

CptS 451- Introduction to Database Systems

SQL : Views and Indexes (DMS Ch-25.8)

Instructor: Sakire Arslan Ay



World Class. Face to Face.

Views, Security, Indexes



Topics

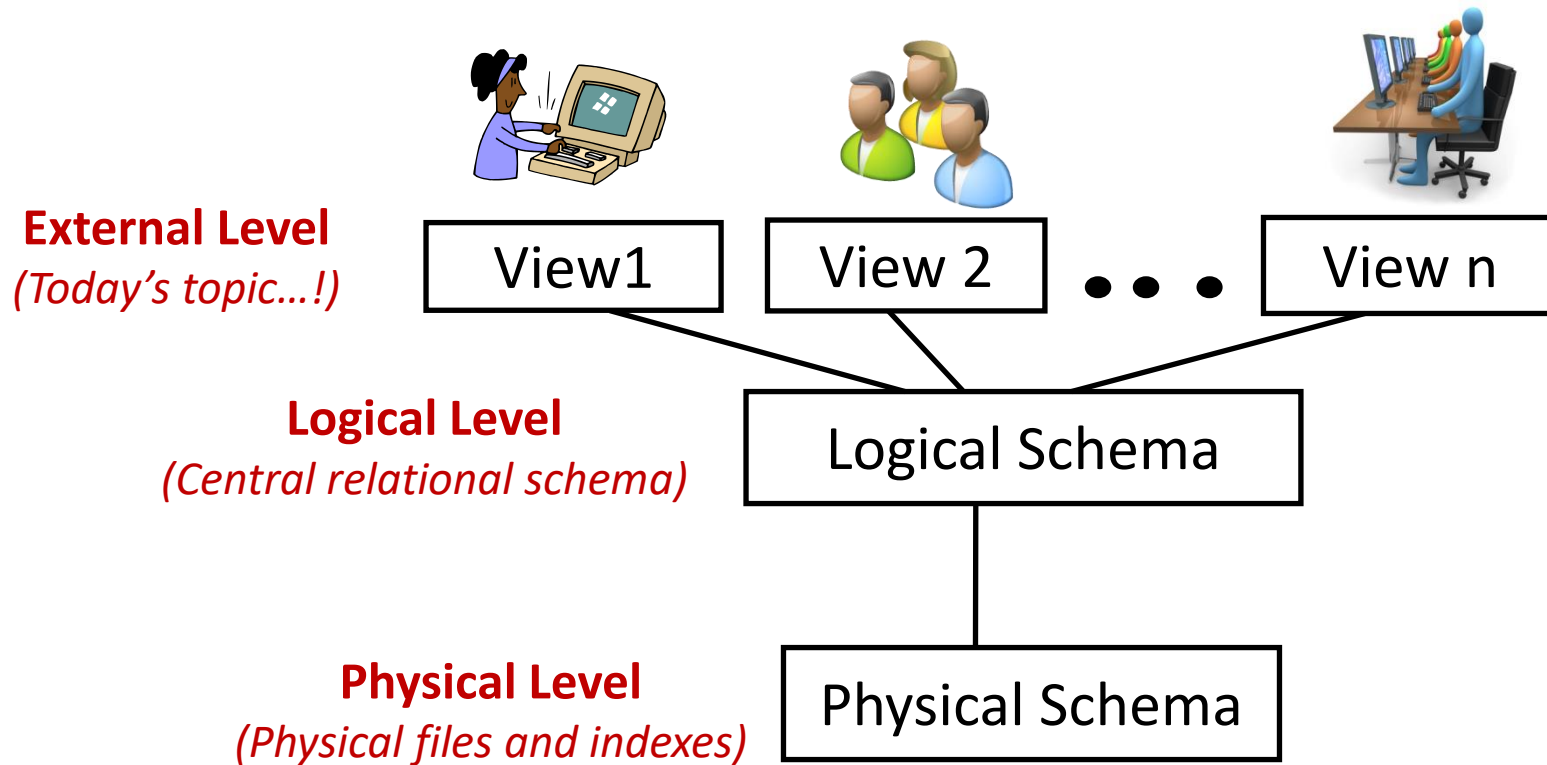
- SQL Views
- Indexes in SQL

SQL Views & Security

- Outline
 - The three-schema architecture
 - View definitions and query execution
 - Views and updates
 - View updates using triggers
 - Views for data independence

