DISCLAIMER

During the course of this presentation, TIBCO or its representatives may make forward-looking statements regarding future events, TIBCO’s future results or our future financial performance. Although we believe that the expectations reflected in the forward-looking statements contained in this presentation are reasonable, these expectations or any of the forward-looking statements could prove to be incorrect and actual results or financial performance could differ materially from those stated herein.

TIBCO could experience factors that could cause actual results or financial performance to differ materially from those contained in any forward-looking statement made in connection with this presentation. TIBCO does not undertake to update any forward-looking statements that may be made from time to time or on its behalf.

This document (including, without limitation, any product roadmap or statement of direction data) illustrates the planned testing, release and availability dates for TIBCO products and services. This document is provided for informational purposes only and its contents are subject to change without notice. TIBCO makes no warranties, express or implied, in or relating to this document or any information in it, including, without limitation, that the information is error-free or meets any conditions of merchantability or fitness for a particular purpose. This document may not be reproduced or transmitted in any form or by any means without our prior written permission.

The material provided is for informational purposes only, and should not be relied on in making a purchasing decision. The information is not a commitment, promise or legal obligation to deliver any material, code, or functionality. The development, release, and timing of any features or functionality described for our products remains at our sole discretion.
We live in a world that is becoming increasingly connected and blended.
Why a Graph Database?

• We live in a **connected** world!
• The Big Data movement has resulted in **more data** being collected at **higher rates**.
• Data structures are **no longer consistent**.
• Information is **related, dynamic, and constantly evolving**.
Relational Model: An Example

CREATE TABLE [Purchasing].[ProductVendor]
  ([ProductID] [int] NOT NULL,
   [BusinessEntityID] [int] NOT NULL,
   [AverageLeadTime] [int] NOT NULL,
   [StandardPrice] [money] NOT NULL,
   ...,
   [UnitMeasureCode] [nchar](3) NOT NULL,
   [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE()))
  ON [PRIMARY];

CREATE TABLE [Production].[Product]
  ([ProductID] [int] IDENTITY (1, 1) NOT NULL,
   [Name] [nvarchar] NOT NULL,
   [ProductNumber] [nvarchar](25) NOT NULL,
   ...,
   [DiscontinuedDate] [datetime] NULL,
   [rowguid] uniqueidentifier ROWGUIDCOL NOT NULL DEFAULT (NEWID()),
   [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE()))
  ON [PRIMARY];

CREATE TABLE [Purchasing].[Vendor]
  ([BusinessEntityID] [int] NOT NULL,
   [AccountNumber] [nvarchar] NOT NULL,
   [Name] [nvarchar] NOT NULL,
   [CreditRating] [tinyint] NOT NULL,
   [PreferredVendorStatus] [Flag] NOT NULL DEFAULT (1),
   [ActiveFlag] [Flag] NOT NULL DEFAULT (1),
   [PurchasingWebServiceURL] [nvarchar](1024) NULL,
   [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE()))
  ON [PRIMARY];

“Find the names of all products, for a particular subcategory, and the names of their vendors.”

Source: AdventureWorks2008R2 Microsoft sample.
Determine related tables from static data model.

Purchasing.ProductVendor : Join Table
Relational Model: An Example

CREATE TABLE [Purchasing].[ProductVendor]
(  
  [ProductID] [int] NOT NULL,
  [BusinessEntityID] [int] NOT NULL,
  [AverageLeadTime] [int] NOT NULL,
  [StandardPrice] [money] NOT NULL,
  ...
  [UnitMeasureCode] [nchar](3) NOT NULL,
  [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE())
) ON [PRIMARY];

CREATE TABLE [Production].[Product]
(  
  [ProductID] [int] IDENTITY (1, 1) NOT NULL,
  [Name] [Name] NOT NULL,
  [ProductNumber] [nvarchar](25) NOT NULL,
  ...
  [DiscontinuedDate] [datetime] NULL,
  [rowguid] uniqueidentifier ROWGUIDCOL NOT NULL DEFAULT (NEWID()),
  [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE())
) ON [PRIMARY];

CREATE TABLE [Purchasing].[Vendor]
(  
  [BusinessEntityID] [int] NOT NULL,
  [AccountNumber] [AccountNumber] NOT NULL,
  [Name] [Name] NOT NULL,
  [CreditRating] [tinyint] NOT NULL,
  [PreferredVendorStatus] [Flag] NOT NULL DEFAULT (1),
  [ActiveFlag] [Flag] NOT NULL DEFAULT (1),
  [PurchasingWebServiceURL] [nvarchar](1024) NULL,
  [ModifiedDate] [datetime] NOT NULL DEFAULT (GETDATE())
) ON [PRIMARY];

SELECT p.Name, v.Name
FROM Production.Product p
JOIN Purchasing.ProductVendor pv
ON p.ProductID = pv.ProductID
JOIN Purchasing.Vendor v
ON pv.BusinessEntityID = v.BusinessEntityID
WHERE ProductSubcategoryID = 15
ORDER BY v.Name;
Relational Model: Limitations

Relational models struggle with highly connected, dynamic data.

