Oracle Database 23aiは、「AI for Data」をコンセプトに、データとAIの統合を強化し、開発者の生産性を向上させるための多くの新機能が導入されたLong Term Releaseです。

### Oracle Database 23ai: 主要機能と実装例

#### 1\. DML (Data Manipulation Language)

データの追加、変更、削除を行うためのSQL文です。

 \* \*\*INSERT (データの挿入)\*\*

 \* \*\*機能:\*\* テーブルに新しい行を追加します。

 \* \*\*Oracle 23aiでの拡張:\*\* 従来の1行ずつの挿入に加え、`VALUES`句で複数行を一度に挿入できるようになりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のINSERT (21cまで)

 INSERT INTO employees (employee\_id, first\_name, last\_name, email) VALUES (101, 'John', 'Doe', 'john.doe@example.com');

 INSERT INTO employees (employee\_id, first\_name, last\_name, email) VALUES (102, 'Jane', 'Smith', 'jane.smith@example.com');

 -- Oracle 23aiでの複数行INSERT

 INSERT INTO employees (employee\_id, first\_name, last\_name, email) VALUES

 (101, 'John', 'Doe', 'john.doe@example.com'),

 (102, 'Jane', 'Smith', 'jane.smith@example.com'),

 (103, 'Peter', 'Jones', 'peter.jones@example.com');

 ```

 \* \*\*UPDATE (データの更新)\*\*

 \* \*\*機能:\*\* テーブルの既存の行のデータを変更します。

 \* \*\*Oracle 23aiでの拡張:\*\* `FROM`句を使用して、他のテーブルとの結合に基づいて更新できるようになりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のUPDATE (21cまで、サブクエリを使用)

 UPDATE employees e

 SET e.salary = e.salary \* 1.10

 WHERE e.department\_id IN (SELECT d.department\_id FROM departments d WHERE d.department\_name = 'IT');

 -- Oracle 23aiでの直接JOINによるUPDATE

 UPDATE employees e

 SET e.salary = e.salary \* 1.10

 FROM departments d

 WHERE d.department\_id = e.department\_id

 AND d.department\_name = 'IT';

 ```

 \* \*\*DELETE (データの削除)\*\*

 \* \*\*機能:\*\* テーブルから行を削除します。

 \* \*\*Oracle 23aiでの拡張:\*\* `FROM`句を使用して、他のテーブルとの結合に基づいて削除できるようになりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のDELETE (21cまで、サブクエリを使用)

 DELETE FROM employees e

 WHERE e.department\_id IN (SELECT d.department\_id FROM departments d WHERE d.department\_name = 'HR');

 -- Oracle 23aiでの直接JOINによるDELETE

 DELETE FROM employees e

 FROM departments d

 WHERE d.department\_id = e.department\_id

 AND d.department\_name = 'HR';

 ```

 \* \*\*MERGE (結合と更新/挿入)\*\*

 \* \*\*機能:\*\* ソーステーブルのデータに基づいて、ターゲットテーブルの行を挿入または更新します。

 \* \*\*実装例:\*\*

 ```sql

 -- 例: products\_staging から products テーブルをMERGE

 MERGE INTO products p

 USING products\_staging ps

 ON (p.product\_id = ps.product\_id)

 WHEN MATCHED THEN

 UPDATE SET p.price = ps.price, p.quantity = ps.quantity

 WHEN NOT MATCHED THEN

 INSERT (product\_id, product\_name, price, quantity)

 VALUES (ps.product\_id, ps.product\_name, ps.price, ps.quantity);

 ```

#### 2\. DDL (Data Definition Language)

データベースオブジェクトの作成、変更、削除を行うためのSQL文です。

 \* \*\*CREATE (オブジェクトの作成)\*\*

 \* \*\*機能:\*\* テーブル、ビュー、インデックスなどのデータベースオブジェクトを作成します。

 \* \*\*Oracle 23aiでの拡張:\*\* `IF NOT EXISTS`句を使用できるようになり、オブジェクトが存在しない場合にのみ作成する条件付きDDLが可能になりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のCREATE

 CREATE TABLE my\_table (id NUMBER, name VARCHAR2(100));

 -- Oracle 23aiでのCREATE IF NOT EXISTS

 CREATE TABLE IF NOT EXISTS my\_table (id NUMBER, name VARCHAR2(100));

 -- (my\_tableが存在しない場合のみ作成される)

