Database Systems

Week 11-Lecture-1-2-3

Triggers

Indexes & Views

Data Control Language

Database Trigger

A procedure that starts automatically if specified changes occur to the DBMS

- Triggers are invoked when you insert data into a table, update data, or delete data.
- By defining one or more triggers on a table, you can specify which data-modification actions will cause the trigger to fire.
- This type of functionality is generally referred to as **active databases**.
- SQL support three type of trigger
  - Insert
  - Update
  - Delete

Trigger

- **Three parts:**
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run) [Optional]
  - Action (what happens if the trigger runs)
- When event occurs, and condition is satisfied, the action is performed.
- Events could be :

  `BEFORE|AFTER INSERT|UPDATE|DELETE ON <tableName>`

  e.g.: `BEFORE INSERT ON Professor`

Syntax

CREATE TRIGGER <trigger name>

{ BEFORE | AFTER }

{ INSERT | DELETE | UPDATE [OF <column list>] }

ON <table name> [REFERENCING <alias options>]

[FOR EACH { ROW | STATEMENT }]

[WHEN (<search condition>)]

<triggered SQL statements>

Trigger

Assume our DB has a relation schema :

Professor (pNum, pName, salary)

We want to write a trigger that :

Ensures that any new professor inserted has salary >= 60000

Trigger

```
CREATE TRIGGER minSalary BEFORE INSERT ON Professor

     FOR EACH ROW

BEGIN

   IF (:new.salary < 60000)
    THEN RAISE_APPLICATION_ERROR (-20004,          'Violation of
   Minimum Professor Salary');
   END IF;

END;
```

Example 1

```
CREATE TRIGGER reminder1
ON Sales.Customer
AFTER INSERT, UPDATE
AS RAISERROR ('Notify Customer Relations', 16, 10);
```

Indexes & Views

Week 11-Lecture-2

Indexes

REALLY important to speed up query processing time.

Suppose we have a relation

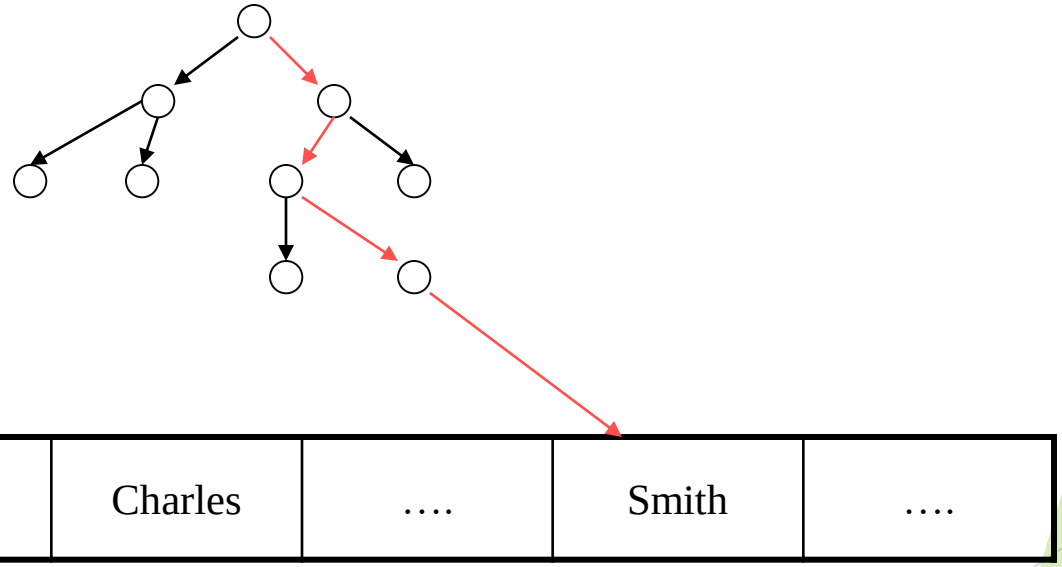
Person (name, age, city)

```
SELECT *
FROM   Person
WHERE  name = “Smith”
```

Sequential scan of the file Person may take long

Indexes

- Create an index on name:



- B+ trees have fan-out of 100s: max 4 levels !