- Requires the use of pre-defined schemas and join tables.
- Difficult and expensive to model n-degrees of separation.
- SQL joins in queries can become complex.
- Complex to maintain primary key / foreign key relationships with highly connected data, and often difficult to properly index for performance.

**Fixed Schema**

As relationships become more dynamic, and domains more connected, degree of separation becomes too great to handle.

**Limited Performance**

As datasets grow and overall structure becomes more complex/less uniform, system query performance significantly diminishes.

**Finite Value**

Difficult to deliver the performance required to engage with real-time activity. Knowledge about entity relationships is inferred or difficult to model.
Graph Model: Natural Approach w/Greater Efficiency

Filter Graph to Find Results
ProductSubcategory = 15
Returns list of relevant nodes and edges.
Graph databases transform a complex web of dynamic data into meaningful (and understandable) relationships.

- Stream Analytics Contextual Data.
- Efficient Entity Link Analysis Triggered by Event Arrival.
- Leverage Connected Data during Stream Processing.
- Eliminate complex joins and self-joins in a relational model.

**Intelligent Schema**

Assumes objects and nodes are linked by relationships; designed to constantly evolve, without impacting performance of existing queries and app functionality.

**Consistent Performance**

Index-free adjacency negates requirement for index lookups – enabling query performance to remain relatively consistent, even as datasets grow.

**Increased Value**

Enables quick extraction of new insight from large and complex databases. Helps uncover unknown interactions and relationships. Provides valuable insight into semantic context.
TIBCO® Graph Database

• Built due to **increased demand** from our customers for storing, accessing, and querying connected data.
  – Extension of graph problems that TIBCO has been solving for years within various industries.

• Built to **emphasis storage and retrieval efficiency**, for large scale, highly-connected datasets.

• Designed to fit within **TIBCO and non-TIBCO environments**.
  – C-99 based server with open client API.
TIBCO® Graph Database: Enterprise & Open Source

TIBCO® Graph Database

Translytical database that transforms a complex web of dynamic data into meaningful, comprehensible and traversable relationships delivered at the speed of transactions.

• Native NoSQL graph store built from the ground up to handle today’s connected world.
  • C99 based server
  • Dynamic schema management
  • ACID
  • Page cache management
• Community Edition freely downloadable.
• Client API open source and available at https://github.com/TIBCOSoftware/tgdb-client.
  • Customers may implement their own client or graph language.
• Enterprise Edition designed and built to handle large scale connected datasets.
## Moving From Relational To Graph Representation

<table>
<thead>
<tr>
<th>Relational Concepts</th>
<th>Graph Database Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Row</td>
<td>Node</td>
</tr>
<tr>
<td>Table Column</td>
<td>Node Attribute</td>
</tr>
<tr>
<td>Business Primary Key</td>
<td>Index</td>
</tr>
<tr>
<td>Indexes</td>
<td>Index</td>
</tr>
<tr>
<td>Foreign Keys</td>
<td>Replaced with Edges (Relationships)</td>
</tr>
<tr>
<td>Join Tables</td>
<td>Replaced with Edges (Relationships)</td>
</tr>
<tr>
<td>Table Schema</td>
<td>Graph Metadata (Dynamic)</td>
</tr>
</tbody>
</table>

© Copyright 2000-2017 TIBCO Software Inc.
Use Cases: IoT

- IoT devices typically are part of a connected network.
  - Edge device / Gateway device
  - Network / Telecom

- The relationships across this connected network are best represented as a graph.
  - Schema is not locked, facilitating the addition of devices and device attributes over time.
  - Both the nodes and edges have associated data, including direction.
  - Complex web of IoT devices is difficult and expensive to represent in a relational model.

- Facilitates analytics over the entire IoT network.
  - Dependency Traversal & Search
  - Network & Root Cause Analysis / Operational Diagnostics
  - Filtering
Use Cases: Dynamic Resource Management

- **Resource management can involve a complex network:**
  - Resources (users, machines) & Capabilities (skill sets)
  - Tasks (units of work)
  - Organization

- Representing these entities as a graph allows one to **easier answer questions** such as:
  - What resources are free and capable to perform a task(s)?
  - What task can be pulled from a work queue for this resource?
  - Who are the manager(s) in any sublevel of a particular organization?

- **Efficiently allocate limited resources** according to constraints.
Use Cases: Fraud Detection

Layer 1: Endpoint Centric
- Context of users and the endpoints they use.
- (e.g. secure browsing)

Layer 2: Navigation Centric
- Analysis of navigation behavior compared to expected patterns.

Layer 3: User/Acct Centric (Specific Channel)
- Analysis of user/acct behavior/transactions using rules & models.

Layer 4: User/Acct Centric (Cross Channel)
- Similar to Layer 3, but across channels and products.

Layer 5: Entity Link Analysis
- Analysis of relationships among internal or external entities & their attributes.

Examples:
- Fraud ring detection.
- Insurance fraud.
- Online purchasing fraud.
- ...

Key Role of Graph Database
Use Cases: Trade Surveillance & Monitoring

• Various use cases involve understanding associated entities and relationships.

