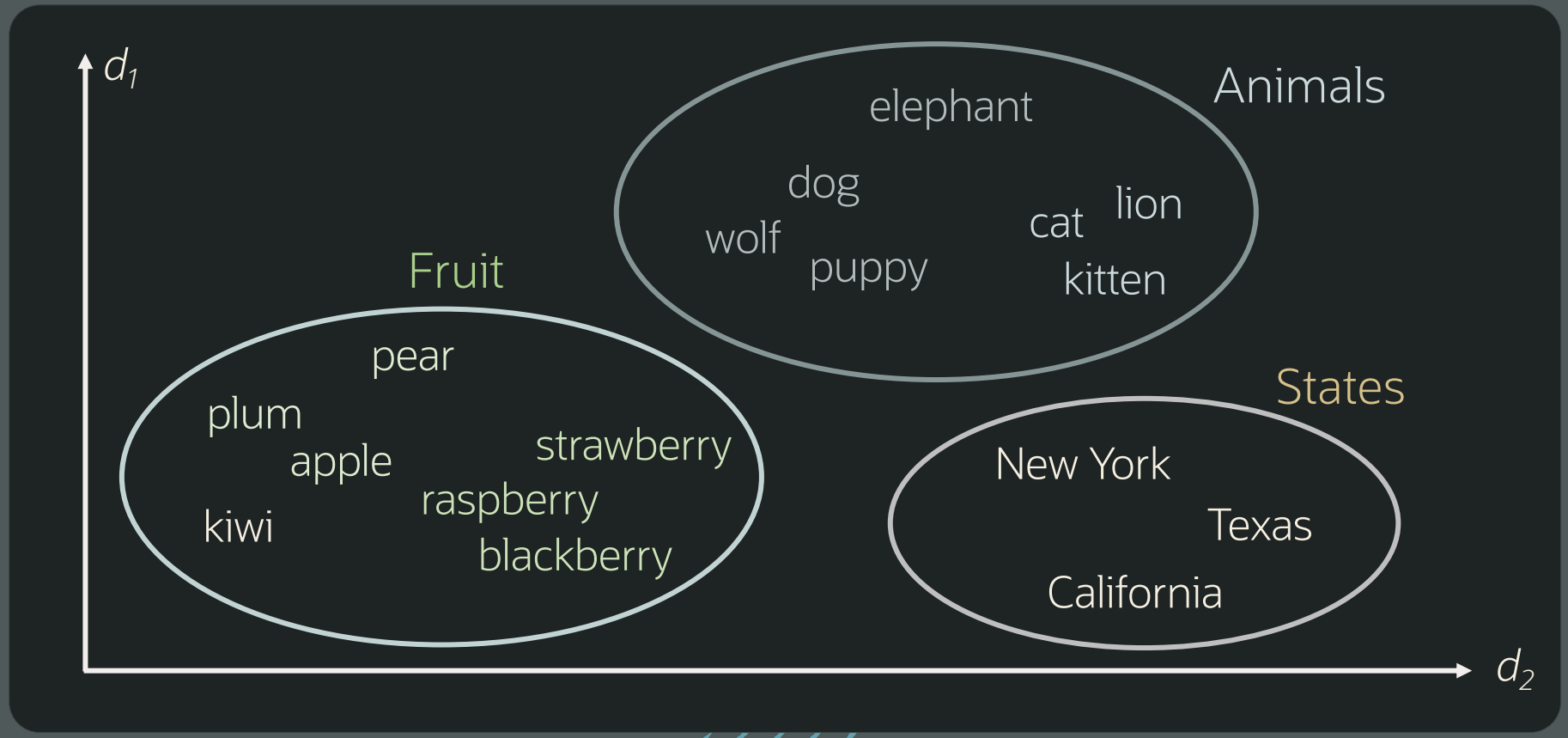
House vectors when collapsed into 2 dimensions instead of hundreds could look like this



The distance between the vectors is proportional to their semantic similarity



Word similarity works the same way
Word vectors that are close are more semantically similar



Documents also work the same way

Documents vectors that represent similar content are closer in distance than those representing dissimilar content

But answering end-user questions requires business data

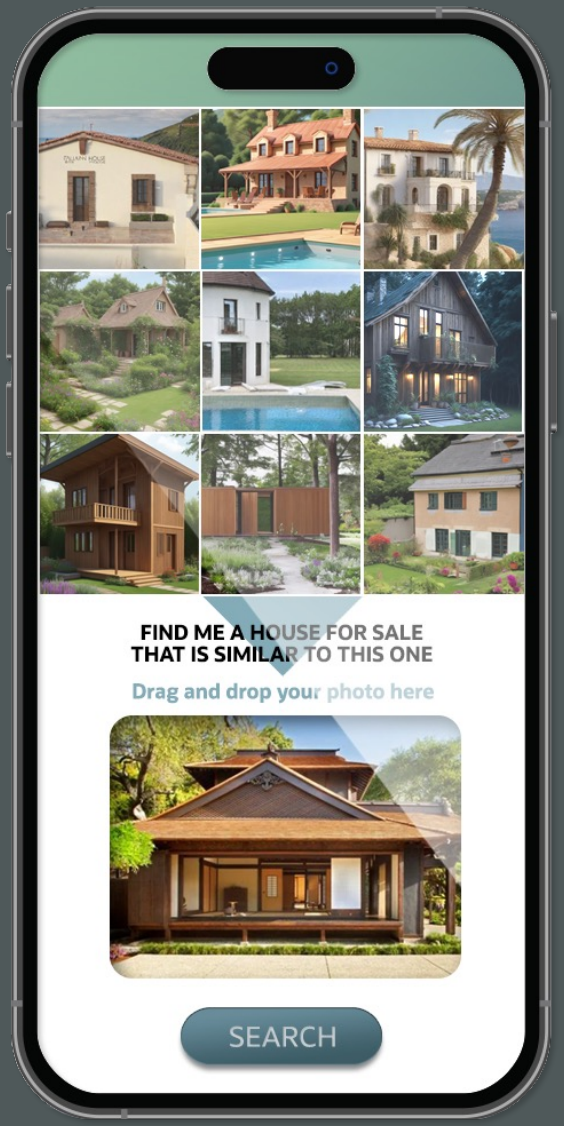
Oracle Database is the **leading** repository of business data

End-user data

Buying history, interests, balance, location, etc.

