Database Preliminaries

Data Models

		Relational		Non-Relational
Analytics	Proprietary Storage	Amazon Redshift EMC Greenplum HP Vertica	IBM Netezza Oracle Teradata MPP	
	Hadoop Storage	Cloudera Impala Presto	Hive SQL-on-Hadoop	MapReduce
Operational		Traditional SQL	NewSQL	NoSQL
	Proprietary Storage	Oracle DB2 SQL Server MySQL	User-Sharded MySQL NuoDB Clustrix On-Disk MemSQL VoltDB In-Memory	Key Value: Aerospike, Riak Column Family: Cassandra Document: MongoDB Graph: Neo4j, InfiniteGraph
	Hadoop Storage		Splice Machine On-Hadoop	Column Family: HBase

Picture source: www.sqlservercentral.com

Relational Database Concept

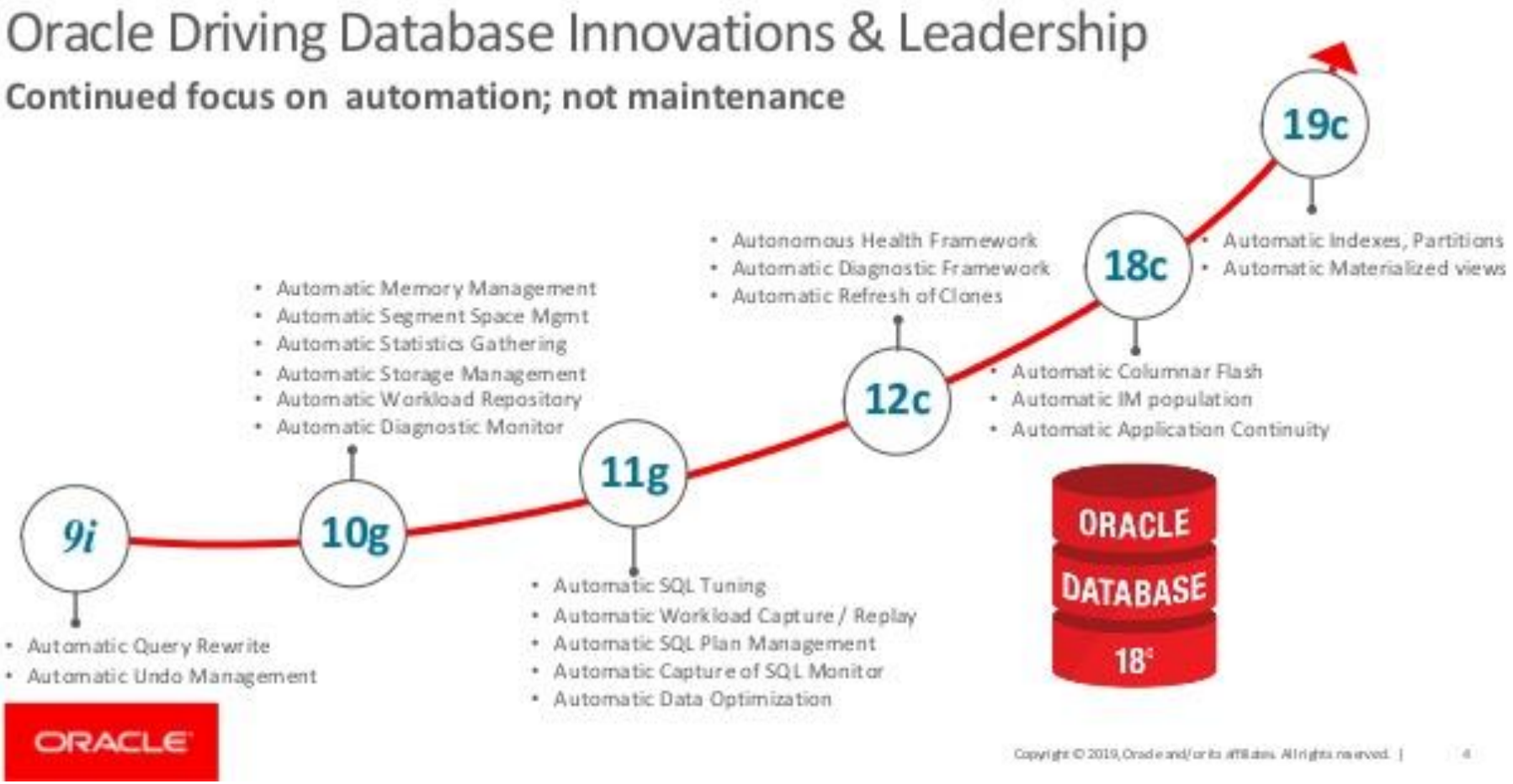
- Dr. E. F. Codd proposed the relational model for database systems in 1970
  - A paper entitled: *A Relational Model of Data for Large Shared Data Banks.*
  - The common models used at that time were hierarchical and network, or even simple flat-file data structures.
- It is the basis for the relational database management system (RDBMS).
  - Soon became very popular, especially for their ease of use and flexibility in structure
- The relational model consists of the following:
  - Collection of objects or relations
  - Set of operators to act on the relations
  - Data integrity for accuracy and consistency

More information: *An Introduction to Database Systems, Eighth Edition* (Addison-Wesley: 2004), written by Chris Date.

