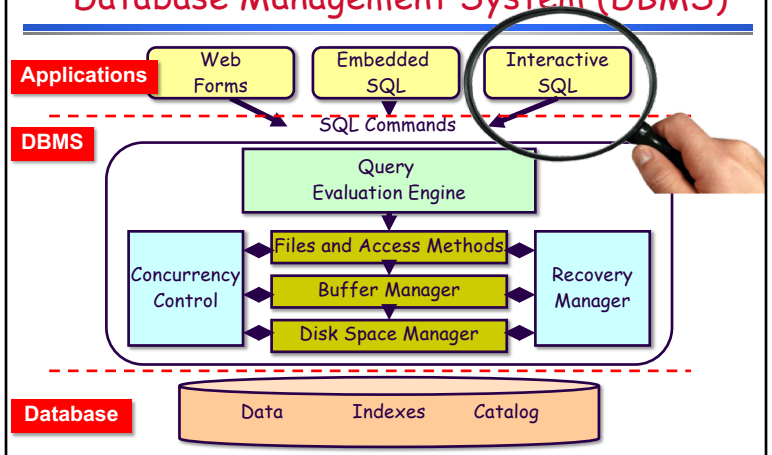
SQL Views

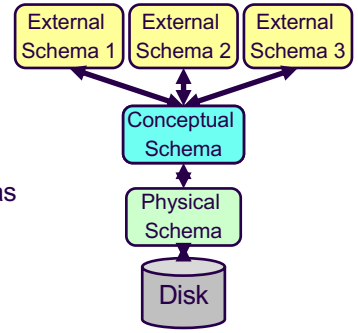
- Views
- Materialized views
- Temporary tables

Database Management System (DBMS)



Levels of Abstraction in a DBMS

- The data in a DBMS is described at three levels of abstraction:
  1. *Conceptual Schema*
  2. *Physical Schema*
  3. *External Schema*



- Many external schemas
- plus one conceptual
- plus one physical

External Schema

Course Name	*Total Enrollment*

Course Name	*Total Enrollment*
DB	2
SW	0
OS	1

SID	*Name*	*Age*	*GPA*
546007	Peter	18	3.8
546100	Bob	19	3.65
546500	Bill	20	3.7

CID	*CName*
CSC 343	DB
CSC 207	SW
CSC 369	OS

SID	*CID*	*Grade*
546007	CSC 343	A
546007	CSC 369	B+
546100	CSC 343	B

Create View

- A view is a table derived from base tables and other views
- Views can be queried as if they were base tables

```
CREATE VIEW CS_STUDENT
   AS SELECT *
   FROM STUDENT
   WHERE Major = 'CS';
```

```
SELECT Class, Count(*)
FROM CS_STUDENT
GROUP BY Class;
```


What is a view?

- It is a table:
  - as it can be queried just like a table!
- It is not a table:
  - as it does not physically exist!
- A view is a **"virtual table"** derived from **base tables**
- A view is a **"named query"**

Advantages of Views

1. Logical independence
2. For **convenience** and clarity when writing queries
   - Views can be used just like tables
3. For **security**
   - Different data **access privileges** can be given to different users (i.e., authorization)

Query Rewriting

View

```
CREATE VIEW CS_Students AS
SELECT name, age
FROM Student
WHERE Major = 'CS';
```

Original Query (user)

```
SELECT name
FROM CS_Students
where age > 19;
```

Modified Query (DBMS)

```
SELECT name
FROM Student
WHERE Major = 'CS'
AND age>19;
```

Modify & Drop a View

- Modify a view

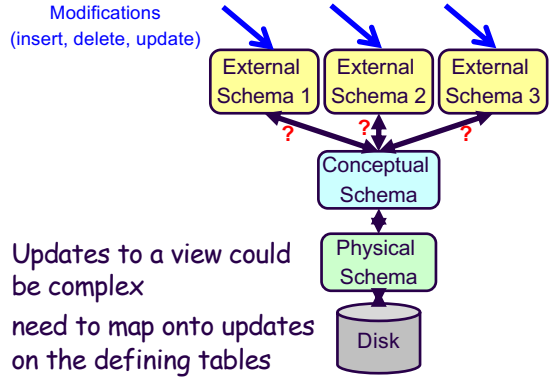
```
CREATE OR REPLACE VIEW CS_STUDENT (Class, Num)
    AS SELECT Class, COUNT(*)
    FROM STUDENT
    WHERE Major = 'CS'
    GROUP BY Class;
```