Three-Layer Schema Architecture

- Three levels of schemas

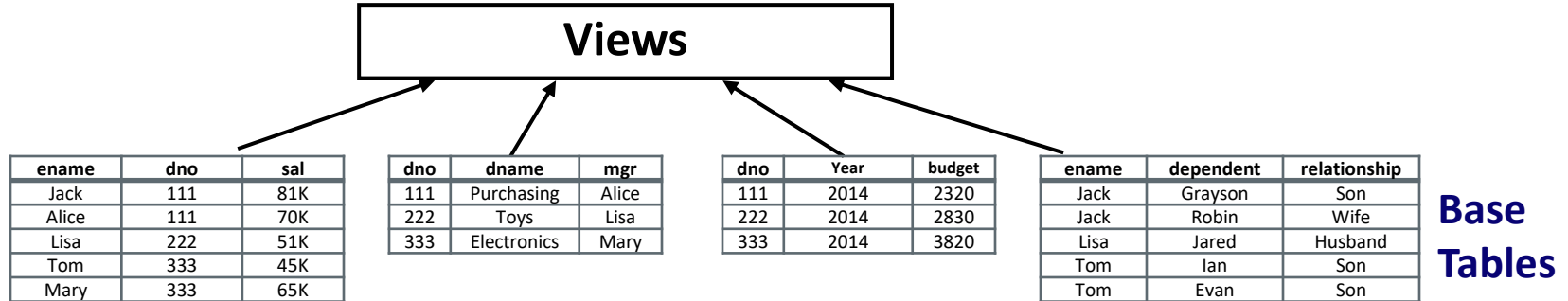


Views

- SQL syntax:

Virtual: **CREATE VIEW <name> AS <query>;**

Materialized: **CREATE VIEW [MATERIALIZED]<name> AS <query>;**



- The uses of “views” include:
 - Provide *logical data independence*
 - Serve as a unit of access control for security purposes
 - Simplify common/complex queries by predefining useful “tables”
 - Tailor DB schema to meet the needs of different applications

Views

- A **view** is a relation defined in terms of stored tables (called **base tables**) and other views.
- Two kinds:
 1. **Virtual** = not stored in the database; just a query for constructing the relation.
 2. **Materialized** = actually constructed and stored.

View Examples

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Dept(dno, dname, mgr)

dno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

- **Example-1:** Employees in the “Purchasing” department.
 - Their names, salaries, departments, and managers

```
CREATE VIEW PurchEmpInfo AS
  SELECT ename, sal, Emp.dno, mgr
  FROM Emp, Dept
  WHERE Emp.dno = Dept.dno AND dname = 'Purchasing';
```

View Examples

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Dept(dno, dname, mgr)

dno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

- **Example-1:** Employees in the “Purchasing” department.
 - Their names, salaries, departments, and managers

```
CREATE VIEW PurchEmpInfo AS
SELECT ename, sal, Emp.dno, mgr
FROM Emp, Dept
WHERE Emp.dno = Dept.dno AND dname = 'Purchasing';
```


View Examples

- **Example-2:** Employees in the Purchasing department.
 - Their names, salaries, and managers

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Dept(dno, dname, mgr)

dno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

Some attributes can be dropped (e.g., dno)
and others renamed (e.g., mgr)

PurchEmpInfo (ename, sal, manager)

ename	sal	manager
Jack	81K	Alice
Alice	70K	Alice

```
CREATE VIEW PurchEmpInfo AS
```

```
SELECT ename, sal, mgr AS manager
```

```
FROM Emp, Dept
```

