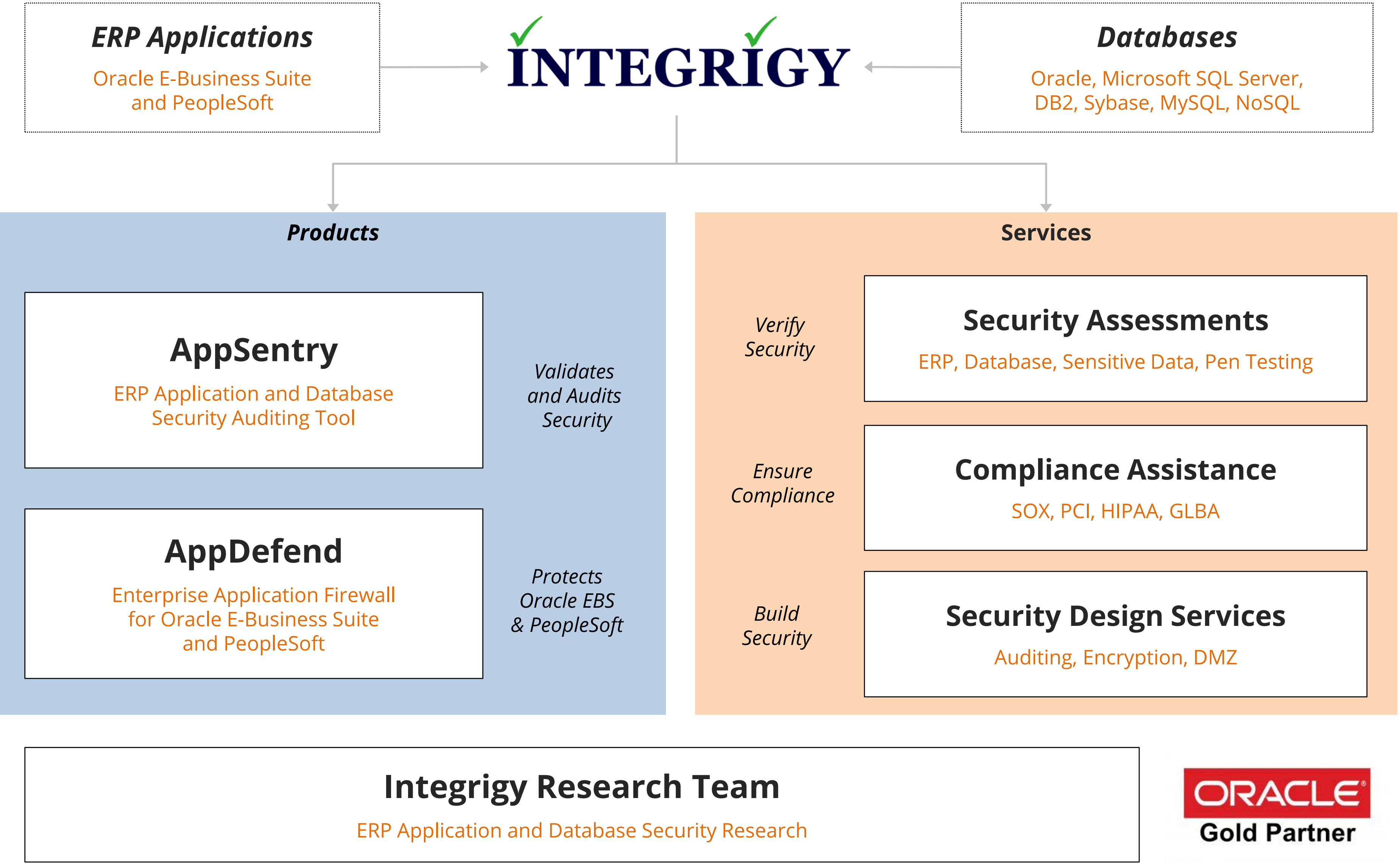
Effectively Using Oracle Blockchain Tables

