Database Systems 2

Lecture 14

Physical Design 2

Overview of the Methodology

<u>Conceptual Database Design</u>

1 Build local conceptual data model for each user view.

<u>Logical Database Design</u>

2 Build and validate local logical data model <u>for each user view</u>

3 Build and validate global logical data model

<u>Physical Database Design</u>

4 Translate global logical data model for target DBMS

5 Design physical representation

6 Design security mechanisms

7 Monitor and tune operational system

Physical Database Design

6 Design security mechanisms

- 6.1 Design user views
- 6.2 Design access rules

7 Monitor and tune operational system

6.1 Design User Views

6.1 Design User Views

Up to this point, we have been taking the individual user data models (Dreamhome Supervisor, Dreamhome Manager) and combining them to produce a set of base tables.

We now work backwards and define a number of user views which the DBMS can offer to the users. We will use the SQL VIEW command to do this.

In the Dreamhome scenario, there are only two. In a more complex scenario, you could combine them if it simplifies things:

DataEntry - consisting of DataEntry and Auditing views
Branch - consisting of Director and Manager views.
Everyday - consisting of Supervisor, Assistant and Client views.

In large systems, a user would not be given the required privileges to access the base tables directly.

Their user accounts would only have the necessary privileges to access a particular set of virtual 'view tables' which reflect their 'user view' of the database.

SQL View Command

```
CREATE VIEW Staff3
AS
SELECT staffNo, fName, lName, position, sex
FROM Staff
WHERE branchNo = 'B003';
```

In general, any changes to the data in a base table are reflected in the view, and any change to the view is reflected in the base table - subject to certain limitations.

A view can be derived from one table, or a multi-table join.

The ISO standard specifies that a view is only updatable if it conforms to the following conditions.

View updatability

Any modifications, including UPDATE, INSERT, and DELETE statements, must reference columns from only one base table.

The columns being modified in the view must directly reference the underlying data in the table columns. The columns cannot be derived in any other way, such as through the following:

- An aggregate function: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR, and VARP.
- A computation. The column cannot be computed from an expression that uses other columns. Columns that are formed by using the set operators UNION, UNION ALL, CROSSJOIN, EXCEPT, and INTERSECT amount to a computation and are also not updatable.

The columns being modified are not affected by GROUP BY, HAVING, or DISTINCT clauses.

In short:

For a view to be updatable, the DBMS must be able to trace any row or column back to its row or column in the base table.

General Syntax

```
CREATE VIEW ViewName [(newColumnName [, … ])]
AS definingQuery
[WITH [CASCADED | LOCAL ] CHECK OPTION]
```

Rows exist in a view because they satisfy the WHERE condition of the defining query. If a row is altered so that it no longer satisfies it, it disappears from the view (and vice versa)

These are known as migrating rows.

Generally, the WITH CHECK OPTION clause will prevent rows from migrating out of a view. CASCADED and LOCAL are applicable to view hierarchies.

Advantages of views

Data Independence

Provides a level of abstraction between the base tables and the users application.

Currency

Changes to any of the base tables is always reflected in the view

Security

Each user can be given a suitable level of privilege such that they can only access the views that are appropriate for their job.

Reduce Complexity

A query can be made simpler if it is based on a view that takes data from several base tables.

Convenience

Users are only presented with that part of the database that they need to use.

Disadvantages of Views

Update restrictions

It is not always possible to update views

Structure restriction

If the structure of a base table changes, then a view will only refer to the columns that were in the base table when the view was created. You will need to drop the view and recreate if you wish the new columns to be seen in the view.