• Examples:
  – Maintain associations between:
    • orders associated with trading sessions
    • associated with FIX engines
    • associated with physical servers / racks / networks / etc.
  – When an anomaly is detected in a component, traverse the graph and remediate the affected component(s).
  – Fixed income price / calculation relationships.
  – Market data / event time series storage.
Use Cases: Master / Metadata Management

- Complex master data or metadata typically contains a series of relationships:
  - Customer is part of a social network.
  - Product is part of a product family.
  - Stores are part of a distribution network.
  - Relationship between components of a power grid (meters, transformers, substations, etc.)
- Maintaining part or all of this data as a graph allows for efficient retrieval and management.
- Graph database may be used to augment an existing relational model.
Use Cases: General

• Impact Analysis
• Product Provenance
• Path Determination
• Root Cause Analysis
• Asset Management

And Various Others...

• Trade Surveillance
• Conversational UIs
• Influencer / Social Analysis
• Path Travelled Optimization
• Recommendation Engines

© Copyright 2000-2017 TIBCO Software Inc.
Technical Overview
Nodes / Edges / Attributes

Attributes
+ Primary Key
+ Indices (NodeType)

memberName (PK)
crownName	houseHead
yearBorn
yearDied
reignStart
reignEnd
crownTitle

Edge (Bi-Directional, Directed, or Undirected)

Attributes (EdgeType)
reltype

Node

Node

Node
Indices

• One or more indices can be defined against attributes in a nodetype.
  – Unique or non-unique.
  – Single or composite attributes ("columns").

• Nodetype primary key is a created as a unique, non-clustered index.

• Defined indices are currently represented as on-disk, non-clustered, B-Tree indices.
  – Stored in segments, with a defined page size.

• May be preloaded into cache, according to defined cache sizes.
Queries / Filters

• Index-aware, node based filters.
• Filters (“where clause”) are based on SQL-92 constructs (join and aggregate free).
  – Examples:
    • numPackages > 50
    • @nodetype in ('Company', 'Employee') and node.packageType in ('FedEx', 'UPS', 'USPS')
    • product like ('BMW%') or productYear > 2010 and productYear < 2015
    • address is not null
• Iterable resultset, with ability to jump to a specific entity.
• Persistence via a series of disk-based “segments”, divided into “pages”.
• Comprehensive/configurable memory management, including server memory, cache, and shared memory.
• Platform-specific IO optimization, with support for various file systems.
• Configurable transaction, query, and redo log settings.
Persistence & Storage

• Graph Database consists of “segments” on disk.
  – Max. 256 segments

• A “segment” is a collection of “pages”.
  – Configurable segment count and page size.
  – Configurable page size for text and BLOB.

• Segments hold data (each up to 1 TB in size).
• Node has a root page, and can span multiple pages.
• Page manager manages any cached pages (e.g. index pages) by loading & evicting pages according to MRU/LRU strategies.
Memory Management

- Server divided into Shared Memory and Server Memory allocations.
- Cache is set as a percentage of server memory.
  - Controllable with min/max threshold settings.
- Configurable redo queue length in the WAL.
- Monitor memory and cache usage with the admin console.
Transactions

• Supports ACID properties.
  – Currently read-committed isolation level.
  – Implicit “begin transaction” at the point of commit.
  – Queries/gets not part of the transaction.
• Node level locks with optimistic locking.
  – Plus any edges owned by the node.
• Write-ahead log maintained in shared memory, backed by a file.
• Supports sticky session-based transaction processors.
• Asynchronous & coalesced disk writers.
Connection Management

- Multi-port / multi-host net listener.
  - Includes IPV6 support.
- Configurable connection limit per net listener.
- Monitor via the command line administration tool.
  - View number of connections on each listener.
- Includes various connection management techniques for ensuring database stability.
  - Idle connection management
  - Rogue TCP connection prevention (e.g. Telnet), with fixed time for handshake.
Client API

- Open source open client APIs.
  - Java (bundled with distribution)
  - Node.js (available on GitHub)
- Allows one to create their own client API, using a domain-specific language.
Administration

• Bundled command-line based administrative tool.
  – Supports both scripted and interactive modes of operation.

• Various metadata operations:
  – Create/query attribute descriptors
  – Create/query indices, nodetypes, primary keys

• Monitor server health and view information on memory, transactions, cache, and connections.

• Kill connections, create users, and dynamically set log levels.
More Information

**TIBCO® Blog and Community:**
http://www.tibco.com/blog/
https://community.tibco.com/products/tibco-graph-database

**Getting Started:**
https://community.tibco.com/wiki/tibcor-graph-database-getting-started

**Client API:**
https://github.com/TIBCOSoftware/tgdb-client
Summary

• The **TIBCO® Graph Database** is a **core component** of the TIBCO product suite, and a “must-have” for solving today’s business issues.

• By moving towards graph storage and a “polyglot persistence” architectural approach, business can **optimize their data driven solutions**, and **realize the benefits of an intelligent data network**.

• TIBCO and the Graph Database **allows organizations to represent their connected data in an efficient manner** that enables the next generation of digital solutions.