Database Programming with SQL  
14-1: Intro to Constraints; NOT NULL and UNIQUE Constraints

* **UNIQUE Constraint -** Every value in a column or set of columns (a composite key) must be unique
* **NOT NULL Constraint -** For every row entered into the table, there must be a value for that column
* **PRIMARY KEY -** Constraint ensures that the column contains no null values and uniquely identifies each row of the table
* **CHECK Constraint -** Specifies a condition for a column that must be true for each row of data
* **FOREIGN KEY -** Identifies that table and column in the parent table
* **UNIQUE Constraint -** An integrity constraint that requires every value in a column or set of columns be unique
* **FOREIGN KEY -** Designates a column (child table) that establishes a relationship between a primary key in the same table and a different table (parent table)
* **Table-level Constraint -** References one or more columns and is defined separately from the definitions of the columns in the table
* **Constraint -** Database rule.
* **Column-level Constraint -** Database rule that references a single column

Global Fast Foods has been very successful this past year and has opened several new stores. They need to add a table to their database to store information about each of their store’s locations. The owners want to make sure that all entries have an identification number, date opened, address, and city and that no other entry in the table can have the same email address. Based on this information, answer the following questions about the global\_locations table. Use the table for your answers.

Global Fast Foods global\_locations Table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| NAME | TYPE | LENGTH | PRECISION | SCALE | NULLABLE | DEFAULT |
| ID | **NUMBER** |  |  |  | **NOT NULL** |  |
| name | **VARCHAR2** |  |  |  | **NOT NULL** |  |
| date\_opened | **DATE** |  |  |  | **NOT NULL** |  |
| address | **VARCHAR2** |  |  |  | **NOT NULL** |  |
| city | **VARCHAR2** |  |  |  | **NOT NULL** |  |
| zip/postal code | **VARCHAR2** |  |  |  | **NULL** |  |
| phone | **VARCHAR2** |  |  |  | **NULL** |  |
| email | **VARCHAR2** |  |  |  | **UNIQUE** |  |
| manager\_id | **NUMBER** |  |  |  | **NULL** |  |
| Emergency contact | **VARCHAR2** |  |  |  | **NULL** |  |

1. What is a “constraint” as it relates to data integrity?

**A constraint is a rule enforced on a database table's column(s) to maintain data integrity. Constraints ensure that the data adheres to specific rules, such as preventing null values (NOT NULL), ensuring unique values (UNIQUE), enforcing valid foreign key references (FOREIGN KEY), or checking specific conditions (CHECK).**

2. What are the limitations of constraints that may be applied at the column level and at the table level?

* **Column-level constraints**: Applied directly within a column definition and can only reference that column. They are used for rules specific to one column, such as NOT NULL or UNIQUE.
* **Table-level constraints**: Defined separately after all columns in the CREATE TABLE statement and can reference multiple columns (e.g., composite UNIQUE or FOREIGN KEY constraints). These constraints are suitable for rules that involve relationships between columns.

3. Why is it important to give meaningful names to constraints?

**Meaningful constraint names improve database readability and maintainability. They help developers and administrators understand the purpose of the constraint quickly, making it easier to debug issues or update database structures.**

4. Based on the information provided by the owners, choose a datatype for each column. Indicate the length, precision, and scale for each NUMBER datatype.

**^see table**  
5. Use “nullable” to indicate those columns that can have null values.

* **Columns that cannot have null values: id, name, date\_opened, address, city.**
* **Columns that can have null values: zip/postal code, phone, email, manager\_id, emergency\_contact.**

6. Write the CREATE TABLE statement for the Global Fast Foods locations table to define the constraints at the column level.

**CREATE TABLE global\_locations (**

**id NUMBER(10) CONSTRAINT pk\_global\_locations\_id PRIMARY KEY NOT NULL,**

**name VARCHAR2(50) NOT NULL,**

**date\_opened DATE NOT NULL,**

**address VARCHAR2(100) NOT NULL,**

**city VARCHAR2(50) NOT NULL,**

**zip\_postal\_code VARCHAR2(10),**

**phone VARCHAR2(15),**

**email VARCHAR2(100) CONSTRAINT unq\_global\_locations\_email UNIQUE,**

**manager\_id NUMBER(10),**

**emergency\_contact VARCHAR2(100)**

**);**7. Execute the CREATE TABLE statement in Oracle Application Express.  
8. Execute a DESCRIBE command to view the Table Summary information.

**DESCRIBE global\_locations;**  
9. Rewrite the CREATE TABLE statement for the Global Fast Foods locations table to define the UNIQUE constraints at the table level. Do not execute this statement

**CREATE TABLE global\_locations (**

**id NUMBER(10) CONSTRAINT pk\_global\_locations\_id PRIMARY KEY NOT NULL,**

**name VARCHAR2(50) NOT NULL,**

**date\_opened DATE NOT NULL,**

**address VARCHAR2(100) NOT NULL,**

**city VARCHAR2(50) NOT NULL,**

**zip\_postal\_code VARCHAR2(10),**

**phone VARCHAR2(15),**

**email VARCHAR2(100),**

**manager\_id NUMBER(10),**

**emergency\_contact VARCHAR2(100),**

**CONSTRAINT unq\_global\_locations\_email UNIQUE (email)**

**);**

14-2: PRIMARY KEY, FOREIGN KEY, and CHECK Constraints

* **ON DELETE CASCADE -** Allows a foreign key row that is referenced to a primary key row to be deleted
* **CHECK Constraint -** Explicitly defines a condition that must be met
* **PIMARY KEY -** A column or set of columns that uniquely identifies each row in a table
* **NOT NULL Constraint -** Constraint ensures that the column contains no null values
* **ON DELETE SET NULL -** Allows a child row to remain in a table with null values when a parent record has been deleted
* **FOREIGN KEY -** Establishes a relationship between the foreign key column and a  
  primary key or unique key in the same table or a different table