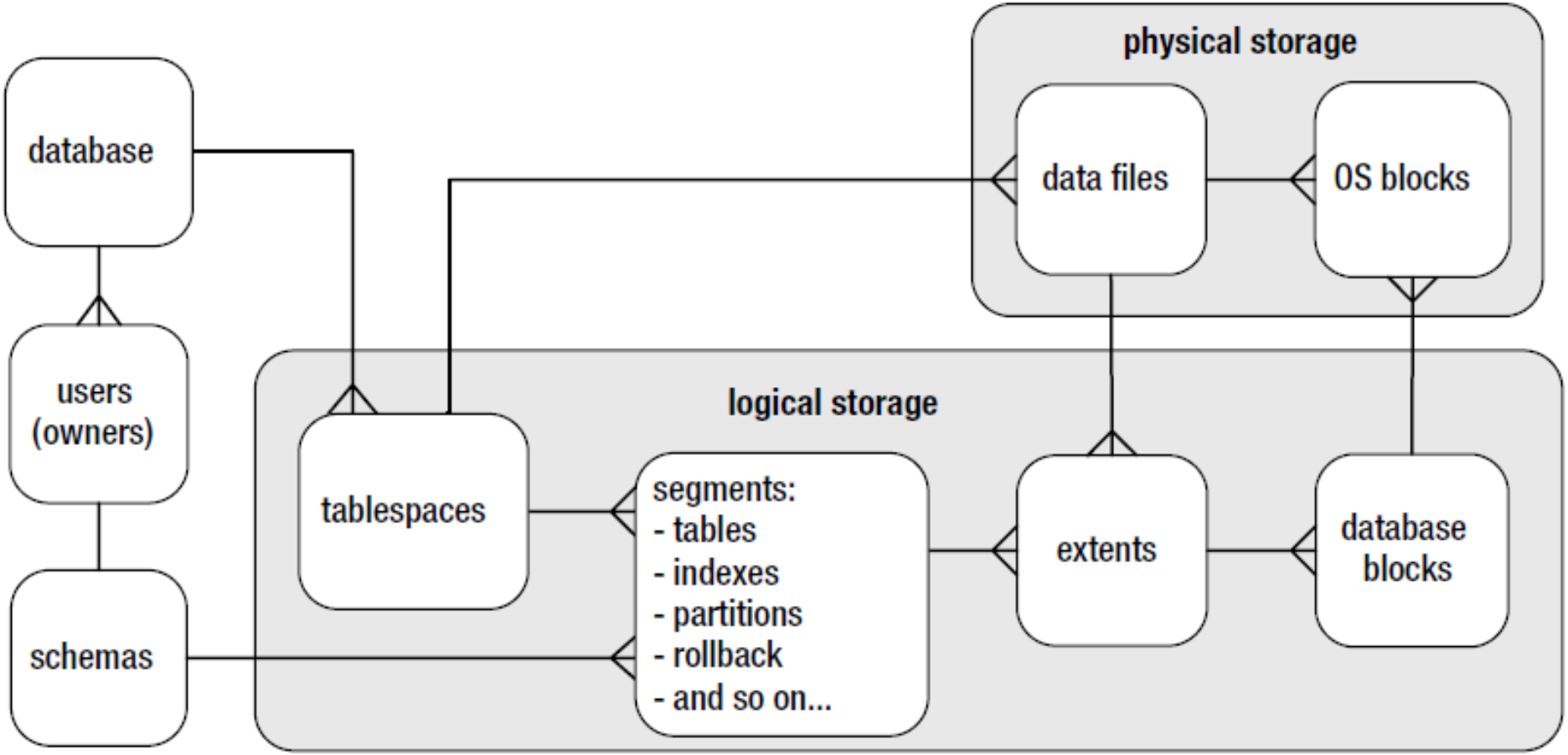
Oracle Database Preliminaries

Oracle Academy Study Materials

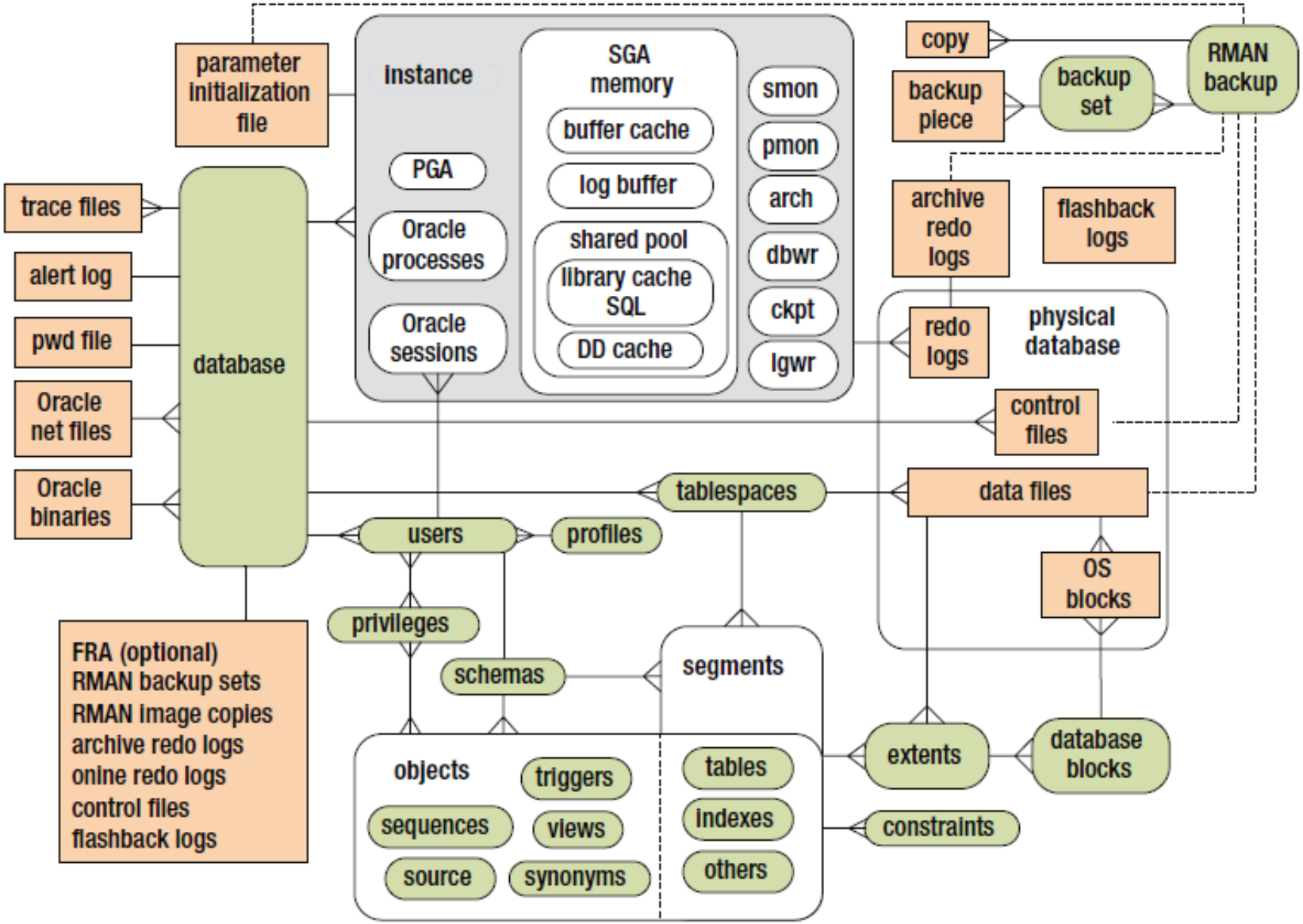
Oracle Database – Release History



Oracle Database Structure



