Database Management Systems
L7

Umass Boston
Summer 2023
Cristina Maier

Some slides are based on “Database Management System, 3rd ed., by Ramakrishnan and Gehrke

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design: the Entity-Relationship Model
- Structured Query Language (SQL)
- Database Security and Authorization
- Schema Refinement and Normal Forms
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

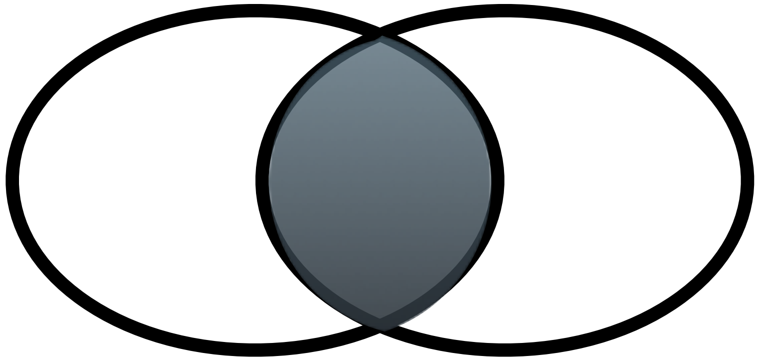
SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Count
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators

Join operators

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- NATURAL JOIN
- CROSS JOIN

Inner Join



TableOrViewExpression [INNER] JOIN TableOrViewExpression

ON BooleanCondition

Example

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

boats

bid	name	color
101	interlake	red
102	clipper	green

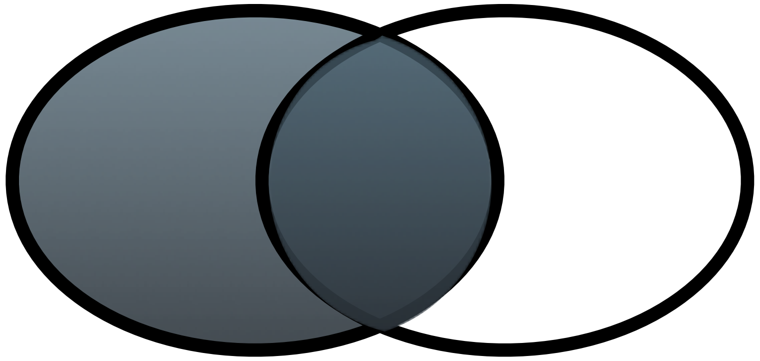
```
SELECT * FROM sailors s JOIN reserves r on s.sid=r.sid ;
```

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/22
22	dustin	7	45.0	22	102	10/20/22
58	rusty	10	35.0	58	101	10/11/22

SAME AS

```
Select * FROM sailors s, reserves r
WHERE s.sid=r.sid;
```

Left Outer Join



TableOrViewExpression LEFT [OUTER] JOIN TableOrViewExpression

ON BooleanCondition

Example

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

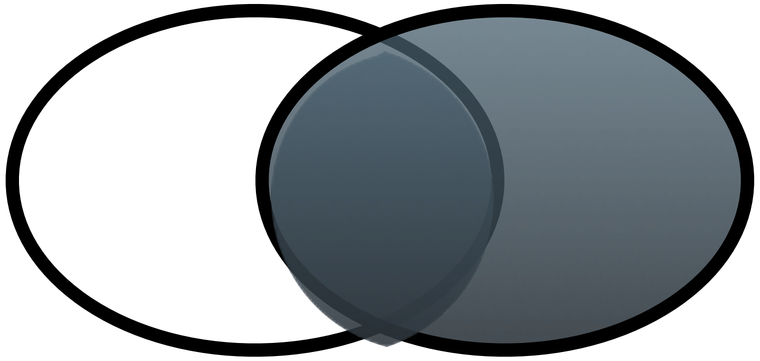
boats

bid	name	color
101	interlake	red
102	clipper	green

```
SELECT * FROM sailors s LEFT JOIN reserves r on s.sid=r.sid ;
```

sid	sname	rating	age	sid	bid	day
22	dustin	7	45.0	22	101	10/10/22
22	dustin	7	45.0	22	102	10/20/22
31	lubber	8	55.0	NULL	NULL	NULL
58	rusty	10	35.0	58	101	10/11/22
59	rusty	10	45.0	NULL	NULL	NULL

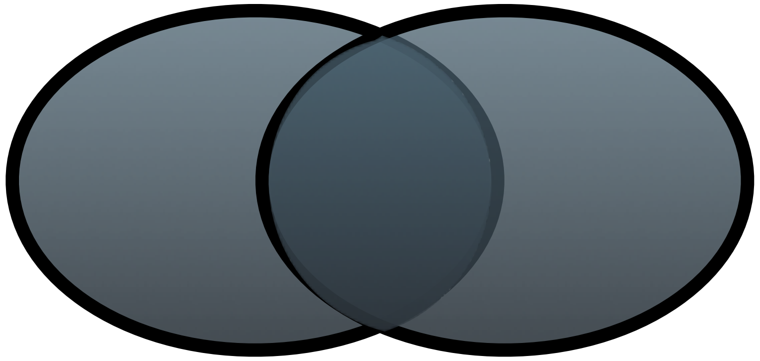
Right Outer Join



TableOrViewExpression RIGHT [OUTER] JOIN TableOrViewExpression

ON BooleanCondition

Full Outer Join



FROM TableOrViewExpression FULL [OUTER] JOIN
TableOrViewExpression

ON BooleanCondition

Join operators

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- NATURAL JOIN
- CROSS JOIN

Natural Join

- FROM TableOrViewExpression NATURAL JOIN TableOrViewExpression2
- Like INNER JOIN on all columns sharing the same name
- No duplicated columns
- Columns sharing the same name appear only once in the result (Same as natural join from relational algebra!!)

NATURAL JOIN Example

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

boats

bid	name	color
101	interlake	red
102	clipper	green

```
SELECT * FROM sailors s NATURAL JOIN reserves r;
```

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/22
22	dustin	7	45.0	102	10/20/22
58	rusty	10	35.0	101	10/11/22

sid column appears only once!

CROSS JOIN

- SELECT * FROM TableOrViewExpression CROSS JOIN TableOrViewExpression2;
- Same as
- SELECT FROM TableOrViewExpression, TableOrViewExpression2;

Summary

- SQL shorthands for expressions we already saw
  - Cross product
    - Sailors CROSS JOIN Reserves
  - Condition Join
    - Sailors JOIN Reserves on <condition>
  - Natural Join
    - Sailors NATURAL JOIN Reserves
- Outer Joins

Summary (cont.)

- Outer Joins : include in the result the non-matching tuples
- Result tuple padded with NULL Values
    - FULL: non-matching tuples in both relations included in the result
    - LEFT: non-matching tuples in left relation included in the result
    - RIGHT: non-matching tuples in right relation included in the result

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators
- UPDATE, DELETE records

UPDATING records

```
UPDATE tableName
SET columnName=<value>[,
        columnName2=<value>…]
[WHERE condition];
```

Example

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

```
UPDATE Sailors
SET rating=9
WHERE sid=31
```

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	9	55.0
58	rusty	10	35.0
59	rusty	10	45.0

Example 2

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	9	55.0
58	rusty	10	35.0
59	rusty	10	45.0

```
UPDATE Sailors
SET rating=10,  sname='lubber2'
WHERE sid=31;
```

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber2	10	55.0
58	rusty	10	35.0
59	rusty	10	45.0

Example 3

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber2	10	55.0
58	rusty	10	35.0
59	rusty	10	45.0

```
UPDATE Sailors
SET rating=8;
```

sailors

sid	sname	rating	age
22	dustin	8	45.0
31	lubber2	8	55.0
58	rusty	8	35.0
59	rusty	8	45.0

Delete Records from Table

```
DELETE FROM tableName
WHERE <condition>;
```

- Deletes all records that satisfy the <condition>

Delete Records from Table

```
DELETE FROM tableName;
```

- Deletes all records from table tableName.
- To be used carefully!!!!

