

1

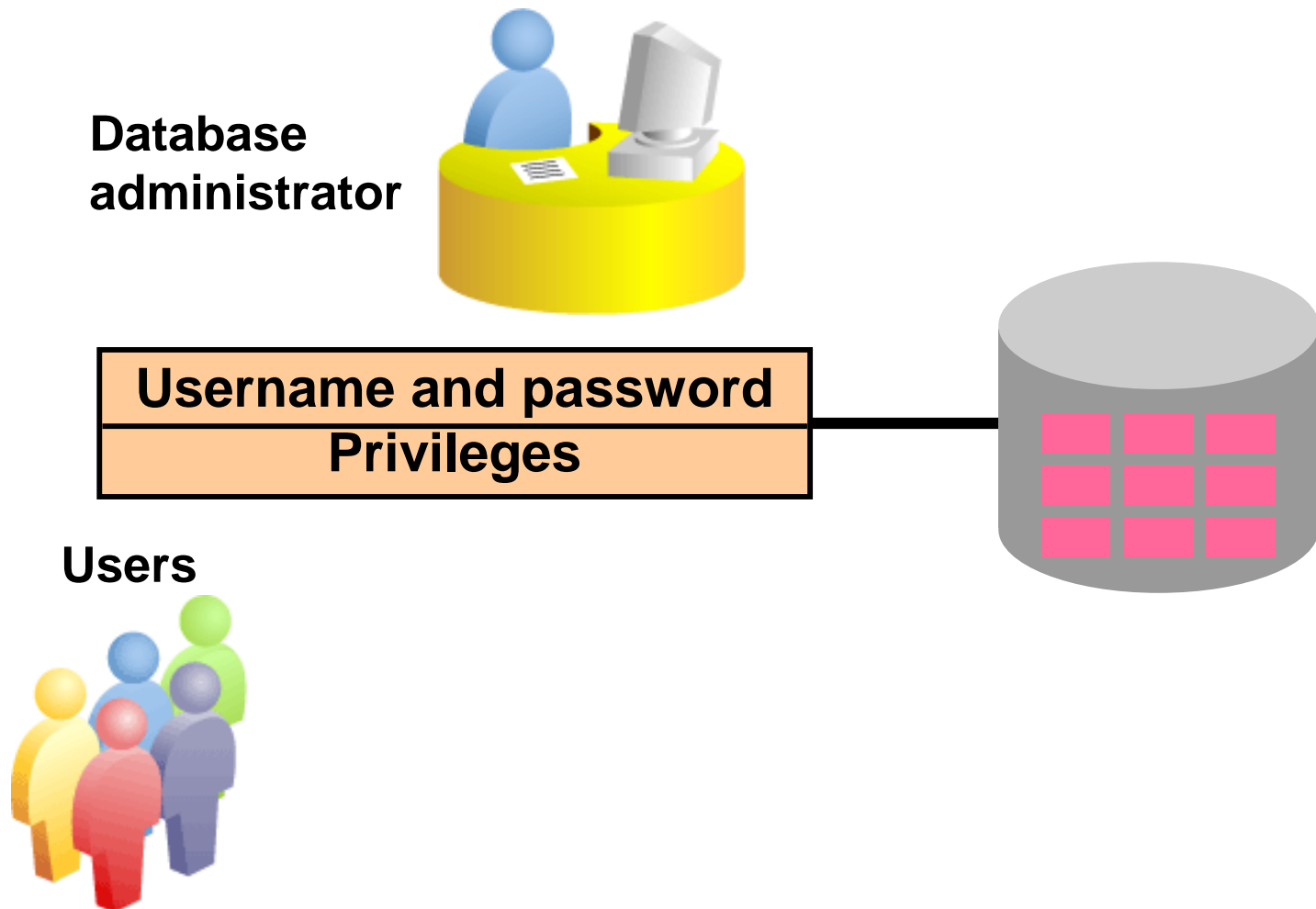
Controlling User Access

Objectives

After completing this lesson, you should be able to do the following:

- **Differentiate system privileges from object privileges**
- **Grant privileges on tables**
- **View privileges in the data dictionary**
- **Grant roles**
- **Distinguish between privileges and roles**

Controlling User Access



Privileges

- **Database security:**
 - System security
 - Data security
- **System privileges: Gaining access to the database**
- **Object privileges: Manipulating the content of the database objects**
- **Schemas: Collection of objects such as tables, views, and sequences**

System Privileges

- **More than 100 privileges are available.**
- **The database administrator has high-level system privileges for tasks such as:**
 - **Creating new users**
 - **Removing users**
 - **Removing tables**
 - **Backing up tables**

Creating Users

The DBA creates users with the **CREATE USER** statement.

```
CREATE USER user  
IDENTIFIED BY password;
```

```
CREATE USER HR  
IDENTIFIED BY HR;  
User created.
```

User System Privileges

- After a user is created, the DBA can grant specific system privileges to that user.

```
GRANT privilege [, privilege...]  
TO user [, user| role, PUBLIC...];
```

- An application developer, for example, may have the following system privileges:
 - CREATE SESSION
 - CREATE TABLE
 - CREATE SEQUENCE
 - CREATE VIEW
 - CREATE PROCEDURE

Granting System Privileges

The DBA can grant specific system privileges to a user.

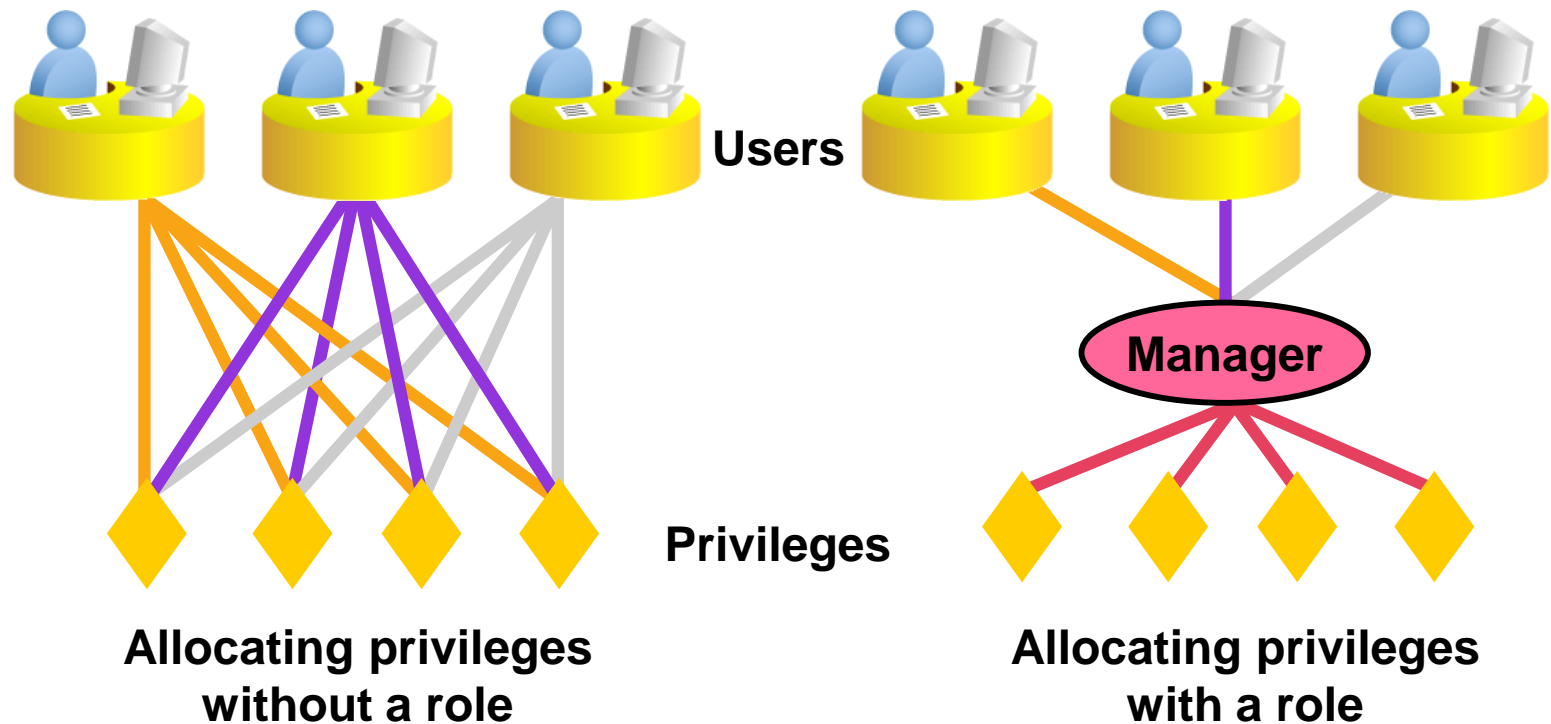
```
GRANT  create session, create table,  
       create sequence, create view  
TO      hr;  
Grant succeeded.
```


Passing On Your Privileges

- Give a user authority to pass along privileges.

```
GRANT  create session, create table,  
       create sequence, create view  
TO      hr  
WITH    ADMIN OPTION;  
Grant succeeded.
```

What Is a Role?