Oracle Database Structure



Development Environments

- SQL Developer
- SQL*Plus
- SQLCL

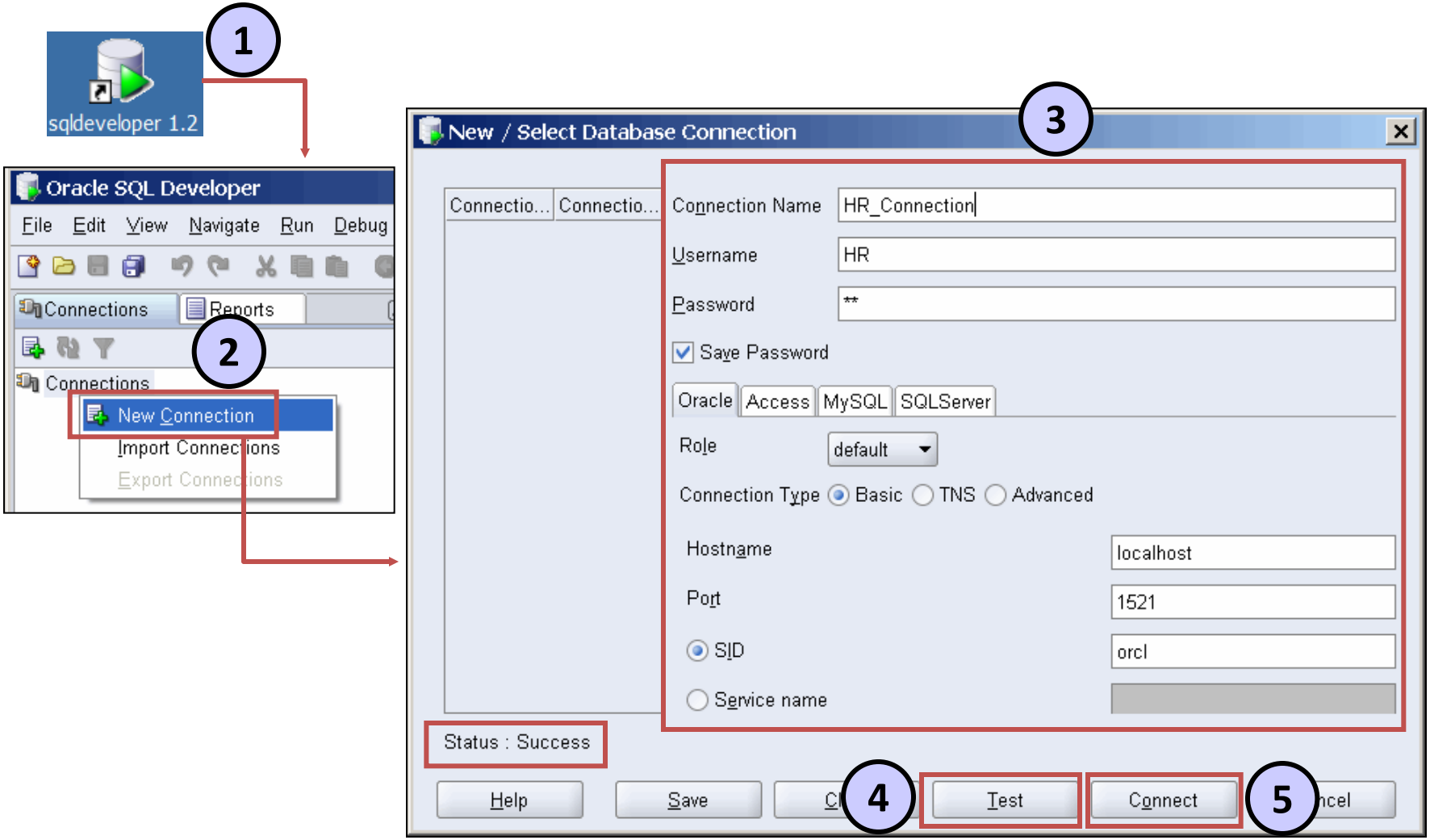
Development Environments

What Is Oracle SQL Developer?

