Chapter 5

More SQL: Complex Queries, Indexes, Views, and Schema Modification

Chapter 5 Outline

- More Complex SQL Retrieval Queries
  - Nested queries, joined tables, outer joins, aggregate functions, and grouping
- Indexes in SQL
- Views (Virtual Tables) in SQL
- Schema Change Statements in SQL

Comparisons Involving NULL

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS` or `IS NOT NULL`

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries**
  - Complete select-from-where blocks within WHERE clause of another query
  - Comparison operator `IN`
    - Compares value *v* with a set (or multiset) of values *V*
    - Evaluates to `TRUE` if *v* is one of the elements in *V*

Nested Queries

List all suppliers from the USA, UK, OR Japan

```
1.  SELECT Id, CompanyName, City, Country
2.    FROM Supplier
3.   WHERE Country IN ('USA', 'UK', 'Japan')
```

. List all products that are not exactly $10, $20, $30, $40, or $50

```
1.  SELECT Id, ProductName, UnitPrice
2.    FROM Product
3.   WHERE UnitPrice NOT IN (10,20,30,40,50)
```

Nested Queries

List all customers that are from the same countries as the suppliers.

```
1.  SELECT Id, FirstName, LastName, Country
2.    FROM Customer
3.   WHERE Country IN
4.         (SELECT Country
5.            FROM Supplier)
```

Can we do it in another way ??

Nested Queries

- Use other comparison operators to compare a single value *v*
  - `= ANY` (or `= SOME`) operator
    - Returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
  - Other operators that can be combined with `ANY` (or `SOME`): >, >=, <, <=, and <>

```
SELECT empno, sal
FROM   emp
WHERE  sal > ALL (2000, 3000, 4000);
```

```
-- Transformed to equivalent statement without ALL.
```

```
SELECT empno, sal
FROM   emp
WHERE  sal > 2000 AND sal > 3000 AND sal > 4000;
```

The EXISTS and UNIQUE Functions in SQL

- `EXISTS` function
  - Check whether the result of a correlated nested query is empty or not
  - The result of EXISTS is a Boolean value **TRUE** if the nested query result contains at least one tuple, or **FALSE** if the nested query result contains no tuples.

Problem: Find suppliers with products over $100.

```
1.  SELECT CompanyName
2.    FROM Supplier
3.   WHERE EXISTS
4.         (SELECT ProductName
5.            FROM Product
6.           WHERE SupplierId = Supplier.Id
7.             AND UnitPrice > 100)
```

The EXISTS and UNIQUE Functions in SQL

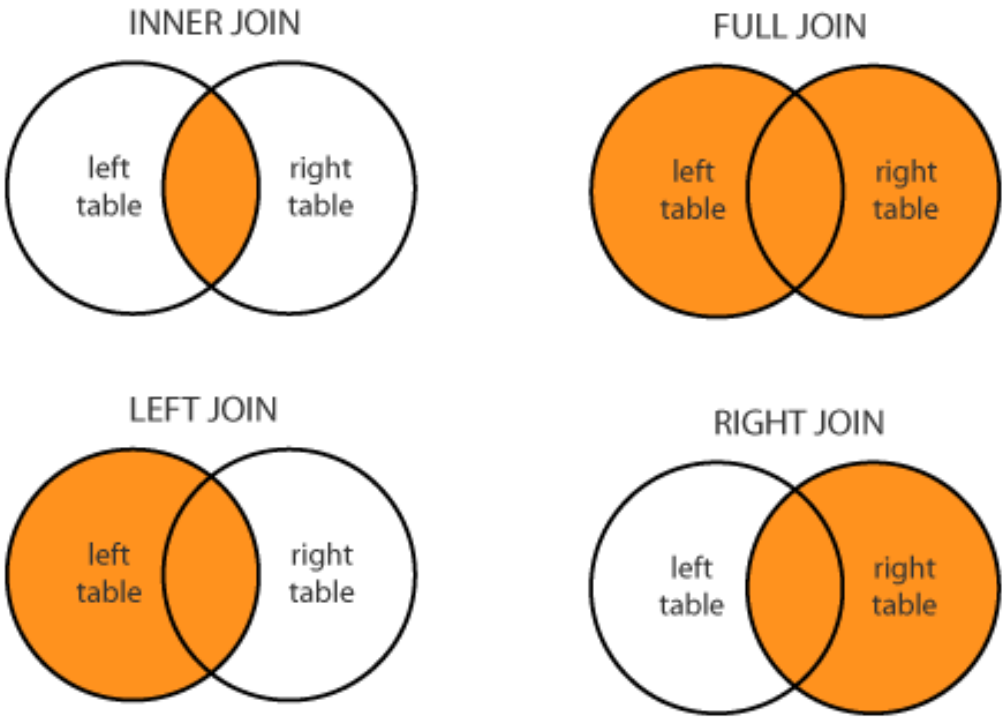
- `NOT EXISTS` returns `TRUE` if there are no tuples in the result of the nested query and returns `FALSE` otherwise.
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