- Dropping a view:

```
DROP VIEW CS_STUDENT;
```

- Note: No **`REPLACE`** *only* **`CREATE MATERIATIZED VIEW`** in Oracle

View Updatability



- Updates to a view could be complex
- need to map onto updates on the defining tables

Example 1

Student

SID	*Name*	*Age*	*GPA*	*Major*
546007	Peter	18	3.8	CS
546100	Bob	19	3.6	EE
546500	Peter	20	3.0	CS

Updating a view:

```
UPDATE HStudents
SET gpa= 4.0
WHERE sid=546007;
```

✓

```
CREATE VIEW HStudents AS
SELECT sid, gpa
FROM Student
WHERE gpa> 3.5;
```


Example 2

Student

SID	*Name*	*Age*	*GPA*	*Major*
546007	Peter	18	3.8	CS
546100	Bob	19	3.6	EE
546500	Peter	20	3.0	CS

Updating a view:

```
INSERT INTO
      HStudents
      VALUES (Jane, 4.0);
```

✗

What is Jane's SID?!

```
CREATE VIEW HStudents AS
SELECT name, gpa
FROM Student
WHERE gpa> 3.0;
```

Example 3

Student

SID	Name	Age	GPA	Major
546007	Peter	18	3.8	CS
546100	Bob	19	3.6	EE
546500	Peter	20	3.0	CS

Updating a view:

```
UPDATE Majors
SET agpa= 3.6
WHERE major='CS';
```

✗

```
CREATE VIEW Majors (major, agpa) AS
SELECT major, avg(gpa)
FROM Student
GROUP BY Major;
```

Infinite possibilities of values!

View Updateability

- In general, a view is called **updateable** if:

  all updates on the view can be unambiguously translated back to tuples in the base tables

- A view update is **unambiguous** if:

  Only one update on the base tables can accomplish the desired update effect on the view

- In general, a view is **not updateable** if:

  an update on a view can be mapped to more than one possible update on the base tables

SQL Standard for View Updateability

1. A view with a single defining table is updatable if the view attributes contain the primary key
2. Views defined using aggregate functions are not updatable
3. Views defined on multiple tables using joins are generally not updatable

Updating a View

```
CREATE VIEW CS_STUDENT
   AS SELECT *
   FROM STUDENT
   WHERE Major = 'CS';
```

```
INSERT INTO CS_STUDENT (128, 'Ping Chen', 'CS');
UPDATE CS_STUDENT
  SET Name = 'Shimin Chen'
  WHERE SID = 128;
```

Migrating Tuples

- What is the outcome of the update:

```
UPDATE CS_STUDENT
  SET Major = 'MATH'
  WHERE SID = 128;
```

- Migrating tuples out of updateable views: an update or insert may eliminate a tuple from the view
- Prevent migration with WITH CHECK OPTION

```
CREATE VIEW CS_STUDENT
AS SELECT * FROM STUDENT WHERE Major = 'CS'
WITH CHECK OPTION;
```


Efficient View Implementation

- A DBMS implements views in two ways:
  - ✓ Query Rewriting / Modification
  - ✓ View Materialization
- With expected trade-offs…

Query Rewriting

- Query rewriting:
  - presents the view query in terms of a query on the underlying base tables
- Disadvantage:
  - **re-compute** the view with every query
    - E.g., multiple queries `SELECT` name `FROM` IT_Students `where` age > 19, 20, 21, …
  - inefficient for views defined via complex queries (e.g., aggregate queries)