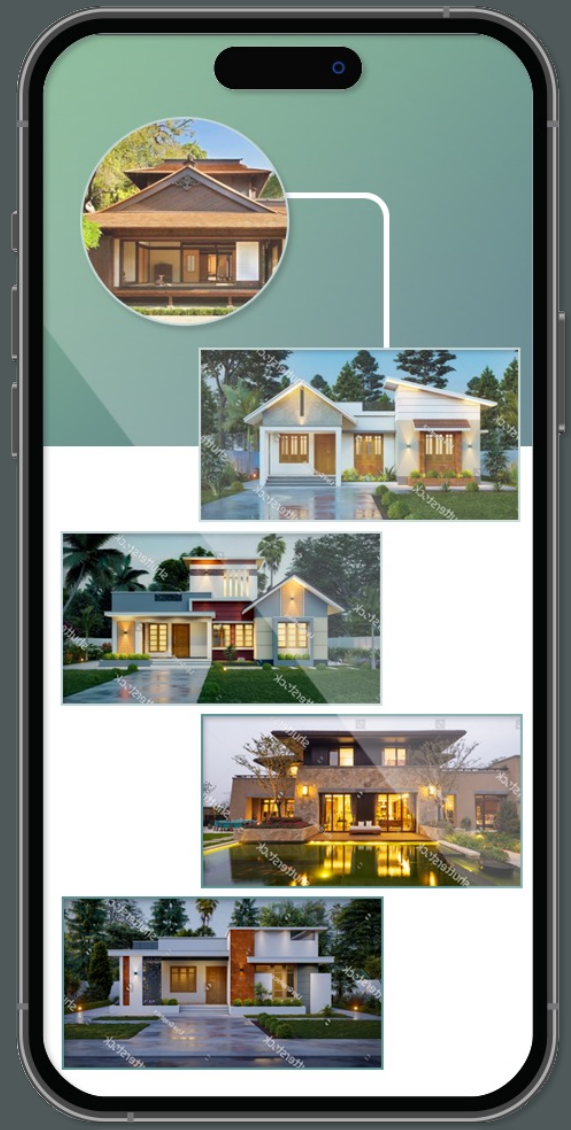
Product Data

Product attributes, inventory, limitations, configurations, etc.



Let's look at an example

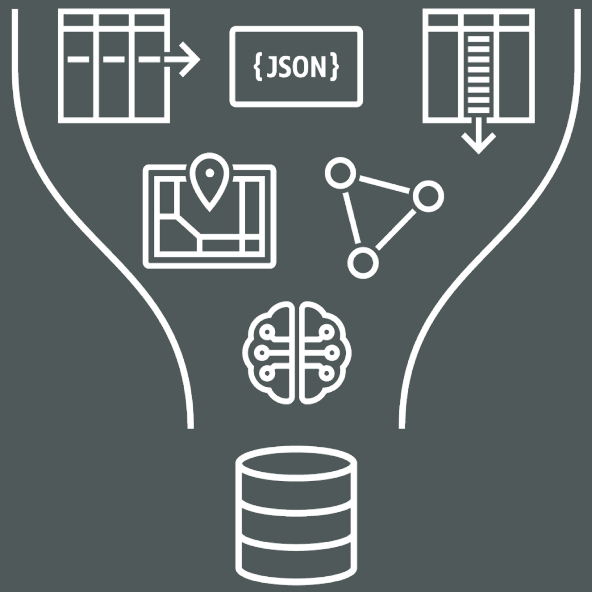
Imagine a house-hunting app that helps customers find houses for sale that are similar to a picture the customer uploads



Finding a good match requires combining semantic picture search with searches on business data including:

- Customer data such as location preference and budget
- Product data such as houses available for sale by location and their price

Searches on a combination of business and semantic data are more effective if both types of data are stored together