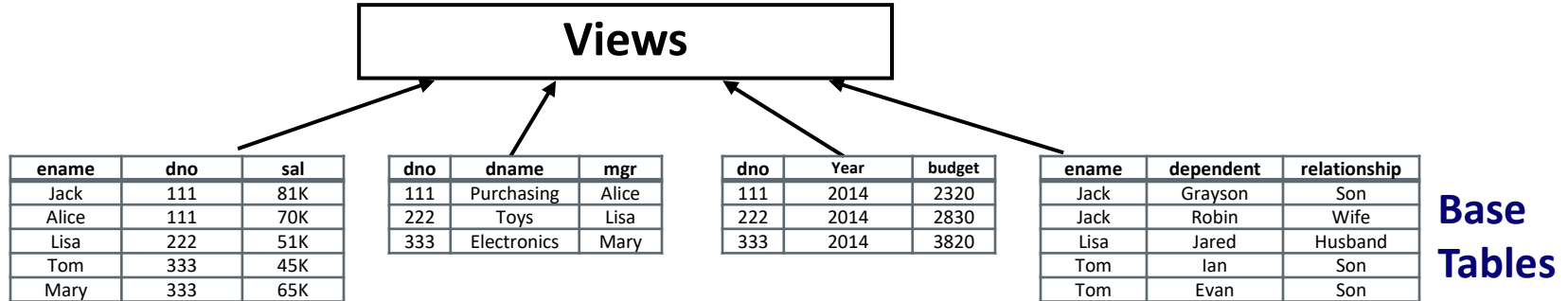
```
WHERE Emp.dno = Dept.dno AND dname = 'Purchasing';
```

Views

- SQL syntax:

Virtual: **CREATE VIEW <name> AS <query>;**

Materialized: **CREATE VIEW [MATERIALIZED]<name> AS <query>;**



- The uses of “views” include:
 - Provide *logical data independence*
 - Serve as a unit of access control for security purposes
 - Simplify common/complex queries by predefining useful “tables”
 - Tailor DB schema to meet the needs of different applications

Views for Data Independence

- Make existing programs compatible with DB schema changes
 - Ex: `Emp(emp, dno, sal)`, `Dept(dno, dname, mgr)`
(All applications currently use this old schema)
 - Suppose for some reason we change the schema to:
`E(emp, deptno, sal)`, `D(deptno, dname, mgr)`
(Old applications will not work with this new schema!)
 - Solution: using views, recreate the **old** schema from the **new** schema

`CREATE VIEW Emp(ename, dno, sal) AS ...`
`CREATE VIEW Dept(dno, dname, mgr) AS ...`

→ *Now old applications and queries will still run (modulo updates)!*

Emp (ename, dno, sal)		
ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Dept(dno, dname, mgr)		
dno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

Views for Data Independence (cont.)

Old schema

Queries

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Dept(dno, dname, mgr)

dno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

New schema

Queries

CREATE VIEW Emp(ename, **dno**, sal) AS
SELECT ename, **deptno**, sal FROM E;

CREATE VIEW Dept(**dno**, dname, mgr) AS
SELECT **deptno**, dname, mgr FROM D;

E (ename, **deptno**, sal)

ename	deptno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

D (**deptno**, dname, mgr)

deptno	dname	mgr
111	Purchasing	Alice
222	Toys	Lisa
333	Electronics	Mary

View Examples

- **Example-3:** A department summary for managers to use
 - Department dno s, number of employees and average salaries.

```
CREATE VIEW DeptSummary AS
  SELECT d.dno, COUNT (*) as headcnt, AVG(e.sal) as avgsal
  FROM Dept d, Emp e
  WHERE e.dno = d.dno
  GROUP BY d.dno;
```

deptSummary(dno, dname, mgr, headcnt, avgsal)

dno	headcnt	avgsal
111	2	70K
222	1	80K
333	2	65K

- With this view, managers don't have to be looking at multiple tables, doing joins, doing aggregations, etc.

Queries on Views

- Views can be used just like other tables in queries

```
SELECT ename,manager
FROM PurchEmpInfo
WHERE sal>=35000;
```

```
SELECT dno, avgsal
FROM DeptSummary
WHERE headcnt>1;
```

- How does the system answer a query on views?
 - “Replace” it (using view definitions) to get a query over base relations

```
SELECT S_emp.ename, S_emp.manager
FROM (SELECT ename, sal, mgr as manager
      FROM Emp, Dept
      WHERE Emp.dno = Dept.dno AND dname = 'Purchasing') AS S_emp
WHERE S_emp.sal>= 35000;
```

←(Note: FROM clause may contain subqueries!)