Creating and Granting Privileges to a Role

- **Create a role**

```
CREATE ROLE manager;  
Role created.
```

- **Grant privileges to a role**

```
GRANT create table, create view  
TO manager;  
Grant succeeded.
```

- **Grant a role to users**

```
GRANT manager TO DE HAAN, KOCHHAR;  
Grant succeeded.
```

Predefined Roles

CONNECT	CREATE SESSION
RESOURCE	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
SCHEDULER_ ADMIN	CREATE ANY JOB, CREATE EXTERNAL JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
DBA	Most system privileges, several other roles. Do not grant to nonadministrators.
SELECT_ CATALOG_ ROLE	No system privileges, but HS_ADMIN_ROLE and over 1,700 object privileges on the data dictionary

Changing Your Password

- The DBA creates your user account and initializes your password.
- You can change your password by using the **ALTER USER** statement.

```
ALTER USER HR  
IDENTIFIED BY employ;  
User altered.
```

Object Privileges

Object Privilege	Table	View	Sequence	Procedure
ALTER	√		√	
DELETE	√	√		
EXECUTE				√
INDEX	√			
INSERT	√	√		
REFERENCES	√			
SELECT	√	√	√	
UPDATE	√	√		

Object Privileges

- Object privileges vary from object to object.
- An owner has all the privileges on the object.
- An owner can give specific privileges on that owner's object.

```
GRANT      object_priv [(columns)]  
ON         object  
TO         {user|role|PUBLIC}  
[WITH GRANT OPTION];
```

Granting Object Privileges

- Grant query privileges on the **EMPLOYEES** table.

```
GRANT  select
ON      employees
TO      sue, rich;
Grant succeeded.
```

- Grant privileges to update specific columns to users and roles.

```
GRANT  update (department_name, location_id)
ON      departments
TO      scott, manager;
Grant succeeded.
```


Passing On Your Privileges

- Give a user authority to pass along privileges.

```
GRANT  select, insert
ON     departments
TO     scott
WITH   GRANT OPTION;
Grant succeeded.
```

- Allow all users on the system to query data from Alice's DEPARTMENTS table.

```
GRANT  select
ON     alice.departments
TO     PUBLIC;
Grant succeeded.
```

Confirming Privileges Granted

Data Dictionary View	Description
ROLE_SYS_PRIVS	System privileges granted to roles
ROLE_TAB_PRIVS	Table privileges granted to roles
USER_ROLE_PRIVS	Roles accessible by the user
USER_TAB_PRIVS_MADE	Object privileges granted on the user's objects
USER_TAB_PRIVS_RECD	Object privileges granted to the user
USER_COL_PRIVS_MADE	Object privileges granted on the columns of the user's objects
USER_COL_PRIVS_RECD	Object privileges granted to the user on specific columns
USER_SYS_PRIVS	System privileges granted to the user

Revoking Object Privileges

- You use the **REVOKE** statement to revoke privileges granted to other users.
- Privileges granted to others through the **WITH GRANT OPTION** clause are also revoked.

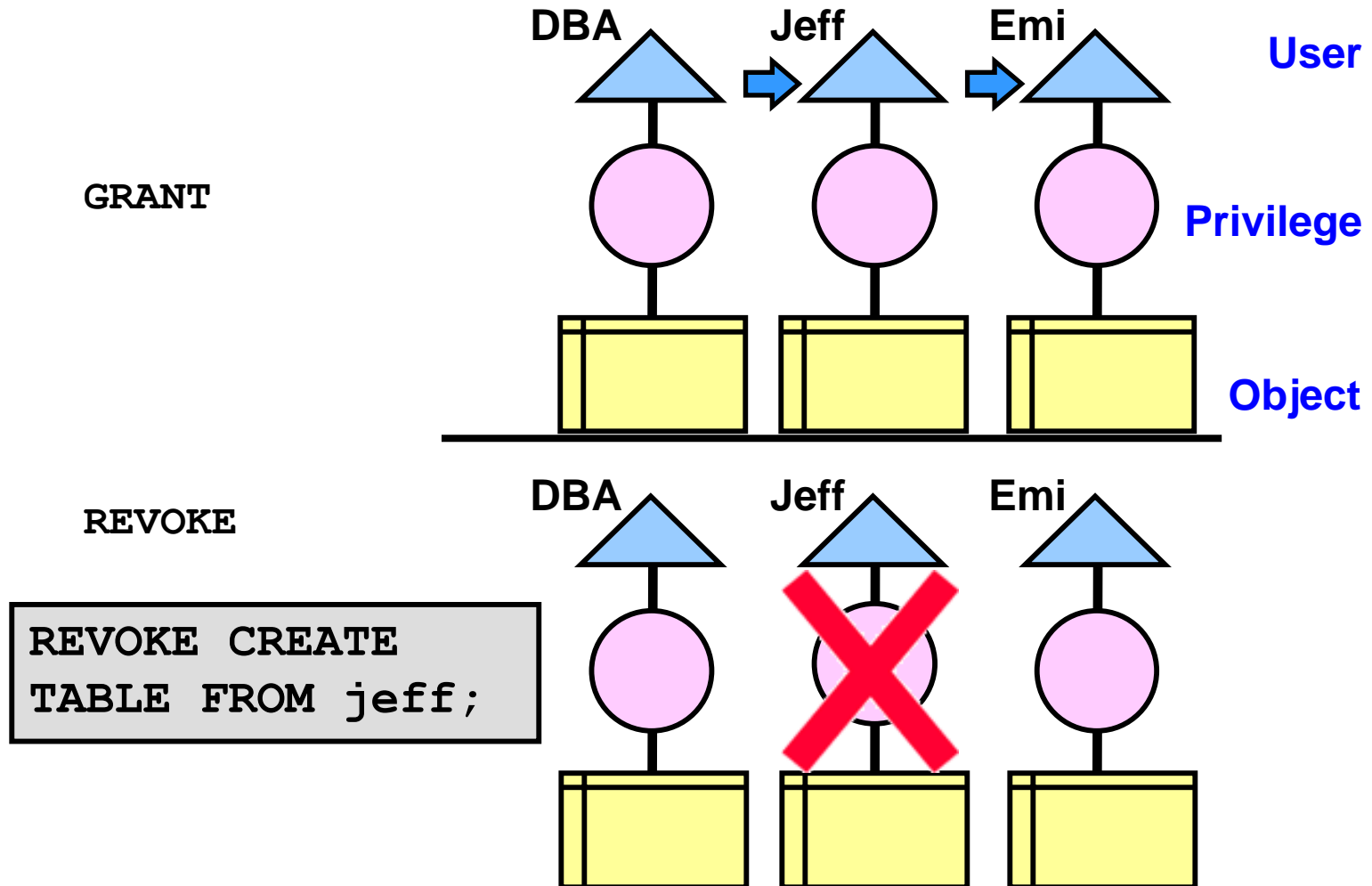
```
REVOKE {privilege [, privilege...]|ALL}  
ON      object  
FROM    {user[, user...]|role|PUBLIC}  
[CASCADE CONSTRAINTS];
```