Example 3

sailors

sid	sname	rating	age
22	dustin	8	45.0
31	lubber2	8	55.0
58	rusty	8	35.0
59	rusty	8	45.0

```
DELETE FROM Sailors

WHERE sname='rusty';
```

sailors

sid	sname	rating	age
22	dustin	8	45.0
31	lubber2	8	55.0

Example 3

sailors

sid	sname	rating	age
22	dustin	8	45.0
31	lubber2	8	55.0

DELETE FROM Sailors;

Deletes all data from the table. The table schema will be kept (i.e. table will not be deleted, it will just become empty).

sailors

INSERT FROM SELECT Statement

- INSERT INTO tableName

  SELECT …;
- INSERT INTO tableName(col1,col2…)

  SELECT col1, col2,….;

Example

- Schamas:
  - Sailors (sid:int, sname: string, rating: int, age:real)
  - Sailors3 (sid:int, sname: string, rating: int, age:real, salary: int)

```
INSERT INTO Sailors(sid,sname,rating,age)
SELECT  sid,sname, rating, age from Sailors3;
```

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators
- Integrity Constraints

Integrity Constraints (IC)

- Describe some conditions that need to be satisfied
- INSERTs, DETELEs, UPDATEs that violate IC are not allowed

Types of integrity constraints

- Domain Type
- NULL constraints
- Primary Key
- Foreign Key
- General Constraints

Domain Constraints

- Values must be of right type
- Always enforced
- E.g. if column rating is int, and we want to insert a string value, the insert is rejected with an error

NULL constraints

- You always have to provide a value for the NOT NULL columns (this includes the primary keys, as they are NOT NULL by default)

PRIMARY KEYS

- Uniqueness constraint
  - If we try to insert a record that has a primary key that is already present in another record, the insert is rejected
  - Same applies to updates
- Not null
  - When we insert record, primary key must always be present

Foreign Keys

- The key that is referenced by the foreign key must always be present in the table we reference
- If we insert a record whose foreign key value does not exist in the table the foreign key references, an error is returned
- When deleting a record from a table we get an error if there is another table that references that record's key.
- That's why the order or CREATING tables and DROPPING tables matter!
- Also when having foreign keys, the order of deleting or inserting records matter.

Example

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

boats

bid	name	color
101	interlake	red
102	clipper	green

```
INSERT INTO reserves(sid,bid,day) VALUES(20,101,TO_DATE('10/26/2022','mm/dd/yyyy');
```

If we have a foreign key defined on reserves sid,
this INSERT will return an error, because sid=20 is not present
In table sailors

Complex Constraints: Check Clause

- Useful for more general IC
- Constraints can be named
- Standalone check for single table only

Example when a constraint is needed

-
  ```
  CREATE TABLE Sailors(
          sid NUMBER(9) PRIMARY KEY,
          sname VARCHAR(20),
          rating NUMBER(2),
          age NUMBER(4,3)
  );
  ```

```
INSERT INTO Sailors VALUES(100,'joe',11,33);
```

Will work fine. But what happens if we don't want to allow ratings > 10 ?

Example constraint on attribute

```
CREATE TABLE Sailors5(
    sid NUMBER(9) PRIMARY KEY,
    sname VARCHAR(20),
    rating NUMBER(2) CHECK (rating >=1 and rating<=10),
    age NUMBER(4,3)


);
```

```
INSERT INTO Sailors5 VALUES(100,'joe',11,33);
```

Will be rejected as it violates the check constraint!

Example with naming the constraint

```
CREATE TABLE Sailors4(
      sid NUMBER(9) PRIMARY KEY,
      sname VARCHAR(20),
      rating NUMBER(2),
      age NUMBER(4,3),
     CONSTRAINT RatingRange
     CHECK (rating >=1 and rating<=10)

);
```

```
INSERT INTO Sailors4 VALUES(100,'joe',11,33);
```

Will be rejected as it violates RatingRange constraint!

Oracle Session

- PracticeSessionSQL6.sql

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators
- Integrity Constraints

Foreign Keys - Referential Integrity

- INSERT: If we try to insert a tuple whose foreign key does not exist on the table it references, the insert is rejected (i.e. error)

Example1 - Referential Integrity (Foreign Key Constraint)

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

boats

bid	name	color
101	interlake	red
102	clipper	green

```
INSERT INTO reserves(sid,bid,day) VALUES(20,101,TO_DATE('10/26/2022','mm/dd/yyyy');
```

If we have a foreign key defined on reserves sid that references sailors, then this INSERT will return an error, because sid=20 is not present in table sailors.

Example2 - Referential Integrity (Foreign Key Constraint)

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

boats

bid	name	color
101	interlake	red
102	clipper	green

DELETE FROM sailors WHERE sid=22;

If we have a foreign key defined on reserves sid that references sailors, then this delete will be rejected as it violates the integrity constraint !

DELETE FROM sailors WHERE sid=59;

This would work. Why?

Foreign Keys - Referential Integrity

- SQL/92, 99: support all these three options for DELETE and UPDATE operations

- Default is NO ACTION: which means Delete/Update operation is rejected

- CASCADE (not implemented in Oracle!): delete/update all tuples that refer to the deleted/updated tuple

- SET NULL/SET DEFAULT (not implemented in Oracle!): sets foreign key value of referencing tuple

Example (this will not work in Oracle, as this feature is not implemented)

```
CREATE TABLE Reserves(
     sid NUMBER(9),
     bid NUMBER(9),
     day DATE,
     PRIMARY KEY(sid,bid),
     FOREIGN KEY(sid) REFERENCES Sailors ON
DELETE SET DEFAULT ON UPDATE CASCADE,
     FOREIGN KEY(bid) REFERENCES Boats
);
```

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators
- Integrity Constraints
- Setting default values

Setting Default Values

```
CREATE TABLE Sailors(
        sid NUMBER(9) PRIMARY KEY,
        sname VARCHAR(20),
        rating NUMBER(2) DEFAULT 5,
        age NUMBER(4,3)
);
```

When CREATING or ALTERING a table
you can specify DEFAULT Values!

Example with default rating

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0

```
INSERT INTO Sailors(sid,sname,age)
VALUES(80,'Mary',25);
```

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0
80	Mary	5	25

```
INSERT INTO Sailors(sid,sname,rating,age)
VALUES(81,'Anne',9,35);
```

sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0
59	rusty	10	45.0
80	Mary	5	25
81	Anne	9	35

Note on Default Values

- You cannot have both NOT NULL and DEFAULT

-
  ```
  CREATE TABLE Sailors(
      sid NUMBER(9) PRIMARY KEY,
      sname VARCHAR(20),
      rating NUMBER(2) NOT NULL DEFAULT 5,
      age NUMBER(4,3)
  );
  ```

- Will return an error!

How to get all tables owned by current user

- SELECT TABLE_NAME
  FROM user_tables;

By running this command, you can see all tables that were created by you and present in the DB.