Give list of customers who participated in only one sale

```
SELECT FirstName, LastName
 FROM CUSTOMER
 WHERE UNIQUE
  (SELECT CustomerID FROM SALES
   WHERE SALES.CustomerID = CUSTOMER.CustomerID);
```

Joined Tables in SQL and Outer Joins

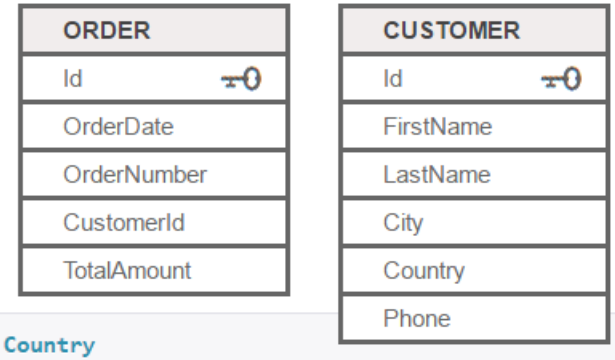
- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query



Joined Tables in SQL and Outer Joins

» Join & Inner Join

» INNER JOIN is the same as JOIN; the keyword INNER is optional.



Problem: List all orders with customer information

```
1.  SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
2.    FROM [Order] JOIN Customer
3.      ON [Order].CustomerId = Customer.Id
```

In this example using table aliases for [Order] and Customer might have been useful.

Results: 830 records.

OrderNumber	TotalAmount	FirstName	LastName	City	Country
542378	440.00	Paul	Henriot	Reims	France
542379	1863.40	Karin	Josephs	Münster	Germany
542380	1813.00	Mario	Pontes	Rio de Janeiro	Brazil
542381	670.80	Mary	Saveley	Lyon	France
542382	3730.00	Pascale	Cartrain	Charleroi	Belgium
542383	1444.80	Mario	Pontes	Rio de Janeiro	Brazil
542384	625.20	Yang	Wang	Bern	Switzerland
...					

Joined Tables in SQL and Outer Joins

» Left Join

» LEFT JOIN and LEFT OUTER JOIN are the same.

```
1.  SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country
2.    FROM Customer C LEFT JOIN  Order  O
3.      ON O.CustomerId = C.Id
4.  ORDER BY TotalAmount
```

This will list all customers, whether they placed any order or not.
The ORDER BY TotalAmount shows the customers without orders first (i.e. TotalMount is NULL).

Results: 832 records

OrderNumber	TotalAmount	FirstName	LastName	City	Country
NULL	NULL	Diego	Roel	Madrid	Spain
NULL	NULL	Marie	Bertrand	Paris	France
542912	12.50	Patricio	Simpson	Buenos Aires	Argentina
542937	18.40	Paolo	Accorti	Torino	Italy
542897	28.00	Pascale	Cartrain	Charleroi	Belgium
542716	28.00	Maurizio	Moroni	Reggio Emilia	Italy
543028	30.00	Yvonne	Moncada	Buenos Aires	Argentina
543013	36.00	Fran	Wilson	Portland	USA
...					

Joined Tables in SQL and Outer Joins

» Right Join

» RIGHT JOIN and RIGHT OUTER JOIN are the same.

Problem: List customers that have not placed orders

```
1.  SELECT TotalAmount, FirstName, LastName, City, Country
2.    FROM  Order  O RIGHT JOIN Customer C
3.      ON O.CustomerId = C.Id
4.  WHERE TotalAmount IS NULL
```

This returns customers that, when joined, have no matching order.