1. What is the purpose of a

a. PRIMARY KEY**: Ensures that each row in a table is uniquely identifiable. It enforces uniqueness and does not allow null values.**  
b. FOREIGN KEY: **Establishes a relationship between a column in one table (child table) and a primary key or unique key in another table (parent table). Ensures referential integrity.**   
c. CHECK CONSTRAINT: **Defines a condition that must be met for each row in a table. For example, a CHECK constraint might ensure that a salary column only contains values greater than 0.**

2. Using the column information for the animals table below, name constraints where applicable at the table level, otherwise name them at the column level. Define the primary key (animal\_id). The license\_tag\_number must be unique. The admit\_date and vaccination\_date columns cannot contain null values.

animal\_id NUMBER(6) - **PRIMARY KEY**  
name VARCHAR2(25) - **None**  
license\_tag\_number NUMBER(10) - **UNIQUE**  
admit\_date DATE – **NOT NULL**  
adoption\_id NUMBER(5), - **FOREIGN KEY (adoption\_id in adoptions)**  
vaccination\_date DATE – **NOT NULL**

3. Create the animals table. Write the syntax you will use to create the table.

**CREATE TABLE animals (**

**animal\_id NUMBER(6) CONSTRAINT pk\_animals\_animal\_id PRIMARY KEY,**

**name VARCHAR2(25),**

**license\_tag\_number NUMBER(10) CONSTRAINT unq\_animals\_license\_tag UNIQUE,**

**admit\_date DATE NOT NULL,**

**adoption\_id NUMBER(5),**

**vaccination\_date DATE NOT NULL**

**);**  
4. Enter one row into the table. Execute a SELECT \* statement to verify your input. Refer to the graphic below for input.

**INSERT INTO animals (animal\_id, name, license\_tag\_number, admit\_date, adoption\_id, vaccination\_date)**

**VALUES (101, 'Spot', 35540, TO\_DATE('10-Oct-2004', 'DD-Mon-YYYY'), 205, TO\_DATE('12-Oct-2004', 'DD-Mon-YYYY'));**

**SELECT \* FROM animals;**

5. Write the syntax to create a foreign key (adoption\_id) in the animals table that has a  
corresponding primary- key reference in the adoptions table. Show both the column-level and table-level syntax. Note that because you have not actually created an adoptions table, no adoption\_id primary key exists, so the foreign key cannot be added to the animals table.

* Column-level syntax:

**ALTER TABLE animals**

**ADD CONSTRAINT fk\_animals\_adoption\_id FOREIGN KEY (adoption\_id) REFERENCES adoptions (adoption\_id);**

* Table-level syntax:

**CREATE TABLE animals (**

**animal\_id NUMBER(6) CONSTRAINT pk\_animals\_animal\_id PRIMARY KEY, name VARCHAR2(25),**

**license\_tag\_number NUMBER(10) CONSTRAINT unq\_animals\_license\_tag UNIQUE,**

**admit\_date DATE NOT NULL,**

**adoption\_id NUMBER(5),**

**vaccination\_date DATE NOT NULL,**

**CONSTRAINT fk\_animals\_adoption\_id FOREIGN KEY (adoption\_id) REFERENCES adoptions (adoption\_id)**

**);**  
6. What is the effect of setting the foreign key in the ANIMAL table as:

* **a.** ON DELETE CASCADE: When **a parent row (in adoptions table) is deleted, all child rows (in animals table) referencing it will also be automatically deleted.**
* **b.** ON DELETE SET NULL: **When a parent row is deleted, all child rows referencing it will have the foreign key column (adoption\_id) set to NULL.**

7. What are the restrictions on defining a CHECK constraint?

* **The condition must be a Boolean expression that evaluates to TRUE or FALSE.**
* **CHECK constraints cannot reference columns in other tables.**
* **The condition must not include subqueries or user-defined functions.**
* **It is possible to define multiple CHECK constraints on the same column, but each must have a unique name.**

14-3: Managing Constraints

* **DISABLE -** To deactivate an integrity constraint
* **CASCADE -** Disables dependent integrity constraints
* **ALTER TABLE -** To add, modify, or drop columns from a table
* **ENABLE -** To activate an integrity constraint currently disabled
* **DROP CONTRAINT -** Removes a constraint from a table
* **DROP COLUMN -** Allows user to delete a column from a table
* **ON DELETE/ON UPDATE -** Defines the actions the database server takes when a user attempts to delete or update a key to which existing foreign keys point

1. What are four functions that an ALTER statement can perform on constraints?

1. **Add a new constraint.**
2. **Drop an existing constraint.**
3. **Enable a disabled constraint.**
4. **Disable an enabled constraint.**

2. Since the tables are copies of the original tables, the integrity rules are not passed onto the new  
tables; only the column datatype definitions remain. You will need to add a PRIMARY KEY  
constraint to the copy\_d\_clients table. Name the primary key copy\_d\_clients\_pk . What is the  
syntax you used to create the PRIMARY KEY constraint to the copy\_d\_clients.table?