April 22, 2021

Stephen Kost
Chief Technology Officer
Integrigy Corporation

Phil Reimann
Director of Business Development
Integrigy Corporation

About Integrigy



ERP Applications

Oracle E-Business Suite and PeopleSoft

INTEGRIGY

Databases

Oracle, Microsoft SQL Server, DB2, Sybase, MySQL, NoSQL

Products

AppSentry

ERP Application and Database Security Auditing Tool

Validates and Audits Security

AppDefend

Enterprise Application Firewall for Oracle E-Business Suite and PeopleSoft

Protects Oracle EBS & PeopleSoft

Services

Verify Security

Security Assessments

ERP, Database, Sensitive Data, Pen Testing

Ensure Compliance

Compliance Assistance

SOX, PCI, HIPAA, GLBA

Build Security

Security Design Services

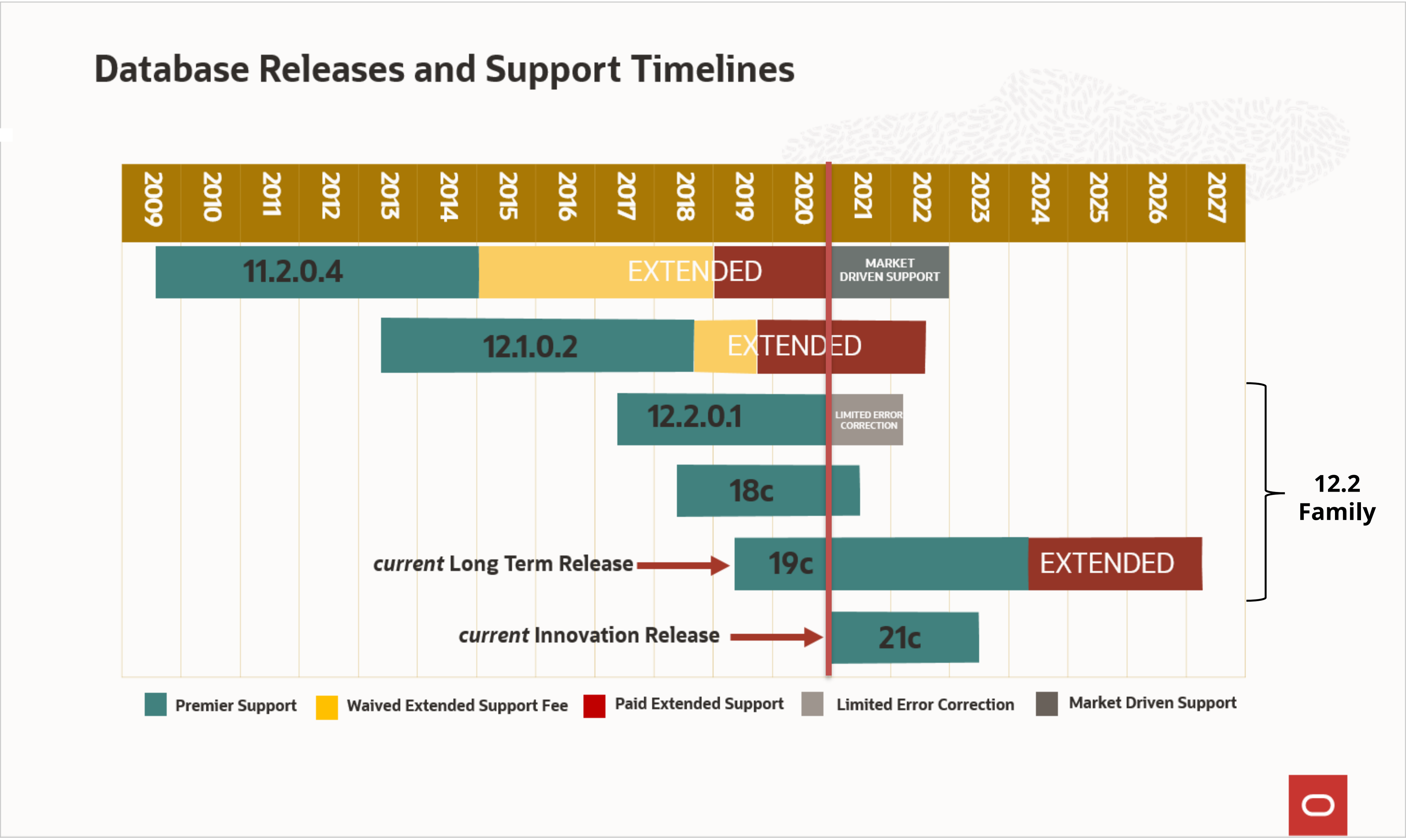
Auditing, Encryption, DMZ

Integrigy Research Team

ERP Application and Database Security Research

ORACLE Gold Partner

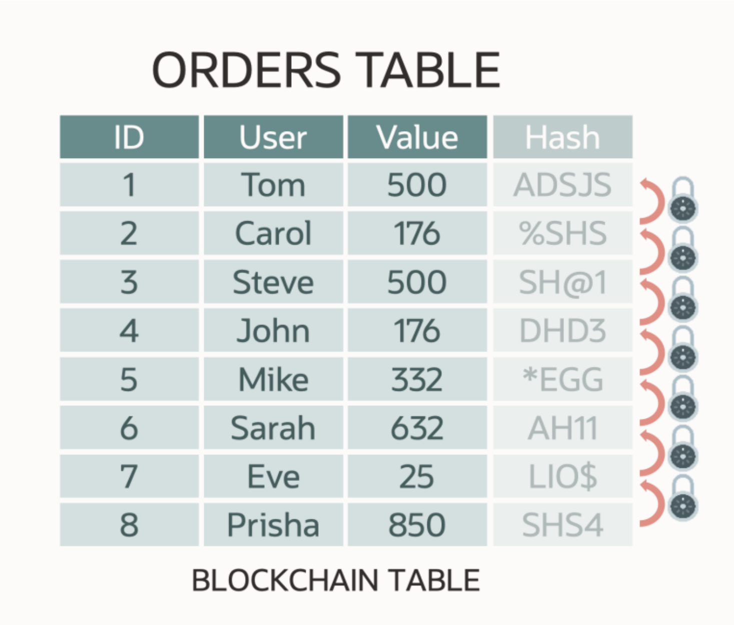
Oracle Database Releases



Source – Oracle Database (RDBMS) Releases Support Status Summary (Doc ID 161818.1) – January 15, 2021

Oracle Blockchain Tables

- Insert-only, tamper-resistant table
- Rows are chained using a cryptographic hashing approach
- Data cannot be modified by DBAs or other users
- Optional row signing by user for additional fraud protection
- Generally, operates as a standard database table
- Common use cases are for audit trails, compliance data, ledgers, and chain of custody or provenance information
- **Available in 19.10 (January 2021) and 21c**



ORDERS TABLE

ID	User	Value	Hash
1	Tom	500	ADSJS
2	Carol	176	%SHS
3	Steve	500	SH@1
4	John	176	DHD3
5	Mike	332	*EGG
6	Sarah	632	AH11
7	Eve	25	LIO$
8	Prisha	850	SHS4

BLOCKCHAIN TABLE

19c Blockchain Tables

- **Requires 19.10 minimum (January 2021)**
- **19.10 (January 2021) – must apply patch 32431413**
  - Required Blockchain Tables library is missing
  - Results in **ORA-901: Invalid Create Statement** when creating a Blockchain table
  - See Oracle Support Note Doc ID 2768266.1
  - 32431413: 19.10 RU FOR ORACLE IS MISSING QCPLK.O WHICH GETS LINKED INTO LIBGENERIC19.A
- **19.11 (April 2021) – no patches required**