 ```

 \* \*\*ALTER (オブジェクトの変更)\*\*

 \* \*\*機能:\*\* 既存のデータベースオブジェクトの構造を変更します。

 \* \*\*実装例:\*\*

 ```sql

 ALTER TABLE employees ADD (phone\_number VARCHAR2(20));

 ALTER TABLE employees MODIFY (email VARCHAR2(255));

 ```

 \* \*\*DROP (オブジェクトの削除)\*\*

 \* \*\*機能:\*\* データベースオブジェクトを削除します。

 \* \*\*Oracle 23aiでの拡張:\*\* `IF EXISTS`句を使用できるようになり、オブジェクトが存在する場合にのみ削除する条件付きDDLが可能になりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のDROP

 DROP TABLE my\_table;

 -- Oracle 23aiでのDROP IF EXISTS

 DROP TABLE IF EXISTS my\_table;

 -- (my\_tableが存在する場合のみ削除される)

 ```

#### 3\. SELECT

データベースからデータを取得するためのSQL文です。

 \* \*\*基本的なSELECT\*\*

 \* \*\*機能:\*\* 指定したテーブルからデータを取得します。

 \* \*\*Oracle 23aiでの拡張:\*\* `FROM`句なしで、関数や式の結果を直接選択できるようになりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のSELECT (DUALテーブルを使用)

 SELECT SYSDATE FROM DUAL;

 SELECT 1 + 1 FROM DUAL;

 -- Oracle 23aiでのFROM句なしSELECT

 SELECT SYSDATE;

 SELECT 1 + 1;

 -- テーブルからのデータ取得

 SELECT employee\_id, first\_name, last\_name, salary FROM employees;

 ```

#### 4\. ソート (ORDER BY)

クエリ結果を指定した列の順序で並べ替えます。

 \* \*\*機能:\*\* `ORDER BY`句を使用して、昇順 (ASC) または降順 (DESC) で結果を並べ替えます。

 \* \*\*実装例:\*\*

 ```sql

 SELECT first\_name, last\_name, salary

 FROM employees

 ORDER BY salary DESC, last\_name ASC; -- 給与の降順、次に姓の昇順

 ```

#### 5\. 単一行関数 (Single-Row Functions)

各行に対して1つの結果を返す関数です。

 \* \*\*機能:\*\* 数値、文字、日付などの操作を行います。

 \* \*\*実装例:\*\*

 ```sql

 SELECT

 employee\_id,

 UPPER(first\_name) AS upper\_first\_name, -- 大文字に変換

 LENGTH(last\_name) AS last\_name\_length, -- 文字列の長さ

 SUBSTR(email, 1, INSTR(email, '@') - 1) AS username, -- サブ文字列の抽出

 ROUND(salary, 0) AS rounded\_salary, -- 数値の丸め

 TO\_CHAR(hire\_date, 'YYYY/MM/DD') AS formatted\_hire\_date, -- 日付の書式設定

 NVL(commission\_pct, 0) AS actual\_commission\_pct -- NULL値の置換

 FROM employees;

 ```

#### 6\. GROUP BY

指定した列の値に基づいて行をグループ化し、集計関数を適用します。

 \* \*\*機能:\*\* `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`などの集計関数と組み合わせて使用します。

 \* \*\*Oracle 23aiでの拡張:\*\* `SELECT`リストで定義した列の別名を`GROUP BY`句で使用できるようになりました。

 \* \*\*実装例:\*\*

 ```sql

 -- 従来のGROUP BY (21cまで、式を繰り返す)

 SELECT department\_id, COUNT(employee\_id) AS num\_employees

 FROM employees

 GROUP BY department\_id;

 -- Oracle 23aiでのGROUP BY (列別名を使用)

 SELECT department\_id AS dept\_id, COUNT(employee\_id) AS num\_employees

 FROM employees

 GROUP BY dept\_id; -- dept\_id 別名を使用

 ```

#### 7\. 結合 (Joins)

複数のテーブルから関連するデータを結合して取得します。

 \* \*\*機能:\*\* `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL OUTER JOIN`, `CROSS JOIN`, `SELF JOIN`などがあります。

 \* \*\*Oracle 23aiでの拡張:\*\* `UPDATE`文と`DELETE`文で直接`JOIN`句を使用できるようになりました（上記DMLのセクション参照）。

 \* \*\*実装例:\*\*

 ```sql

 -- INNER JOIN (部門に所属する従業員)

 SELECT e.first\_name, e.last\_name, d.department\_name

 FROM employees e

 INNER JOIN departments d ON e.department\_id = d.department\_id;

 -- LEFT JOIN (すべての部門と、所属する従業員がいれば表示)

 SELECT d.department\_name, e.first\_name, e.last\_name

 FROM departments d

 LEFT JOIN employees e ON d.department\_id = e.department\_id;