Queries on Views

- Note that views can be layered on other views (and so on)...
 - Example:

```
CREATE VIEW costlyDepts AS  
  SELECT dno, headcnt, avgsal, headcnt*avgsal AS spending  
  FROM deptSummary  
  WHERE avgsal >= 70000;
```

costlyDepts(dno, headcnt, avgsal, spending)

dno	headcnt	avgsal	spending
222	2	70K	140K
111	1	80K	80K

Modifying Views

- How can we modify a view if it is supposedly virtual?
 - INSERT, DELETE, UPDATE
- Many views cannot be modified, as we will soon see
- Some views can be modified (called updatable views)
 - Example:

INSERT INTO PurchEmpInfo VALUES ('Pete', 111);

Modifications on Views

```
CREATE TABLE Emp(
  ename char(20) PRIMARY KEY,
  dno int,
  sal float default 0);
```

```
CREATE VIEW PurchEmpInfo AS
  SELECT ename, dno
  FROM Emp
  WHERE dno = 111;
```

```
INSERT INTO PurchEmpInfo VALUES ('Pete', 111);
```

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K
Pete	111	0

PurchEmpInfo(ename, dno)

ename	dno
Jack	111
Alice	111
Pete	111

- Insertion of a tuple into a view
 - Insertion of corresponding tuple(s) into its base table(s)
 - Missing column values lead to insertion of NULL or default values
 - The inserted base tuple(s) should generate the new view tuple(s)!

Modifications on Views

```
CREATE TABLE Emp(
  ename char(20) PRIMARY KEY,
  dno int,
  sal float default 0);
```

```
CREATE VIEW PurchEmpInfo AS
  SELECT ename
  FROM Emp
  WHERE dno = 111;
```

```
INSERT INTO PurchEmpInfo VALUES ('Pete');
```

Emp (ename, dno, sal)

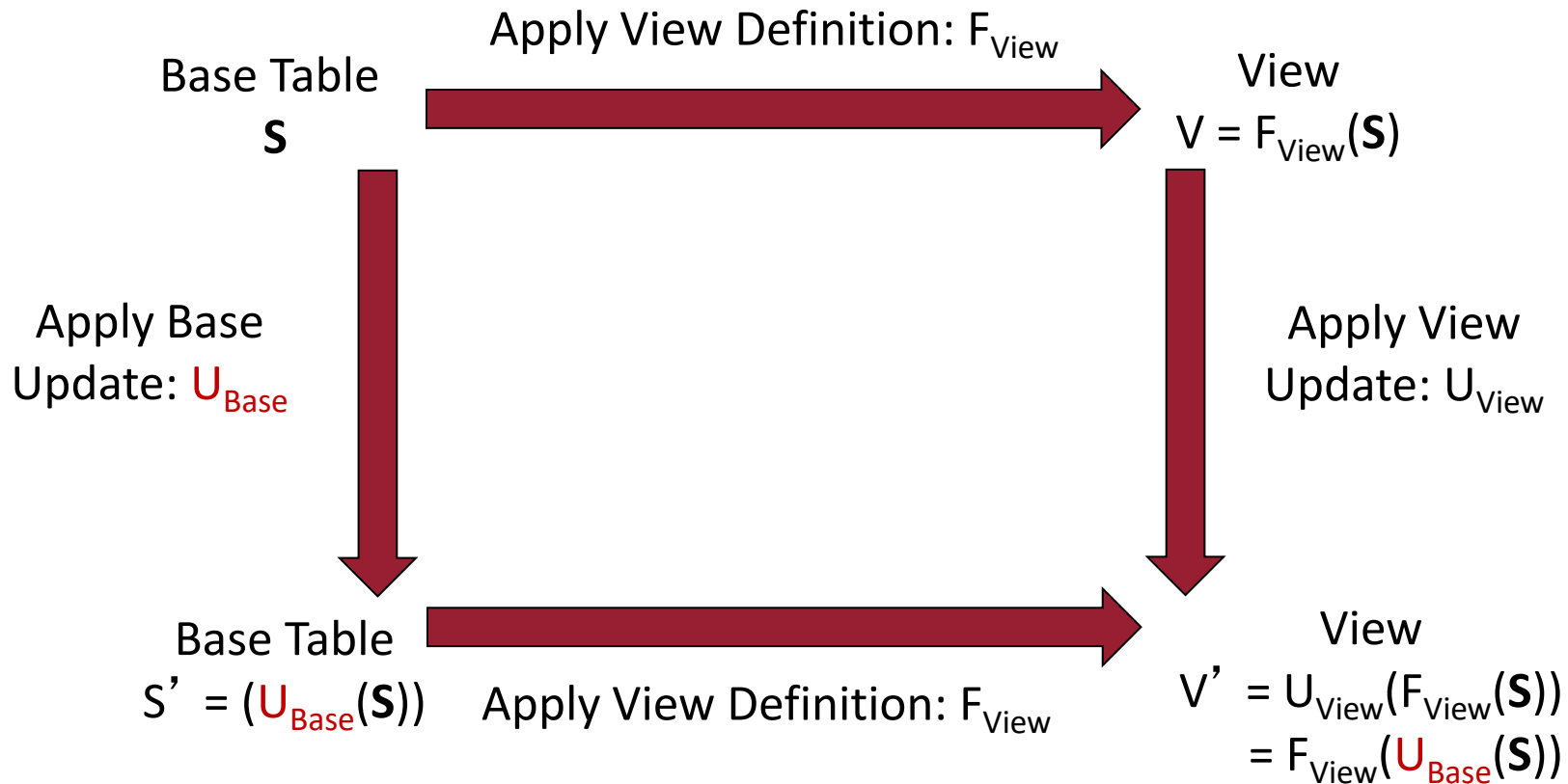
ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K
Pete	????	0