Creating Indexes

Syntax:

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

Creating Indexes

Indexes can be created on more than one attribute:

Example:

```
CREATE INDEX doubleindex ON
                    Person (age, city)
```

Helps in:

```
SELECT *
FROM    Person
WHERE age = 55 AND city = “Seattle”
```

But not in:

```
SELECT *
FROM    Person
WHERE city = “Seattle”
```

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

Defining Views

Views are relations, except that they are not physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = “Development”
```

Payroll has access to Employee, others only to Developers

A Different View

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Seattle-view  AS

        SELECT  name,buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                    Person.name = Purchase.buyer
```

We have a new virtual table:
Seattle-view(buyer, seller, product, store)

```
CREATE VIEW  Seattle-view  AS

        SELECT  buyer, seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                    Person.name = Purchase.buyer
```

A Different View

We can later use the view:

```
SELECT   name, store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
           Product.category = “shoes”
```

What Happens When We Query a View ?

```
SELECT   name, Seattle-view.store
FROM     Seattle-view, Product
WHERE    Seattle-view.product = Product.name  AND
         Product.category = "shoes"
```

```
SELECT  name, Purchase.store
FROM    Person, Purchase, Product
WHERE   Person.city = "Seattle"    AND
        Person.name = Purchase.buyer   AND
        Purchase.poduct = Product.name  AND
        Product.category = "shoes"
```

Types of Views

- Virtual views:
  - Used in databases
  - Computed only on-demand – slow*er* at runtime
  - Always up to date
- Materialized views
  - Used in data warehouses
  - Precomputed offline – fast*er* at runtime
  - May have stale data

Updating Views

How can I insert a tuple into a table that doesn't exist?

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = "Development"
```

If we make the following insertion:

```
INSERT INTO  Developers
VALUES("Joe", "Optimizer")
```

It becomes:

```
INSERT INTO  Employee
VALUES(NULL, "Joe", NULL, "Optimizer", NULL)
```

Non-Updatable Views

```
CREATE VIEW  Seattle-view  AS

        SELECT  seller, product, store
        FROM    Person, Purchase
        WHERE   Person.city = "Seattle"    AND
                    Person.name = Purchase.buyer
```

How can we add the following tuple to the view?

("Joe", "Shoe Model 12345", "Nine West")

We need to add "Joe" to Person first, but we don't have all its attributes

Answering Queries Using Views

- What if we want to *use* a set of views to answer a query.
- Why?
  - The obvious reason…
  - Answering queries over web data sources.
- *Very* cool stuff!

Query Rewriting Using Views

Rewritten query:

```
SELECT  buyer, seller
FROM    SeattleView
WHERE   product= 'gizmo'
```

Original query:

```
SELECT  buyer, seller
FROM    Person, Purchase
WHERE   Person.city = 'Seattle'    AND
            Person.per-name = Purchase.buyer AND
            Purchase.product='gizmo'.
```

Week -11
Lecture-01

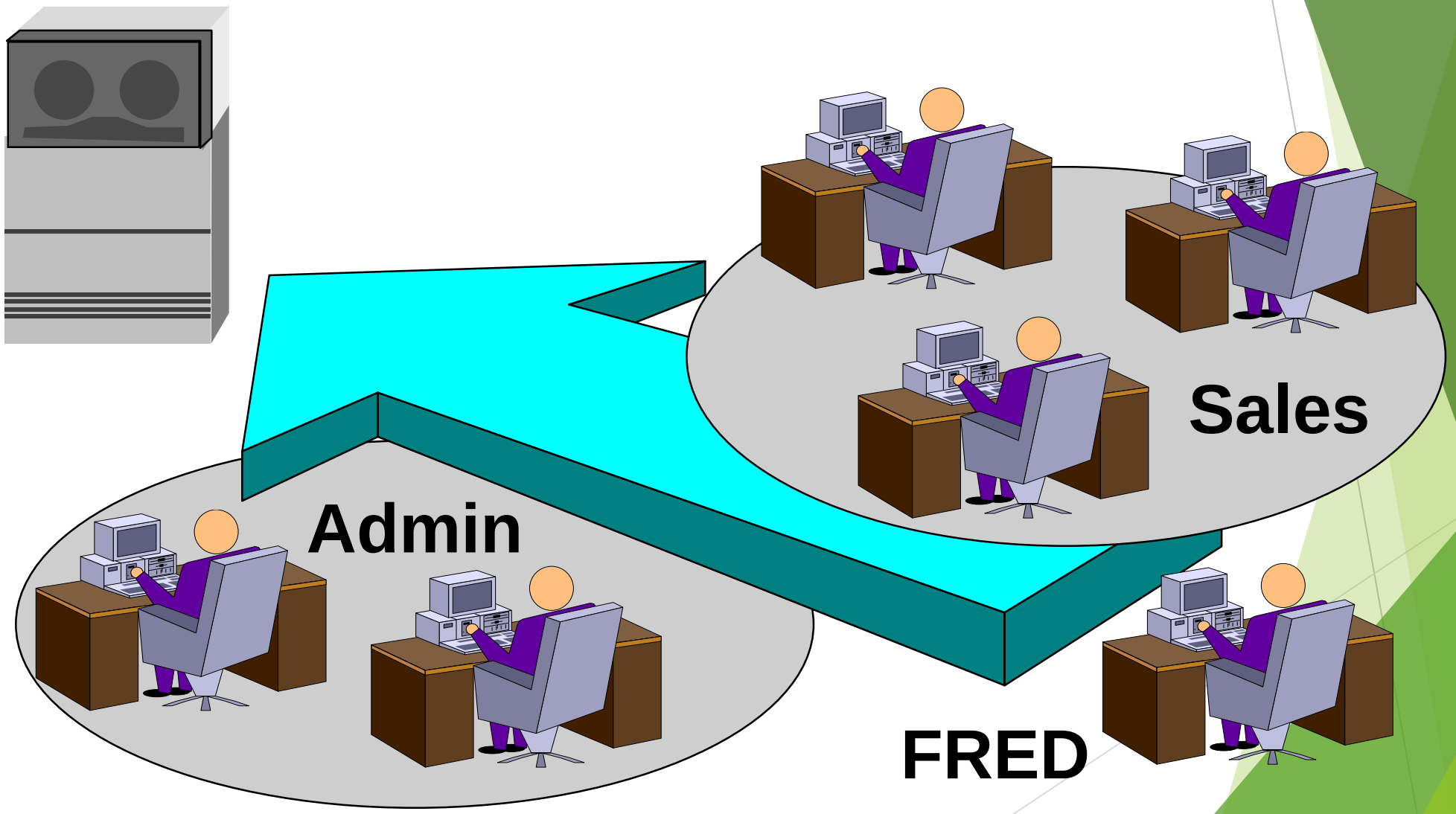
DCL

- The SQL security scheme is based on three central concepts
- Users. The actor in the database. Each time they retrieves, inserts, delete or update data.