19c Blockchain Tables

- **Initialization parameter COMPATIBLE must be set to 19.10.0 or greater**
  - Default for 19c is 19.0.0
  - Locks pluggable databases to 19.10.0 and may not be moved to lower versions
  - `alter system set compatible='19.10.0' scope=spfile;`

```
ORA-00406: COMPATIBLE parameter needs to be 19.10.0.0.0 or greater
ORA-00722: Feature "Blockchain table"
00406. 00000 -  "COMPATIBLE parameter needs to be %s or greater"
*Cause:    The COMPATIBLE initialization parameter is not high
           enough to allow the operation. Allowing the command would make
           the database incompatible with the release specified by the
           current COMPATIBLE parameter.
*Action:   Shutdown and startup with a higher compatibility setting.
```

Blockchain Table Creation

`CREATE BLOCKCHAIN TABLE <table> (<columns>)`	▪ Create table DDL similar to standard tables
`NO DROP [ UNTIL <0+> DAYS IDLE ]`	▪ `NO DROP` without days will prevent table from ever being dropped ▪ Use `1 DAYS` during testing so table can be dropped ▪ Don’t use `0 DAYS` as this may cause errors
`NO DELETE { [ LOCKED ] | (UNTIL <16+> DAYS AFTER INSERT [ LOCKED ]) }`	▪ `NO DELETE` prevents rows from ever being deleted – cannot be changed ▪ `UNTIL number DAYS AFTER INSERT` prevents rows from deleted for x days ▪ `LOCKED` does not allow setting to be changed ▪ Retention periods can only be increased
`HASHING USING sha2_512 VERSION v1`	▪ `sha2_512` hash and `v1` version are fixed in this version

Blockchain Table DDL and DML

`DROP TABLE`	▪ Cannot drop until after NO DROP days has expired ▪ ORA-05723: drop blockchain table <> not allowed
`ALTER TABLE`	▪ Cannot modify table structure (add, drop, rename columns) or move tablespace ▪ ORA-05715: operation not allowed on the blockchain table
`DROP TABLESPACE`	▪ ORA-05723: drop blockchain table <> not allowed
`TRUNCATE TABLE`	▪ Never allowed ▪ ORA-05715: operation not allowed on the blockchain table
`UPDATE`	▪ Never allowed ▪ ORA-05715: operation not allowed on the blockchain table
`DELETE`	▪ Never allowed – use DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS ▪ ORA-05715: operation not allowed on the blockchain table
`DROP USER CASCADE`	▪ ORA-00604/ORA-05723 if user has unexpired rows

Add ORA-05723 and ORA-05715 to list of monitored Oracle error messages.

Blockchain Table Hidden Columns

`ORABCTAB_INST_ID$`	▪ RAC instance ID
`ORABCTAB_CHAIN_ID$`	▪ Each table may have up to 32 chains (0-31) in current use to allow for parallelism
`ORABCTAB_SEQ_NUM$`	▪ Row number in a chain
`ORABCTAB_CREATION_TIME$`	▪ Row creation timestamp, always UTC
`ORABCTAB_USER_NUMBER$`	▪ USER_ID of the user who inserted row (DBA_USERS.USER_ID)
`ORABCTAB_HASH$`	▪ Calculated row hash (SHA2_512, v1)

`ORABCTAB_SIGNATURE$,`
`ORABCTAB_SIGNATURE_ALG$,`
`ORABCTAB_SIGNATURE_CERT$`
	▪ Signature information when row signing is used
	▪ Signature based on certificate and ORABCTAB_HASH$

`ORABCTAB_SPARE$`	▪ Future use

Blockchain Table Data Dictionary Views

{CDB|DBA|ALL|USER}_ BLOCKCHAIN_TABLES	▪ Information about blockchain tables including row retention period, table retention period, and hashing algorithm used to chain rows ▪ View over the SYS.BLOCKCHAIN_TABLE$ table

```
SELECT row_retention "Row Retention Period", row_retention_locked "Row Retention Lock",
table_inactivity_retention "Table Retention Period", hash_algorithm "Hash Algorithm"
FROM dba_blockchain_tables WHERE table_name='BANK_LEDGER';

Row Retention Period Row Retention Lock   Table  Retention Period Hash Algorithm
-------------------- ------------------  ------------------------ --------------
                  16 YES                                       31 SHA2_512
```