PurchEmpInfo(ename)

ename
Jack
Alice
Pete

- Insertion of a tuple into this view
 - Not allowed: What would we insert into Emp.dno? → View not updatable!
 - The DBMS isn't "smart" enough to "know" that the inserted dno value should be 111
 - If dno was inserted as NULL, the new view tuple ("Pete") would not be generated

Principles of Updatable Views



Modifying Views

- Updatable views:
 - An **updatable view** **V(R)**'s definition must satisfy certain requirements (SQL92):
 - **FROM**: Consists of one occurrence of R (and no other relations)
 - **WHERE**: Does not involve R in a subquery
 - **SELECT**: Includes “enough” attributes, has only simple field references (no arithmetic expressions or aggregates), and not **DISTINCT**
 - No **GROUP BY** or **HAVING** clauses
 - A modification of V is translated to a corresponding modification of R
 - Note: Some vendors support more or less in this area

View Updates Using Triggers

- Remember triggers: ECA rules
 - **Event:** INSERT, DELETE, UPDATE
 - **Condition:** WHEN <some condition on table>
 - **Action:** Some operation
- Use **INSTEAD OF** triggers on views
 - Specify what to do in case of an update of the view, e.g.:

```
CREATE TRIGGER PurchEmpInsert
INSTEAD OF INSERT on PurchEmpInfo
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Emp (ename, dno)
VALUES (NewRow.ename, 111);
```

Deletes from Updatable Views

- When deleting tuples from a view, system should delete all tuples from base table(s) that can potentially produce the deleted view tuples (and only those tuples, of course)
- Example:

```
DELETE FROM PurchEmpInfo
WHERE ename = 'Jack';
```

will be translated to:

```
DELETE FROM Emp
WHERE ename = 'Jack' AND dno = 111;
```

```
CREATE VIEW PurchEmpInfo
AS
SELECT ename, dno
FROM Emp
WHERE dno = 111;
```

PurchEmpInfo(ename, dno)

ename	dno
Jack	111
Alice	111

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	81K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Updates to Updatable Views

- Will update all tuples in the base relations that produce the updated tuples in the view
- Example:

```
CREATE VIEW PurchEmpInfo AS
SELECT ename, dno, sal
FROM Emp
WHERE dno = 111;
```

```
UPDATE PurchEmpInfo SET sal = sal * 0.9
WHERE ename = 'Jack'
```

will be translated to:

```
UPDATE Emp SET sal = sal * 0.9
WHERE ename = 'Jack' AND dno = 111;
```