SQL

- Create, alter, delete tables
- Insert Statement
- Basic Select Statement Structure
- LIKE, AS keywords, Dates, Text case sensitivity
- Set Operations
- Aggregates
- Nested Queries
- SQL Division
- NULL Constraints
- Join Operators
- Integrity Constraints
- Setting default values
- Views

Views

- Virtual Relations that act as a relation (i.e. table)
- Data in views is not stored on disk
- Data from views is generated every time a view is accessed
- Note: Some DBMS also offer support for Materialized Views

View

```
CREATE VIEW viewName [(col1,col2,..)]
AS Query;
```

Example View creation

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)

```
CREATE VIEW
SailorsAndBoats(sid,sname,bid,bname,color,day)
AS SELECT s.sid,s.sname,b.bid,b.name,b.color,r.day
FROM sailors s, reserves r, boats b
WHERE s.sid=r.sid AND r.bid=b.bid;
```

How do we use a view

- Find the id and name of sailors who reserved green boats:

```
SELECT sid,sname
FROM SailorsAndBoats
WHERE color='green';
```

Drop view

- DROP VIEW SailorsAndBoats;

Note

- Data from View is not stored on disk
- The query inside the view gets executed when the view is invoked

Oracle Session Views

- PracticeSessionSQL7_Views.sql

Indexes

- A database Index is a data structure that improves the speed of data retrieval
- An index can be created on columns
- A Primary Key always has an index on it
- It uses additional storage

Indexes

- Pros
  - Rapid look-ups
  - Can speed up queries, if columns that have an index are in the WHERE clause
- Cons
  - Requires additional storage
  - Write (INSERT, UPDATE, DELETE) require more time

Indexes

- Beneficial to create on high-cardinality columns that appear often in WHERE clauses

How to CREATE an INDEX

```
CREATE INDEX indexName
ON tableNAME(col);

CREATE INDEX indexName
ON tableNAME(col1,col2,..);
```

Example INDEX creation

- sailors(sid:int, sname: string, rating:int, age:, salary:real)
- boats(bid:int, name:string, color:string, manufacturer:string, prod_year:date)
- reserves (sid:int, bid:int, day:date)
- Assuming we have many queries that filter (WHERE clause) by the day of reservation, it might make sense to create an index on the day column from reserves

```
CREATE INDEX indResDay
ON reserves(day);
```

Example 2 INDEX creation

- Customers(ssn: string,name:string, address:string, phone:string)
- Accounts(number: int, type:string, balance:real)
- Has(ssn:string, number:int)
- Supposing we have many queries in which we use the ssn
- SSN column already has an index because it is the primary key

Example 3 INDEX creation

- Customers(ssn: string,name:string, address:string, phone:string)
- Accounts(number: int, type:string, balance:real)
- Has(ssn:string, number:int)
- Supposing we have many queries on table Customers in which we use the filtering on phone number
- We would probably want to create an INDEX on the phone column from table Customers

```
CREATE INDEX indxPhone
ON Customers(phone);
```

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design: the Entity-Relationship Model
- Structured Query Language (SQL)
- Database Security and Authorization
- Schema Refinement and Normal Forms
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

Definitions

- Security policy
  - specifies who is authorized to do what
- Security mechanism
  - allows to enforce a chosen security policy
- Terminology
  - Users = Subjects or Principals
  - Data = Objects
- Two important functions needed to achieve security
  - Authentication (AuthN)
  - Authorization (AuthZ)

Authentication

- Establishing the identity of the user, or who the user is
- Subjects (users) present authentication credentials
  - Username/Password combination – “what user knows”
  - Digital certificates (cryptographic tokens) – “what user has”
  - Biometrics – “what user is”
- Some credential types stronger than others
  - For high-security applications, multi-factor authentication
  - E.g., password + fingerprint

Authorization

- Once we know who the user is, what can s/he access?
  - What objects (data) the subjects is allowed access to?
  - What kind of operations is the subject allowed to perform?
    - Read-only, modify, append
  - Authorization also referred to as access control
- Two main categories of access control
  - Discretionary: object owner decides authorization policy for its objects (Unix system)
  - Mandatory: system-wide rules that dictate who gets to access what (multi-level security, Bell-LaPadula)

Discretionary Access Control

- Based on the concept of access rights or privileges
  - Privileges for objects (tables and views)
  - Mechanisms for granting and revoking privileges
- Object creator automatically gets all privileges on it
  - DBMS keeps track of who subsequently gains and loses privileges
  - DBMS ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed

GRANT Command

GRANT privilege_list ON object TO user_list [WITH GRANT OPTION]

- The following privileges can be specified:
  - SELECT
    - can read all columns
    - including those added later via ALTER TABLE command
  - INSERT(col-name)
    - can insert tuples with non-null or non-default values in this column
    - INSERT means same right with respect to all columns
  - DELETE
    - can delete tuples
  - REFERENCES (col-name)
    - can define foreign keys (in other tables) that refer to this column

GRANT Command (cont.)

- If a privilege is granted with GRANT OPTION, the grantee can pass privilege on to other users
  - Special ALL PRIVILEGES privilege
- Only owner can execute CREATE, ALTER, and DROP

Example

```
create user mary identified by abc12 ;
alter user mary quota 10000k on USERS;

create user mary identified by abc12 default tablespace
USERS temporary tablespace TEMP;
grant connect, resource to mary;
```

Other Example

```
create user <user> identified by <pass>
default tablespace USERS temporary
 tablespace TEMP;
grant connect, resource to <user>;

alter user <user> quota 10000k on USERS;
grant create view to <user>;
```

Examples

GRANT INSERT, SELECT ON Sailors TO Horatio

- Horatio can query Sailors or insert tuples into it

GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION

- Yuppy can delete tuples, and also authorize others to do so

GRANT INSERT (*rating*) ON Sailors TO Dustin

- Dustin can insert (only) the *rating* field of Sailors tuples

Revoke Command

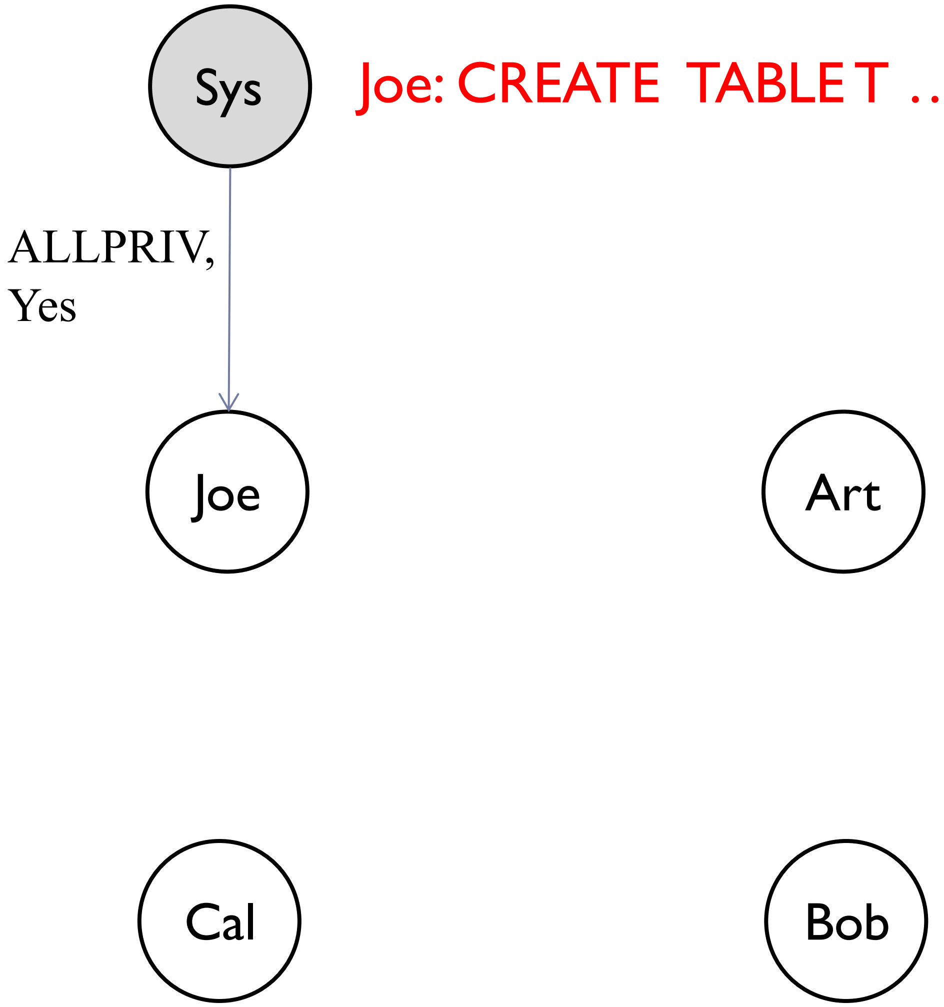
REVOKE [GRANT OPTION FOR] privilege_list ON object
FROM user_list [CASCADE | RESTRICT]