**ALTER TABLE copy\_d\_clients**

**ADD CONSTRAINT copy\_d\_clients\_pk PRIMARY KEY (client\_number);**

3. Create a FOREIGN KEY constraint in the copy\_d\_events table. Name the foreign key  
copy\_d\_events\_fk. This key references the copy\_d\_clients table client\_number column. What is  
the syntax you used to create the FOREIGN KEY constraint in the copy\_d\_events table?

**ALTER TABLE copy\_d\_events**

**ADD CONSTRAINT copy\_d\_events\_fk FOREIGN KEY (client\_number)**

**REFERENCES copy\_d\_clients (client\_number);**

4. Use a SELECT statement to verify the constraint names for each of the tables. Note that the  
tablenames must be capitalized.

**SELECT constraint\_name, constraint\_type, table\_name**

**FROM user\_constraints**

**WHERE table\_name IN ('COPY\_D\_CLIENTS', 'COPY\_D\_EVENTS');**

a. The constraint name for the primary key in the copy\_d\_clients table is **COPY\_D\_CLIENTS\_PK**.  
b. The constraint name for the foreign key in the copy\_d\_events table is **COPY\_D\_EVENTS\_FK**.

5. Drop the PRIMARY KEY constraint on the copy\_d\_clients table. Explain your results.

**ALTER TABLE copy\_d\_clients**

**DROP CONSTRAINT copy\_d\_clients\_pk;**

**Results: The primary key constraint is removed. This allows duplicate or null values in the client\_number column unless other constraints are applied**

**.**  
6. Add the following event to the copy\_d\_events table. Explain your results.

**INSERT INTO copy\_d\_events (id, name, event\_date, description, cost, venue\_id, package\_code, theme\_code, client\_number)**

**VALUES (140, 'Cline Bas Mitzvah', TO\_DATE('15-Jul-2004', 'DD-Mon-YYYY'), 'Church and Private Home', 4500, 105, 87, 77, 7125);**

7. Create an ALTER TABLE query to disable the primary key in the copy\_d\_clients table. Then add  
the values from #6 to the copy\_d\_events table. Explain your results.  
 **ALTER TABLE copy\_d\_clients**

**DISABLE CONSTRAINT copy\_d\_clients\_pk;**

**INSERT INTO copy\_d\_events (id, name, event\_date, description, cost, venue\_id, package\_code, theme\_code, client\_number)**

**VALUES (140, 'Cline Bas Mitzvah', TO\_DATE('15-Jul-2004', 'DD-Mon-YYYY'), 'Church and Private Home', 4500, 105, 87, 77, 7125);**

**Results: Disabling the primary key removes the integrity check, so the new row can be added even if client\_number does not exist in copy\_d\_clients.**

8. Repeat question 6: Insert the new values in the copy\_d\_events table. Explain your results.

**When the primary key constraint is disabled, the new values can be added without triggering a referential integrity error. However, this could lead to orphaned rows in the child table.**

9. Enable the primary-key constraint in the copy\_d\_clients table. Explain your results.  
 **ALTER TABLE copy\_d\_clients**

**ENABLE CONSTRAINT copy\_d\_clients\_pk;**

**Results: Enabling the primary key constraint reactivates the integrity check. If the table contains any duplicate or null values in client\_number, enabling the constraint will fail.**

10. If you wanted to enable the foreign-key column and reestablish the referential integrity between  
these two tables, what must be done?

1. **Ensure all client\_number values in copy\_d\_events match existing client\_number values in copy\_d\_clients.**
2. **Use the following query:**

**ALTER TABLE copy\_d\_events**

**ENABLE CONSTRAINT copy\_d\_events\_fk;**

11. Why might you want to disable and then re-enable a constraint?

* **To temporarily bypass data integrity checks for bulk data loads or updates.**
* **To fix data integrity issues or clean up invalid data in a table.**
* **To reorganize or migrate database schema while avoiding constraint violations.**

12. Query the data dictionary for some of the constraints that you have created. How does the data  
dictionary identify each constraint type?

**SELECT constraint\_name, constraint\_type, status, table\_name**

**FROM user\_constraints**

**WHERE table\_name IN ('COPY\_D\_CLIENTS', 'COPY\_D\_EVENTS');**

* **Constraint Types:**
  + **P: Primary key.**
  + **R: Foreign key (referential integrity).**
  + **U: Unique key.**
  + **C: Check constraint.**

15-1: Creating Views

* **VIEW -** A subset of data from one or more tables that is generated from a query and stored as a virtual table
* **View Name -** Name of view
* **FORCE View -** Creates a view regardless of whether or not the base tables exist
* **Simple View -** Derives data from a table, no functions or groups, performs DML operations through the view
* **NOFORCE View -** Creates the view only if the base table exists
* **CREATE VIEW -** Statement used to create a new view
* **Alias -** Specifies a name for each expression selected by the view’s query
* **Subquery -** A complete SELECT statement
* **Complex View -** Derives data from more than one table, contains functions or groups of data, and does not always allow DML operations through the view
* **CREATE OR REPLACE View -** Re-creates the view if it already exists

1. What are three uses for a view from a DBA’s perspective?

1. **Data Security: Restrict access to specific columns or rows of a table, ensuring users only see the necessary data.**
2. **Simplified Queries: Provide users with simplified access to complex queries by encapsulating them in a view.**
3. **Logical Data Organization: Organize and present data in a logical structure, derived from multiple tables, without altering the physical database schema.**

2. Create a simple view called view\_d\_songs that contains the ID, title, and artist from the DJs on  
Demand table for each “New Age” type code. In the subquery, use the alias “Song Title” for the  
title column.