- Oracle SQL Developer is a free graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using the standard Oracle database authentication.
- You can use either SQL Developer or SQL*Plus in this course.

Development Environments

Starting SQL Developer and Creating a Database Connection

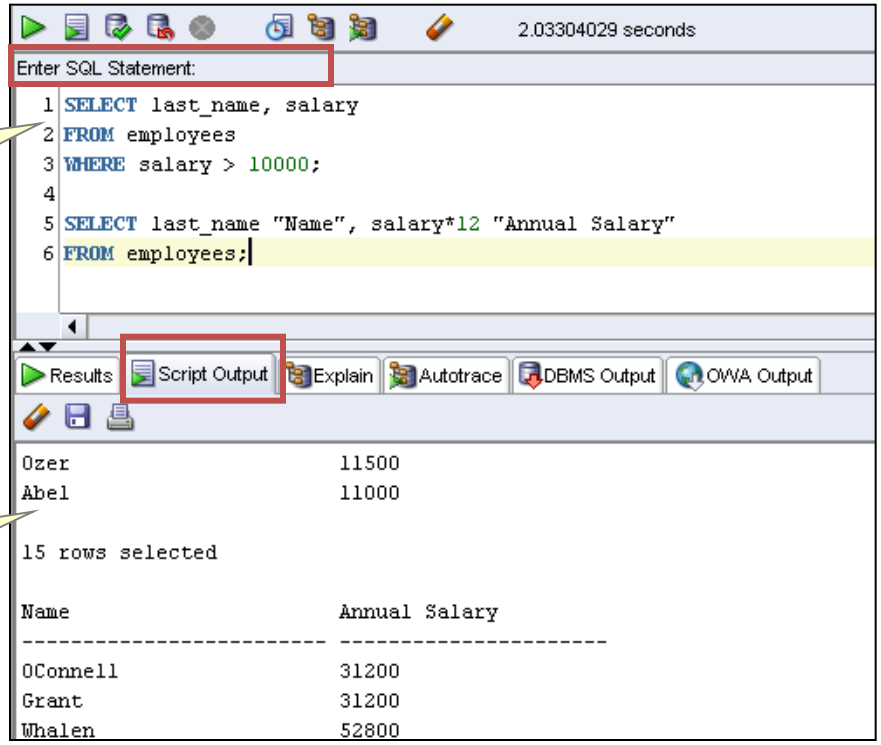


Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.

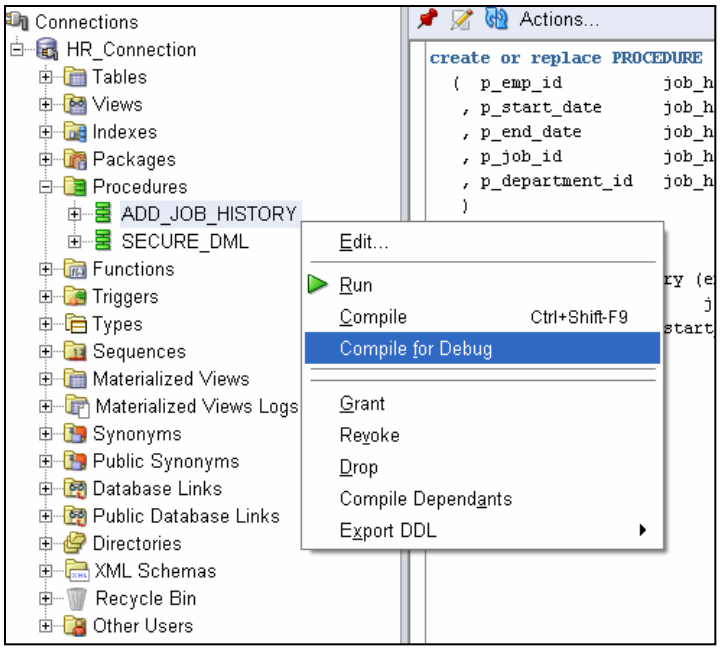
Use the **Enter SQL Statement** box to enter single or multiple SQL statements.

View the results on the **Script Output** tabbed page.