Revoking Object Privileges

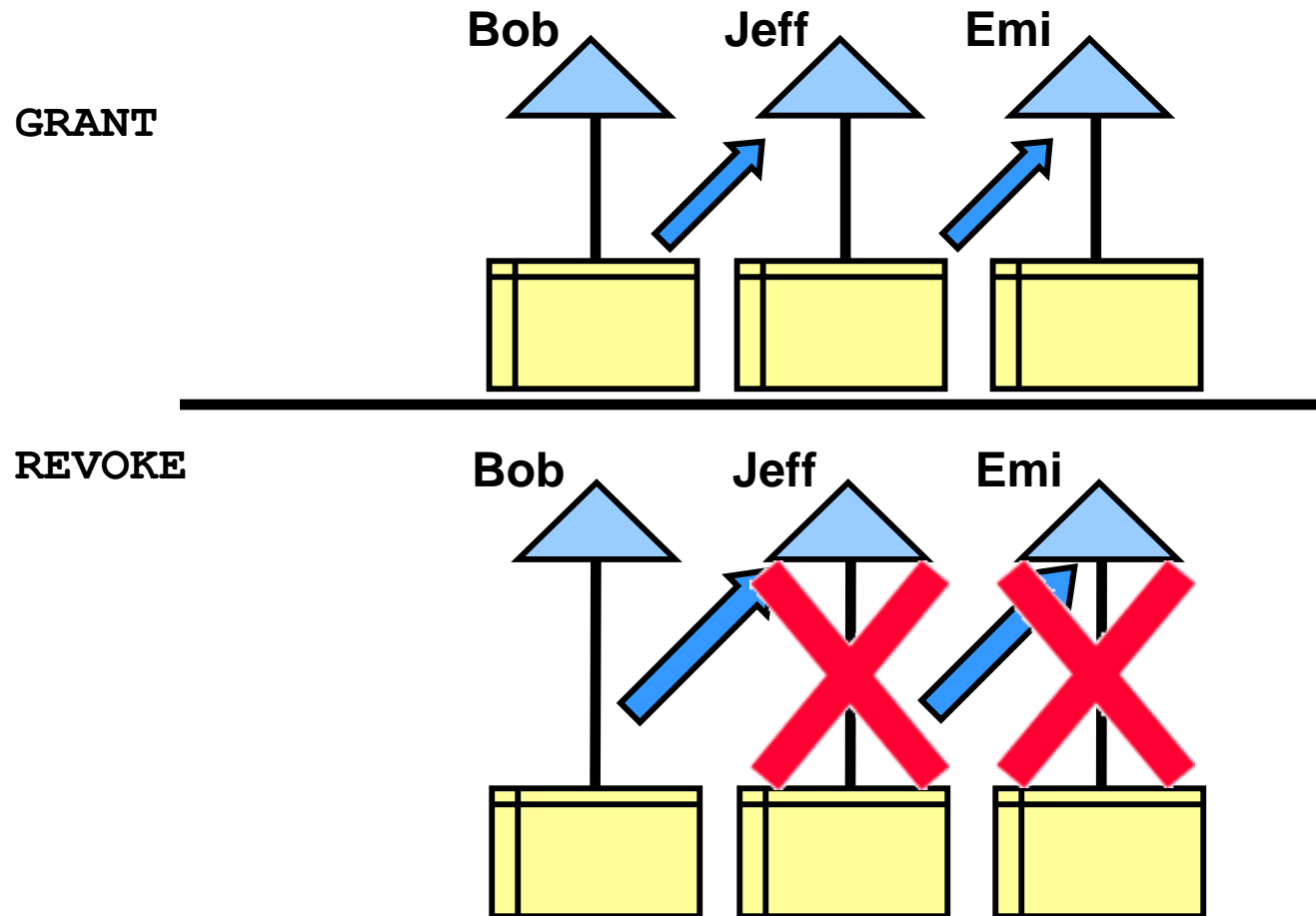
As user Alice, revoke the SELECT and INSERT privileges given to user Scott on the DEPARTMENTS table.

```
REVOKE  select, insert
ON      departments
FROM    scott;
Revoke succeeded.
```

Revoking System Privileges with ADMIN OPTION



Revoking Object Privileges with GRANT OPTION



Secure Roles

- Roles may be protected through authentication.

```
CREATE ROLE secure_application_role  
IDENTIFIED BY <password>;
```

- Roles may also be secured programmatically.

```
CREATE ROLE secure_application_role  
IDENTIFIED USING <security_procedure_name>;
```

- Creating a role.

```
CREATE ROLE test_role;  
Role created.
```

- Creating a role protected by authentication.

```
CREATE ROLE pwd_role  
IDENTIFIED BY pwd123;  
Role created.
```

Setting default role

- A default role means that the role is always enabled for the current session at logon.
- Setting new roles to user and review

```
GRANT test_role, pwd_role TO myuser;  
Grant succeeded.  
SELECT * FROM dba_role_privs;  
...  
n row(s) selected.
```

- Setting a default role for the none

```
ALTER USER myuser DEFAULT ROLE test_role;  
User altered.  
SELECT * FROM dba_role_privs;  
n row(s) selected.
```


Setting default role

- A default role means that the role is always enabled for the current session at logon.
- Setting new roles to user

```
GRANT test_role, pwd_role  
TO myuser;  
Grant succeeded.
```

- Review in metadata

```
SELECT * FROM dba_role_privs;  
...  
n row(s) selected.
```

NOTE: user's every role are default

Setting default role

- Setting no default role to user and review

```
ALTER USER myuser DEFAULT ROLE none;  
User altered.  
SELECT * FROM    dba_role_privs;  
...  
n row(s) selected.
```

- Setting default role to user and review

```
ALTER USER myuser DEFAULT ROLE test_role;  
User altered.  
SELECT * FROM    dba_role_privs;  
...  
n row(s) selected.
```

Setting default role ALL ... EXCEPT

- Setting no default role to user

```
ALTER USER myuser  
DEFAULT ROLE ALL EXCEPT pwd_role;  
User altered.
```

- Reviewing

```
SELECT *  
FROM   dba_role_privs  
ORDER BY 1;  
...  
n row(s) selected.
```

Secure Roles

- Roles may be nondefault.

```
SET ROLE role;
```

- Roles may be nondefault.

```
SET ROLE role_name IDENTIFIED BY password;
```

- Setting role in a session.

```
CONNECT myuser/mypassword  
Connected.  
SELECT * FROM session_roles;  
1 row selected.
```

```
SET ROLE pwd_role IDENTIFIED BY pwd123;  
Role set.
```

```
SELECT * FROM session_roles;  
1 row selected.
```

Dropping users

- Use the **DROP USER** statement to remove a database user and optionally remove the user's objects

- **Syntax**

```
DROP USER user [ CASCADE ] ;
```

- **Drop the user and its objects**

```
DROP USER myuser CASCADE;  
User dropped.
```

Dropping roles

- Once a role has been created in Oracle, you might at some point need to drop the role.