- REVOKE
  - Revokes privileges
- CASCADE: when a privilege is revoked from X, it is also revoked from all users who got it *solely* from X
  - Privilege is said to be ABANDONED
  - A graph with the granting relationship is maintained
- RESTRICT: if revoke causes some privilege to be abandoned, it is NOT executed

Authorization Graph

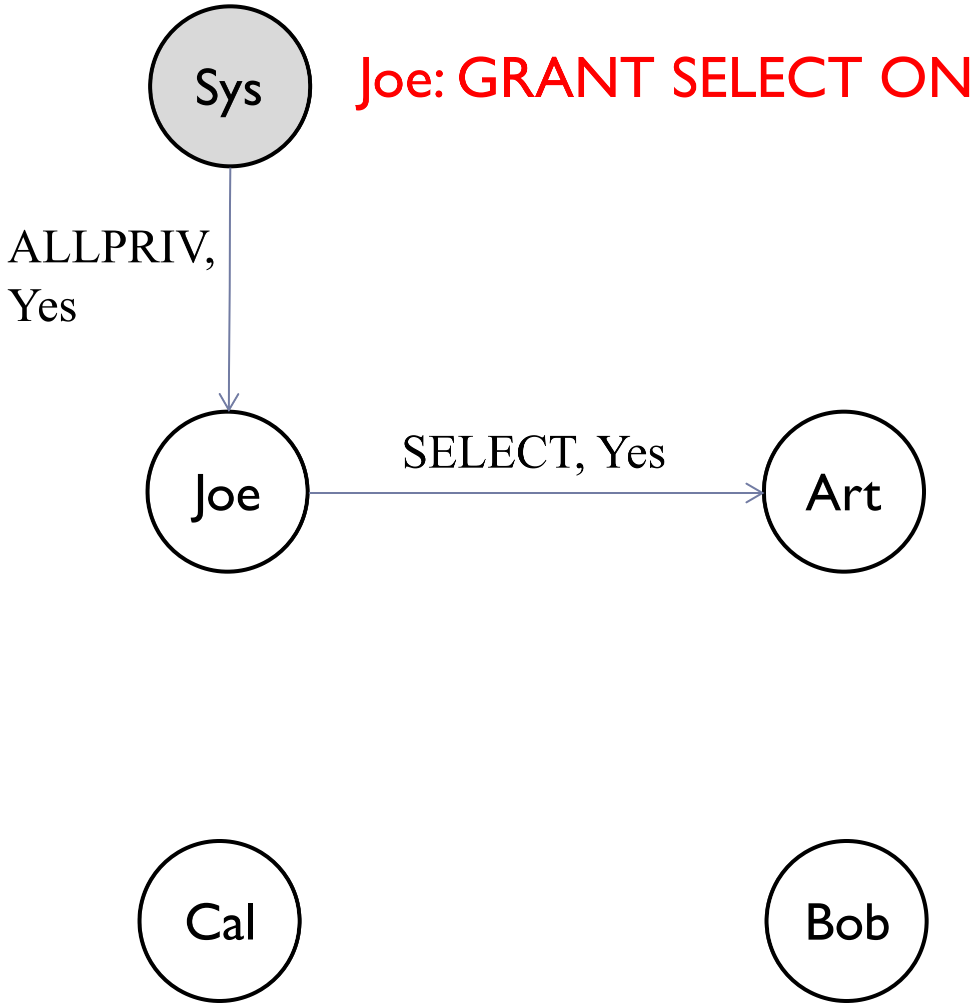
- Keeps track of active authorization on objects
  - Each authorization ID (user) corresponds to a node
  - Granting a privilege adds labeled edge to graph
  - Removing privilege deletes one or more edges from graph
  - Special “System” node that originates all privileges
  - Note: it is possible to have multiple edges between same pair of nodes (with same direction)!
- How to determine if access is allowed for an ID?
  - There must be a path from System to that ID formed of privileges equal (or stronger) than the one required