 ```

#### 8\. 副問い合わせ (Subqueries)

別のSQL文の内側にネストされたクエリです。

 \* \*\*機能:\*\* 外部クエリの条件として使用したり、データソースとして使用したりします。

 \* \*\*実装例:\*\*

 ```sql

 -- 単一行サブクエリ (平均給与より高い給与の従業員)

 SELECT first\_name, last\_name, salary

 FROM employees

 WHERE salary > (SELECT AVG(salary) FROM employees);

 -- 複数行サブクエリ (IT部門またはHR部門の従業員)

 SELECT first\_name, last\_name

 FROM employees

 WHERE department\_id IN (SELECT department\_id FROM departments WHERE department\_name IN ('IT', 'HR'));

 -- FROM句でのインラインビュー (部門ごとの従業員数)

 SELECT d.department\_name, emp\_counts.num\_employees

 FROM departments d

 JOIN (SELECT department\_id, COUNT(\*) AS num\_employees FROM employees GROUP BY department\_id) emp\_counts

 ON d.department\_id = emp\_counts.department\_id;

 ```

#### 9\. 集合演算 (Set Operations)

複数の`SELECT`文の結果を結合または比較します。

 \* \*\*機能:\*\* `UNION`, `UNION ALL`, `INTERSECT`, `MINUS`があります。

 \* \*\*実装例:\*\*

 ```sql

 -- UNION (重複を除いて結合)

 SELECT employee\_id, first\_name FROM employees WHERE salary > 10000

 UNION

 SELECT employee\_id, first\_name FROM employees WHERE hire\_date > TO\_DATE('2023-01-01', 'YYYY-MM-DD');

 -- UNION ALL (重複を含めて結合)

 SELECT employee\_id, first\_name FROM employees WHERE salary > 10000

 UNION ALL

 SELECT employee\_id, first\_name FROM employees WHERE hire\_date > TO\_DATE('2023-01-01', 'YYYY-MM-DD');

 -- INTERSECT (両方のクエリに存在する行)

 SELECT employee\_id, first\_name FROM employees WHERE salary > 10000

 INTERSECT

 SELECT employee\_id, first\_name FROM employees WHERE department\_id = 50;

 -- MINUS (最初のクエリに存在し、2番目のクエリに存在しない行)

 SELECT employee\_id, first\_name FROM employees WHERE salary > 10000

 MINUS

 SELECT employee\_id, first\_name FROM employees WHERE department\_id = 50;

 ```

#### 10\. スキーマオブジェクト (Schema Objects)

データベース内に存在する論理的な構造です。

 \* \*\*主要なスキーマオブジェクト:\*\*

 \* \*\*TABLES (表):\*\* データを格納する基本的なオブジェクト。

 ```sql

 CREATE TABLE customers (

 customer\_id NUMBER PRIMARY KEY,

 customer\_name VARCHAR2(100) NOT NULL,

 city VARCHAR2(50)

 );

 ```

 \* \*\*VIEWS (ビュー):\*\* 1つ以上のテーブルから派生した仮想テーブル。

 ```sql

 CREATE VIEW high\_salary\_employees AS

 SELECT employee\_id, first\_name, salary

 FROM employees

 WHERE salary > 15000;

 ```

 \* \*\*INDEXES (索引):\*\* データ検索のパフォーマンスを向上させるための構造。

 ```sql

 CREATE INDEX idx\_employees\_last\_name ON employees (last\_name);

 ```

 \* \*\*SEQUENCES (シーケンス):\*\* 一意の数値を生成するオブジェクト（主キーの自動生成などに使用）。

 ```sql

 CREATE SEQUENCE customer\_id\_seq START WITH 1 INCREMENT BY 1;

 ```

 -- 事前にシーケンスを作成

 CREATE SEQUENCE your\_sequence\_name

 START WITH 1

 INCREMENT BY 1

 NOCACHE

 NOCYCLE;

 -- テーブル作成時にシーケンスを指定

 CREATE TABLE your\_table\_name (

 your\_id\_column\_name NUMBER DEFAULT your\_sequence\_name.NEXTVAL ON NULL, -- シーケンス値をデフォルトとして利用

 column2 VARCHAR2(100),

 column3 DATE,

 -- ... その他の列

 CONSTRAINT pk\_your\_table PRIMARY KEY (your\_id\_column\_name)