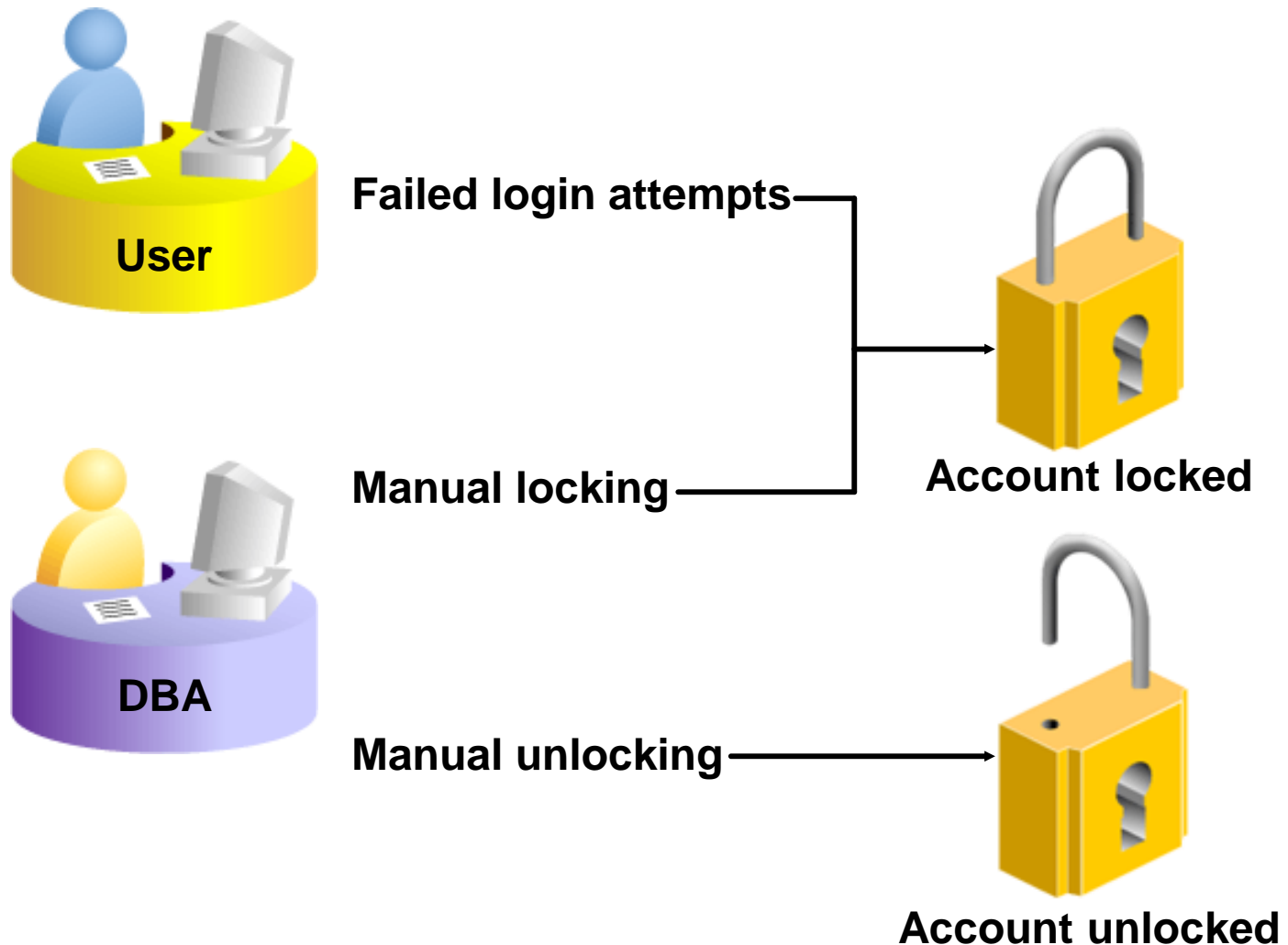
- Syntax

```
DROP ROLE role_name;
```

- Drop the user and its objects

```
DROP ROLE test_role;  
Role dropped.  
DROP ROLE pwd_role;  
Role dropped.
```

Locking and Unlocking Accounts



Locking and Unlocking Accounts

- To temporarily deny access to the database for a particular user, you can lock the user account. You can unlock the user account when you want to allow database access again for that user.
- Syntax

```
ALTER USER username ACCOUNT LOCK;  
ALTER USER username ACCOUNT UNLOCK;
```

- Drop the user and its objects

```
ALTER USER hr ACCOUNT UNLOCK;  
User altered.
```


Summary

In this lesson, you should have learned about statements that control access to the database and database objects.

Statement	Action
CREATE USER	Creates a user (usually performed by a DBA)
GRANT	Gives other users privileges to access the objects
CREATE ROLE	Creates a collection of privileges (usually performed by a DBA)
ALTER USER	Changes a user's password and others
REVOKE	Removes privileges on an object from users
SET ROLE	Set roles in a session
DROP ROLE	Drop a role from the database
DROP USER	Drop a user from the database

Practice 1: Overview

This practice covers the following topics:

- **Granting other users privileges to your table**
- **Modifying another user's table through the privileges granted to you**
- **Creating a synonym**
- **Querying the data dictionary views related to privileges**

1

Using DDL Statements to Create and Manage Tables

Objectives

After completing this lesson, you should be able to do the following:

- **Categorize the main database objects**
- **Review the table structure**
- **List the data types that are available for columns**
- **Create a simple table**
- **Understand how constraints are created at the time of table creation**
- **Describe how schema objects work**

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Naming Rules

Table names and column names:

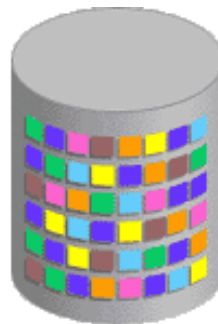
- **Must begin with a letter**
- **Must be 1–30 characters long**
- **Must contain only A–Z, a–z, 0–9, _, \$, and #**
- **Must not duplicate the name of another object owned by the same user**
- **Must not be an Oracle server reserved word**

CREATE TABLE Statement

- You must have:
 - CREATE TABLE privilege
 - A storage area

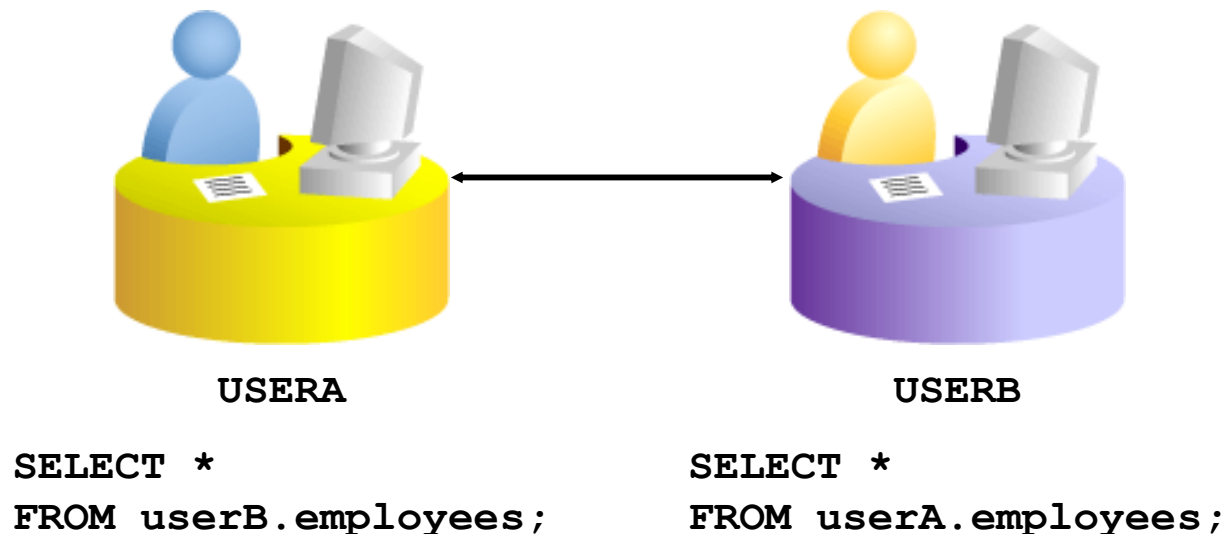
```
CREATE TABLE [schema.]table  
      (column datatype [DEFAULT expr][, ...]);
```

- You specify:
 - Table name
 - Column name, column data type, and column size



Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.



DEFAULT Option

- Specify a default value for a column during an insert.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates  
  (id          NUMBER(8),  
   hire_date DATE DEFAULT SYSDATE);
```

Table created.

Creating Tables

- **Create the table.**

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

Table created.

- **Confirm table creation.**

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

Data Types

Data Type	Description
VARCHAR2 (<i>size</i>)	Variable-length character data
CHAR (<i>size</i>)	Fixed-length character data
NUMBER (<i>p</i> , <i>s</i>)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Datetime Data Types

You can use several datetime data types:

Data Type	Description
TIMESTAMP	Date with fractional seconds
INTERVAL YEAR TO MONTH	Stored as an interval of years and months
INTERVAL DAY TO SECOND	Stored as an interval of days, hours, minutes, and seconds



Datetime Data Types

- The **TIMESTAMP** data type is an extension of the **DATE** data type.
- It stores the year, month, and day of the **DATE** data type plus hour, minute, and second values as well as the fractional second value.
- You can optionally specify the time zone.