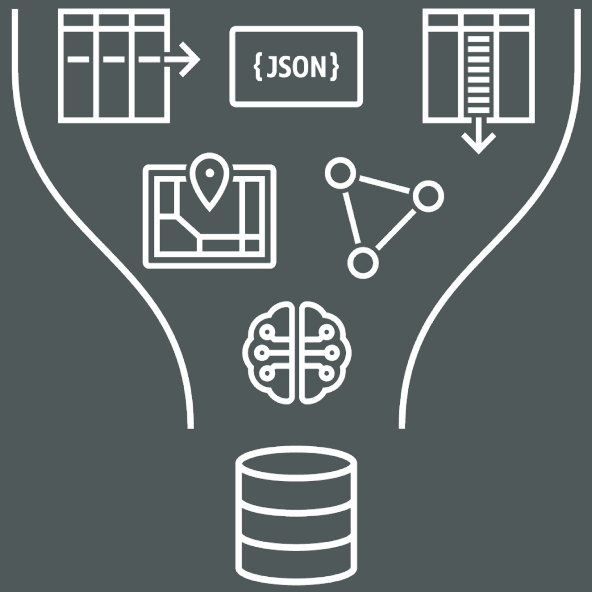
Business Database

One solution is to continuously send your business data to a vector database

User and Product Data

Vector Database

The business data that is relevant to a question varies widely

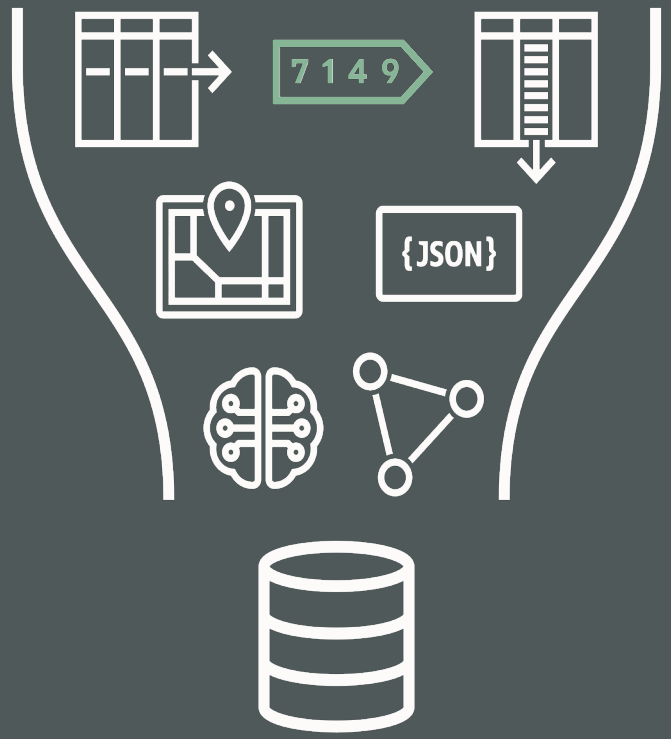


Business Database

You need to send lots of business data since you can't predict the question

Vector Database

Dedicated vector databases are not good at searching or securing business data

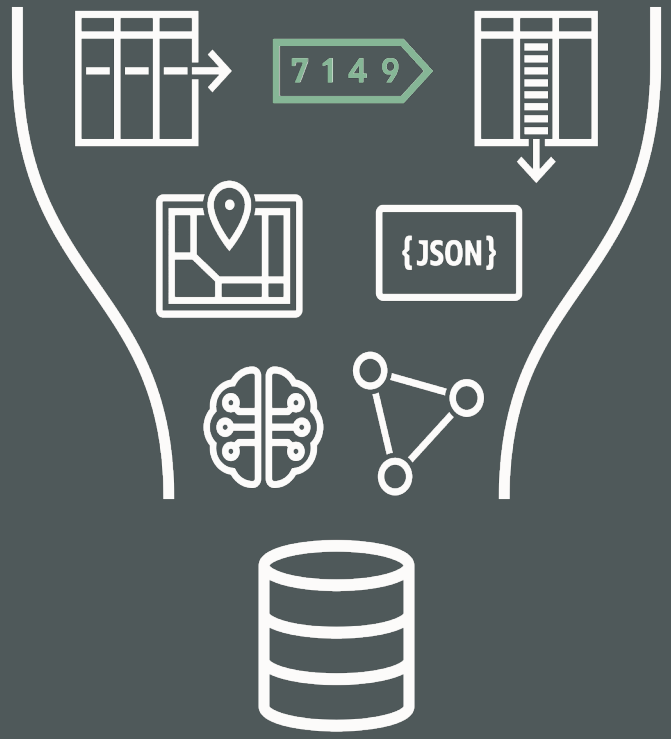


Converged Database

The best solution is to add vector search to your business database

Allows you to use both business data and vectors when answering a question

- No need to move and synchronize data, manage multiple products, etc.



Introducing:

AI Vector Search in Oracle Database 23ai

Unification of AI and Databases Delivers Developer Nirvana:

> Vector search to find similar unstructured data combined with the power of relational search on business data

Oracle Database 23ai can store vectors using a new vector data type

```
CREATE TABLE house_for_sale (house_id     number,
                             price        number,
                             city         varchar2(400),
                             house_photo  blob,
                             house_vector vector
);
```

Allows finding data that is semantically similar to an input

No ML expertise required

DBAs and Developers can learn to use AI vector search in minutes

Find houses that are similar to this picture

```
SELECT   …
FROM     house_for_sale
ORDER BY vector_distance(house_vector, :input_vector)
FETCH FIRST 10 APPROXIMATE ROWS;
```

Allows queries that combine AI vector search with business data about customers and products