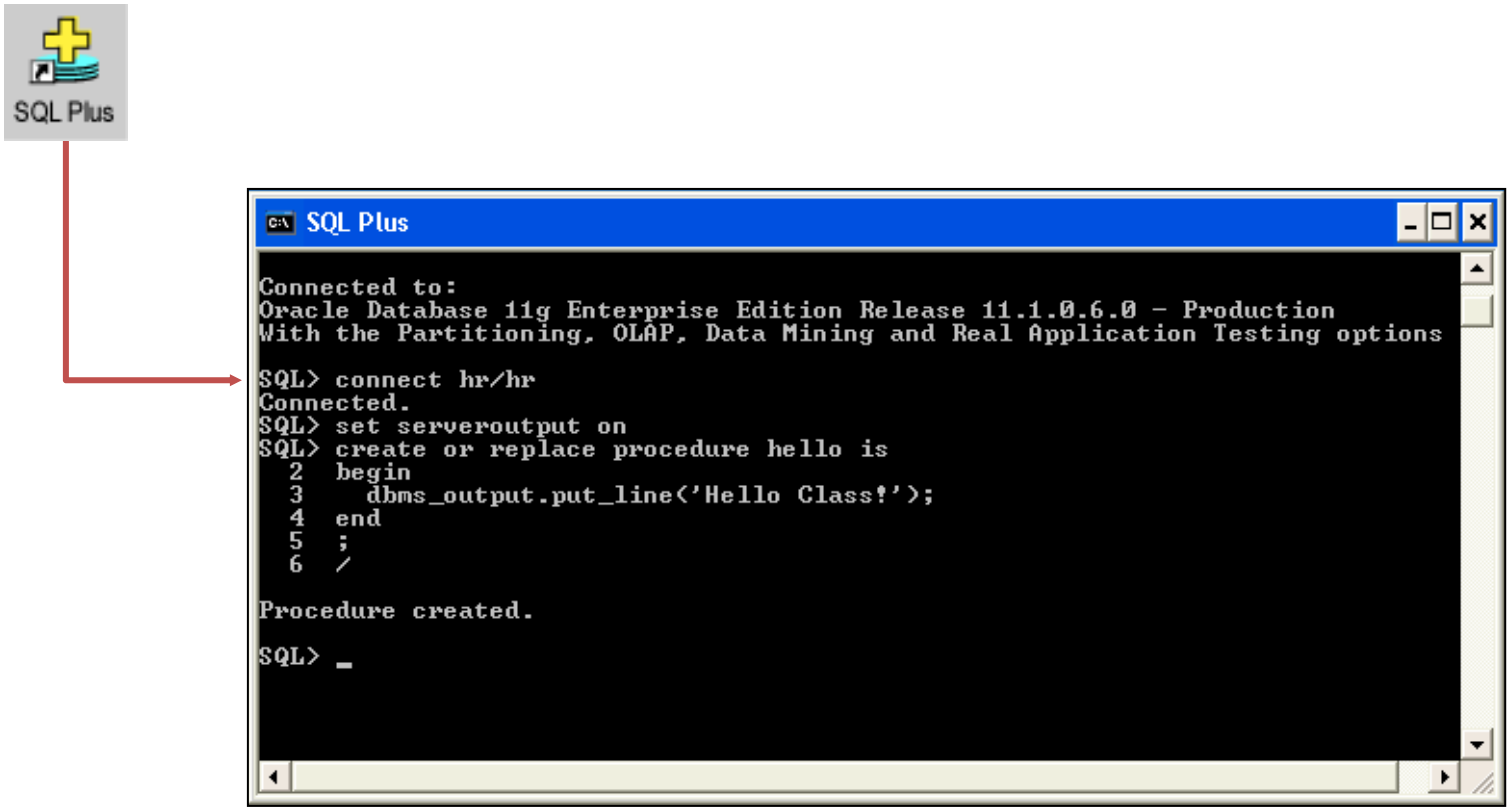
Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform step into and step over tasks.



Development Environments

Coding PL/SQL in SQL*Plus



SQL - Review

Oracle Academy Study Materials

Accessing Data in an RDBMS

- A relational database-management system (RDBMS) organizes data into related rows and columns.
- To access the data in a database, you do not need to know where the data is located physically, nor do you need to specify an access route to the tables.
- You simply use structured query language (SQL) statements and operators.

Using SQL to Query Your Database

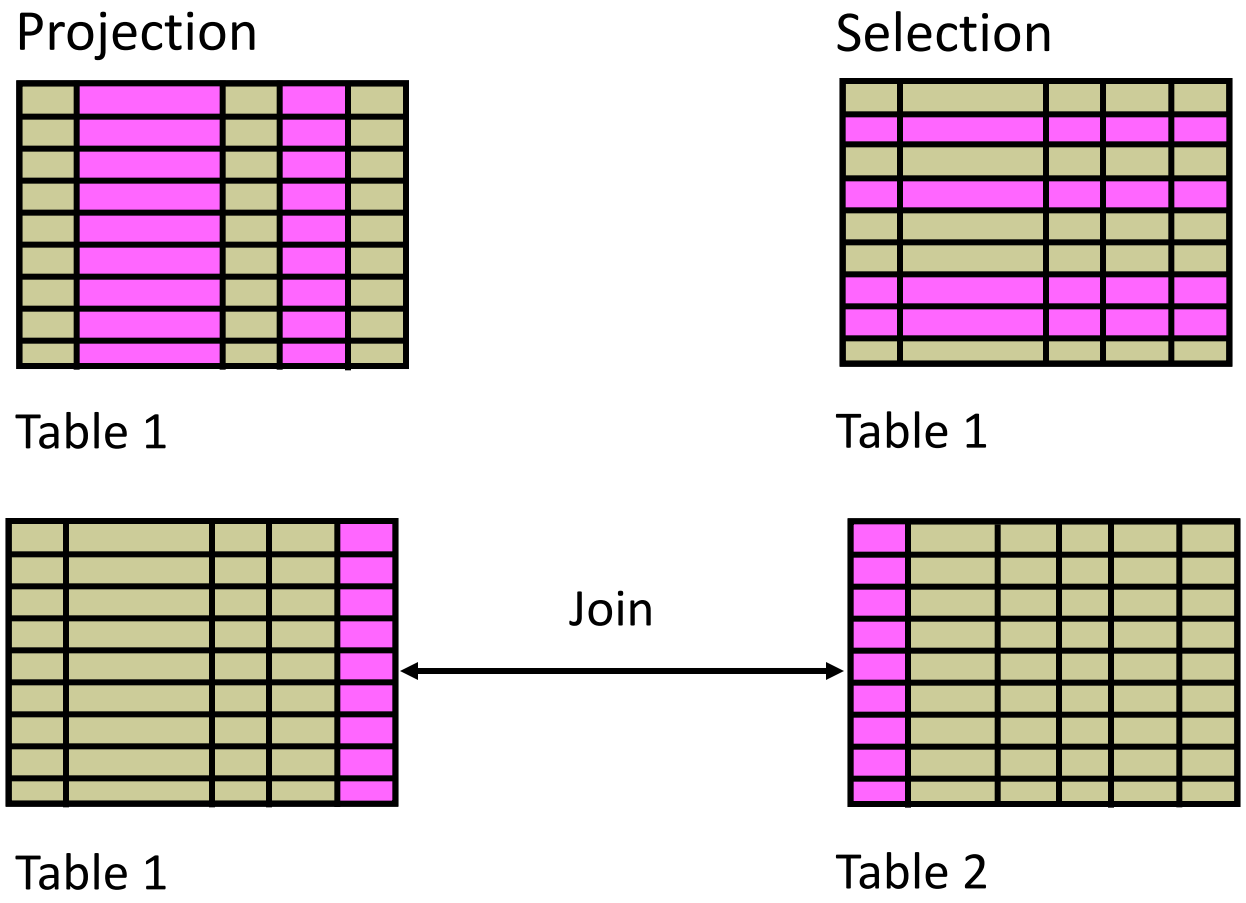
- Structured query language (SQL) is:
  - The ANSI standard language for operating relational databases
  - Efficient, easy to learn, and use
  - Functionally complete (with SQL, you can define, retrieve, and manipulate data in the tables)