```
TIMESTAMP[ (fractional_seconds_precision) ]
```

```
TIMESTAMP[ (fractional_seconds_precision) ]  
WITH TIME ZONE
```

```
TIMESTAMP[ (fractional_seconds_precision) ]  
WITH LOCAL TIME ZONE
```

Datetime Data Types

- The **INTERVAL YEAR TO MONTH** data type stores a period of time using the **YEAR** and **MONTH** datetime fields:

```
INTERVAL YEAR [(year_precision)] TO MONTH
```

- The **INTERVAL DAY TO SECOND** data type stores a period of time in terms of days, hours, minutes, and seconds:

```
INTERVAL DAY [(day_precision)]  
            TO SECOND [(fractional_seconds_precision)]
```

Including Constraints

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



Constraint Guidelines

- You can name a constraint, or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
 - At the same time as the table is created
 - After the table has been created
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

Defining Constraints

- **Syntax:**

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

- **Column-level constraint:**

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

- **Table-level constraint:**

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

Defining Constraints

- **Column-level constraint:**

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

1

- **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));
```

2

NOT NULL Constraint

Ensures that null values are not permitted for the column:

EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...
20 rows selected.

↑
NOT NULL constraint
(No row can contain
a null value for
this column.)

↑
NOT NULL
constraint

↑
Absence of NOT NULL
constraint
(Any row can contain
a null value for this
column.)

UNIQUE Constraint

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL
100	King	SKING
101	Kochhar	NKOCHHAR
102	De Haan	LDEHAAN
103	Hunold	AHUNOLD
104	Ernst	BERNST

...



INSERT INTO

208	Smith	JSMITH
209	Smith	JSMITH

← Allowed
← Not allowed:
already exists

UNIQUE constraint

UNIQUE Constraint


Defined at either the table level or the column level:

```
CREATE TABLE employees (  
    employee_id      NUMBER(6) ,  
    last_name        VARCHAR2(25) NOT NULL ,  
    email            VARCHAR2(25) ,  
    salary           NUMBER(8,2) ,  
    commission_pct   NUMBER(2,2) ,  
    hire_date        DATE NOT NULL ,  
    ...  
    CONSTRAINT emp_email_uk UNIQUE(email));
```

PRIMARY KEY Constraint

DEPARTMENTS

PRIMARY KEY



DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

Not allowed
(null value)

 INSERT INTO



	Public Accounting		1400
50	Finance	124	1500

Not allowed
(50 already exists)

FOREIGN KEY Constraint

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

**PRIMARY
KEY** →

...

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

← **FOREIGN
KEY**

...



INSERT INTO

200	Ford	9
201	Ford	60

**Not allowed
(9 does not
exist)**

← **Allowed**

FOREIGN KEY Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(  
    employee_id      NUMBER(6),  
    last_name        VARCHAR2(25) NOT NULL,  
    email            VARCHAR2(25),  
    salary           NUMBER(8,2),  
    commission_pct   NUMBER(2,2),  
    hire_date        DATE NOT NULL,  
    ...  
    department_id    NUMBER(4),  
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)  
        REFERENCES departments(department_id),  
    CONSTRAINT emp_email_uk UNIQUE(email));
```


FOREIGN KEY Constraint: Keywords

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES:** Identifies the table and column in the parent table
- **ON DELETE CASCADE:** Deletes the dependent rows in the child table when a row in the parent table is deleted
- **ON DELETE SET NULL:** Converts dependent foreign key values to null

CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
 - References to CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns
 - Calls to SYSDATE, UID, USER, and USERENV functions
 - Queries that refer to other values in other rows

```
..., salary  NUMBER(2)  
CONSTRAINT emp_salary_min  
CHECK (salary > 0),...
```

CREATE TABLE: Example

```
CREATE TABLE employees
( employee_id      NUMBER(6)
  CONSTRAINT emp_employee_id PRIMARY KEY
, first_name       VARCHAR2(20)
, last_name        VARCHAR2(25)
  CONSTRAINT emp_last_name_nn NOT NULL
, email            VARCHAR2(25)
  CONSTRAINT emp_email_nn     NOT NULL
  CONSTRAINT emp_email_uk     UNIQUE
, phone_number     VARCHAR2(20)
, hire_date        DATE
  CONSTRAINT emp_hire_date_nn NOT NULL
, job_id           VARCHAR2(10)
  CONSTRAINT emp_job_nn       NOT NULL
, salary           NUMBER(8,2)
  CONSTRAINT emp_salary_ck    CHECK (salary>0)
, commission_pct   NUMBER(2,2)
, manager_id       NUMBER(6)
, department_id    NUMBER(4)
  CONSTRAINT emp_dept_fk      REFERENCES
    departments (department_id));
```

Violating Constraints

```
UPDATE employees  
SET    department_id = 55  
WHERE  department_id = 110;
```

```
UPDATE employees  
*  
ERROR at line 1:  
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)  
violated - parent key not found
```

Department 55 does not exist.

Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE      department_id = 60;
```

```
DELETE FROM departments
      *
ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)
violated - child record found
```

Creating a Table by Using a Subquery

- Create a table and insert rows by combining the **CREATE TABLE** statement and the **AS *subquery*** option.

```
CREATE TABLE table  
            [ (column, column...) ]  
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

Creating a Table by Using a Subquery

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

Table created.

```
DESCRIBE dept80
```

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

Dropping a Table

- All data and structure in the table are deleted.
- Any pending transactions are committed.
- All indexes are dropped.
- All constraints are dropped.
- You *cannot* roll back the DROP TABLE statement.

```
DROP TABLE dept80;  
Table dropped.
```


DROP TABLE ... PURGE

```
DROP TABLE dept80 PURGE;
```

The FLASHBACK TABLE Statement

- **Repair tool for accidental table modifications**
 - Restores a table to an earlier point in time
 - Benefits: Ease of use, availability, fast execution
 - Performed in place
- **Syntax:**

```
FLASHBACK TABLE[schema.]table[,  
[ schema.]table ]...  
TO { TIMESTAMP | SCN } expr  
[ { ENABLE | DISABLE } TRIGGERS ];
```

The FLASHBACK TABLE Statement

```
DROP TABLE emp2;  
Table dropped
```

```
SELECT original_name, operation, droptime,  
FROM recyclebin;
```

ORIGINAL_NAME	OPERATION	DROPTIME
EMP2	DROP	2004-03-03:07:57:11

...

```
FLASHBACK TABLE emp2 TO BEFORE DROP;  
Flashback complete
```

Summary

In this lesson, you should have learned how to use the `CREATE TABLE` statement to create a table and include constraints.

- **Categorize the main database objects**
- **Review the table structure**
- **List the data types that are available for columns**
- **Create a simple table**
- **Understand how constraints are created at the time of table creation**
- **Describe how schema objects work**

Practice 9: Overview

This practice covers the following topics:

- **Creating new tables**
- **Creating a new table by using the `CREATE TABLE AS` syntax**
- **Verifying that tables exist**
- **Dropping tables**

1

Manage Schema Objects

Objectives

After completing this lesson, you should be able to do the following:

- **Add constraints**
- **Create indexes**
- **Create and manage indexes**
- **Creating function-based indexes**
- **Drop columns and set column UNUSED**
- **Perform FLASHBACK operations**
- **Create and use external tables**

The ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

The ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns.

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
DROP         (column);
```

Adding a Column

- You use the ADD clause to add columns.

```
ALTER TABLE dept80
ADD      (job_id VARCHAR2(9)) ;
Table altered.
```

- The new column becomes the last column.

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
145	Russell	14000	01-OCT-96	
146	Partners	13500	05-JAN-97	
147	Errazuriz	12000	10-MAR-97	
148	Cambrault	11000	15-OCT-99	
149	Zlotkey	10500	29-JAN-00	

...

Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE dept80  
MODIFY      (last_name VARCHAR2(30)) ;  
Table altered.
```

- A change to the default value affects only subsequent insertions to the table.

Dropping a Column

Use the **DROP COLUMN** clause to drop columns you no longer need from the table.

```
ALTER TABLE dept80
DROP COLUMN job_id;
Table altered.
```

EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
145	Russell	14000	01-OCT-96
146	Partners	13500	05-JAN-97
147	Errazuriz	12000	10-MAR-97
148	Cambrault	11000	15-OCT-99
149	Zlotkey	10500	29-JAN-00

The SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.