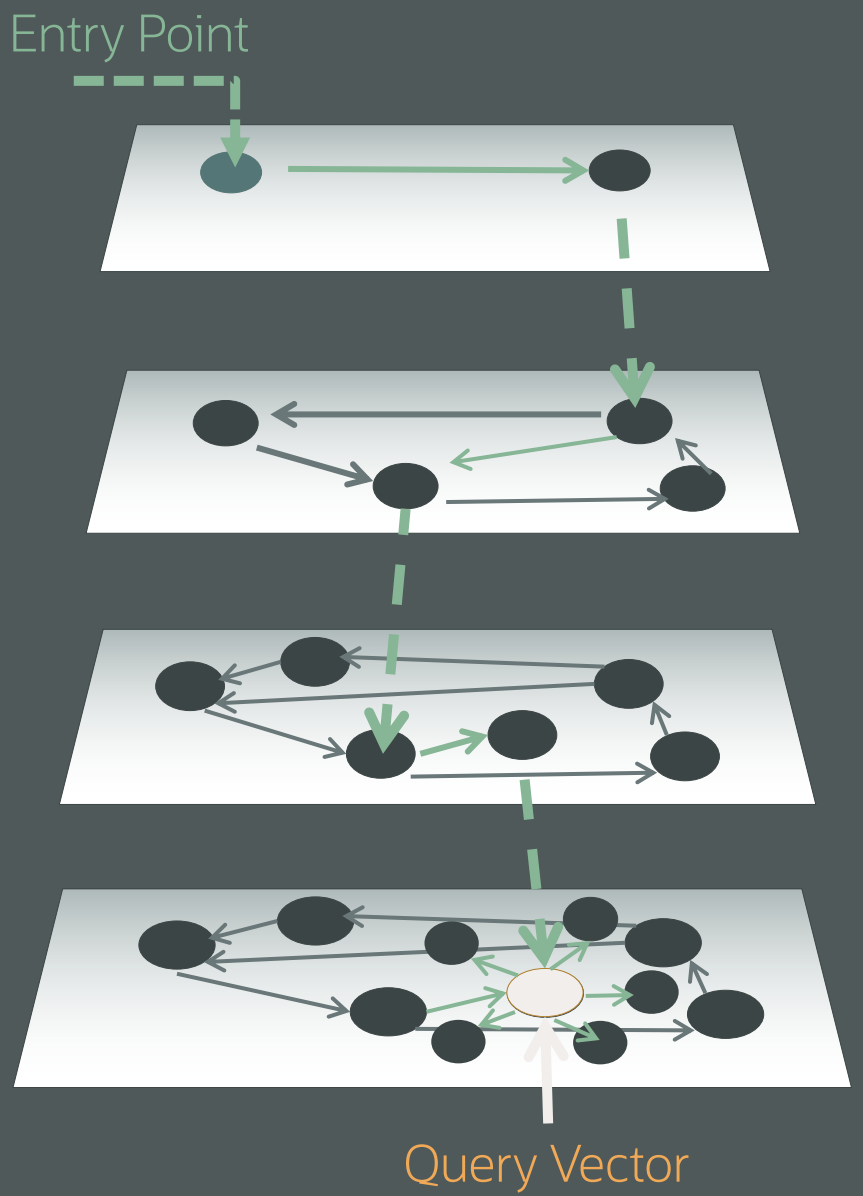
Combines customer data, product data, and AI search in 5 lines of SQL!

A single integrated solution

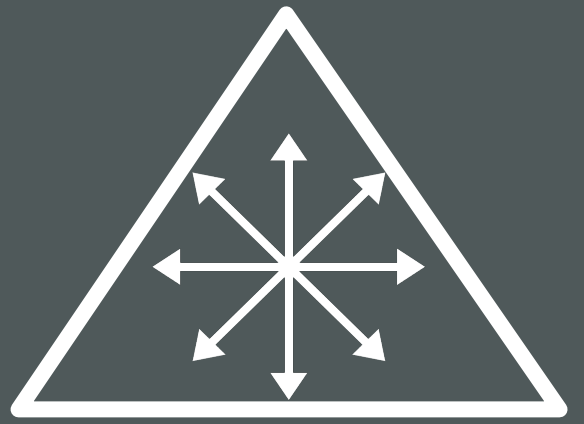
All data is fully consistent

Find houses that are similar to this picture and match the customer's preferred city and budget

```
SELECT …
FROM   house_for_sale
WHERE  price <= (SELECT budget      FROM customer …)
AND    city  in (SELECT search_city FROM customer …)
ORDER BY vector_distance(house_vector, :input_vector)
FETCH FIRST 10 APPROXIMATE ROWS;
```