**CREATE VIEW view\_d\_songs AS**

**SELECT id, title AS "Song Title", artist**

**FROM d\_songs**

**WHERE type\_code = 'New Age';**

3. SELECT \*FROM view\_d\_songs. What was returned?

**SELECT \* FROM view\_d\_songs;**

**Results: The query returns all rows where the type\_code is "New Age", displaying the id, Song Title, and artist.**  
4. REPLACE view\_d\_songs. Add type\_code to the column list. Use aliases for all columns.

**CREATE OR REPLACE VIEW view\_d\_songs AS**

**SELECT id AS "Song ID", title AS "Song Title", artist AS "Artist Name", type\_code AS "Type Code"**

**FROM d\_songs**

**WHERE type\_code = 'New Age';**

5. Jason Tsang, the disk jockey for DJs on Demand, needs a list of the past events and those  
planned for the coming months so he can make arrangements for each event’s equipment setup.  
As the company manager, you do not want him to have access to the price that clients paid for  
their events. Create a view for Jason to use that displays the name of the event, the event date,  
and the theme description. Use aliases for each column name.

**CREATE VIEW view\_d\_events AS**

**SELECT name AS "Event Name", event\_date AS "Event Date", theme\_description AS "Theme"**

**FROM d\_events;**

6. It is company policy that only upper-level management be allowed access to individual employee  
salaries. The department managers, however, need to know the minimum, maximum, and  
average salaries, grouped by department. Use the Oracle database to prepare a view that  
displays the needed information for department managers

**CREATE VIEW view\_department\_salaries AS**

**SELECT department\_id AS "Department",**

**MIN(salary) AS "Minimum Salary",**

**MAX(salary) AS "Maximum Salary",**

**AVG(salary) AS "Average Salary"**

**FROM employees**

**GROUP BY department\_id;**

15-2: DML Operations and Views

* **ROWNUM -** A pseudocolumn which assigns a sequential value starting with 1 to each of the rows returned from the subquery
* **WITH CHECK OPTION -** Specifies that INSERTS and UPDATES performed through the view can’t create rows which the view cannot select
* **WITH READ ONLY -** Ensures that no DML operations can be performed on this view

Use the DESCRIBE statement to verify that you have tables named copy\_d\_songs, copy\_d\_events,  
copy\_d\_cds, and copy\_d\_clients in your schema. If you don't, write a query to create a copy of each.  
1. Query the data dictionary USER\_UPDATABLE\_COLUMNS to make sure the columns in the base  
tables will allow UPDATE, INSERT, or DELETE. Use a SELECT statement. All table names in the  
data dictionary are stored in uppercase.

**SELECT table\_name, column\_name, updatable, insertable, deletable**

**FROM user\_updatable\_columns**

**WHERE table\_name IN ('COPY\_D\_SONGS', 'COPY\_D\_EVENTS', 'COPY\_D\_CDS', 'COPY\_D\_CLIENTS');**  
2. Use the CREATE or REPLACE option to create a view of all the columns in the copy\_d\_songs table called view\_copy\_d\_songs.

**CREATE OR REPLACE VIEW view\_copy\_d\_songs AS**

**SELECT \* FROM copy\_d\_songs;**  
3. Use view\_copy\_d\_songs to INSERT the following data into the underlying copy\_d\_songs table. Execute a SELECT \* from copy\_d\_songs to verify your DML command. See the graphic.

ID TITLE DURATION ARTIST TYPE\_CODE  
88 Mello Jello 2 The What 4

**INSERT INTO view\_copy\_d\_songs (id, title, duration, artist, type\_code)**

**VALUES (88, 'Mello Jello', 2, 'The What', 4);**

**SELECT \* FROM copy\_d\_songs;**  
4. Create a view based on the DJs on Demand COPY\_D\_CDS table. Name the view read\_copy\_d\_cds. Select all columns to be included in the view. Add a WHERE clause to restrict the year to 2000. Add the WITH READ ONLY option.

**CREATE OR REPLACE VIEW read\_copy\_d\_cds AS**

**SELECT \***

**FROM copy\_d\_cds**

**WHERE year = 2000**

**WITH READ ONLY;**  
5. Using the read\_copy\_d\_cds view, execute a DELETE FROM read\_copy\_d\_cds WHERE cd\_number = 90;

**DELETE FROM read\_copy\_d\_cds WHERE cd\_number = 90;**6. Use REPLACE to modify read\_copy\_d\_cds. Replace the READ ONLY option with WITH CHECK OPTION CONSTRAINT ck\_read\_copy\_d\_cds. Execute a SELECT \* statement to verify that the view exists.