```
ALTER TABLE <table_name>  
SET UNUSED(<column_name>);
```

OR

```
ALTER TABLE <table_name>  
SET UNUSED COLUMN <column_name>;
```

```
ALTER TABLE <table_name>  
DROP UNUSED COLUMNS;
```

Adding a Constraint Syntax

Use the **ALTER TABLE** statement to:

- Add or drop a constraint, but not modify its structure
- Enable or disable constraints
- Add a **NOT NULL** constraint by using the **MODIFY** clause

```
ALTER TABLE  <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>) ;
```

Adding a Constraint

Add a FOREIGN KEY constraint to the EMP2 table indicating that a manager must already exist as a valid employee in the EMP2 table.

```
ALTER TABLE emp2
modify employee_id Primary Key;
Table altered.
```

```
ALTER TABLE emp2
ADD CONSTRAINT emp_mgr_fk
FOREIGN KEY(manager_id)
REFERENCES emp2(employee_id) ;
Table altered.
```

ON DELETE CASCADE

Delete child rows when a parent key is deleted.

```
ALTER TABLE Emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments ON DELETE CASCADE);  
Table altered.
```


Deferring Constraints

Constraints can have the following attributes:

- DEFERRABLE or NOT DEFERRABLE
- INITIALLY DEFERRED or INITIALLY IMMEDIATE

```
ALTER TABLE dept2  
ADD CONSTRAINT dept2_id_pk  
PRIMARY KEY (department_id)  
DEFERRABLE INITIALLY DEFERRED
```

Deferring constraint on
creation

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE
```

Changing a specific
constraint attribute

```
ALTER SESSION  
SET CONSTRAINTS= IMMEDIATE
```

Changing all constraints for a
session

Dropping a Constraint

- Remove the manager constraint from the EMP2 table.

```
ALTER TABLE emp2  
DROP CONSTRAINT emp_mgr_fk;  
Table altered.
```

- Remove the PRIMARY KEY constraint on the DEPT2 table and drop the associated FOREIGN KEY constraint on the EMP2.DEPARTMENT_ID column.

```
ALTER TABLE dept2  
DROP PRIMARY KEY CASCADE;  
Table altered.
```

Disabling Constraints

- Execute the **DISABLE** clause of the **ALTER TABLE** statement to deactivate an integrity constraint.
- Apply the **CASCADE** option to disable dependent integrity constraints.

```
ALTER TABLE emp2  
DISABLE CONSTRAINT emp_dt_fk;  
Table altered.
```

Enabling Constraints

- **Activate an integrity constraint currently disabled in the table definition by using the `ENABLE` clause.**

```
ALTER TABLE      emp2
ENABLE CONSTRAINT emp_dt_fk;
Table altered.
```

- **A `UNIQUE` index is automatically created if you enable a `UNIQUE` key or `PRIMARY KEY` constraint.**

Cascading Constraints

- The **CASCADE CONSTRAINTS** clause is used along with the **DROP COLUMN** clause.
- The **CASCADE CONSTRAINTS** clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The **CASCADE CONSTRAINTS** clause also drops all multicolumn constraints defined on the dropped columns.

Cascading Constraints

Example:

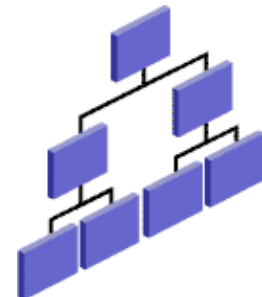
```
ALTER TABLE emp2  
DROP COLUMN employee_id CASCADE CONSTRAINTS;  
Table altered.
```

```
ALTER TABLE test1  
DROP (pk, fk, col1) CASCADE CONSTRAINTS;  
Table altered.
```

Indexes

An index:

- Is a schema object
- Is used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk I/O by using a rapid path access method to locate data quickly
- Is independent of the table that it indexes
- Is used and maintained automatically by the Oracle server



How Are Indexes Created?

- **Automatically:** A unique index is created automatically when you define a **PRIMARY KEY** or **UNIQUE** constraint in a table definition.



- **Manually:** Users can create nonunique indexes on columns to speed up access to the rows.



Overview of Indexes

Indexes are created:

- **Automatically**
 - PRIMARY KEY creation
 - UNIQUE KEY creation
- **Manually**
 - CREATE INDEX statement
 - CREATE TABLE statement

Creating an Index

- Create an index on one or more columns:

```
CREATE INDEX index  
ON table (column[, column]...);
```

- Improve the speed of query access to the **LAST_NAME** column in the **EMPLOYEES** table:

```
CREATE INDEX emp_last_name_idx  
ON employees(last_name);  
Index created.
```

Index Creation Guidelines

Create an index when:	
<input checked="" type="checkbox"/>	A column contains a wide range of values
<input checked="" type="checkbox"/>	A column contains a large number of null values
<input checked="" type="checkbox"/>	One or more columns are frequently used together in a WHERE clause or a join condition
<input checked="" type="checkbox"/>	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table
Do not create an index when:	
<input checked="" type="checkbox"/>	The columns are not often used as a condition in the query
<input checked="" type="checkbox"/>	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
<input checked="" type="checkbox"/>	The table is updated frequently
<input checked="" type="checkbox"/>	The indexed columns are referenced as part of an expression

CREATE INDEX with CREATE TABLE Statement

```
CREATE TABLE NEW_EMP  
(employee_id NUMBER(6)  
    PRIMARY KEY USING INDEX  
    (CREATE INDEX emp_id_idx ON  
    NEW_EMP(employee_id)),  
first_name  VARCHAR2(20),  
last_name   VARCHAR2(25));  
Table created.
```

```
SELECT INDEX_NAME, TABLE_NAME  
FROM    USER_INDEXES  
WHERE   TABLE_NAME = 'NEW_EMP';
```

INDEX_NAME	TABLE_NAME
EMP_ID_IDX	NEW_EMP

Function-Based Indexes

- A function-based index is based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user-defined functions.

```
CREATE INDEX upper_dept_name_idx  
ON dept2 (UPPER(department_name)) ;
```

Index created.

```
SELECT *  
FROM   dept2  
WHERE  UPPER(department_name) = 'SALES' ;
```

Removing an Index

- Remove an index from the data dictionary by using the **DROP INDEX** command:

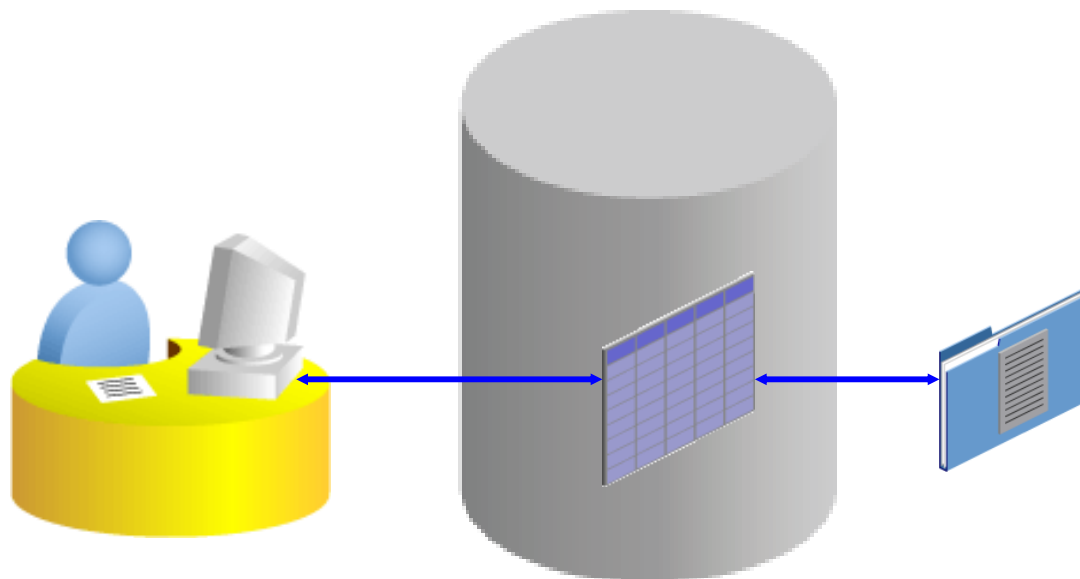
```
DROP INDEX index;
```

- Remove the **UPPER_LAST_NAME_IDX** index from the data dictionary:

```
DROP INDEX emp_last_name_idx;  
Index dropped.
```

- To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege.