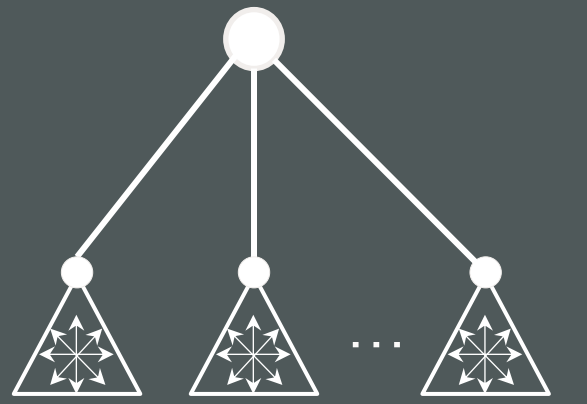


Oracle database accelerates AI vector search using sophisticated vector indexes



Oracle can partition vector indexes for improved performance

Vector index of house images



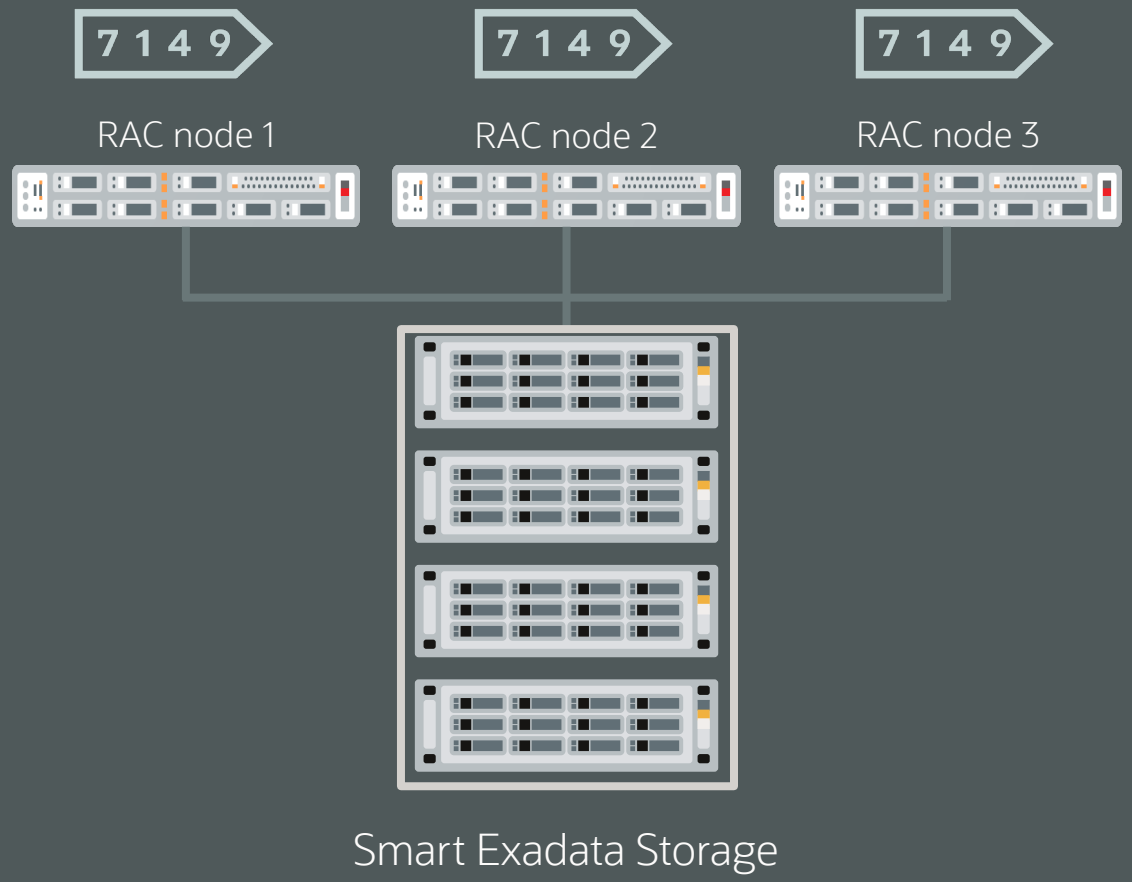
Partitioned by city

House image vectors can be partitioned by city

Creates a vector index for each city

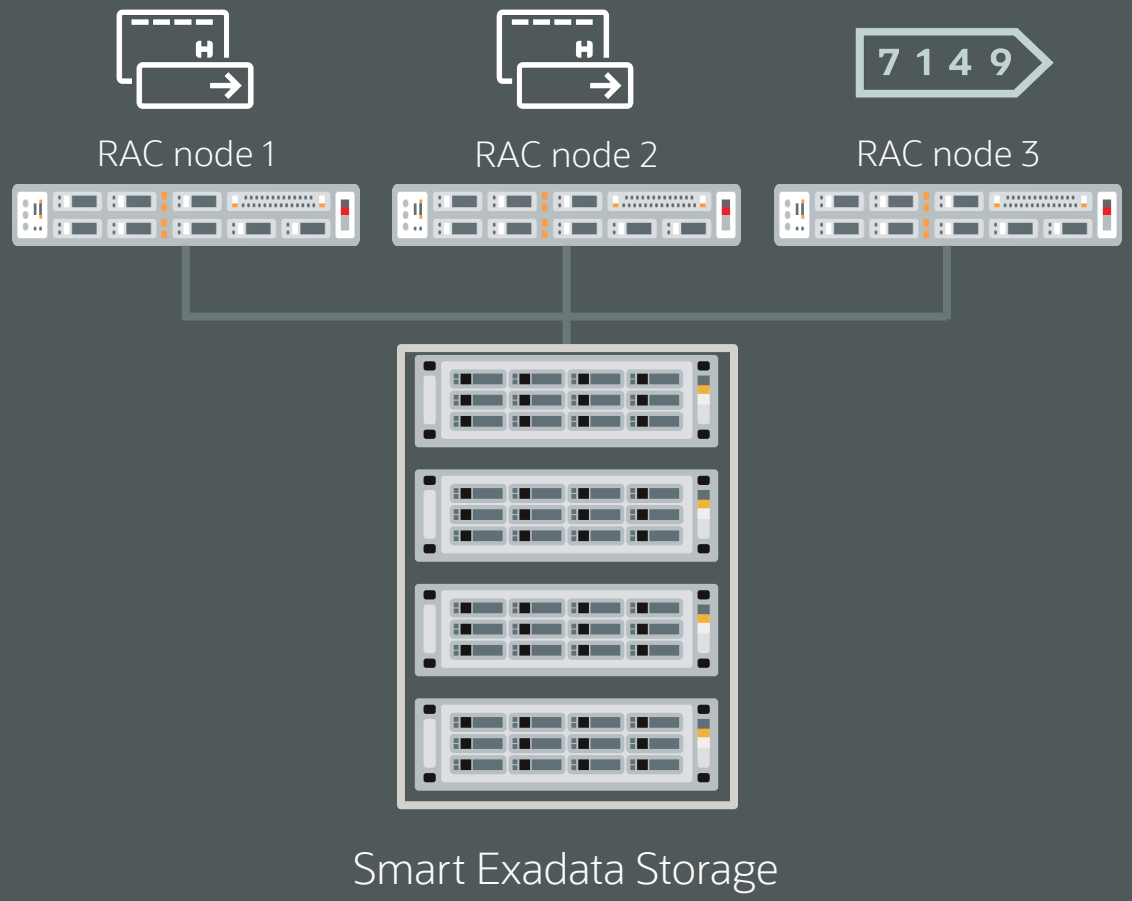
No need to search images of houses in an unwanted city