**CREATE OR REPLACE VIEW read\_copy\_d\_cds AS**

**SELECT \***

**FROM copy\_d\_cds**

**WHERE year = 2000**

**WITH CHECK OPTION CONSTRAINT ck\_read\_copy\_d\_cds;**

**SELECT \* FROM read\_copy\_d\_cds;**

**Verification**: Execute the SELECT query to ensure the view is updated.  
7. Use the read\_copy\_d\_cds view to delete any CD of year 2000 from the underlying copy\_d\_cds.

**DELETE FROM read\_copy\_d\_cds WHERE year = 2000;**  
8. Use the read\_copy\_d\_cds view to delete cd\_number 90 from the underlying copy\_d\_cds table.

**DELETE FROM read\_copy\_d\_cds WHERE cd\_number = 90;**  
9. Use the read\_copy\_d\_cds view to delete year 2001 records.

**DELETE FROM read\_copy\_d\_cds WHERE year = 2001;**  
10. Execute a SELECT \* statement for the base table copy\_d\_cds. What rows were deleted?

**SELECT \* FROM copy\_d\_cds;**11. What are the restrictions on modifying data through a view?

1. **Modifications are restricted if the view includes:**
   * **Group functions (e.g., SUM, AVG).**
   * **DISTINCT clause.**
   * **Aggregate functions or expressions.**
   * **JOIN statements (in some cases).**
   * **WITH READ ONLY option.**
2. **The WITH CHECK OPTION ensures that DML operations do not insert or update rows that violate the view’s WHERE clause.**

12. What is Moore’s Law? Do you consider that it will continue to apply indefinitely? Support your opinion with research from the internet.

* + **Definition: Moore’s Law states that the number of transistors on a microchip doubles approximately every two years, increasing computational power while reducing cost per transistor.**
  + **Continuance: Moore’s Law faces physical and economic challenges as transistors approach atomic scale. Innovations such as quantum computing, 3D chip designs, and other materials like graphene may extend computational advancements, but the traditional form of Moore’s Law is unlikely to continue indefinitely.**

13. What is the “singularity” in terms of computing?

* **Definition: The singularity refers to a hypothetical future point when artificial intelligence (AI) surpasses human intelligence, leading to exponential technological growth that fundamentally transforms society.**
* **Significance: It is a concept discussed in the context of AI and futurism, often associated with thinkers like Ray Kurzweil. Proponents believe it will revolutionize human capabilities, while critics warn of potential risks.**

15-3: Managing Views

* **Top-N Analysis -** Asks for the N largest or smallest values in a column
* **DROP VIEW -** Removes a view
* **Inline View -** Subquery with an alias that can be used within a SQL statement

1. Create a view from the copy\_d\_songs table called view\_copy\_d\_songs that includes only the title and artist. Execute a SELECT \* statement to verify that the view exists.

**CREATE VIEW view\_copy\_d\_songs AS**

**SELECT title, artist**

**FROM copy\_d\_songs;**

**SELECT \* FROM view\_copy\_d\_songs;**  
2. Issue a DROP view\_copy\_d\_songs. Execute a SELECT \* statement to verify that the view has been deleted.

**DROP VIEW view\_copy\_d\_songs;**

**SELECT \* FROM view\_copy\_d\_songs**

3. Create a query that selects the last name and salary from the Oracle database. Rank the salaries from highest to lowest for the top three employees.

**SELECT last\_name, salary, RANK() OVER (ORDER BY salary DESC) AS rank**

**FROM employees**

**WHERE RANK() OVER (ORDER BY salary DESC) <= 3;**  
4. Construct an inline view from the Oracle database that lists the last name, salary, department ID, and maximum salary for each department. Hint: One query will need to calculate maximum salary by department ID.

**SELECT e.last\_name, e.salary, e.department\_id, d.max\_salary**

**FROM employees e**

**JOIN (**

**SELECT department\_id, MAX(salary) AS max\_salary**

**FROM employees**

**GROUP BY department\_id**

**) d**

**ON e.department\_id = d.department\_id;**  
5. Create a query that will return the staff members of Global Fast Foods ranked by salary from  
lowest to highest.

**SELECT staff\_name, salary, RANK() OVER (ORDER BY salary ASC) AS rank**

**FROM global\_fast\_foods\_staff;**  
Extension Exercises  
1. Create a new table called my\_departments and add all columns and all rows to it using a subquery from the Oracle departments table. Do a SELECT \* from my\_departments to confirm that you have all the columns and rows.

**CREATE TABLE my\_departments AS**

**SELECT \* FROM departments;**

**SELECT \* FROM my\_departments;**  
2. To view any constraints that may affect the my\_departments table, DESCRIBE my\_departments to check if any constraints were carried over from the departments table. If there are constraints on my\_departments, use an ALTER TABLE command to DISABLE all constraints on my\_departments.