External Tables



Creating a Directory for the External Table

Create a DIRECTORY object that corresponds to the directory on the file system where the external data source resides.

```
CREATE OR REPLACE DIRECTORY emp_dir  
AS '/.../emp_dir';  
  
GRANT READ ON DIRECTORY emp_dir TO hr;
```


Creating an External Table

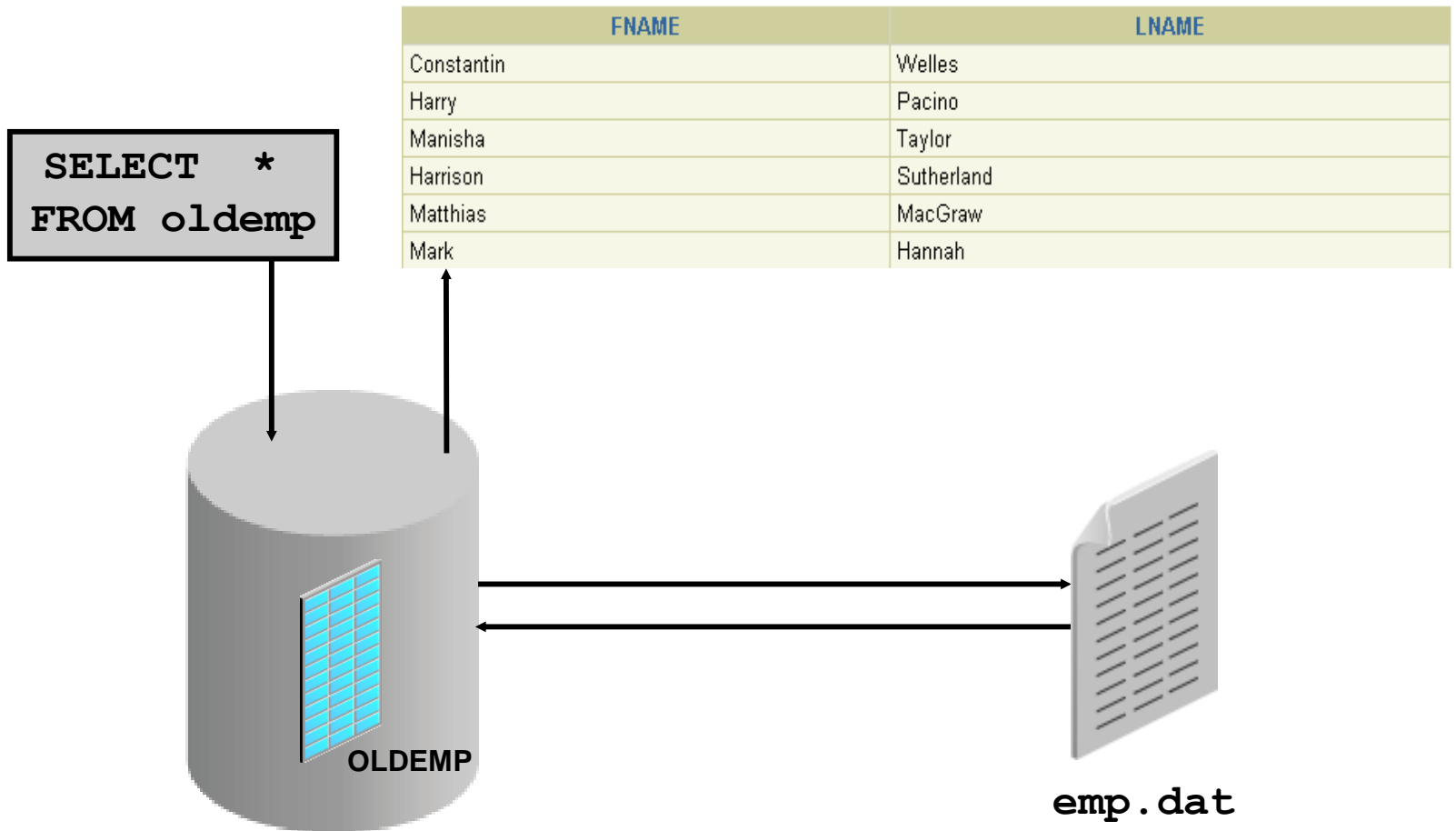
```
CREATE TABLE <table_name>
  ( <col_name> <datatype>, ... )
  ORGANIZATION EXTERNAL
    (TYPE <access_driver_type>
      DEFAULT DIRECTORY <directory_name>
      ACCESS PARAMETERS
        (... ) )
    LOCATION ('<location_specifier>') )
  REJECT LIMIT [0 | <number> | UNLIMITED];
```

Creating an External Table Using ORACLE_LOADER

```
CREATE TABLE oldemp (  
  fname char(25), lname CHAR(25))  
ORGANIZATION EXTERNAL  
  (TYPE ORACLE_LOADER  
   DEFAULT DIRECTORY emp_dir  
   ACCESS PARAMETERS  
     (RECORDS DELIMITED BY NEWLINE  
      NOBADFILE  
      NOLOGFILE  
      FIELDS TERMINATED BY ',')  
   LOCATION ('emp.dat'))  
PARALLEL 5  
REJECT LIMIT 200;
```

Table created.

Querying External Tables



Summary

In this lesson, you should have learned how to:

- **Add constraints**
- **Create indexes**
- **Create a primary key constraint using an index**
- **Create indexes using the `CREATE TABLE` statement**
- **Creating function-based indexes**
- **Drop columns and set column `UNUSED`**
- **Perform `FLASHBACK` operations**
- **Create and use external tables**

Practice 2: Overview

This practice covers the following topics:

- **Altering tables**
- **Adding columns**
- **Dropping columns**
- **Creating indexes**
- **Creating external tables**

1

Creating Other Schema Objects

Objectives

After completing this lesson, you should be able to do the following:

- **Create simple and complex views**
- **Retrieve data from views**
- **Create, maintain, and use sequences**
- **Create and maintain indexes**
- **Create private and public synonyms**

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

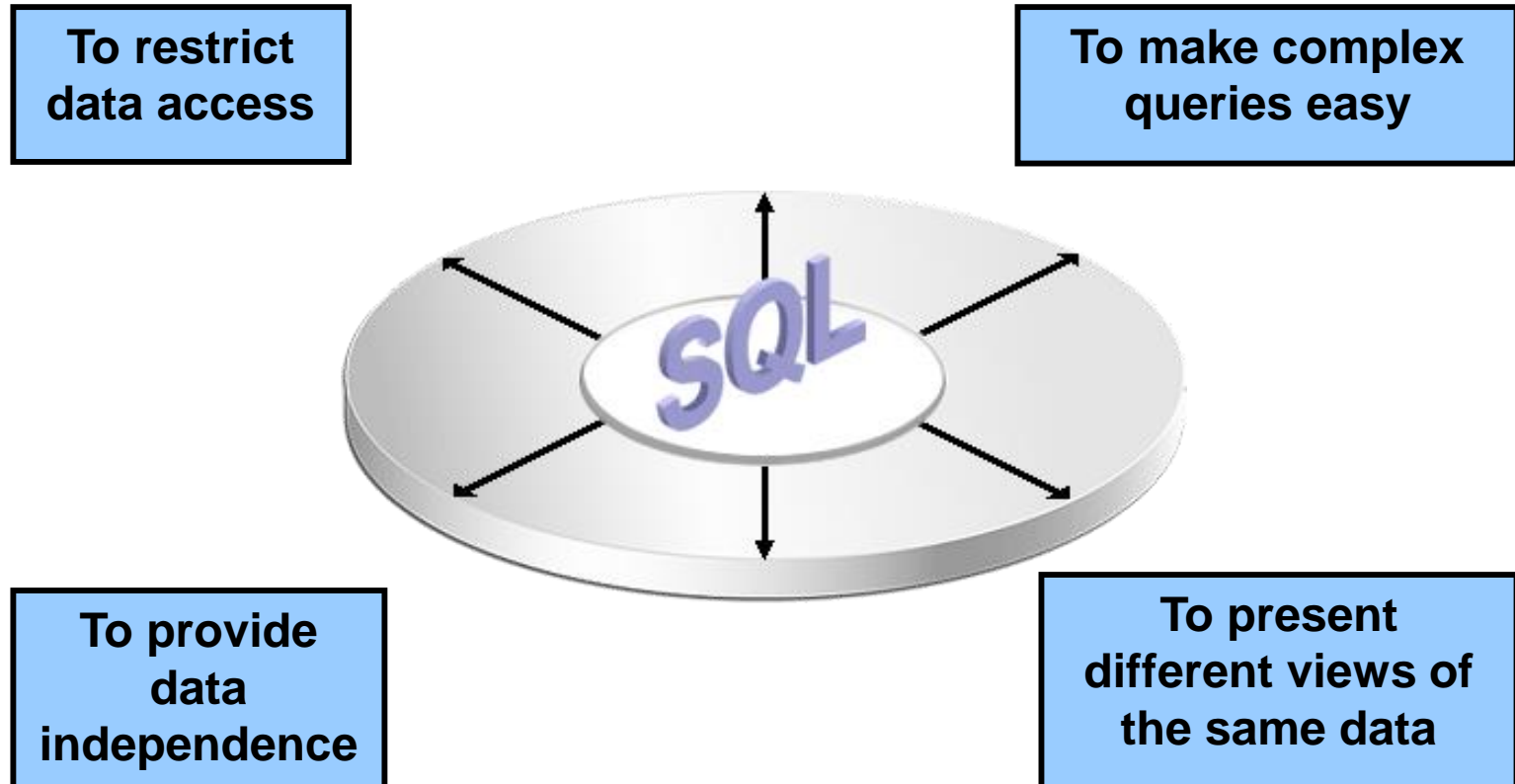
What Is a View?

EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_FRES	2400
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	1700
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	1700
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-98	IT_PROG	4200
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800
141	Trenna	Rae	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100
143	Randall	Mateo	RMATEO	650.121.2074	10-MAR-98	ST_CLERK	2600
149	Zlotkey				26-JUL-96	ST_CLERK	2500
174	Abel				24-JAN-00	SA_MAN	10500
176	Taylor				11-MAY-96	SA_REP	11000
177	Kimberly	Grant	KGRANT	611.44.1044.429203	24-MAR-98	SA_REP	8600
178	Kimberly	Grant	KGRANT	611.44.1044.429203	24-MAY-99	SA_REP	7000
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

20 rows selected.

Advantages of Views



Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

Creating a View

- You embed a subquery in the **CREATE VIEW** statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view  
  [(alias[, alias]...)]  
  AS subquery  
[WITH CHECK OPTION [CONSTRAINT constraint]]  
[WITH READ ONLY [CONSTRAINT constraint]];
```

- The subquery can contain complex **SELECT** syntax.

Creating a View

- Create the EMPVU80 view, which contains details of employees in department 80:

```
CREATE VIEW empvu80
AS SELECT employee_id, last_name, salary
FROM employees
WHERE department_id = 80;
```

View created.

- Describe the structure of the view by using the SQL*Plus DESCRIBE command:

```
DESCRIBE empvu80
```

Creating a View

- Create a view by using column aliases in the subquery:

```
CREATE VIEW    salvu50
  AS SELECT    employee_id ID_NUMBER, last_name NAME,
              salary*12 ANN_SALARY
  FROM        employees
  WHERE       department_id = 50;
View created.
```

- Select the columns from this view by the given alias names:

Retrieving Data from a View

```
SELECT *  
FROM salvu50;
```

ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

Modifying a View

- **Modify the EMPVU80 view by using a CREATE OR REPLACE VIEW clause. Add an alias for each column name:**

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' '
           || last_name, salary, department_id
  FROM      employees
  WHERE     department_id = 80;
```

View created.

- **Column aliases in the CREATE OR REPLACE VIEW clause are listed in the same order as the columns in the subquery.**



Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT      d.department_name, MIN(e.salary),
              MAX(e.salary), AVG(e.salary)
  FROM      employees e, departments d
  WHERE      e.department_id = d.department_id
  GROUP BY   d.department_name;
```

View created.

Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views. 
- You cannot remove a row if the view contains the following:
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
 - The pseudocolumn ROWNUM keyword

Rules for Performing DML Operations on a View

You cannot modify data in a view if it contains:

- **Group functions**
- **A GROUP BY clause**
- **The DISTINCT keyword**
- **The pseudocolumn ROWNUM keyword**
- **Columns defined by expressions**

Rules for Performing DML Operations on a View

You cannot add data through a view if the view includes:

- **Group functions**
- **A GROUP BY clause**
- **The DISTINCT keyword**
- **The pseudocolumn ROWNUM keyword**
- **Columns defined by expressions**
- **NOT NULL columns in the base tables that are not selected by the view**

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
   FROM        employees
   WHERE       department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

View created.

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

- You can ensure that no DML operations occur by adding the `WITH READ ONLY` option to your view definition.
- Any attempt to perform a DML operation on any row in the view results in an Oracle server error.



Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10  
    (employee_number, employee_name, job_title)  
AS SELECT      employee_id, last_name, job_id  
    FROM        employees  
    WHERE       department_id = 10  
    WITH READ ONLY ;
```

View created.

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;  
View dropped.
```


Practice 10: Overview of Part 1

This practice covers the following topics:

- **Creating a simple view**
- **Creating a complex view**
- **Creating a view with a check constraint**
- **Attempting to modify data in the view**
- **Removing views**

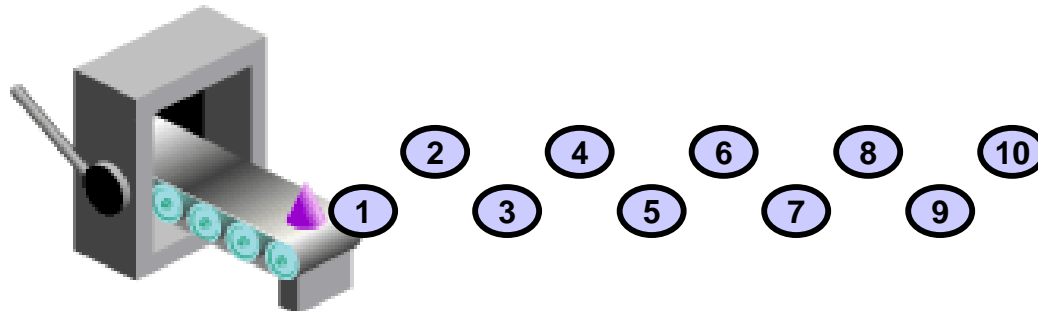
Sequences

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a sharable 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



CREATE SEQUENCE Statement: Syntax

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}];
```

Creating a Sequence

- Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.
- Do not use the CYCLE option.

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

Sequence created.

NEXTVAL and CURRVAL Pseudocolumns

- **NEXTVAL** returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- **CURRVAL** obtains the current sequence value.
- **NEXTVAL** must be issued for that sequence before **CURRVAL** contains a value.

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 row created.
```

- View the current value for the DEPT_DEPTID_SEQ sequence:

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

Caching Sequence Values

- **Caching sequence values in memory gives faster access to those values.**
- **Gaps in sequence values can occur when:**
 - **A rollback occurs**
 - **The system crashes**
 - **A sequence is used in another table**

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq  
        INCREMENT BY 20  
        MAXVALUE 999999  
        NOCACHE  
        NOCYCLE;
```

Sequence altered.

Guidelines for Modifying a Sequence

- You must be the owner or have the **ALTER** privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the **DROP** statement:

```
DROP SEQUENCE dept_deptid_seq;  
Sequence dropped.
```

Synonyms

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

Synonyms

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

- **Create an easier reference to a table that is owned by another user**
- **Shorten lengthy object names**

```
CREATE [PUBLIC] SYNONYM synonym  
FOR      object;
```

Creating and Removing Synonyms

- Create a shortened name for the DEPT_SUM_VU view:

```
CREATE SYNONYM d_sum  
FOR dept_sum_vu;  
Synonym Created.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;  
Synonym dropped.
```

Summary

In this lesson, you should have learned how to:

- **Create, use, and remove views**
- **Automatically generate sequence numbers by using a sequence generator**
- **Create indexes to improve query retrieval speed**
- **Use synonyms to provide alternative names for objects**

Practice 10: Overview of Part 2

This practice covers the following topics:

- **Creating sequences**
- **Using sequences**
- **Creating nonunique indexes**
- **Creating synonyms**