- Database Objects. The items to which SQL security protection can be applied. These objects are tables and views.
- Privileges: The action that a user is permitted to carry out for a given database object. These privileges are
  SELECT, INSERT, DELETE AND UPDATE.
- To established a security scheme for a database, you use the SLQ GRANT statement to specify which user have which privileges on which database object.

Data Control Language

- Objectives
  - To learn about the security mechanisms implemented in an RDBMS and how to use them
- Contents
  - Identifying Users
  - Privileges
  - The GRANT Statement
  - The REVOKE Statement
  - The System Catalogue

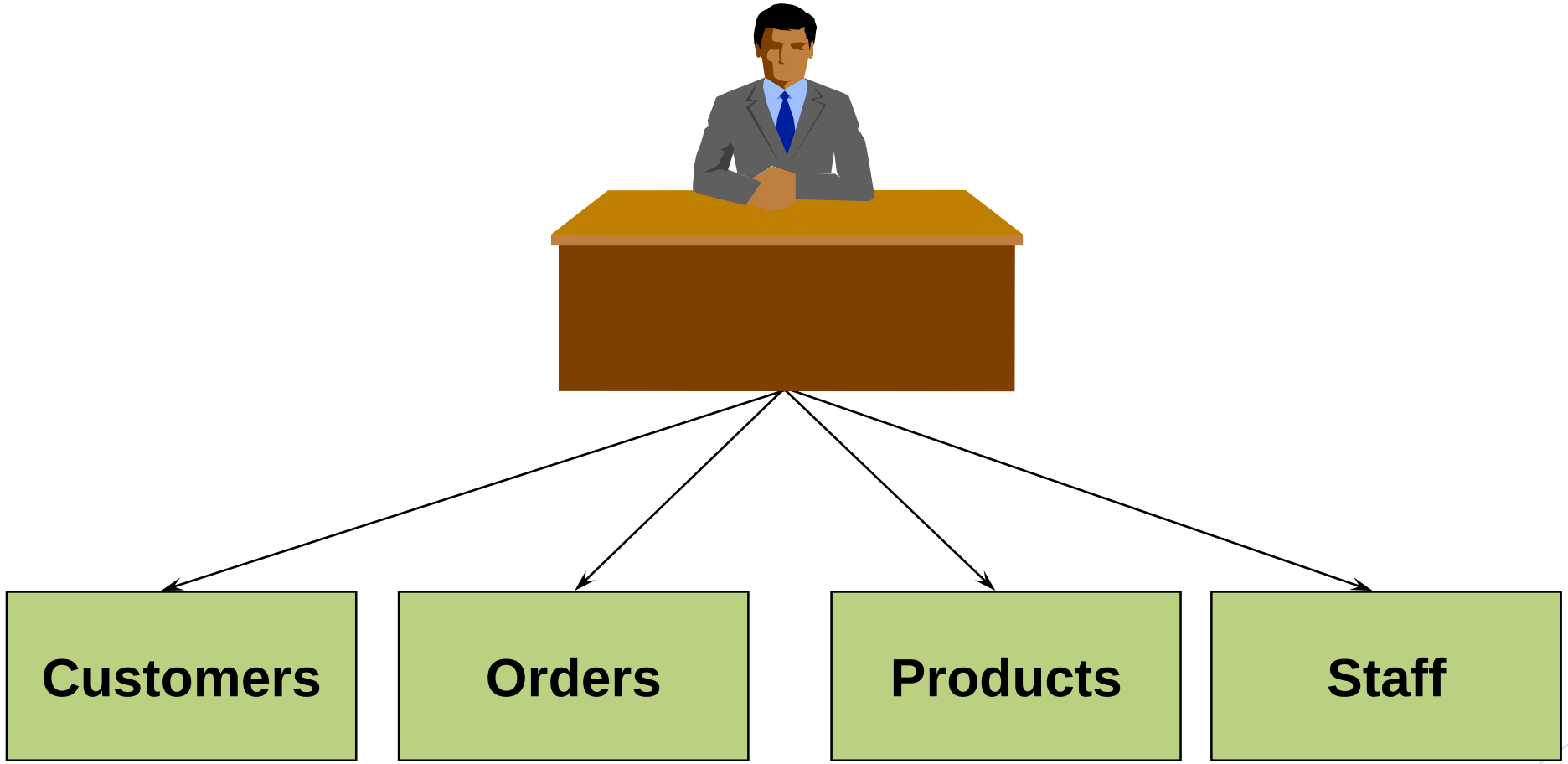
Identifying Users



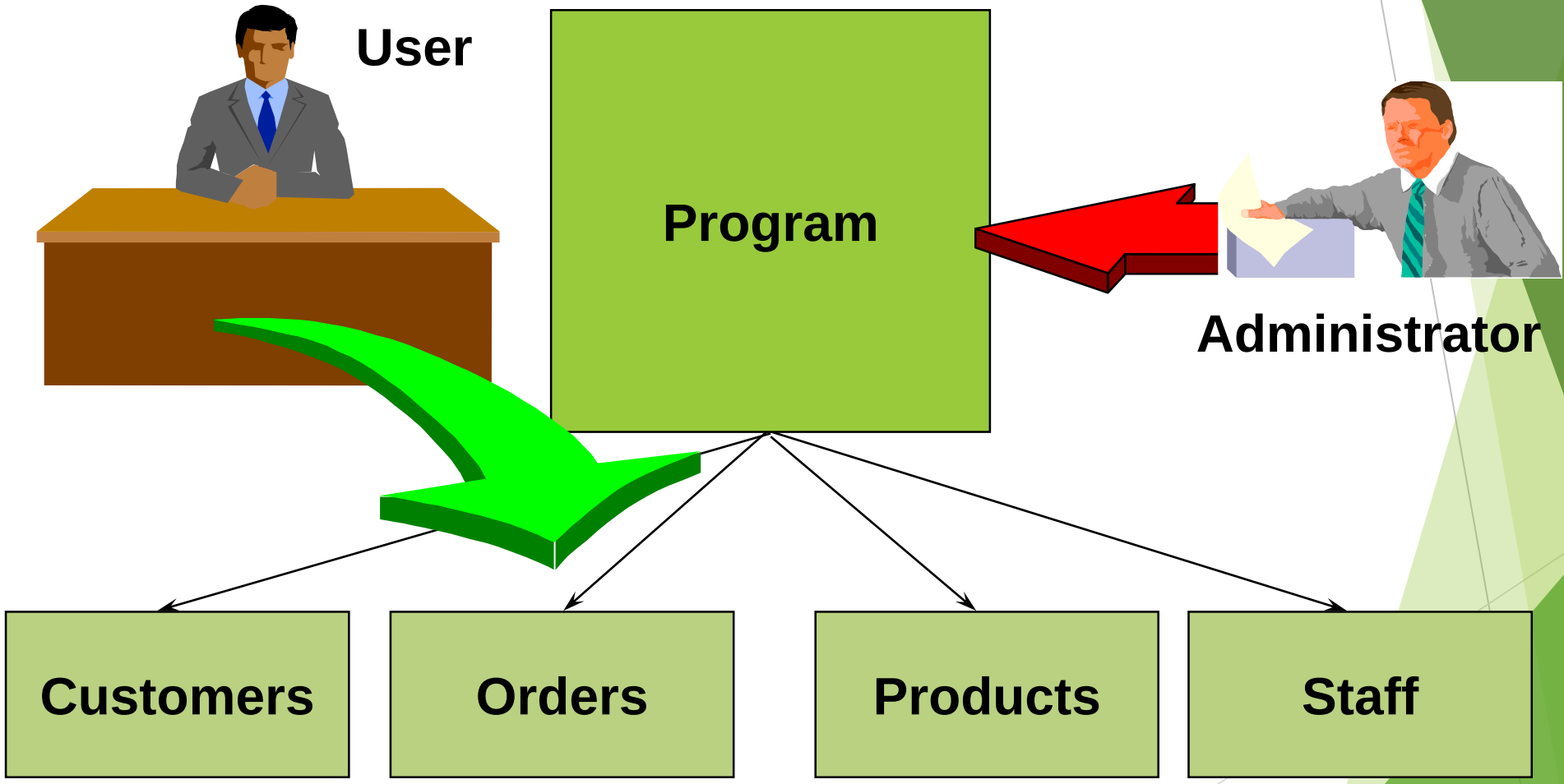
Privileges

- Allowable Privileges
  - SELECT, INSERT, UPDATE, DELETE
  - CREATE Table, View, Procedure, Trigger, Rule, Default
- The owner/creator of a table automatically has all the privileges