Results: 2 records

TotalAmount	FirstName	LastName	City	Country
NULL	Diego	Roel	Madrid	Spain
NULL	Marie	Bertrand	Paris	France

Joined Tables in SQL and Outer Joins

» Full Join

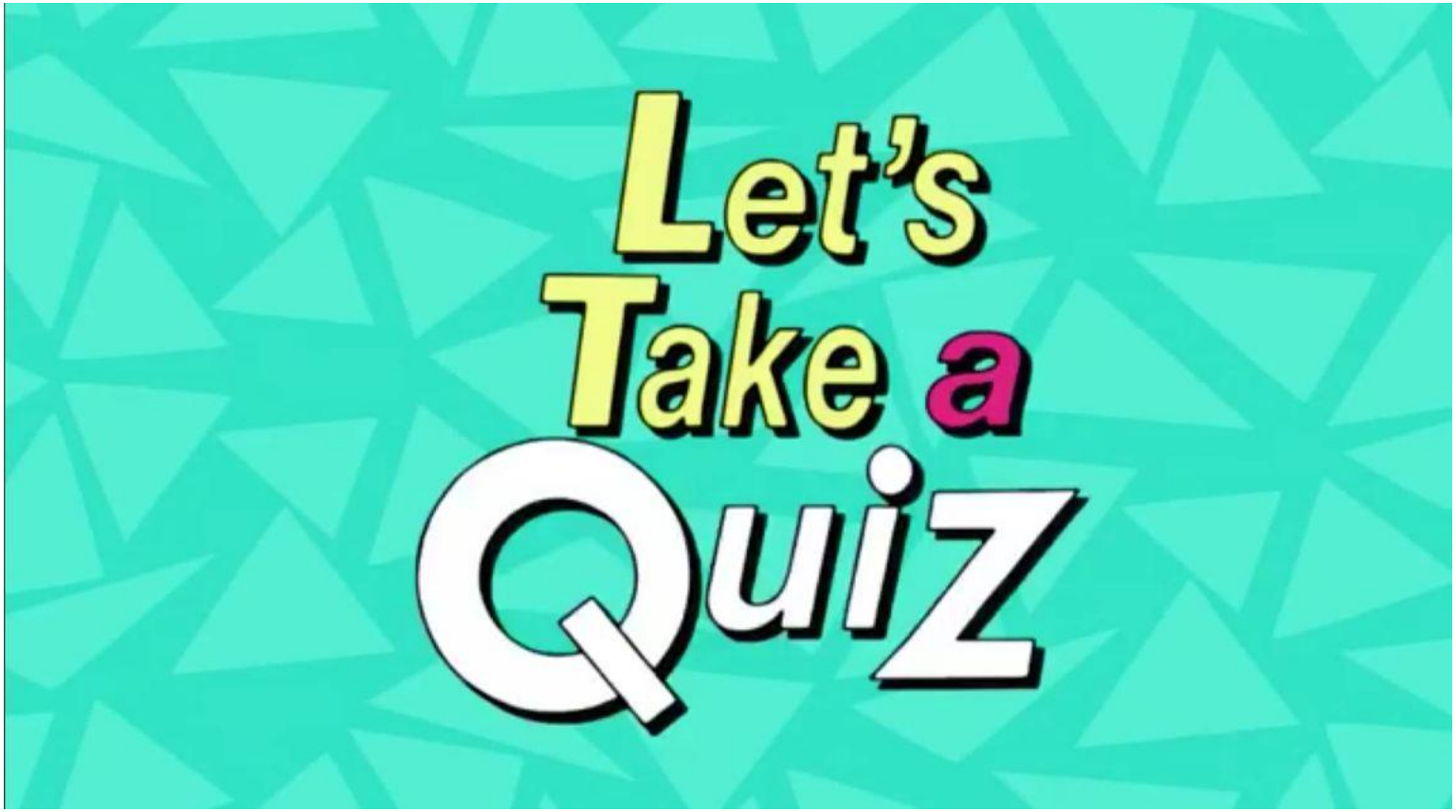
» FULL JOIN and FULL OUTER JOIN are the same.