Authorization Graph



Joe: CREATE TABLE T …

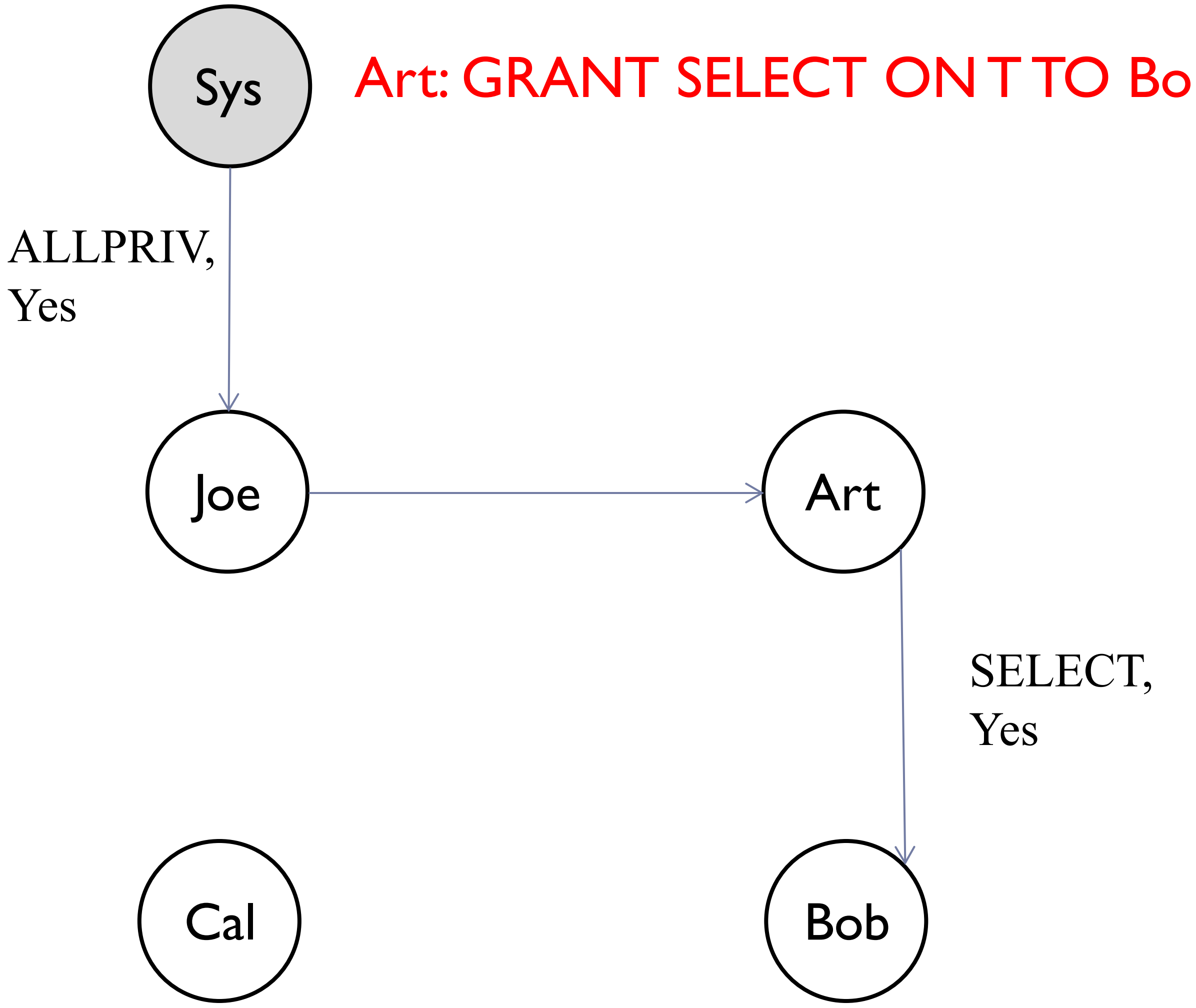
Authorization Graph



Joe: GRANT SELECT ON T TO Art WITH GRANT OPTION

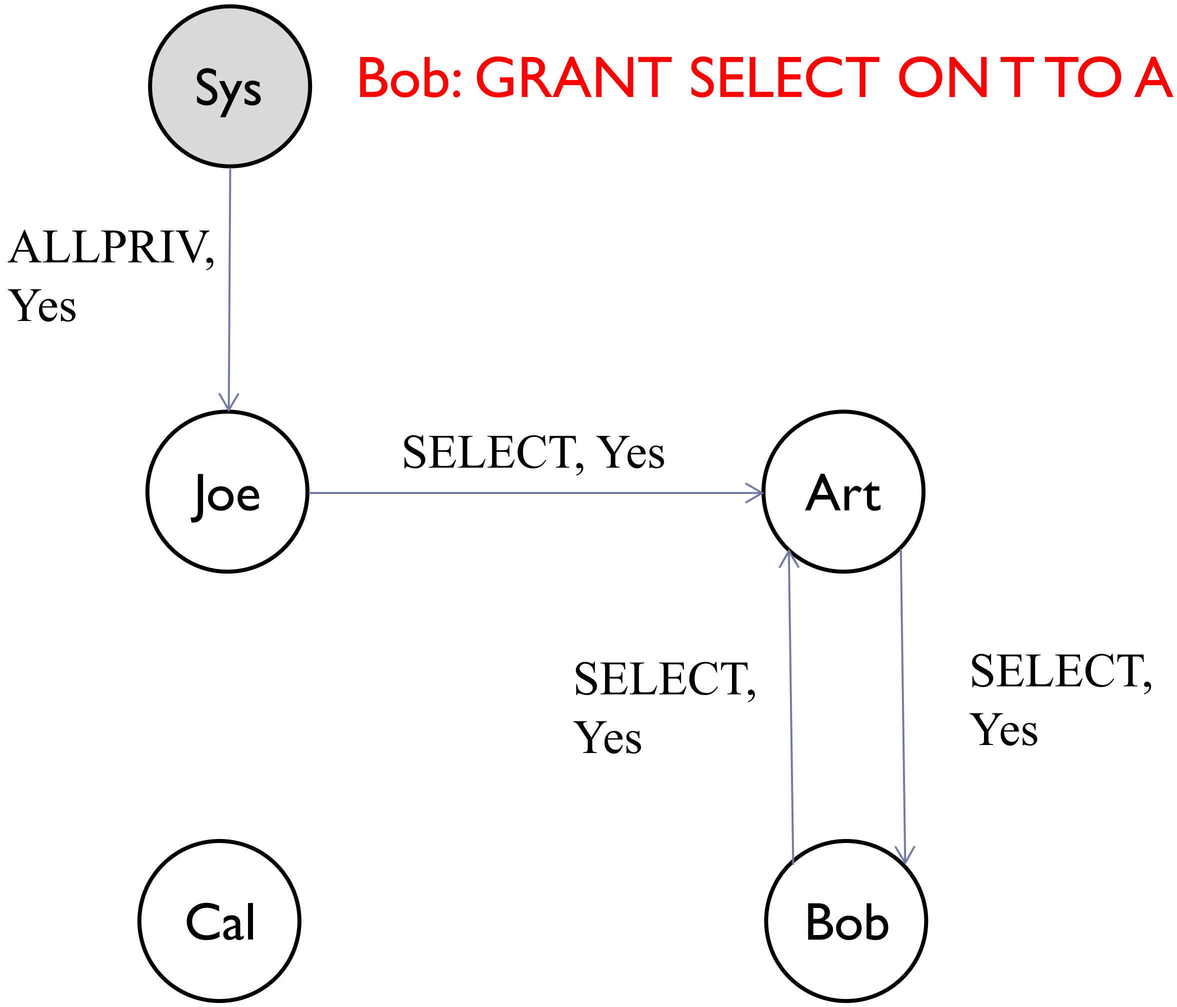
Authorization Graph

Art: GRANT SELECT ON T TO Bob WITH GRANT OPTION