Direct Privileges



Indirect Privileges



- Creating User

CREATE USER username IDENTIFIED BY Password;

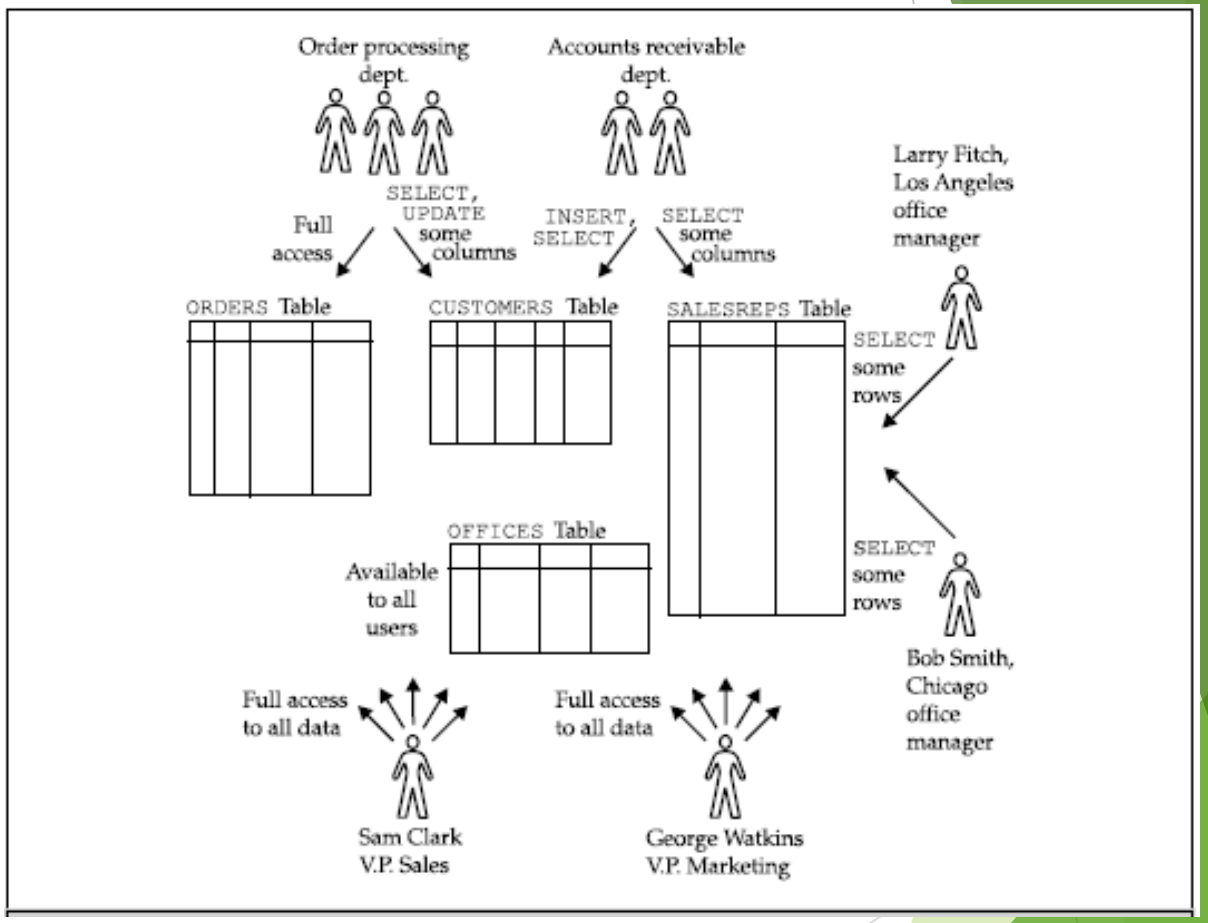
GRANT CONNECT TO user;

GRANT CONNECT, RESOURCE, DBA TO user;

GRANT SELECT, INSERT, UPDATE, DELETE ON table-name TO user;

GRANT the Privileges

For example, here is a GRANT statement that lets Sam Clark retrieve and insert data in the OFFICES table of the sample database:




```
GRANT SELECT, INSERT
ON OFFICES
TO SAM
```

If you want to take away these grants then apply revoke statement

```
REVOKE SELECT, INSERT
ON OFFICE
FROM SAM
```

GRANT/ REVOKE

- GRANT privilege ON tablename TO list
  [WITH GRANT OPTION]

- For example
  GRANT ALL ON dept TO John
  GRANT SELECT ON dept TO sally
  GRANT SELECT, UPDATE, INSERT ON dept TO Jim, Mike, Howard

  REVOKE privilege ON tablename FROM list
  e.g REVOKE SELECT ON dept FROM Sally

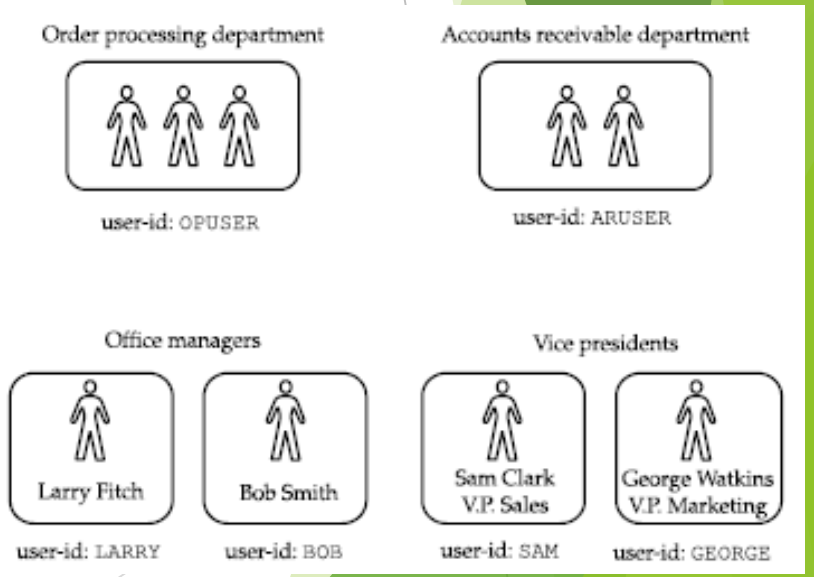
- However largely done these days via checkboxes in grids in GUI admin tools

Grant & Revoke specific attribute in the relation

- Grant select(attribute) on table to user;
- Revoke select(attribute) on table to user;

USER ID

- Each user of a SQL-based database is typically assigned a user-id, to identify the user to the DBMS software.
- User-id determine whether the statement will be permitted or prohibited by the DBMs.
- Assigned by DBA.
- Creating user syntax
- Create user username identified by password;
- Create user sam identified by welcome123;



Group of users

- BUNDLE OF PRIVILIGES TO MANY USERS USE ROLE
- Role define set of privileges.

  Grant create session, create user, create table, insert, delete to role name;

  Grant select delete update on table name to r1;
- Assigned these grants to new user

  Grant r1 to user1,user2,user3,user4….,usern;
- If a group of user perform same task the we create role for group of users.
- Suppose Sam, David, Jerry are clerk so we create role for them
- Create role clerk identified by login;
- Now grant privileges to clerk

  Grant r1 to clerk;

Group of users

- Now grant this clerks role to users like this
- grant clerks to sami, scott, ashi, tanya ;
- Now Sami, Scott, Ashi and Tanya have all the privileges granted on clerks role.
- Suppose after one month you want grant delete on privilege on emp table all these users then just delete these privilege to clerks role and automatically all the users will have the privilege.
- grant delete on emp to clerks;
- If you want to take back update privilege on emp table from these users just take it back from clerks role.
- revoke update on emp from clerks;
- To Drop a role
- Drop role clerks;
- To modify role use ALTER command.
- Alter role role name with attribute name;
- Alter role clerk with system_clerk;

Non-ANSI Privileges

- Sample Non ANSI Table Privileges

  GRANT ALTER ON dept TO Sally

  GRANT INDEX ON dept TO John

- Sample Non ANSI Database Privileges

  GRANT CONNECT ON database TO John, Ann

  GRANT RESOURCE TO Alex

  GRANT DBA TO Simon

Summary

- Identifying Users
  - Users in the system can be grouped together to make security handling easier
- Privileges
