Composite Datatypes

- Types:
  - PL/SQL RECORDS
  - PL/SQL TABLES (INDEX BY TABLES)
- Contain internal components
- Are reusable

PL/SQL Records

- Must contain one or more components of any scalar, RECORD, or PL/SQL TABLE datatype, called fields
- Are similar in structure to records in a 3GL
- Are not the same as rows in a database table
- Treat a collection of fields as a logical unit
- Are convenient for fetching a row of data from a table for processing

Creating a PL/SQL Record

Syntax

```
TYPE type_name IS RECORD
       (field_declaration[, field_declaration]…);
identifier    type_name;
```

Where *field_declaration* is

```
field_name {field_type | variable%TYPE
    | table.column%TYPE | table%ROWTYPE}
    [[NOT NULL] {:= | DEFAULT} expr]
```

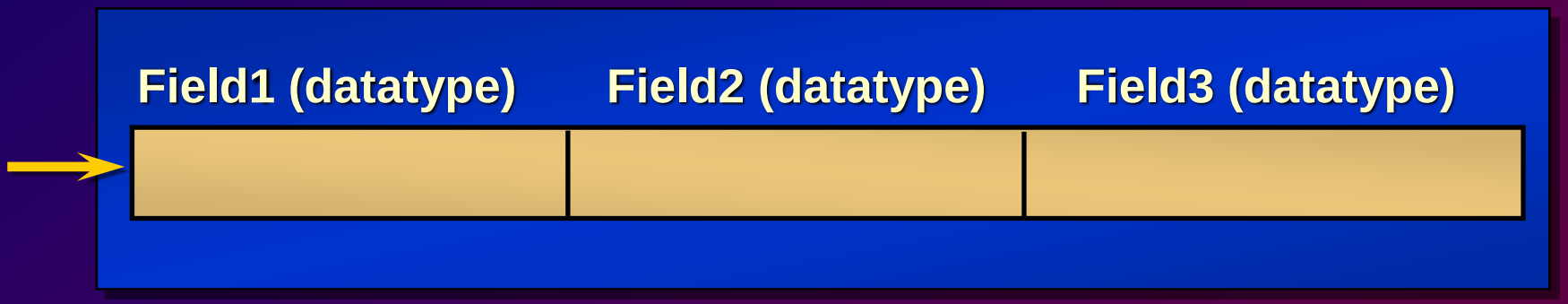
Creating a PL/SQL Record

Declare variables to store the name, job, and salary of a new employee.

Example

```
...
  TYPE emp_record_type IS RECORD
    (ename    VARCHAR2(10),
     job          VARCHAR2(9),
     sal          NUMBER(7,2));
  emp_record      emp_record_type;
...
```

PL/SQL Record Structure



Example

The %ROWTYPE Attribute

- Declare a variable according to a collection of columns in a database table or view.
- Prefix %ROWTYPE with the database table.
- Fields in the record take their names and datatypes from the columns of the table or view.

Advantages of Using %ROWTYPE

- The number and datatypes of the underlying database columns may not be known.
- The number and datatypes of the underlying database column may change at runtime.
- The attribute is useful when retrieving a row with the SELECT statement.

The %ROWTYPE Attribute

Examples

Declare a variable to store the same information about a department as it is stored in the DEPT table.

```
dept_record    dept%ROWTYPE;
```

Declare a variable to store the same information about an employee as it is stored in the EMP table.

```
emp_record     emp%ROWTYPE;
```

PL/SQL Tables

- Are composed of two components:
  - Primary key of datatype BINARY_INTEGER
  - Column of scalar or record datatype
- Increase dynamically because they are unconstrained

Creating a PL/SQL Table

Syntax

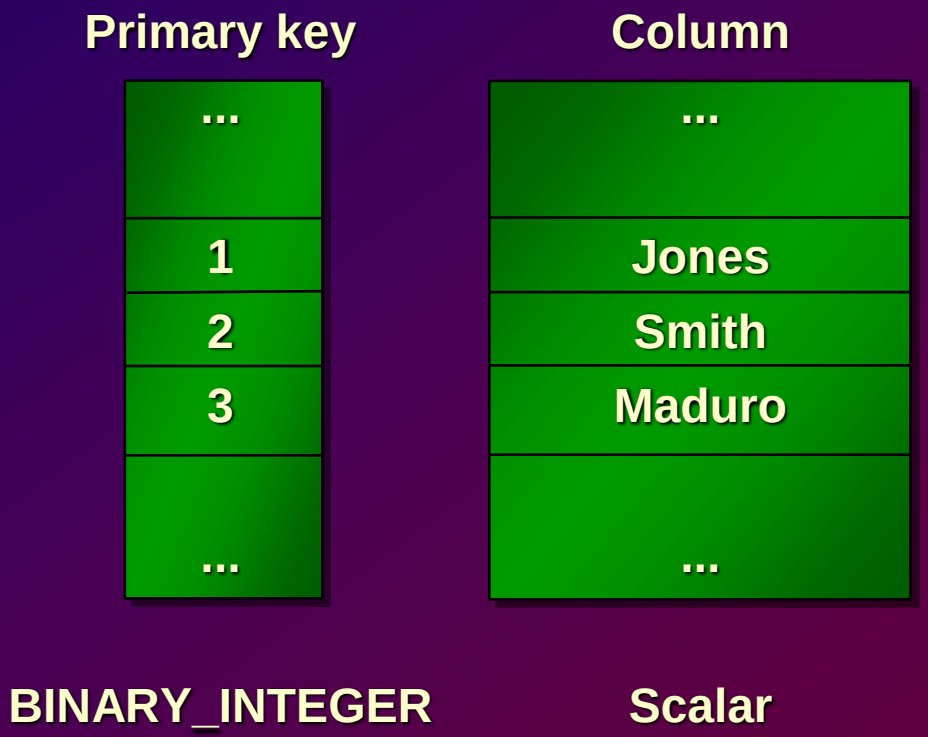
```
TYPE type_name IS TABLE OF
  {column_type | variable%TYPE
  | table.column%TYPE} [NOT NULL]
  [INDEX BY BINARY_INTEGER];
identifier    type_name;
```

Declare a PL/SQL variable to store a name.

Example

```
...
TYPE ename_table_type IS TABLE OF emp.ename%TYPE
  INDEX BY BINARY_INTEGER;
ename_table ename_table_type;
...
```

PL/SQL Table Structure



Creating a PL/SQL Table

```
DECLARE
  TYPE ename_table_type IS TABLE OF emp.ename%TYPE
    INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY BINARY_INTEGER;
  ename_table     ename_table_type;
  hiredate_table  hiredate_table_type;
BEGIN
  ename_table(1) := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
```

Using PL/SQL Table Methods

The following methods make PL/SQL tables easier to use:

- EXISTS
- COUNT
- FIRST and LAST
- PRIOR
- NEXT
- EXTEND
- TRIM
- DELETE

PL/SQL Table of Records

- Define a TABLE variable with the %ROWTYPE attribute.
- Declare a PL/SQL variable to hold department information.

Example

```
DECLARE
  TYPE dept_table_type IS TABLE OF dept%ROWTYPE
    INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```