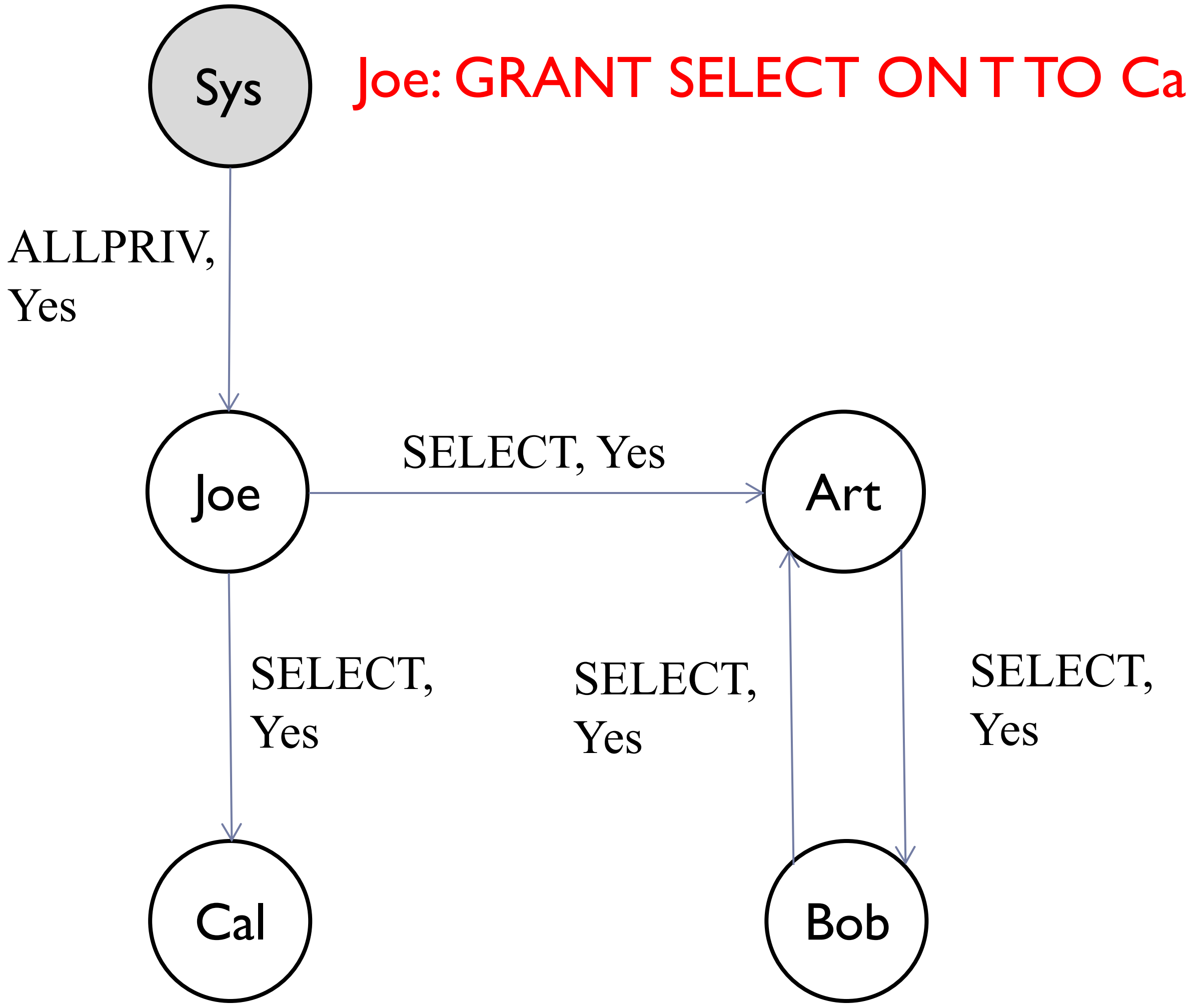
Authorization Graph

Bob: GRANT SELECT ON T TO Art WITH GRANT OPTION



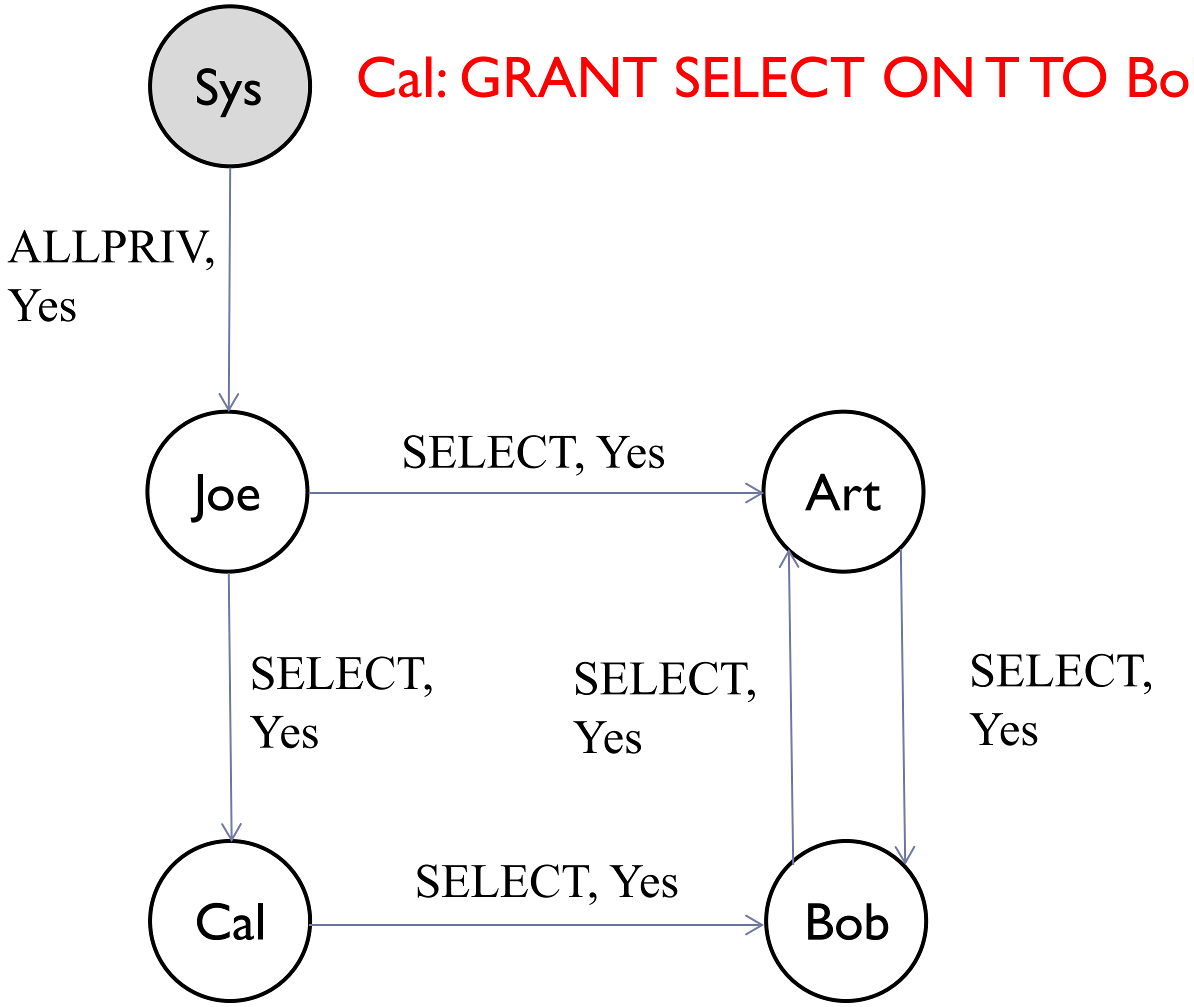
Authorization Graph

Joe: GRANT SELECT ON T TO Cal WITH GRANT OPTION

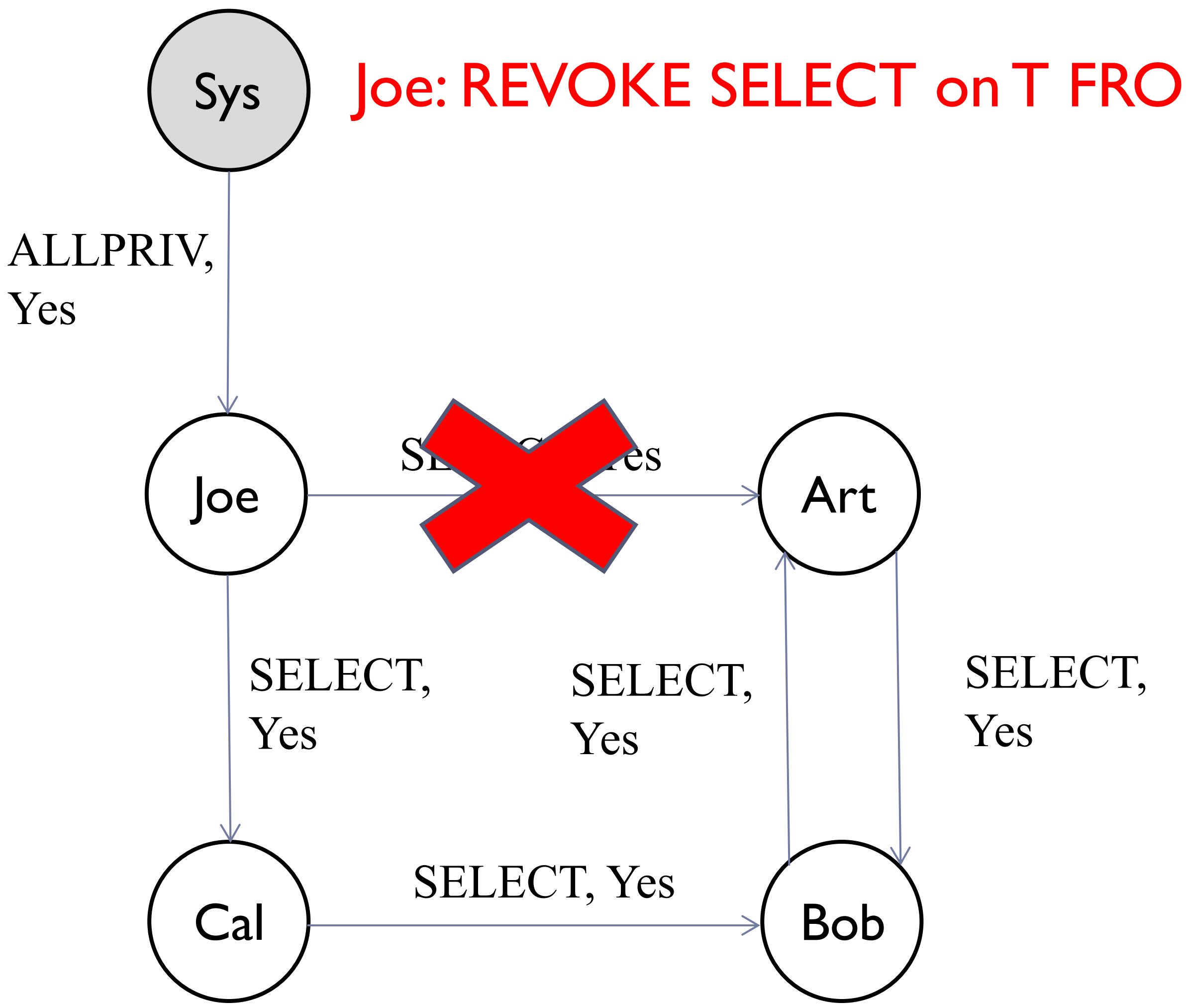