**DESCRIBE my\_departments;**

**ALTER TABLE my\_departments DISABLE CONSTRAINT <constraint\_name>;**

3. Create a view called view\_my\_departments that includes: department\_id and department\_name.

**CREATE VIEW view\_my\_departments AS**

**SELECT department\_id, department\_name**

**FROM my\_departments;**  
4. Add the following data to the my\_departments table using view\_my\_departments.  
department\_id department\_name  
105 Advertising  
120 Custodial  
130 Planning

**INSERT INTO view\_my\_departments (department\_id, department\_name)**

**VALUES (105, 'Advertising'),**

**(120, 'Custodial'),**

**(130, 'Planning');**  
5. Create or enable the department\_id column as the primary key.

**ALTER TABLE my\_departments**

**ADD CONSTRAINT pk\_my\_departments PRIMARY KEY (department\_id);**  
6. Enter a new department named Human Resources into the my\_departments table using view\_my\_departments. Do not add a new department ID.

**INSERT INTO view\_my\_departments (department\_name)**

**VALUES ('Human Resources');**7. Add the Human Resources department, department ID 220, to my\_departments using view\_my\_departments.

**INSERT INTO view\_my\_departments (department\_id, department\_name)**

**VALUES (220, 'Human Resources');**  
8. Verify that the new additions to my\_departments were added using view\_my\_departments. See chart below

**SELECT \* FROM view\_my\_departments;**  
9. Modify view\_my\_departments to include location ID. Do a SELECT \* command to show what columns are present and a DESCRIBE command to view the columns and associated constraints.

**CREATE OR REPLACE VIEW view\_my\_departments AS**

**SELECT department\_id, department\_name, location\_id**

**FROM my\_departments;**

**SELECT \* FROM view\_my\_departments;**

**DESCRIBE view\_my\_departments;**10. Make location\_id a NOT NULL column in the my\_departments table.

**ALTER TABLE my\_departments**

**MODIFY location\_id NOT NULL;**  
11. Using the Oracle database, create a complex view between locations and departments with only  
the following columns: department\_name, street\_address, city, and state. Include only U.S. cities.  
Verify that the view was created using a SELECT \* statement.  
See chart below  
1700200Administration10  
LOCATION\_IDMANAGER\_IDDEPARTMENT  
\_NAME  
DEPARTMENT\_ID

**CREATE VIEW view\_us\_departments\_locations AS**

**SELECT d.department\_name, l.street\_address, l.city, l.state**

**FROM departments d**

**JOIN locations l ON d.location\_id = l.location\_id**

**WHERE l.country = 'US';**

**SELECT \* FROM view\_us\_departments\_locations;**

16-1: Working with Sequences

* **CREATE SEQUENCE -** Command that automatically generates sequential numbers
* **Sequence -** Generates a numeric value
* **NEXTVAL -** Returns the next available sequence value
* **INCREMENT BY -** Specifies the interval between sequence numbers
* **MAXVALUE -** Specifies a maximum value of 10^27 for an ascending sequence and -1 for a descending sequence (default)
* **CURRVAL -** Returns the current sequence value
* **MINVALUE -** specifies the minimum sequence value
* **CYCLE -** specifies whether the sequence continues to generate values after reaching its maximum or minimum values
* **MINVALUE -** specifies a minimum value of 1 for an ascending sequence and – (10^26) for a descending sequence (default)
* **MAXVALUE -** specifies a maximum or default value the sequence can generate
* **START WITH -** specifies the first sequence number to be generated
* **CACHE -** specifies how many values the Server pre-allocates and keeps in memory

1. Using CREATE TABLE AS subquery syntax, create a seq\_d\_songs table of all the columns in the DJs on Demand database table d\_songs. Use the SELECT \* in the subquery to make sure that you have copied all of the columns.

**CREATE TABLE seq\_d\_songs AS**

**SELECT \* FROM d\_songs;**  
2. Because you are using copies of the original tables, the only constraints that were carried over were the NOT NULL constraints. Create a sequence to be used with the primary-key column of the seq\_d\_songs table. To avoid assigning primary-key numbers to these tables that already exist, the sequence should start at 100 and have a maximum value of 1000. Have your sequence increment by 2 and have NOCACHE and NOCYCLE. Name the sequence seq\_d\_songs\_seq.

**CREATE SEQUENCE seq\_d\_songs\_seq**

**START WITH 100**

**INCREMENT BY 2**

**MAXVALUE 1000**

**NOCACHE**

**NOCYCLE;**  
3. Query the USER\_SEQUENCES data dictionary to verify the seq\_d\_songs\_seq SEQUENCE  
settings.

**SELECT sequence\_name, min\_value, max\_value, increment\_by, cache\_size, cycle\_flag**

**FROM user\_sequences**

**WHERE sequence\_name = 'SEQ\_D\_SONGS\_SEQ';**  
4. Insert two rows into the seq\_d\_songs table. Be sure to use the sequence that you created for the ID column. Add the two songs shown in the graphic.

**INSERT INTO seq\_d\_songs (id, title, duration, artist, type\_code)**

**VALUES (seq\_d\_songs\_seq.NEXTVAL, 'Island Fever', 5, 'Hawaiian Islanders', 12);**

**INSERT INTO seq\_d\_songs (id, title, duration, artist, type\_code)**

**VALUES (seq\_d\_songs\_seq.NEXTVAL, 'Castle of Dreams', 4, 'The Wanderers', 77);**  
5. Write out the syntax for seq\_d\_songs\_seq to view the current value for the sequence. Use the DUAL table. (Oracle Application Developer will not run this query.)