Problem: Match all customers and suppliers by country

```
1.  SELECT C.FirstName, C.LastName, C.Country AS CustomerCountry,
2.         S.Country AS SupplierCountry, S.CompanyName
3.    FROM Customer C FULL JOIN Supplier S
4.      ON C.Country = S.Country
5.   ORDER BY C.Country, S.Country
```

This returns suppliers that have no customers in their country,
and customers that have no suppliers in their country,
and customers and suppliers that are from the same country.

FirstName	LastName	CustomerCountry	SupplierCountry	CompanyName
NULL	NULL	NULL	Australia	Pavlova, Ltd.
NULL	NULL	NULL	Australia	G'day, Mate
NULL	NULL	NULL	Japan	Tokyo Traders
Patricio	Simpson	Argentina	NULL	NULL
Yvonne	Moncada	Argentina	NULL	NULL
Sergio	Gutiérrez	Argentina	NULL	NULL
Lúcia	Carvalho	Brazil	Brazil	Refrescos Americanas LTDA
Janete	Limeira	Brazil	Brazil	Refrescos Americanas LTDA
Aria	Cruz	Brazil	Brazil	Refrescos Americanas LTDA


Let's
Take a
QuiZ

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Grouping
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- Functions can be used in the `SELECT` clause or in a `HAVING` clause

Aggregate Functions in SQL

Problem: Find the cheapest product

```
1.  SELECT MIN(UnitPrice)
2.    FROM Product
```

Problem: Find the number of customers

```
1.  SELECT COUNT(Id)
2.    FROM Customer
```

Problem: Find the largest order placed in 2014

```
1.  SELECT MAX(TotalAmount)
2.    FROM  Order
3.   WHERE YEAR(OrderDate) = 2014
```

Problem: Compute the total amount sold in 2013

```
1.  SELECT SUM(TotalAmount)
2.    FROM  Order
3.   WHERE YEAR(OrderDate) = 2013
```

Problem: Find the number of customers

```
1.  SELECT COUNT(*)
2.    FROM Customer
```

Aggregate Functions in SQL

- In the previous slide, the asterisk refers to rows (tuples), so COUNT(*) returns the number of rows in the result of a query
- We may also use the COUNT function to count values in a column rather than tuples

Query 23. Count the number of distinct salary values in the database.

```
Q23:  SELECT  COUNT (DISTINCT Salary)
      FROM    EMPLOYEE;
```

Grouping: The GROUP BY and HAVING Clauses

- **Partition** relation into subsets of tuples
  - Based on **grouping attribute(s)**
  - Apply function to each such group independently
- **`GROUP BY`** clause
  - Specifies grouping attributes
- If NULLs exist in grouping attribute
  - Separate group created for all tuples with a NULL value in grouping attribute

Grouping: The GROUP BY and HAVING Clauses

Problem: List the number of customers in each country

```
1.  SELECT COUNT(Id), Country
2.    FROM Customer
3.   GROUP BY Country
```

Results: 21 records.

Count	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
...	

Grouping: The GROUP BY and HAVING Clauses

- HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.

Problem: List the number of customers in each country. Only include countries with more than 10 customers.

```
1.  SELECT COUNT(Id), Country
2.    FROM Customer
3.   GROUP BY Country
4.  HAVING COUNT(Id) > 10
```

Results: 3 records

Count	Country
11	France
11	Germany
13	USA

Discussion and Summary of SQL Queries

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

Discussion and Summary of SQL Queries (Very Important!)

- A query is evaluated *conceptually* by first applying the FROM clause (to identify all tables involved in the query or to materialize any joined tables), followed by the WHERE clause to select and join tuples, and then by GROUP BY and HAVING.
- For each combination of tuples—one from each of the relations specified in the FROM clause—evaluate the WHERE clause; if it evaluates to TRUE, place the values of the attributes specified in the SELECT clause from this tuple combination in the result of the query.
- Conceptually, ORDER BY is applied at the end to sort the query result.