DBMS_BLOCKCHAIN_TABLE Package

`DELETE_EXPIRED_ROWS`	▪ Deletes all expired rows or rows prior to a date ▪ Must have DELETE on table in order to delete rows
`VERIFY_ROWS`	▪ Verifies all rows or rows between two timestamps and optionally signatures for each row ▪ Must have SELECT on table in order to verify rows
`SIGN_ROW`	▪ Sign a row – user must be the one who inserted the row ▪ A row can only be signed once ▪ Must have INSERT on table in order to sign a row ▪ Must also have SELECT on table to sign a row as instance id, chain id, and row id are required
`VERIFY_TABLE_BLOCKCHAIN`	▪ Verifies rows between two signed rows ▪ Must have SELECT on table in order to verify rows

Blockchain Table Observations

- As blockchain tables are new to Oracle Database 19c and 21c, should be carefully tested as issues and bugs may be encountered for the next 6 to 12 months
- Multiple security vulnerabilities will likely be fixed over the next 6 months due to such issues as bypasses of DROP TABLE
- Blockchain tables should not be used for high volume transactional tables due to overhead required for the blockchain
- No margin for error in determining DROP and DELETE days, so blockchain tables must be well designed from the beginning
  - Set BLOCKCHAIN_TABLE_MAX_NO_DROP to 0 for test and development
- Signing rows requires a certificate for each database user although most applications use a single database account
- Use in combination with Oracle TDE tablespace encryption and Table Compression to help protect against direct manipulation of data by editing data files

Key Blockchain Tables Restrictions and Limitations

- Carefully review the restrictions and limitations for blockchain tables
- Not all datatypes allowed such as no TIMESTAMP WITH TIME ZONE
- No inserting data using parallel DML or direct-path loading
- No distributed transactions or XA transactions
- No flashback table
- No Oracle Virtual Private Database (VPD) policies or Oracle Label Security (OLS) policies
- Oracle Data Pump Export and Import removes the blockchain from the table
- Blockchain table can not be created in the root container database
  - `ORA-05729: blockchain or immutable table cannot be created in root container`

Blockchain Table Auditing

- Audit key blockchain table events, monitor for ORA-05723 and ORA-05715 errors
- Assuming Unified Auditing with 19c and 21c

①

```
CREATE AUDIT POLICY blockchain_table_actions
ACTIONS drop table, truncate table, drop tablespace, drop user;

AUDIT POLICY blockchain_table_actions WHENEVER NOT SUCCESSFUL;
```

②

```
CREATE AUDIT POLICY blockchain_tables
ACTIONS update ON schema.t1, delete ON schema.t1, alter ON schema.t1,
update ON schema.t2, delete ON schema.t2, alter ON schema.t2;

AUDIT POLICY blockchain_tables;
```

③

```
CREATE AUDIT POLICY blockchain_package
ACTIONS EXECUTE ON sys.dbms_blockchain_table;

AUDIT POLICY blockchain_package;
```