1000x faster



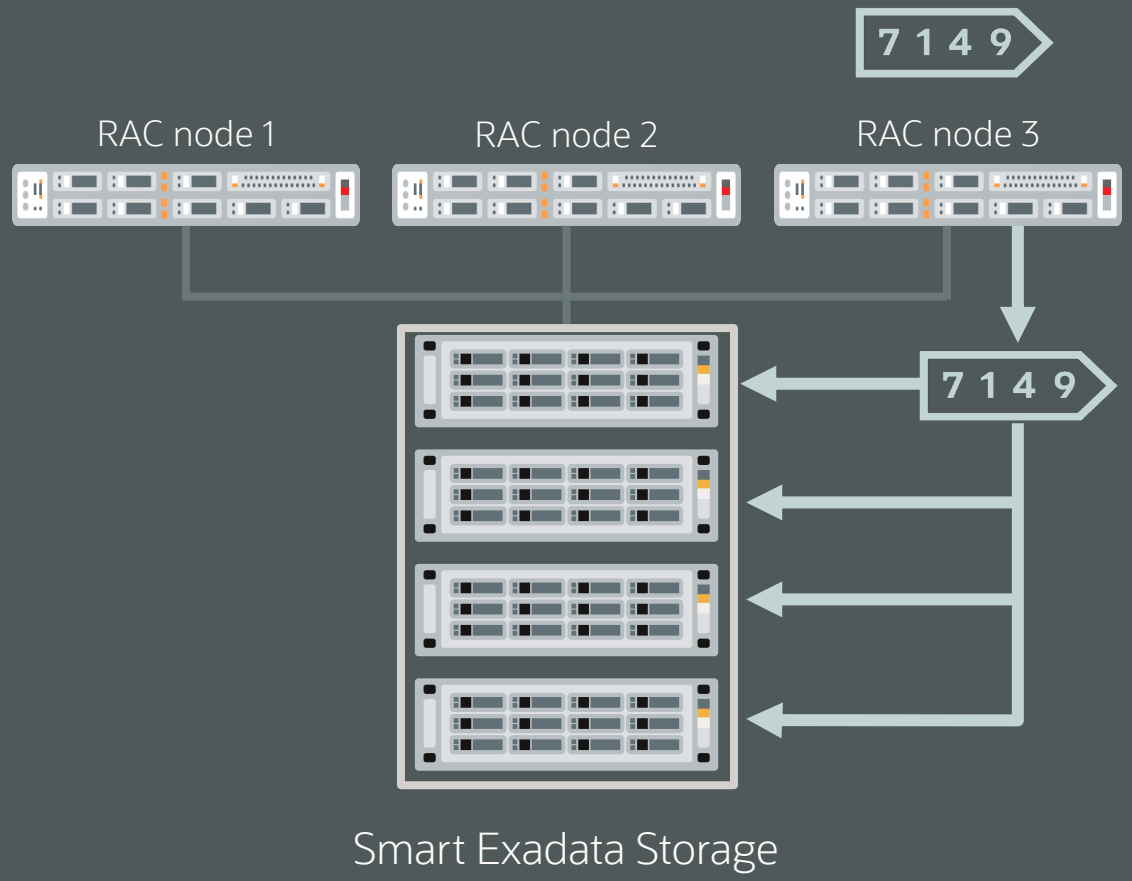
Oracle transparently scales vector processing across the computers in a RAC cluster

With full data consistency

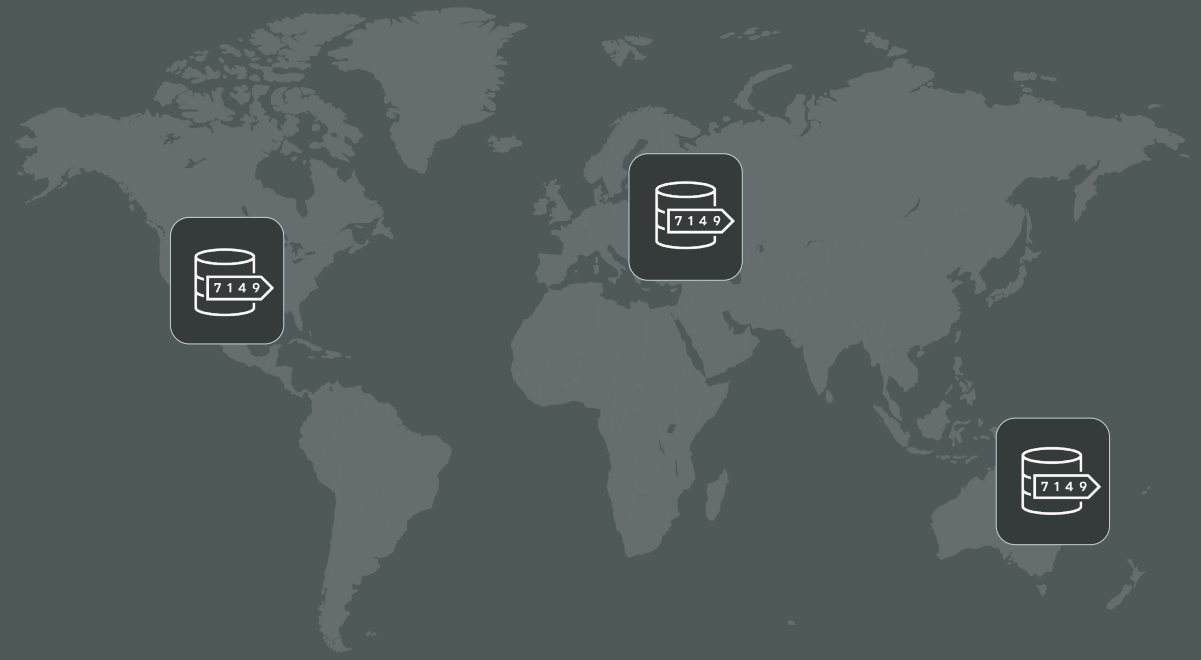


Oracle vector processing can be isolated to a subset of RAC computers to avoid disturbing business OLTP