Authorization Graph

Cal: GRANT SELECT ON T TO Bob WITH GRANT OPTION

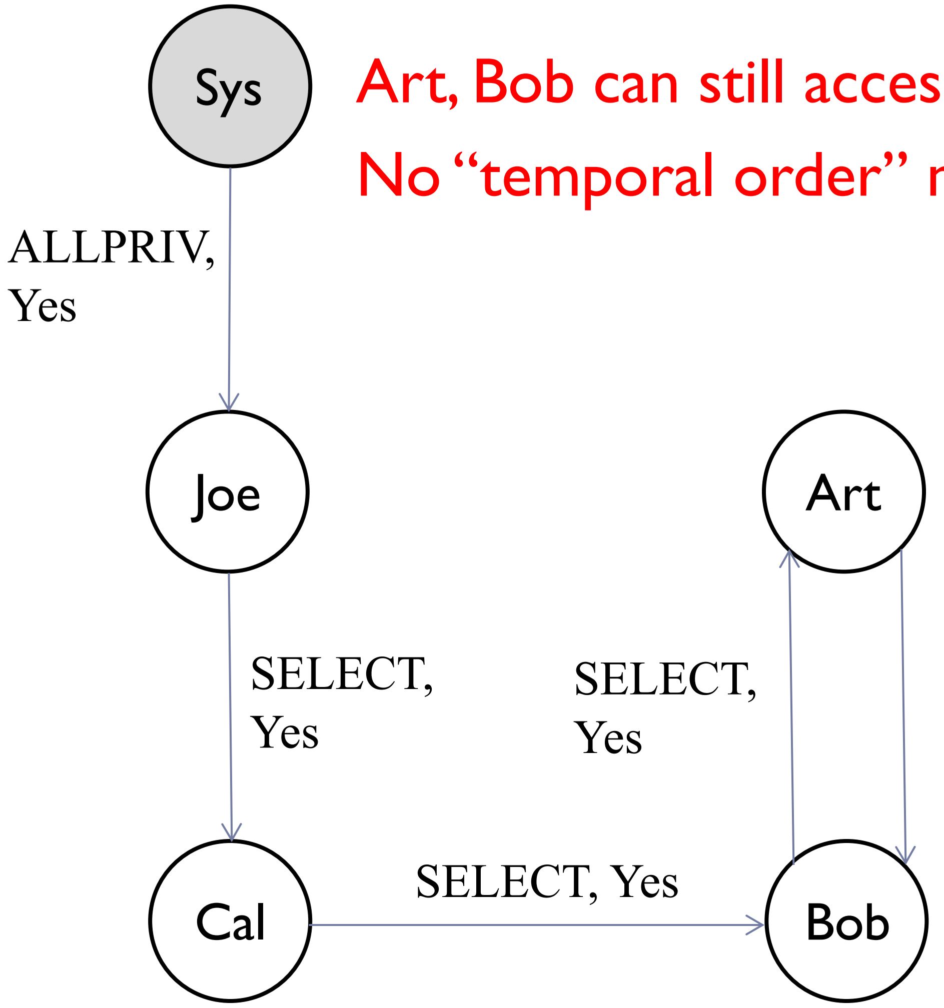


Authorization Graph



Joe: REVOKE SELECT on T FROM Art CASCADE

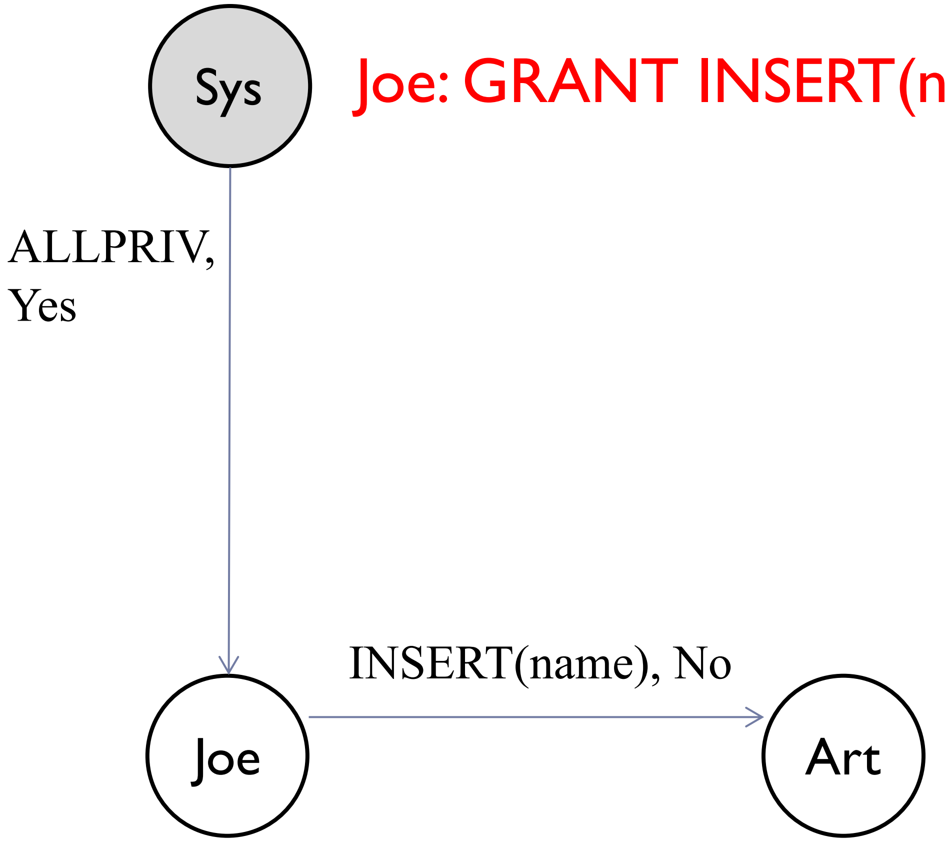
Authorization Graph



Art, Bob can still access T!