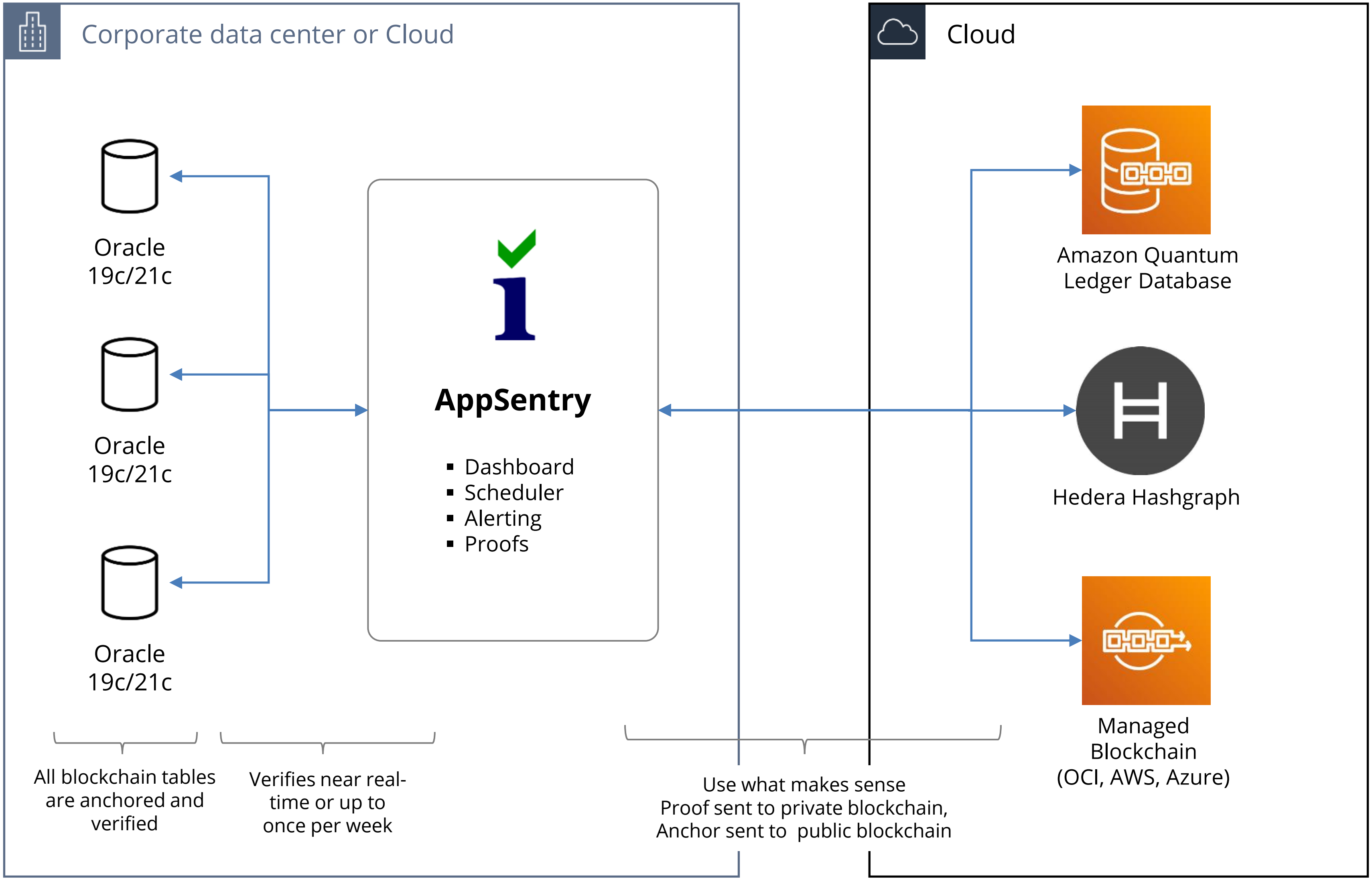
Blockchain Table Integrity

Oracle Database 21c Database Administrator’s Guide

“An important aspect of maintaining the integrity of blockchain table data is to ensure that all rows are intact. Computing a signed digest provides a snapshot of the metadata and data about the last row in all chains at a particular time. You must store this information in [an external] repository. Signed digests generated at various times comprise the input to the DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN procedure. Use this procedure to verify the integrity of rows created between two *specified times.”*

- **Use Integrigy AppSentry to periodically retrieve, store, and verify the integrity of all blockchain tables – “anchor the blockchain”**
  - Fingerprints the database to verify the database
  - Detects all blockchain tables
  - Fingerprints the table to verify the table
  - Generates a signed digest for each blockchain table
  - “Anchors” the signed digests for each blockchain table to AppSentry, AWS Quantum Ledger Database, or Hedera Hashgraph (future Ethereum and Oracle, Azure, and AWS blockchains)
  - Verifies since last signed digest to confirm the integrity of the blockchain table