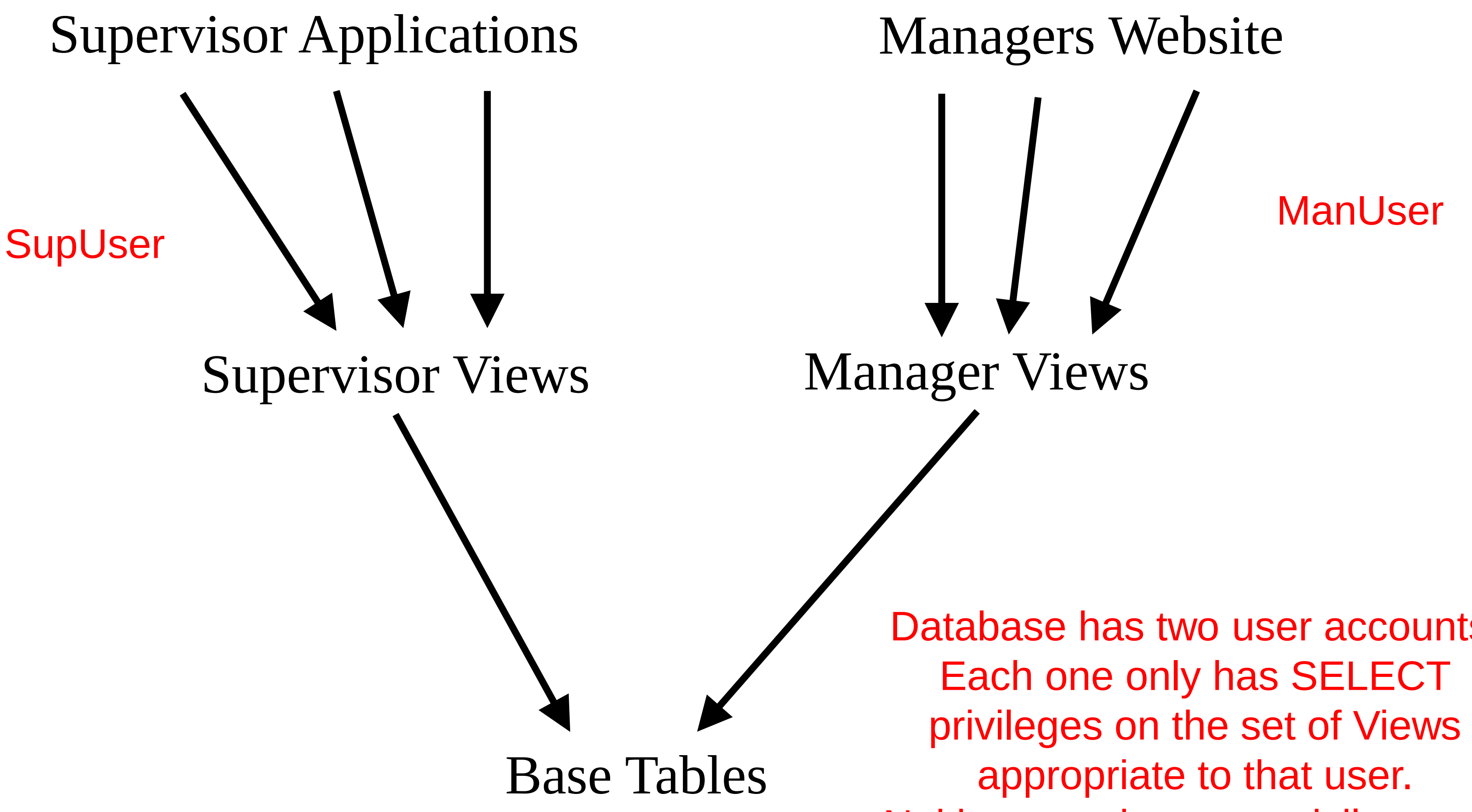
Performance

There is a performance overhead. If a view is based on a multi-table join query, those joins will have to be redone every time the view is accessed.

View materialisation

Cache the view when it is first accessed to speed things up. Problems?

6.2 Design Access Rules



Database has two user accounts. Each one only has SELECT privileges on the set of Views appropriate to that user. Neither user has any privileges on the base tables.

Security Policy

Develop and <u>document</u> a security policy.

For example:

There will be three levels of access:

- Head Office – SELECT / INSERT access to all views for all branches (not including client data)
- Branch Staff – SELECT access to their appropriate job/dept views
- Client – SELECT / INSERT access to their own data only

The Branch Staff level consists of the following posts:

- Branch Manager – can access Supervisor views for all branches
- Desk Clerk – can access Clerk views on data for their branch only
- Loans Officer – can access Loans views on data for their branch only

Creating Users (Oracle)

```
CREATE USER user
   IDENTIFIED {BY password | EXTERNALLY}
   [DEFAULT TABLESPACE tablespace]
   [TEMPORARY TABLESPACE tablespace]
   [QUOTA (integer [K|M] | UNLIMITED} ON tablespace]
   [PROFILE profile]
```