**SELECT seq\_d\_songs\_seq.CURRVAL**

**FROM dual;**  
6. What are three benefits of using SEQUENCEs?

1. **Automatic Number Generation**: Automatically generates unique numeric values for primary keys or other purposes.
2. **Thread Safety**: Sequences ensure thread-safe number generation, preventing duplicates in multi-user environments.
3. **Flexibility**: Can be customized with specific increments, min/max values, and cycle options.\

7. What are the advantages of caching sequence values?

1. **Improved Performance**: Reduces disk I/O by pre-allocating sequence values in memory.
2. **Efficient Resource Usage**: Minimizes access to the sequence definition in the database.
3. **Better Scalability**: Enhances performance in high-concurrency environments.

8. Name three reasons why gaps may occur in a sequence?  
ID TITLE DURATION ARTIST TYPE\_CODE  
Island Fever 5 min Hawaiian Islanders 12  
Castle of Dreams 4 min The Wanderers 77

1. **ROLLBACK**: When a transaction is rolled back, the sequence value generated during the transaction is lost.
2. **System Crashes**: Cached sequence values are lost if the system crashes.
3. **Explicit Usage**: Gaps can occur when a sequence is used outside of table inserts (e.g., in test queries or calculations).

Extension Exercise  
1. Create a table called “students”. You can decide which columns belong in that table and what datatypes these columns require. (The students may create a table with different columns; however, the important piece that must be there is the student\_id column with a numeric datatype. This column length must allow the sequence to fit, e.g. a column length of 4 with a sequence that starts with 1 and goes to 10000000 will not work after student #9999 is entered.)

**CREATE TABLE students (**

**student\_id NUMBER(10),**

**first\_name VARCHAR2(50),**

**last\_name VARCHAR2(50),**

**date\_of\_birth DATE,**

**major VARCHAR2(100)**

**);**  
2. Create a sequence called student\_id\_seq so that you can assign unique student\_id numbers for  
all students that you add to your table.

**CREATE SEQUENCE student\_id\_seq**

**START WITH 1**

**INCREMENT BY 1**

**NOCACHE**

**NOCYCLE;**  
3. Now write the code to add students to your STUDENTS table, using your sequence “database  
object.”

**INSERT INTO students (student\_id, first\_name, last\_name, date\_of\_birth, major)**

**VALUES (student\_id\_seq.NEXTVAL, 'John', 'Doe', TO\_DATE('2000-01-01', 'YYYY-MM-DD'), 'Computer Science');**

**INSERT INTO students (student\_id, first\_name, last\_name, date\_of\_birth, major)**

**VALUES (student\_id\_seq.NEXTVAL, 'Jane', 'Smith', TO\_DATE('1999-05-15', 'YYYY-MM-DD'), 'Biology');**

16-2: Indexes and Synonyms

* **USER\_INDEXES -** Confirms the existence of indexes from the USER\_INDEXES data dictionary view
* **Index -** Schema object that speeds up retrieval of rows
* **Synonym -** To refer to a table by another name to simplify access
* **Composite Index -** An index that you create on multiple columns in a table
* **Implicit Index -** The Oracle Server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE KEY constraint
* **B-Tree Index -** Stores the indexed values and uses the index based on a SELECT statement to retrieve the data
* **DROP INDEX -** Removes an index
* **Synonym -** Gives alternative names to objects

1. What is an index and what is it used for?

**An index is a schema object that improves the performance of queries by allowing faster retrieval of rows from a table. It achieves this by creating a data structure that stores column values in a sorted order, making it quicker to search and retrieve specific rows.**  
2. What is a ROWID, and how is it used?

**ROWID is a unique identifier assigned to every row in a table. It represents the physical location of the row in the database, including the block, file, and row number. ROWID is used for efficient data retrieval in queries, especially in indexed access.**  
3. When will an index be created automatically?

* A column is defined as a **PRIMARY KEY**.
* A column is defined as a **UNIQUE KEY**.
* A **UNIQUE constraint** is applied to one or more columns.

4. Create a nonunique index (foreign key) for the DJs on Demand column (cd\_number) in the D\_TRACK\_LISTINGS table. Use the Oracle Application Developer SQL Workshop Data Browser to confirm that the index was created.

**CREATE INDEX idx\_cd\_number ON d\_track\_listings (cd\_number);**

**SELECT index\_name, table\_name, uniqueness**

**FROM user\_indexes**

**WHERE table\_name = 'D\_TRACK\_LISTINGS';**  
5. Use the join statement to display the indexes and uniqueness that exist in the data dictionary for  
the DJs on Demand D\_SONGS table.

**SELECT i.index\_name, i.table\_name, i.uniqueness**

**FROM user\_indexes i**

**WHERE i.table\_name = 'D\_SONGS';**  
6. Use a SELECT statement to display the index\_name, table\_name, and uniqueness from the data dictionary USER\_INDEXES for the DJs on Demand D\_EVENTS table.

**SELECT index\_name, table\_name, uniqueness**

**FROM user\_indexes**

**WHERE table\_name = 'D\_EVENTS';**  
7. Write a query to create a synonym called dj\_tracks for the DJs on Demand d\_track\_listings table.