Nice Tool!

https://www.khanacademy.org/computing/computer-programming/sql/sql-basics/p/aggregating-data

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

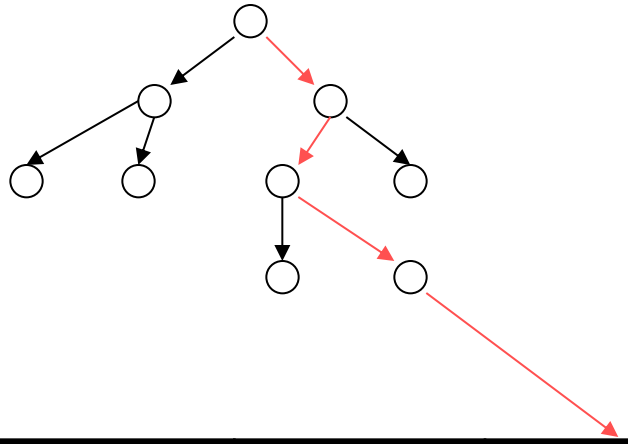
Person (name, age, city)

```
SELECT *
FROM   Person
WHERE  name = "Smith"
```

Sequential scan of the file Person may take long

Indexes

- Create an index on name:



Adam	Betty	Charles	….	Smith	….

Creating Indexes

Syntax:

```
CREATE INDEX  nameIndex ON Person(name)
```

Creating Indexes

Indexes can be useful in range queries too:

```
CREATE INDEX ageIndex ON  Person (age)
```

B+ trees help in:

```
SELECT *
FROM Person
WHERE age > 25 AND age < 28
```

- Why not create indexes on everything?
- When a PRIMARY KEY constraint is specified, DBMS automatically creates an index to support rapid data retrieval for the specified table.

Views (Virtual Tables) in SQL

- Concept of a view in SQL
  - Single table derived from other tables
  - Considered to be a virtual table
    - This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view

Specification of Views in SQL

- **`CREATE VIEW`** command
  - Give table name, list of attribute names, and a query to specify the contents of the view

The view "Current Product List" lists all active products

```
CREATE VIEW Current_Product_List AS
        SELECT      ProductID,ProductName
        FROM        Products
        WHERE       Discontinued=No
```

We can query the view above as follows:

```
SELECT * FROM Current_Product_List
```

Select every product in the "Products" table with a unit price higher than the average unit price:

```
CREATE VIEW Products_Above_Average_Price AS
        SELECT      ProductName,UnitPrice
        FROM        Products
        WHERE       UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

We can query the view above as follows:

```
SELECT * FROM Products_Above_Average_Price
```

Specification of Views in SQL

- Specify SQL queries on a view
- View always up-to-date
  - Responsibility of the DBMS and not the user
  - The view is not realized or materialized at the time of *view definition* but rather at the time when we *specify a query* on the view
- **`DROP VIEW`** command