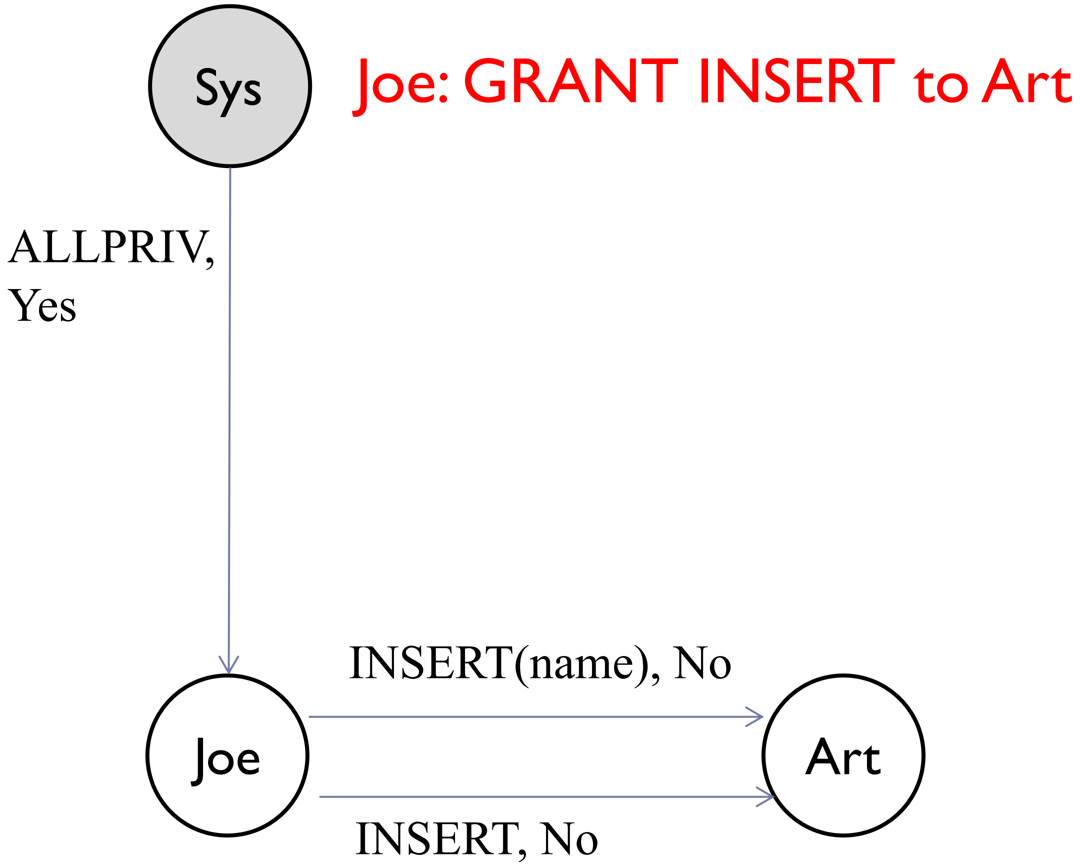
No "temporal order" memorized

Another Example

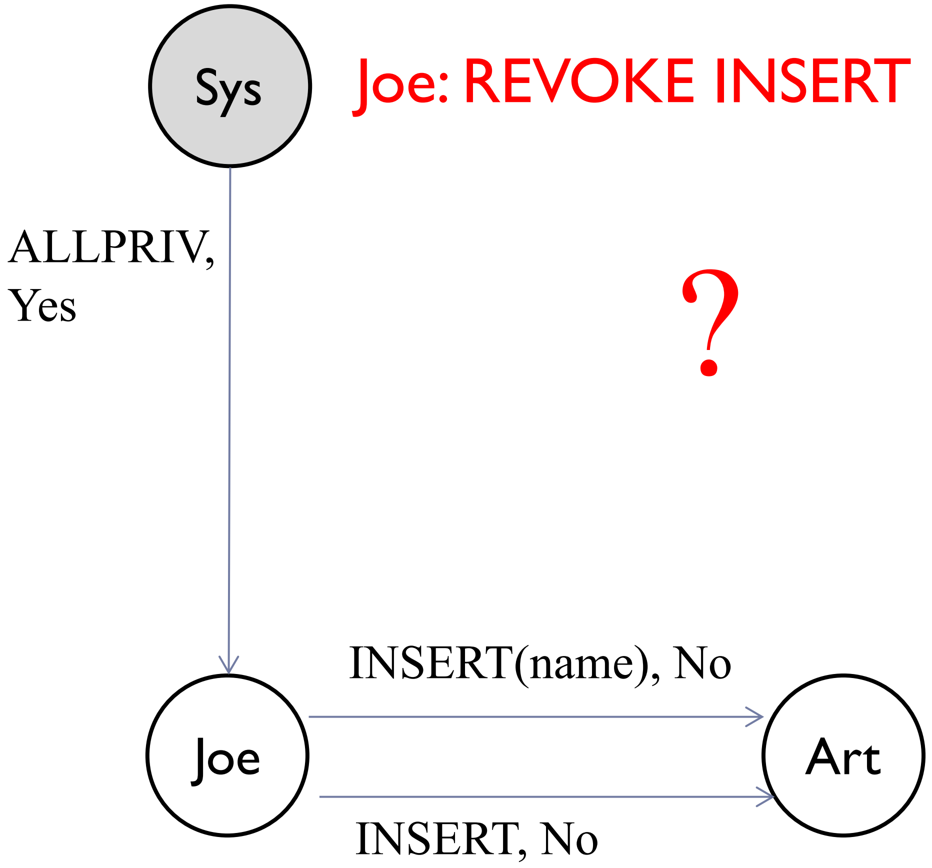


Joe: GRANT INSERT(name) to Art

Another Example



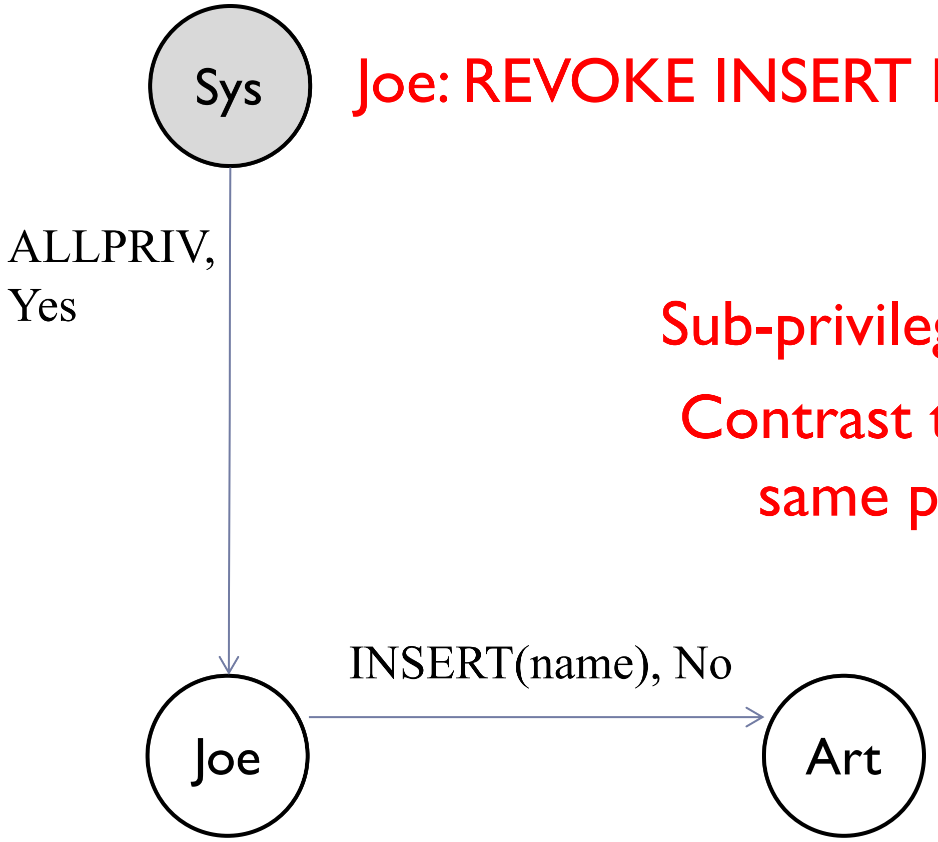
Another Example



Joe: REVOKE INSERT FROM Art

?

Another Example



Joe: REVOKE INSERT FROM Art

Sub-privileges NOT revoked

Contrast this with granting same privilege twice!

Security at the Level of a Field!

- Can create a view that only returns one field of one tuple
  - Then grant access to that view accordingly
- Allows for *arbitrary* granularity of control, *but*:
  - Tedious to specify and maintain policies
  - Performance is unacceptable
    - Too many view creations and look-ups
- Another solution
  - Attach labels to subjects and objects
  - Create rules of access based on labels

Mandatory Access Control

- Based on system-wide policies that cannot be changed by individual users (even if they own objects)
  - Each DB object is assigned a security class
  - Each subject (user or user program) is assigned a clearance for a security class
  - Rules based on security classes and clearances govern who can read/write which objects.
- Many commercial systems do not support mandatory access control
- Some specialized versions do
  - e.g., those used in military applications

Bell-LaPadula Model

- Security classes:
  - Top secret (TS)
  - Secret (S)
  - Confidential (C)
  - Unclassified (U):
  - TS > S > C > U
- Each object (O) and subject (S) is assigned a class
  - S can read O only if class(S) >= class(O) (Simple Security Property or No Read Up)
  - S can write O only if class(S) <= class(O) (*-Property or No Write Down)

Intuition

- Idea is to ensure that information can never flow from a higher to a lower security level
- The mandatory access control rules are applied in addition to any discretionary controls that are in effect

Questions?