AppSentry Blockchain – Blockchain Table Anchor



Corporate data center or Cloud

Oracle 19c/21c

Oracle 19c/21c

Oracle 19c/21c

AppSentry

- Dashboard
- Scheduler
- Alerting
- Proofs

Cloud

Amazon Quantum Ledger Database

Hedera Hashgraph

Managed Blockchain (OCI, AWS, Azure)

All blockchain tables are anchored and verified

Verifies near real-time or up to once per week

Use what makes sense
Proof sent to private blockchain,
Anchor sent to public blockchain

Oracle Immutable Tables

- Immutable = unable to be changed
- Insert-only, tamper-resistant tables without blockchain
- Introduced as part of 19.11 (April 2021) and 21.3 (April 2021)
  - Initialization parameter COMPATIBLE must be set to 19.11.0 or 21.3.0
- Includes same system generated hidden columns as Blockchain Table but only two columns are populated –
  - ORABCTAB_CREATION_TIME$
  - ORABCTAB_USER_NUMBER$
- Support VPD policies, distributed transactions, and XA transactions
- Immutable tables should be used for every audit trail, security log, and compliance table if a blockchain table is not required

Create Immutable Table

```
create immutable table imt_t1 (
  id             number,
  name           varchar2(20),
  quantity       number,
  created_date   date
)
no drop until 0 days idle
no delete until 16 days after insert;
```

Immutable Table Data Dictionary Views

{CDB|DBA|ALL|USER}_ IMMUTABLE_TABLES	▪ Information about blockchain tables including row retention period and table retention period ▪ View over the SYS.IMMUTABLE_TABLE$ table

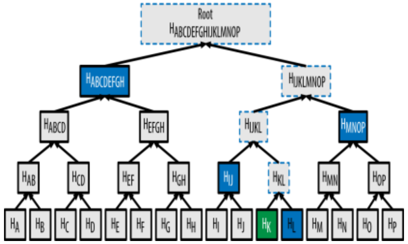
```
SELECT row_retention "Row Retention Period", row_retention_locked "Row Retention Lock",
table_inactivity_retention "Table Retention Period"
FROM dba_immutable_tables
WHERE table_name = 'TRADE_LEDGER';

Row Retention Period Row Retention Locked Table Retention Period
-------------------- -------------------- ----------------------
                 110                   NO                     16
```