 );

 \* \*\*SYNONYMS (シノニム):\*\* オブジェクトの代替名。

 ```sql

 CREATE SYNONYM emp FOR employees;

 ```

 \* \*\*PROCEDURES / FUNCTIONS / PACKAGES / TRIGGERS (プロシージャ/ファンクション/パッケージ/トリガー):\*\* PL/SQLで記述されたプログラム単位。

 \* \*\*Oracle 23aiでの新しいスキーマオブジェクト (JSON機能関連):\*\*

 \* \*\*JSON Relational Duality Views (JSONリレーショナル二面性ビュー):\*\* リレーショナルデータとJSONドキュメントの両方としてアクセスできる新しいタイプのビュー。リレーショナルモデルの効率性とドキュメントAPIの使いやすさを組み合わせます。

 ```sql

 -- 例 (概念的): HRスキーマの部門と従業員を結合した二面性ビュー

 CREATE OR REPLACE JSON RELATIONAL DUALITY VIEW emp\_dept AS

 hr.departments {

 \_id: department\_id,

 name: department\_name,

 employees: hr.employees @insert @update @delete [

 {

 id: employee\_id,

 firstname: first\_name,

 lastname: last\_name,

 email

 }

 ]

 };

 ```

 \* \*\*JSON Collection Tables / Views:\*\* JSONドキュメントを効率的に格納・管理するための新しいテーブル/ビュータイプ。

 @insert:

 このアノテーションが指定されたemployees配列（入れ子になったJSONドキュメントのコレクション）に対して、新しい従業員データ（JSONオブジェクト）を挿入することを許可します。つまり、JSONの形式で新しい従業員を追加する操作が可能になります。

 @update:

 このアノテーションが指定されたemployees配列内の既存の従業員データ（JSONオブジェクト）に対して、そのデータを更新することを許可します。JSONの形式で既存の従業員情報を変更する操作が可能になります。

 @delete:

 このアノテーションが指定されたemployees配列内の既存の従業員データ（JSONオブジェクト）に対して、そのデータを削除することを許可します。JSONの形式で特定の従業員を削除する操作が可能になります。

#### 11\. データディクショナリの機能・実装例 (Data Dictionary Functions/Implementation Examples)

Oracleデータベースのメタデータ（データベースオブジェクトに関する情報）を格納する一連のシステムテーブルとビューです。

 \* \*\*機能:\*\* データベースの構造、ユーザー、権限、オブジェクトの定義などを確認できます。

 \* \*\*主要なデータディクショナリビューのプレフィックス:\*\*

 \* `USER\_`: 現在のユーザーが所有するオブジェクトに関する情報。

 \* `ALL\_`: 現在のユーザーがアクセス権を持つすべてのオブジェクトに関する情報。

 \* `DBA\_`: データベース内のすべてのオブジェクトに関する情報（DBA権限が必要）。

 \* \*\*Oracle 23aiでの拡張:\*\* オブジェクトにメタデータを追加できる「アノテーション (Annotations)」が導入され、データディクショナリビュー (`USER\_ANNOTATIONS`, `ALL\_ANNOTATIONS`, `DBA\_ANNOTATIONS`) で確認できます。

 \* \*\*実装例:\*\*

 ```sql

 -- 現在のユーザーが所有するテーブルの一覧

 SELECT table\_name, status FROM user\_tables;

 -- 現在のユーザーがアクセスできるすべてのビューの一覧

 SELECT owner, view\_name FROM all\_views WHERE owner = 'HR';

 -- 特定のテーブルの列情報

 SELECT column\_name, data\_type, data\_length, nullable

 FROM user\_tab\_columns

 WHERE table\_name = 'EMPLOYEES'

 ORDER BY column\_id;

 -- Oracle 23aiでのアノテーションの例

 -- テーブルにアノテーションを追加

 ALTER TABLE employees ADD ANNOTATION 'This table stores employee information including personal details and salary.';

 -- アノテーションの確認

 SELECT object\_name, annotation\_text

 FROM user\_annotations

 WHERE object\_name = 'EMPLOYEES';

 ```

-----

Oracle 23aiは、開発者向けのSQL機能の強化、JSONデータ処理能力の向上、そしてAI機能の統合に力を入れています。

※調査中

Oracle Database 23aiの主なAI機能

・AI Vector Search (AIベクトル検索)

・SQL ML (SQL Machine Learning)