PurchEmpInfo(ename,

ename	dno	sal
Jack	111	72.9K
Alice	111	70K

Emp (ename, dno, sal)

ename	dno	sal
Jack	111	72.9K
Alice	111	70K
Lisa	222	51K
Tom	333	45K
Mary	333	65K

Materialized Views

- `CREATE MATERIALIZED VIEW <name> AS <query>;`
- *Ex:* `CREATE MATERIALIZED VIEW deptSummary AS ... ;`
- **Problem:** each time a base table changes, the materialized view may change.
 - Cannot afford to re-compute the view with each change.
- **Solution:** A number of policy options are available
 - Incremental maintenance → continual consistency
 - Periodic maintenance → e.g., refresh every 24 hours

Example: A Data Warehouse

- Wal-Mart stores every sale at every store in a database.
- Overnight, the sales for the day are used to update a *data warehouse* which includes materialized views of the sales.
- The warehouse is used by analysts to predict trends and move goods to where they are selling best.

Other Aspects of Views

- Dropping views
 - **DROP VIEW <name>;**
 - *Ex:* **DROP VIEW PurchEmpInfo;**
 - Base tables are NOT affected in any way
 - May impact other views/applications

Indexes in SQL

Indexes

- **Index** : data structure used to speed access to tuples of a relation, given values of one or more attributes.
- For example:
 - a hash index, or
 - a balanced search tree with giant nodes (a full disk page) called a **B+ tree**.

Declaring Indexes

- No standard!
- Typical syntax:

CREATE INDEX dnoInd **ON** Dept(dno);

CREATE INDEX emplInd **ON** Emp(ename,dept);

Using Indexes

- Assume relation R has an index on attributes A1, A2. Given the values v1,v2 for A1,A2; the index takes us to only those tuples that have v1,v2.
- **Example:**

```
CREATE INDEX emplInd ON Emp(ename,dno);
```

```
SELECT ename, dno, sal  
FROM Emp  
WHERE ename='Jack' AND dno = 111;
```

1. Use **emplInd** to get all the employees whose names are “Jack” and that work in department 111
2. Then return the names, department numbers and salaries of those employees.

Using Indexes (cont.)

- **Example:**

```
CREATE INDEX dnoInd ON Emp(dno);
```

```
CREATE INDEX emplInd ON Emp(ename);
```

```
SELECT ename, dno, sal  
FROM Emp  
WHERE ename='Jack' AND dno = 111;
```

Using Indexes (cont.)

- **Example:**

```
CREATE INDEX dnoInd ON Dept(dno);
```

```
CREATE INDEX empDnoInd ON Emp(ename,dno);
```

```
SELECT ename, Emp.dno, sal, mgr  
FROM Emp, Dept  
WHERE ename='Jack' AND Emp.dno=Dept.dno;
```

1. Use **empDnoInd** to get all the employees whose names are “Jack”.
2. Use **dnoInd** to get the departments whose dno s match the tuples in step1.
3. Perform the join on tuples from step1 and 2.
4. Then return the names, department numbers, salaries, and managers of those employees.

Database Tuning

- A major problem in making a database run fast is deciding which indexes to create.
- **Benefit:** An index speeds up queries that can use it.
- **Tradeoff:** An index slows down all modifications (INSERT,DELETE,UPDATE) on its relation because the index must be modified too.

Useful Indexes

- Often, the most useful index on a relation is an index on the relation's key.
 - The keys are usually used frequently in queries
 - Since each tuple has a unique value for the key, the search on index returns either nothing or a single tuple.
 - i.e., a single page access
- On some occasions, an index can be effective, even if the index is not on a key
 - If the attribute is almost a key, i.e., only a few tuples share the same value for the index attribute(s).
 - If the tuples are clustered (on disk) based on the index attribute(s).

Best Indexes to Create

- Will come later:
 - Storage and Indexing

SQL

- Go update your resumes, adding “SQL” to your list of “languages spoken”...!