  - Permissions can be granted at several levels and can be granted directly or indirectly using views and stored procedures
- The GRANT Statement
  - Is used to give people permissions on database objects
- The REVOKE Statement
  - Is used to take permissions away
- The System Catalogue
  - All information about permissions is stored within the catalogue

SQL Session

- An *SQL session* is the connection between some sort of client application and the database.
- The session provides the context in which the authorization identifier executes SQL statements during a single connection.
- The session begins when you start the interactive SQL program, and it lasts until you exit the program. In an application program using programmatic SQL,
- All of the SQL statements used during the session are associated with the user-id specified for the session.
- Usually, you must supply both a user-id and an associated password at the beginning of a session.
- Grant create session to sam;

GRANT statement

- SQL GRANT is a command used to provide access or privileges on the database objects to the users.
- Normally, the GRANT statement is used by the owner of a table or view to give other users access to the data.

  GRANT SELECT, INSERT, DELETE, UPDATE

  ON OREDER

  TO OPUSER
- *Allow Sam Clark to insert or delete an office.*

  GRANT INSERT, DELETE

  ON OFFICES

  TO SAM

GRANTS STATEMENTS

- Assign all privileges to the user using ALL PRIVILEGES statements

  GRANT ALL PRIVILEGES

  ON SALESREPS

  TO SAM

- *Give all users* SELECT *access to the* OFFICES *table.*

  GRANT SELECT

  ON OFFICES

  TO PUBLIC

- *Let order-processing users change company names and salesperson assignments.*

  GRANT UPDATE (COMPANY, CUST_REP)

  ON CUSTOMERS

  TO OPUSER

- *Give accounts receivable users read-only access to the employee number, name, and sales*
- *office columns of the* SALESREPS *table.*

```
GRANT SELECT (EMPL_NUM, NAME, REP_OFFICE)
ON SALESREPS
TO ARUSER
```

- Passing GRANT

```
GRANT SELECT
ON WESTREPS
TO LARRY
WITH GRANT OPTION
```

- Larry can now issue this GRANT statement:

```
GRANT SELECT
ON WESTREPS
TO SUE
```

REVOKE

- The privileges that you have granted with the GRANT statement can be taken away with the REVOKE statement
- A REVOKE statement may take away all or some of the privileges that you previously granted to a user-id.
- *Grant and then revoke some* SALESREPS *table privileges*.

```
GRANT SELECT, INSERT, UPDATE
ON SALESREPS
TO ARUSER, OPUSER


REVOKE INSERT, UPDATE
ON SALESREPS
FROM OPUSER
```

- *Take away* UPDATE *and* DELETE *privileges for two user-ids.*

  REVOKE UPDATE, DELETE

  ON OFFICES

  FROM ARUSER, OPUSER

- *Take away all privileges on the* OFFICES *that were formerly granted to all users.*

  REVOKE ALL PRIVILEGES

  ON OFFICES

  FROM PUBLIC