AppSentry Blockchain – Standard Table Anchor

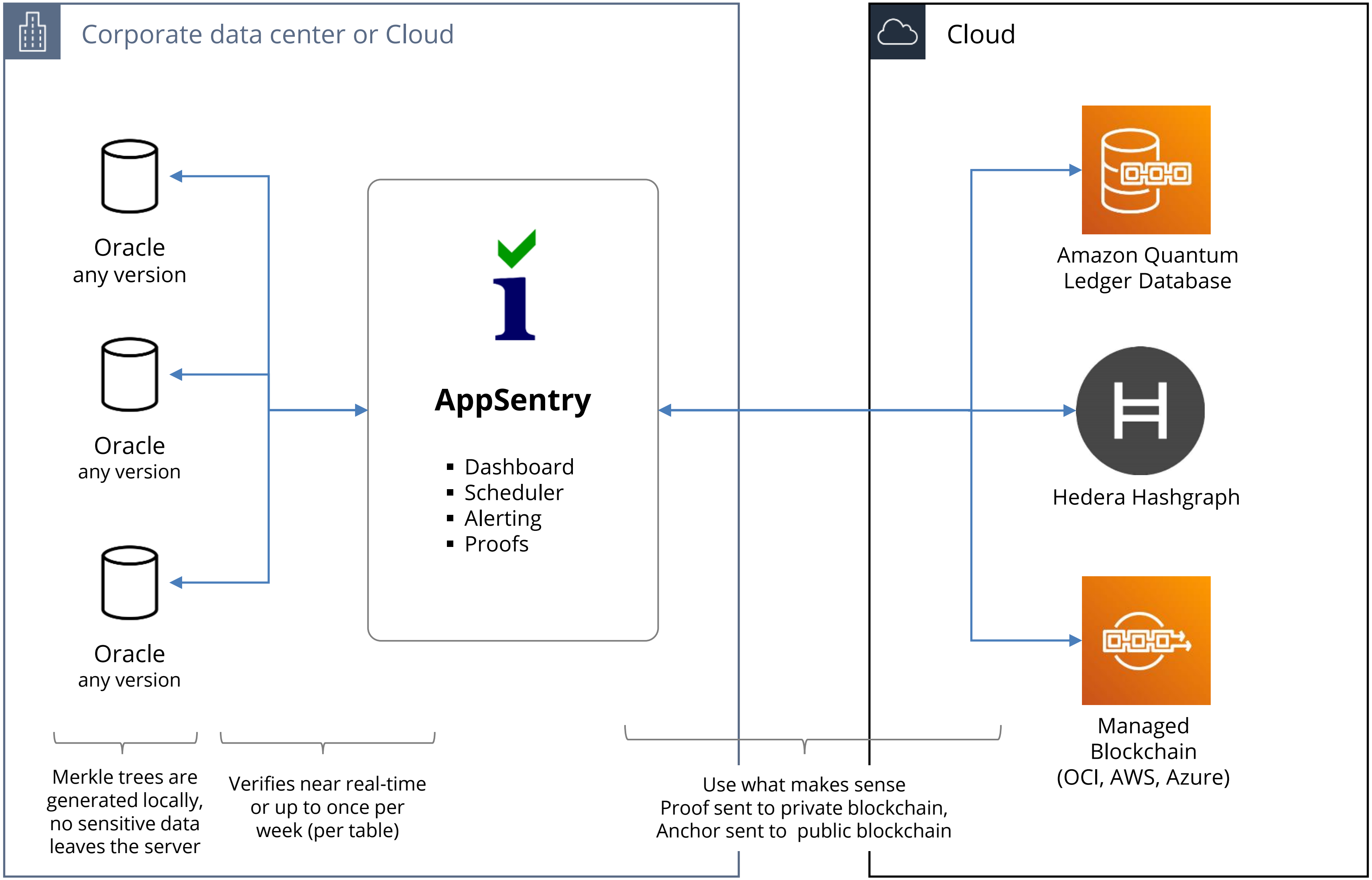
- **AppSentry Blockchain allows you to anchor any Oracle table when you can't use Blockchain or Immutable tables – create digital trust**
  - Pre-19.10 databases
  - Package applications

- **Generates Merkle trees for all new and changed rows**
  - A Merkle tree is a tree of hashes that allow for efficient and secure verification of large structures of data
  - Triggers and Flashback may be used to enhance detection of table inserts and changes
  - Merkle trees are calculated in-database so no sensitive data is transferred outside of the database server



- **Proofs are anchored to private or public blockchains**
  - Amazon Quantum Ledger Database – cloud ledger database
  - Hedera Hashgraph – public distributed ledger with consistent pricing and fast, low-latency transactions
  - Plugin API to integrate any service or blockchain network

AppSentry Blockchain – Standard Table Anchor



- Dashboard
- Scheduler
- Alerting
- Proofs

Merkle trees are generated locally, no sensitive data leaves the server

Verifies near real-time or up to once per week (per table)

Use what makes sense
Proof sent to private blockchain,
Anchor sent to public blockchain

Integrigy Contact Information

Stephen Kost

Chief Technology Officer

Integrigy Corporation

web – **www.integrigy.com**

e-mail – **info@integrigy.com**

blog – **integrigy.com/oracle-security-blog**

youtube – **youtube.com/integrigy**

linkedin – **linkedin.com/company/integrigy**

twitter – **twitter.com/integrigy**