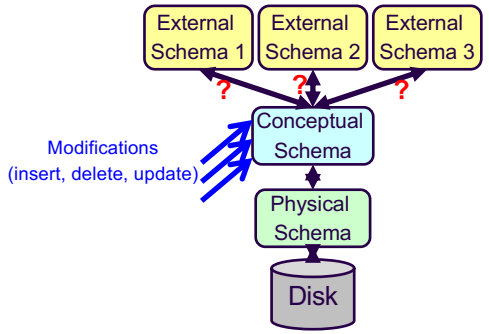
Virtual vs. Materialized Views

- Views:
  - Virtual tables
  - Evaluating a view (query) creates its data
- Materialized Views:
  - **Stored tables**
  - Physically store the view (query) **and its data**

Materialized Views

- Advantage:
  - Avoid re-computing the view with every query
  - Assumption: more queries can use the same view
- **But,** materialized **view maintenance** is needed
  - A materialized view should be updated when any **base table** used in the view definition is updated

Updating Materialized Views



Example of View Materialization

STUDENT

SID	*Name*	*Age*	*GPA*	*Major*
546007	Peter	18	3.8	CS
546100	Bob	19	3.65	CoE
546500	Bill	20	3.7	CS

Update on base table:

```
INSERT INTO Student
VALUES (456, …, CoE);
COMMIT;
```

```
CREATE MATERIALIZED VIEW
        Majors (major, mtotal)
AS SELECT major, count(*)
FROM Student
GROUP BY Major;
```

Materialized View

major	*mtotal*
CS	2
CoE	2

Updating Materialized Views

- Efficient strategies for automatically updating the materialized view when base tables are updated
  - Avoid re-computing the view from "scratch"
  - **Incremental update:**
    - determines what new tuples must be inserted, deleted, or modified in the view when an update is applied to the base tables

Trade-offs in view implementation

	(Virtual) Views	Materialized Views
Queries on Views		
Updates on Base Tables		

Trade-offs in view implementation

	(Virtual) Views	Materialized Views
Queries on Views	Re-compute view 	Re-use view 
Updates on Base Tables	Do nothing 	View Maintenance 

Specifying A Materialized View

```
CREATE MATERIALIZED VIEW Majors (major, mtotal)
 [BUILD METHOD][REFRESH OPTION METHOD]
   AS SELECT major, count(*)
   FROM Student
   GROUP BY Major;
```

- No **REPLACE** *only* **CREATE MATERIATIZED VIEW**
- Build Method:
  - `IMMEDIATE:` Create view and populate it with data
  - `DEFFERED:` Create view but do not populate it
- Refresh Method:
  - `ON COMMIT:` Automatic after a commit
  - `ON DEMAND:` Manually - execute DBMS_MVIEW.REFRESH('<MV-name>')
- Refresh Option:
  - `COMPLETE` (re-computation), `FAST` (incremental), `NEVER`

Full Materialized View Construction

```
CREATE MATERIALIZED VIEW Majors (major, mtotal)
[WITH ENCRYPTION,SCHEMABINDING,VIEW_METADATA]
 [BUILD METHOD][REFRESH OPTION METHOD]
   AS SELECT major, count(*)
   FROM Student
   GROUP BY Major
WITH CHECK OPTION;
```

- **ENCRYPTION**: The definition of the view is stored encrypted
- **SCHEMABINDING**: Prevents the drop of defining tables/views
- **VIEW_METADATA**: It makes visible the metadata on the view but hides the metadata of the defining tables/views.
- **WITH CHECK OPTION:** Prevent migration of tuples out of updateable views

Views Vs Temporary Tables

- No standard but Temporary Tables are
  - visible to the current SQL session
  - automatically dropped at the end of session
  - cannot have foreign key constraints
- SQLServer & MySQL: temporary tables are local
  - SQLServer: Create Table #Yahoo (YID int, YNM Char(3));
  - MySQL: Create Temporary Table (YID int, YNM Char(3));
- Oracle Server: global temporary tables
  - Example: Create Global Temporary Table Yahoo
    on commit preserve rows
    AS Select YID, YNM From TahooBase;