Categories of SQL Statements

Data manipulation language (DML)	Data definition language (DDL)	Data control language (DCL)	Transaction control language (TCL)
SELECT	CREATE	GRANT	COMMIT
INSERT	ALTER	REVOKE	ROLLBACK
UPDATE	DROP		SAVEPOINT
DELETE	RENAME		
MERGE	TRUNCATE		
	COMMENT		

Retrieving Data Using SQL SELECT Statement

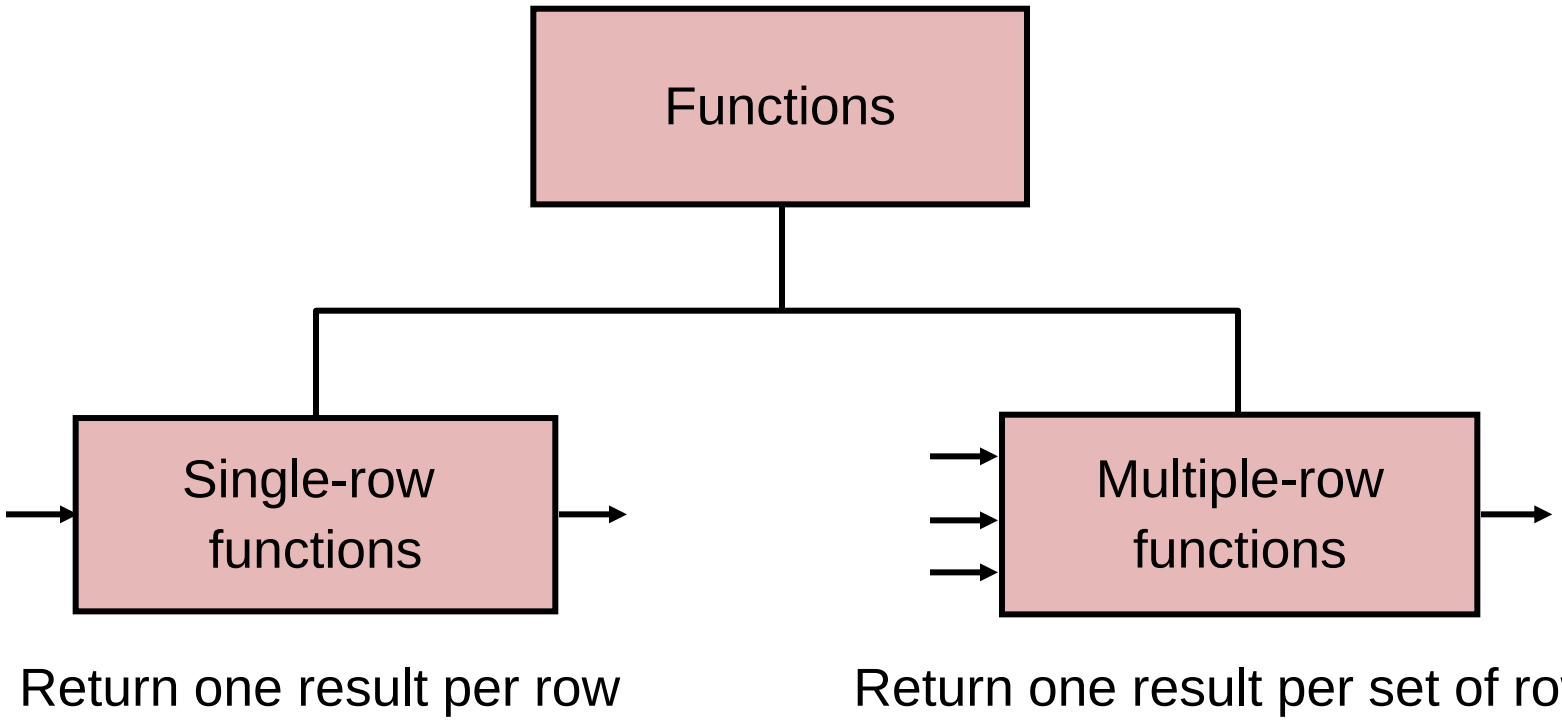
Capabilities of SQL `SELECT` Statements



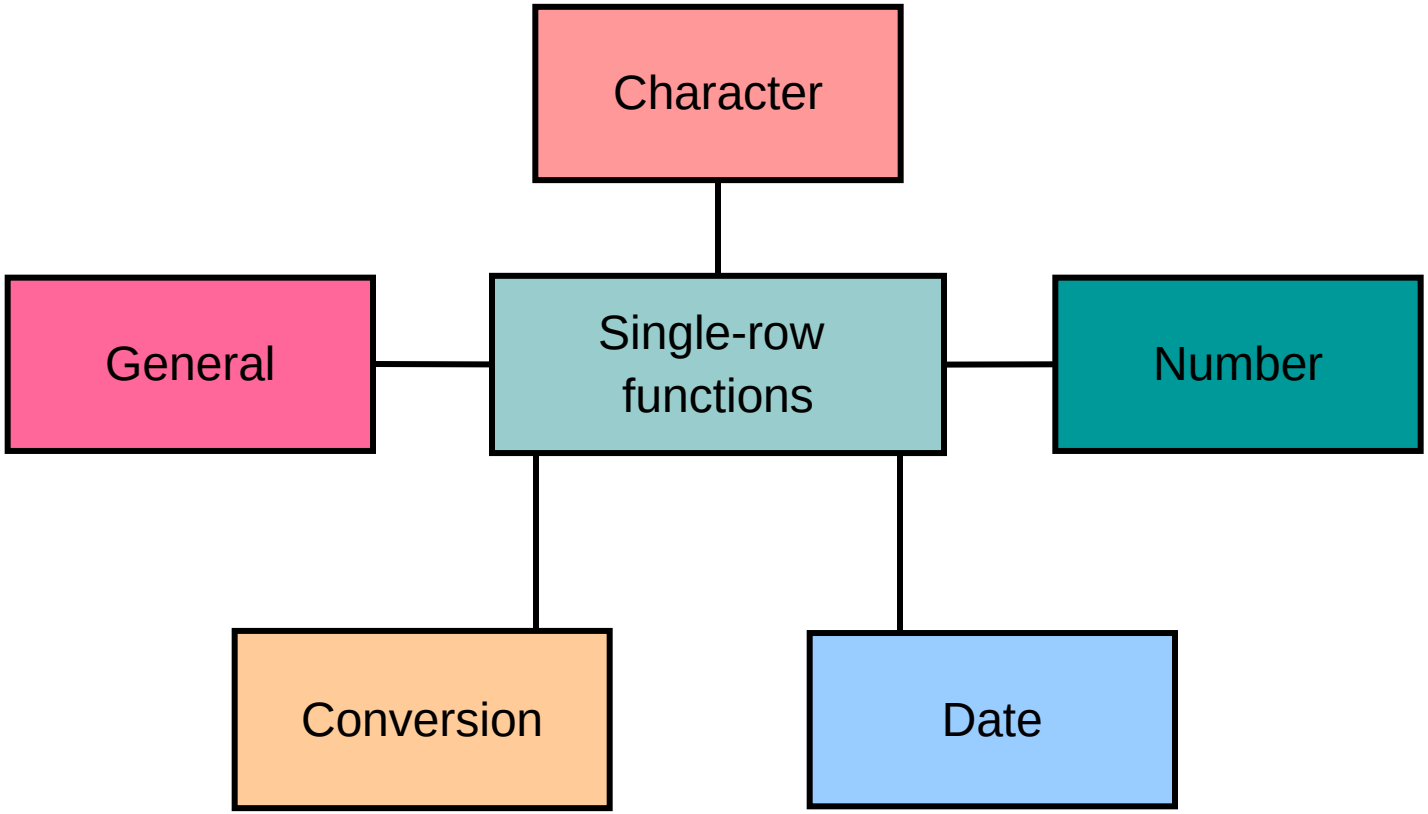
Retrieving Data Using SQL SELECT Statement

```
SELECT  {*|[DISTINCT] column|expression [alias],...}
FROM    table
[WHERE  condition(s)]
[ORDER BY {column, expr, alias} [ASC|DESC]] ;
```

Two Types of SQL Functions



Retrieving Data Using SQL SELECT Statement




```
SELECT last_name,
  UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM   employees
WHERE  department id = 60;
```

Retrieving Data Using SQL SELECT Statement

Types of Group Functions

- AVG
- COUNT
- MAX
- MIN
- STDDEV
- SUM
- VARIANCE

```
SELECT column, group_function(column)
FROM   table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

Retrieving Data Using SQL SELECT Statement

Joining Tables - Types of Joins:

- Natural join with the `NATURAL JOIN` clause
- Join with the `USING` Clause
- Join with the `ON` Clause
- `OUTER` joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross joins

```
SELECT    table1.column, table2.column
FROM      table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Retrieving Data Using SQL SELECT Statement

Using Subqueries to Solve Queries

```
SELECT    select_list FROM  table
WHERE     expr operator
                    (SELECT select_list FROM table);
```

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if at least one element exists in the result-set of the Subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if the relation is TRUE for all elements in the result set of the Subquery.