**CREATE SYNONYM dj\_tracks FOR d\_track\_listings;**

**SELECT synonym\_name, table\_name**

**FROM user\_synonyms**

**WHERE synonym\_name = 'DJ\_TRACKS';**  
8. Create a function-based index for the last\_name column in DJs on Demand D\_PARTNERS table  
 makes it possible not to have to capitalize the table name for searches. Write a SELECT statement that would use this index.

**CREATE INDEX idx\_lower\_last\_name ON d\_partners (LOWER(last\_name));**

**SELECT \***

**FROM d\_partners**

**WHERE LOWER(last\_name) = 'smith';**  
9. Create a synonym for the D\_TRACK\_LISTINGS table. Confirm that it has been created by querying the data dictionary.

**CREATE SYNONYM synonym\_track\_listings FOR d\_track\_listings;**

**SELECT synonym\_name, table\_name**

**FROM user\_synonyms**

**WHERE synonym\_name = 'SYNONYM\_TRACK\_LISTINGS';**  
10. Drop the synonym that you created in question 9

**DROP SYNONYM synonym\_track\_listings;**

**SELECT synonym\_name**

**FROM user\_synonyms**

**WHERE synonym\_name = 'SYNONYM\_TRACK\_LISTINGS';**

17-1: Controlling User Access  
  
1. What are system privileges concerned with?

**System privileges allow users to perform specific administrative tasks or create and manage database objects. Examples include the ability to create tables, users, or views, or to perform database-wide operations like granting privileges.**  
2. What are object privileges concerned with?

**Object privileges define what actions a user can perform on specific database objects (e.g., tables, views, sequences). Examples include SELECT, INSERT, UPDATE, DELETE, and EXECUTE.**3. What is another name for object security?

**Another name for object security is data-level security.**  
4. What commands are necessary to allow Scott access to the database with a password of tiger?

**CREATE USER scott IDENTIFIED BY tiger;**

**GRANT CREATE SESSION TO scott;**  
5. What are the commands to allow Scott to SELECT from and UPDATE the d\_clients table?

**GRANT SELECT, UPDATE ON d\_clients TO scott;**  
6. What is the command to allow everybody the ability to view the d\_songs table?

**GRANT SELECT ON d\_songs TO PUBLIC;**7. Query the data dictionary to view the object privileges granted to you the user.

**SELECT grantee, table\_name, privilege**

**FROM user\_tab\_privs;**  
8. What privilege should a user be given to create tables?

**GRANT CREATE TABLE TO scott;**9. If you create a table, how can you pass along privileges to other users just to view your table?

**GRANT SELECT ON my\_table TO other\_user;**  
10. What syntax would you use to grant another user access to your copy\_employees table?

**GRANT SELECT, INSERT, UPDATE ON copy\_employees TO other\_user;**11. How can you find out what privileges you have been granted for columns in the tables belonging  
to others?

**SELECT grantee, table\_name, column\_name, privilege**

**FROM user\_col\_privs;**

17-2: Creating and Revoking Object Privileges  
  
1. What is a role?

**A role is a database object that groups together a set of privileges, which can then be granted to users or other roles. It simplifies the management of privileges by allowing administrators to assign and revoke multiple privileges in one operation.**  
2. What are the advantages of a role to a DBA?

* **Simplified Privilege Management**: Roles allow a DBA to group and manage privileges collectively instead of individually for each user.
* **Consistency**: Ensures consistent privilege assignments across multiple users with similar needs.
* **Scalability**: Simplifies privilege assignment when new users are added, especially in environments with many users.
* **Easier Revocation**: Revoking a role removes all associated privileges from the user.

3. Give the ability to another user in your class to look at one of your tables. Give him the right to let  
other students have that ability.

**GRANT SELECT ON your\_table TO other\_user WITH GRANT OPTION;**4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?

**A Role**  
5. What is the syntax to accomplish the following?  
a. Create a role of manager that has the privileges to select, insert, and update and delete from the employees table

**CREATE ROLE manager;**

**GRANT SELECT, INSERT, UPDATE, DELETE ON employees TO manager;**b. Create a role of clerk that just has the privileges of select and insert on the employees table

**CREATE ROLE clerk;**

**GRANT SELECT, INSERT ON employees TO clerk;**  
c. Grant the manager role to user scott

**GRANT manager TO scott;**d. Revoke the ability to delete from the employees table from the manager role

**REVOKE DELETE ON employees FROM manager;**  
6. What is the purpose of a database link?

**A database link allows a user to access objects in a remote database as if they were in the local database. It facilitates distributed database operations, such as querying, updating, or inserting data across multiple databases.**

17-3: Regular Expressions  
  
1. Working with the employees table, and using regular expressions, write a query that returns employees whose first names start with a “S” (uppercase) followed by either a “t” (lowercase) or “h” (lowercase).

**SELECT first\_name**

**FROM employees**

**WHERE REGEXP\_LIKE(first\_name, '^S[th]');**  
2. Investigate the LOCATIONS table.  
a. Describe the table.

**DESCRIBE LOCATIONS;**  
b. Perform a select that returns all rows and all columns of that table.

**SELECT \* FROM LOCATIONS;**  
c. Write a query using regular expressions that removes the spaces in the street\_address column in the LOCATIONS table

**SELECT REGEXP\_REPLACE(street\_address, ' ', '') AS cleaned\_street\_address**

**FROM LOCATIONS;**