Example

```
CREATE USER user01 IDENTIFIED BY pass99;

CREATE USER fred
   IDENTIFIED BY fredpasswd
   DEFAULT TABLESPACE slate
   TEMPORARY TABLESPACE temp
   QUOTA 50k ON slate;
```

You can ALTER and DROP users

Roles

Oracle has three predefined roles:

The Connect Role

The Connect role enables the user to select, insert, update, and delete records from tables belonging to other users (after the appropriate permissions have been granted). The user can also create tables, views, sequences, clusters, and synonyms.

The Resource Role

The Resource role gives the user more access to Oracle databases. In addition to the permissions that can be granted to the Connect role, Resource roles can also be granted permission to create procedures, triggers, and indexes.

The DBA Role

The DBA role includes all privileges. Users with this role are able to do essentially anything they want to the database system. You should keep the number of users with this role to a minimum to ensure system integrity.

Don't rely on defaults!

Create your own roles.

```
CREATE ROLE STUDENT_USER;

GRANT
  CREATE SESSION,
  CREATE TABLE,
  CREATE VIEW,
  CREATE ANY INDEX,
  CREATE TRIGGER,
  CREATE SYNONYM,
  CREATE DATABASE LINK,
  CREATE PROCEDURE
TO
STUDENT_USER;
```

Note that you can only create roles and grant privileges if you are connected to the database with the appropriate privileges.

Privileges

An object's owner has all object privileges for that object, and those privileges cannot be revoked.

The object's owner can grant object privileges for that object to other database users.

A user with ADMIN privilege can grant and revoke object privileges from users who do not own the objects on which the privileges are granted.

Granting Privileges

SYNTAX:

```
GRANT {object_priv | ALL [PRIVILEGES]} [ (column [, column]...) ]
   [, (object_priv  ALL [PRIVILEGES]} [ (column [, column] ...)]]
   ...
   ON [schema.]object
   TO {user | role | PUBLIC} [, {user | role | PUBLIC}] ...
   [WITH GRANT OPTION]
```

Example

```
GRANT SELECT, UPDATE
ON user05.myemp
TO fred;
```

Some databases allow you to grant individual privileges on specific columns or data items to specific users

You can use GRANT to grant:

- System privileges to users and roles.

  Eg CREATE TABLE, ALTER TRIGGER or DROP VIEW

- Roles to users and roles. Both privileges and roles are either local, global, or external.

- Object privileges for a particular object (ie a table, or view) to users, roles, and PUBLIC.

  Eg

  ```
  GRANT SELECT, UPDATE
  ON user05.myemp
  TO fred;
  ```

See:

http://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_9013.htm

Having set up a role called STUDENT_USER, with a defined set of privileges, I can then assign that role to as many users as I want.

```
GRANT STUDENT_USER TO fred;
GRANT STUDENT_USER TO tom;
GRANT STUDENT_USER TO testuser01;
```

To remove a GRANT, the command is REVOKE   (not DROP)

REVOKE UPDATE
ON emp_details_view FROM public;

REVOKE DROP ANY TABLE FROM hr, oe;

7 Monitor and Tune System

Monitoring and Tuning

Difficult to provide any general rules, because it depends:

1. On the specific DBMS being used, and it's capabilities.

2. On the specific database design being implemented on that DBMS.

One of the main objectives of physical database design is to store and access data in an efficient way.

There are a number of factors we can use to measure efficiency. The main ones are:

- **Transaction throughput**
  Number of transactions in a given time interval.
- **Response time**
  Time taken for a single transaction
- **Disk storage**
  Amount of disk space needed to store the data.

You need to find out how your DBMS records / logs the above metrics.

You will need to regularly check the logs to see how well or badly the DBMS is performing.

When are the peak loads – are they when you thought they would be?

There is no one factor that is always correct.

You are always trading one thing off against another.

Do not regard your initial physical database design as the finished article.

It is more like an initial estimate of how the operational system might perform.

Benefits of Tuning the Performance

Can avoid having to buy extra hardware.

Might be able to downsize the hardware installation.

Faster response times and better throughput.

Improved staff morale.

Increased customer satisfaction.