With full data consistency



Oracle vector search can be transparently offloaded to smart Exadata storage for faster search



Oracle vector processing can be sharded across geographically distributed databases for unlimited scale or data sovereignty

Parallel SQL

Transactions

Analytics

Disaster Recovery

Security

AI Vector Search also benefits from many other core database capabilities

Adding semantic search to relational search is great, but we can do even better by adding Generative AI

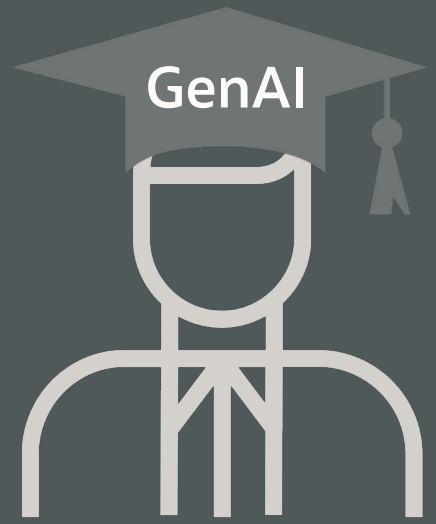
Vector Search + Generative AI enables end-users to simply ask natural language questions

AI Vector Search can map the natural language question to relevant data in the database

The user question plus relevant data can then be passed to a Generative AI to provide an informed answer to the question

Let's look at how this works

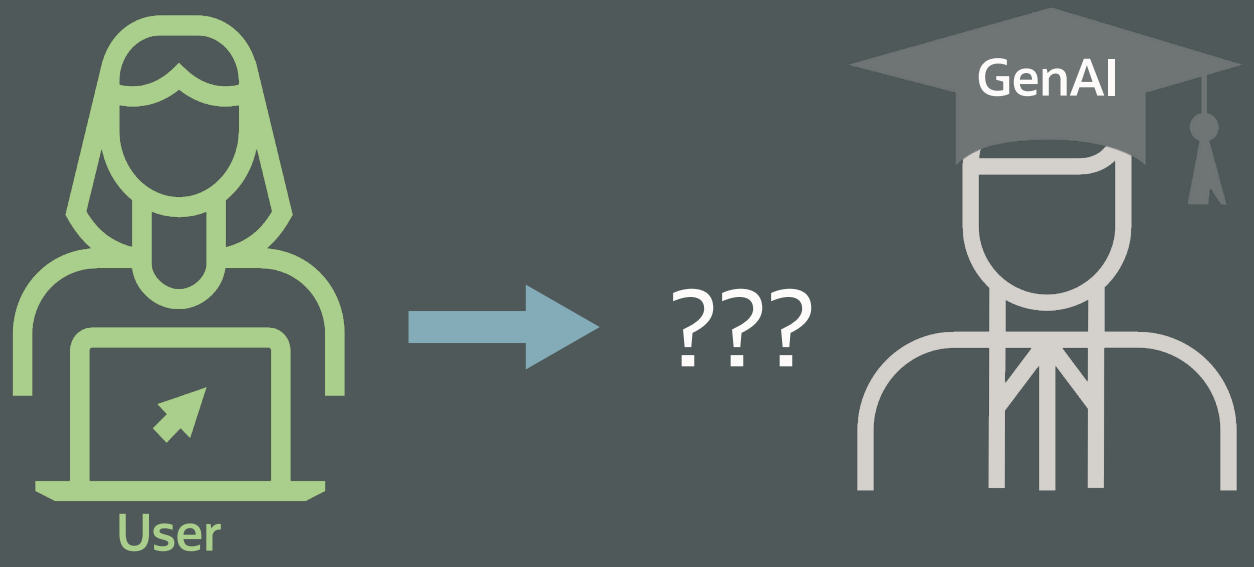
Generative AI is like a smart college graduate



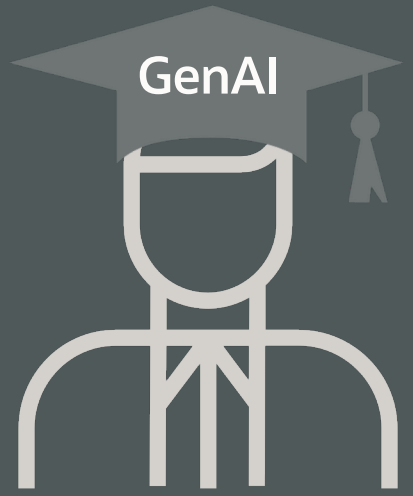
Imagine you hire smart college grads to answer your company's support calls

The grads have lots of general knowledge, but know nothing about your products or past product issues

Left on their own, the grads cannot give good answers to product support questions



If the grads could be instantly augmented with product and product support information, they could provide better answers



That is where vector databases come in

Vector Databases augment Generative AI
by retrieving detailed, often private content
needed to answer questions