Retrieving Data Using SQL SELECT Statement

Single-Row Subqueries

```
SELECT last_name, job_id, salary FROM   employees
WHERE  job_id =
                (SELECT job_id FROM employees WHERE last_name='Taylor')
AND    salary >
                (SELECT salary FROM employees WHERE last_name='Taylor');
```

```
SELECT last_name, job_id, salary FROM   employees
WHERE  salary = (SELECT MIN(salary) FROM   employees);
```

```
SELECT   department_id, MIN(salary) FROM employees
GROUP BY department_id
HAVING   MIN(salary) >  (SELECT MIN(salary) FROM employees
                         WHERE  department_id = 50);
```

Retrieving Data Using SQL SELECT Statement

Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary FROM   employees
WHERE  salary < ANY SELECT salary FROM   employees
                    WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

```
SELECT employee_id, last_name, job_id, salary FROM   employees
WHERE  salary < ALL (SELECT salary FROM   employees
                    WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

```
SELECT employee_id,salary,last_name FROM employees M WHERE EXISTS
(SELECT employee_id FROM employees W
 WHERE (W.manager_id=M.employee_id) AND W.salary > 10000);
```

```
SELECT * FROM departments WHERE NOT EXISTS
(SELECT * FROM employees
 WHERE employees.department_id=departments.department_id);
```

Retrieving Data Using SQL SELECT Statement

Null Values in a Subquery

```
SELECT emp.last_name FROM   employees emp
WHERE  emp.employee_id NOT IN (SELECT mgr.manager_id FROM employees mgr);
```

Subquery returns no rows because one of the values returned by a subquery is Null.

- One of the values returned by the inner query is a null value and, therefore, the entire query returns no rows.
- The reason is that all conditions that compare a null value result in a null. So whenever null values are likely to be part of the results set of a subquery, do not use the NOT IN operator. The NOT IN operator is equivalent to <> ALL.

```
SELECT emp.last_name FROM   employees emp
WHERE  emp.employee_id NOT IN (SELECT mgr.manager_id
                               FROM   employees mgr
                               WHERE  manager_id IS NOT NULL);
```

Manipulating Data

Inserting New Rows

```
INSERT INTO    table [(column [, column...])]
VALUES         (value [, value...]);
```

```
INSERT INTO departments(department_id,
       department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```

```
1 rows inserted
```

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';
```

```
4 rows inserted
```

Manipulating Data

Updating Rows in a Table

```
UPDATE          table
SET             column = value [, column = value, ...]
[WHERE          condition];
```

```
UPDATE employees SET    department_id = 50 WHERE  employee_id = 113;
```

```
1 rows updated
```

```
UPDATE employees SET (job_id,salary) = (SELECT  job_id,salary
                     FROM employees WHERE   employee_id = 205)
        WHERE    employee_id    =  103;
```

```
1 rows updated
```

```
UPDATE  copy_emp
SET     department_id  =  (SELECT department_id FROM employees
                           WHERE employee_id = 100)
WHERE   job_id         =  (SELECT job_id FROM employees
                           WHERE employee_id = 200);
```

```
1 rows updated
```

Manipulating Data

Deleting Rows from a Table

```
DELETE [FROM]     table
[WHERE            condition];
```

```
 DELETE FROM departments
 WHERE  department_name = 'Finance';
1 rows deleted
```

```
DELETE FROM employees
WHERE  department_id IN
                (SELECT department_id FROM   departments
                 WHERE  department_name LIKE '%Public%');
1 rows deleted
```

Removes all rows from a table

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE copy_emp;
```

Using DDL Statements to Create and Manage Tables

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
table DEPT created.
```

```
DESCRIBE dept
```

```
DESCRIBE dept
Name          Null Type
------------- ---- ------------
DEPTNO             NUMBER(2)
DNAME              VARCHAR2(14)
LOC                VARCHAR2(13)
CREATE_DATE        DATE
```

Defining Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table and its contents if there are dependencies.
- The following constraint types are valid:
  - `NOT NULL`
  - `UNIQUE`
  - `PRIMARY KEY`
  - `FOREIGN KEY`
  - `CHECK`

Defining Constraints

```
CREATE TABLE [schema.]table
      (column datatype [DEFAULT expr]
      [column_constraint],
      ...
      [table_constraint][,...]);
```

Column-level constraint syntax:

```
column [CONSTRAINT constraint_name] constraint_type,
```

Table-level constraint syntax:

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Defining Constraints

Example of a column-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```

Example of a table-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

Creating a Table Using a Subquery

```
CREATE TABLE   dept80
  AS
    SELECT  employee_id, last_name,
            salary*12 ANNSAL,
            hire_date
    FROM    employees
    WHERE   department_id = 80;
```

```
table DEPT80 created.
```

```
DESCRIBE dept80
```

```
Name                          Null     Type
----------------------------- -------- -------------
EMPLOYEE_ID                            NUMBER(6)
LAST_NAME                     NOT NULL VARCHAR2(25)
ANNSAL                                 NUMBER
HIRE_DATE                     NOT NULL DATE
```

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a shareable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory

Sequences

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```

```
sequence DEPT_DEPTID_SEQ created.
```

Using a Sequence

– Insert a new department named “Support” in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
1 rows inserted
```

– View the current value for the DEPT_DEPTID_SEQ sequence:

```
SELECT    dept_deptid_seq.CURRVAL
FROM      dual;
```