```
DROP VIEW       WORKS_ON1;
```

The DROP Command

- Drop TABLE behavior options:
  - `CASCADE` and `RESTRICT`
- Example:
  - `DROP TABLE DEPENDENT CASCADE;`
- If the RESTRICT option is chosen, a table is dropped only if it is *not referenced* in any constraints (for example, by foreign key definitions in another relation) or views or by any other elements.
- With the CASCADE option, all such constraints, views, and other elements that reference the table being dropped are also dropped automatically from the schema, along with the table itself.

The DROP Command

- Example:
  - `DROP TABLE DEPENDENT CASCADE;`
- The DROP TABLE command not only deletes all the records in the table if successful, but also removes the *table definition* from the catalog.
- If it is desired to delete only the records but to leave the table definition for future use, then the DELETE command should be used instead of DROP TABLE.
- *The DROP command can also be used to drop other types of named schema elements, such as constraints or domains.*

The ALTER Command

- **Alter table actions** include:
  - Adding or dropping a column (attribute)
  - Changing a column definition
  - Adding or dropping table constraints
- Example:
  -
```
ALTER TABLE EMPLOYEE
  ADD COLUMN Job VARCHAR(12);
```

The ALTER Command

- We must still enter a value for the new attribute Job for each individual EMPLOYEE tuple.
- This can be done either by specifying a DEFAULT clause or by using the UPDATE command individually on each tuple.
- If no default clause is specified, the new attribute will have NULLs in all the tuples of the relation immediately after the command is executed;
  - The NOT NULL constraint is *not allowed* in this case.

The ALTER Command

- To drop a column
  - Choose either `CASCADE` or `RESTRICT`

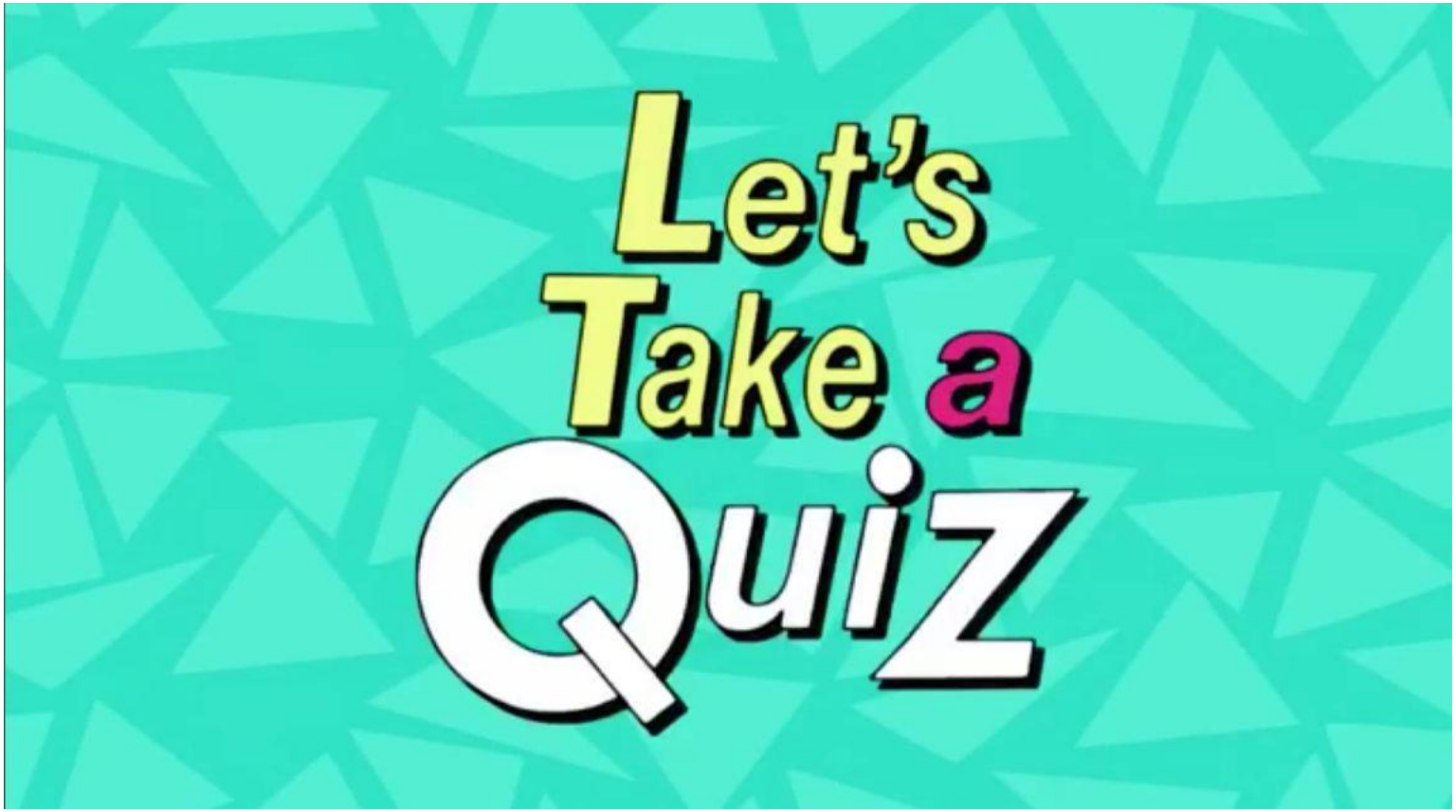
```
ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;
```

- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
- If CASCADE is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.
- If RESTRICT is chosen, the command is successful only if no views or constraints (or other schema elements) reference the column.

The ALTER Command

- It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause. The following examples illustrate this clause:

```
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
    DROP DEFAULT;
ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn
    SET DEFAULT '333445555';
```


Let's
Take a
QuiZ