Called: **Retrieval Augmented Generation (RAG)**

Retrieval Augmented Generation works like this

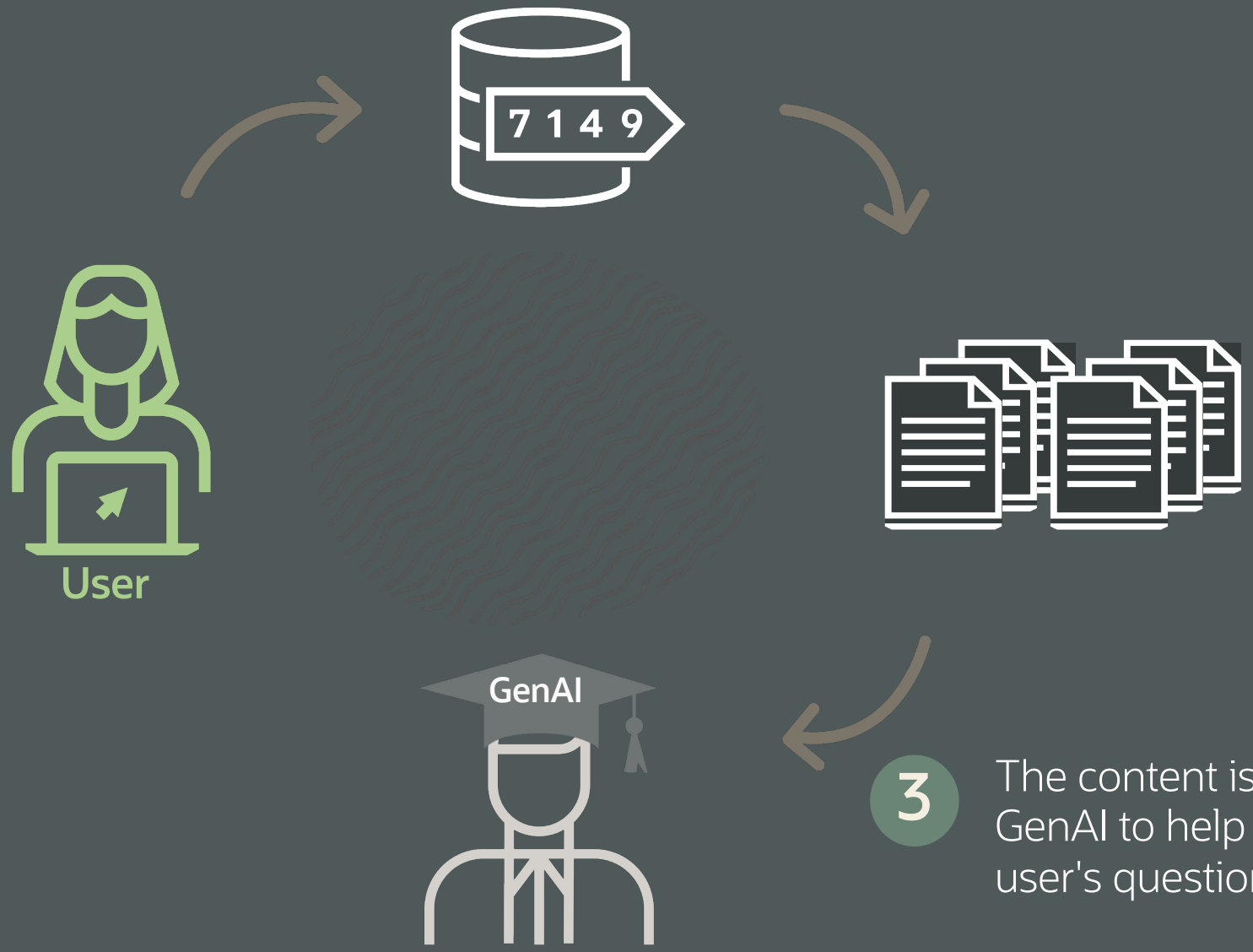
The user's question is encoded as a vector and sent to a Vector DB

1

User

Retrieval Augmented Generation works like this

Retrieval Augmented Generation works like this



3 The content is sent to the GenAI to help answer the user's question

Retrieval Augmented Generation works like this

4

GenAI uses the content plus general knowledge to provide an informed answer

Using a similar process, Generative AI can be used by developers to generate SQL queries, JSON duality, and graph views

Documents

Rows

Graphs

This revolutionizes application development. Developers will be more productive than ever before thanks to the innovations in Oracle Database 23ai

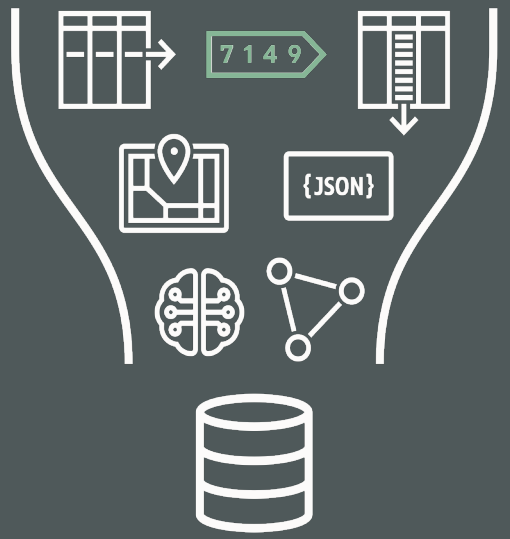
Documents

Rows

Graphs

Key Takeaways

Key takeaways



Oracle is introducing revolutionary new Converged Database technologies that unify relational, JSON, Graph, and AI Vector data models at a fundamental level

Eliminates the simplicity vs power tradeoffs

Enables unprecedented productivity for app dev

Data, AI, and App Dev are rapidly transforming

We encourage you to embrace the change to reap the rewards!

Learn more