Physical Database Design And Tuning

CHAPTER 4

Physical Database Design

- It is the process of transforming a logical data model into a physical model of a database.
- Unlike a logical design, a physical database design is optimized for data-access paths, performance requirements and other constraints of the target environment, i.e. hardware and software.

Physical Database Design

The physical data model is created by transforming the logical data model into a physical implementation based on the DBMS to be used for deployment. To successfully create a physical database design you will need to have a good working knowledge of the features of the DBMS, including

- In-depth knowledge of the database objects supported by the DBMS and the physical structures and files required to support those objects
- Details regarding the manner in which the DBMS supports indexing, referential integrity, constraints, data types, and other features that augment the functionality of database objects
- Detailed knowledge of new and obsolete features for particular versions or releases of the DBMS
- Knowledge of the DBMS configuration parameters that are in place
- Data definition language (DDL) skills to translate the physical design into actual database objects

Data Definition Language.

DDL stands for Data Definition Language.

These commands are used to change the structure of a database and database objects

DDL Operations

- **DDL** commands can be used to add, remove, or modify tables with in a database.
- The DDL commands are:

- CREATE
- ALTER
- DROP
- TRUNCATE
- RENAME

DDL Operations continue

- **1. CREATE :**
  This command is used to create table in the relational database .
  This can be done by specifying the names and datatypes of various columns.
- **Syntax:**

```
CREATE TABLE TABLE_NAME
( column_name1 datatype1,
column_name2 datatype2,
column_name3 datatype3,
column_name4 datatype4 );
```

DDL Operations continue

- The column_name in create table command will tell the name of the column and corresponding datatype will specify the datatype of that column.Here in this table the three column_names namely – Student_id is of type int ,Name is of type varchar and Marks is of type int.
- for example:
- CREATE TABLE Employee (Student_id INT, Name VARCHAR(100), Marks INT);

DDL Operations continue

Student_id	Name	Marks

DDL Operations continue

- **2. ALTER :**
- Alter command is used for altering the table in many forms like:
- Add a column
- Rename existing column
- Drop a column
- Modify the size of the column or change datatype of the column

DDL Operations continue

- **ADD using ALTER –**
  Syntax to add column
- ALTER TABLE table_name ADD(
- column_name datatype);
- The above command will add a new column to the table.And the resulting table will have one more column like this:
- ALTER TABLE Student
- ADD
- (Address VARCHAR(200))

DDL Operations continue

- Here this command will add a new column “Address” in the table Student of datatype varchar(200);

Student_id	Name	Marks	Address

DDL Operations continue

- RENAME using ALTER –
- Syntax to rename column
- ALTER TABLE
- table_name
- RENAME
- old_column_name TO new_column_name;

- The above command will rename the existing column to new column.

DDL Operations continue

- ALTER TABLE
- Employee
- RENAME
- Marks TO Age;
- The command above will change the column_name from Marks to Age;

Student_id	Name	Age	Address

DDL Operations continue

- **DROP using ALTER –**
  Syntax to Drop a column :
- ALTER TABLE
- table_name
- DROp
- (column_name);
- The above command will delete the existing column
- For example:
- ALTER TABLE Employee
- DROP
- (Age);

DDL Operations continue

- Here the column_name =”Age”, has been deleted by this command;

Student_id	Name	Address

DDL Operations continue

- **MODIFY using ALTER –**
  Syntax to Modify a column
- ALTER TABLE
- Employee MODIFY
- (column_name datatype);
- The above command will modify the existing column .
- For example:
- ALTER TABLE
- student
- MODIFY
- (name varchar(300));

DDL Operations continue

- The above command will modify the column_name “Name” by changing the size of that column.

Student_id	Name	Address

DDL Operations continue

- **3. TRUNCATE :**
- This command removes all the records from a table. But this command will not destroy the table's structure.
- Syntax :
- TRUNCATE TABLE table_name
- This will delete all the records from the table.For example the below command will remove all the records from table student.

- Example:

- TRUNCATE TABLE Student;

DDL Operations continue

- 4. DROP :
- This command completely removes the table from the database along with the destruction of the table structure.
- Syntax –
- DROP TABLE table_name
- This will delete all the records as well as the structure of the table.
- This is the main difference between TRUNCATE AND DROP.-TRUNCATE only removes the records whereas DROP completely destroys the table.
- Example:
- DROP TABLE Student;
- This command will remove the table records as well as destroys the schema too.
- This is all about the DDL commands.

Applications of DDL

- Creating Database Objects: DDL statements can be used to create various database objects such as tables, views, indexes, and stored procedures.
- Modifying Database Objects: DDL statements can be used to modify the structure of existing database objects such as adding or dropping columns from tables, modifying the data type of columns, renaming tables or columns, etc.
- Managing Database Constraints: DDL statements can be used to create or alter database constraints such as primary keys, foreign keys, unique constraints, and check constraints.

Applications of DDL

- Granting or Revoking Permissions: DDL statements can be used to grant or revoke permissions to various database objects such as tables, views, stored procedures, and indexes.
- Indexing: DDL statements can be used to create or modify indexes on database tables, which can improve the performance of SQL queries.
- Partitioning: DDL statements can be used to create or modify partitioned tables, which can improve the performance of queries that access large amounts of data.

- Overall, DDL is an essential part of SQL and is used extensively in database management systems to create, modify and manage database objects.

Transform Entities to Tables

The physical counterpart of an entity is a Table. Therefore, the first step in transforming a logical data model into a physical database is to map each entity in the data model to a table in the database.

Transform Attributes to Columns

The physical counterpart of an attribute is a column within a table. When you map entities to tables, map the attributes of each entity to the columns of each respective table.

At least initially, do not change the basic definition of the columns. For example, do not group attributes together into a composite column.

The attributes of each entity should be mapped to the columns of each respective table.

Transform Domains to Data Types

To support the mapping of attributes to table columns you will need to map each logical domain of the attribute to a physical data type and perhaps additional constraints.

Each column must be assigned a data type. Certain data types require you to specify a maximum length. For example, you could specify a character data type as CHAR(20), indicating that up to 20 characters can be stored for the column.

You may need to apply a length to other data types as well, such as graphic, floating point, and decimal (which also require a scale).

Map each logical domain to a physical data type, perhaps coupled with additional constraints.

Transform Domains to Data Types

- Primary Keys

  Be sure to identify a primary key for each physical table you create.

  The Identity Property

  The identity property is a feature supported by several of the most popular relational DBMS products. It can be assigned to a column that has a numeric (usually integer) data type. When the identity property is assigned to a column, the DBMS treats that column in a special way. The database user does not provide values for the column when rows are inserted into the table in which the column exists. Instead, the DBMS increments a counter and automatically uses that value for the column. Usually only one column per table can be assigned the identity property.

- Column Ordering

Before implementing a physical table, be sure to review the order of the columns. Column sequencing can impact performance, therefore, for physical implementation you may need to change the sequence recorded in the logical data model.

Build Referential Constraints for All Relationships

- The physical counterpart of a relationship is a referential constraint. To define a referential constraint you must create a primary key in the parent table and a foreign key in the dependent table. The referential constraint ties the primary key to the foreign key.

The referential constraint ties the primary key to the foreign key.

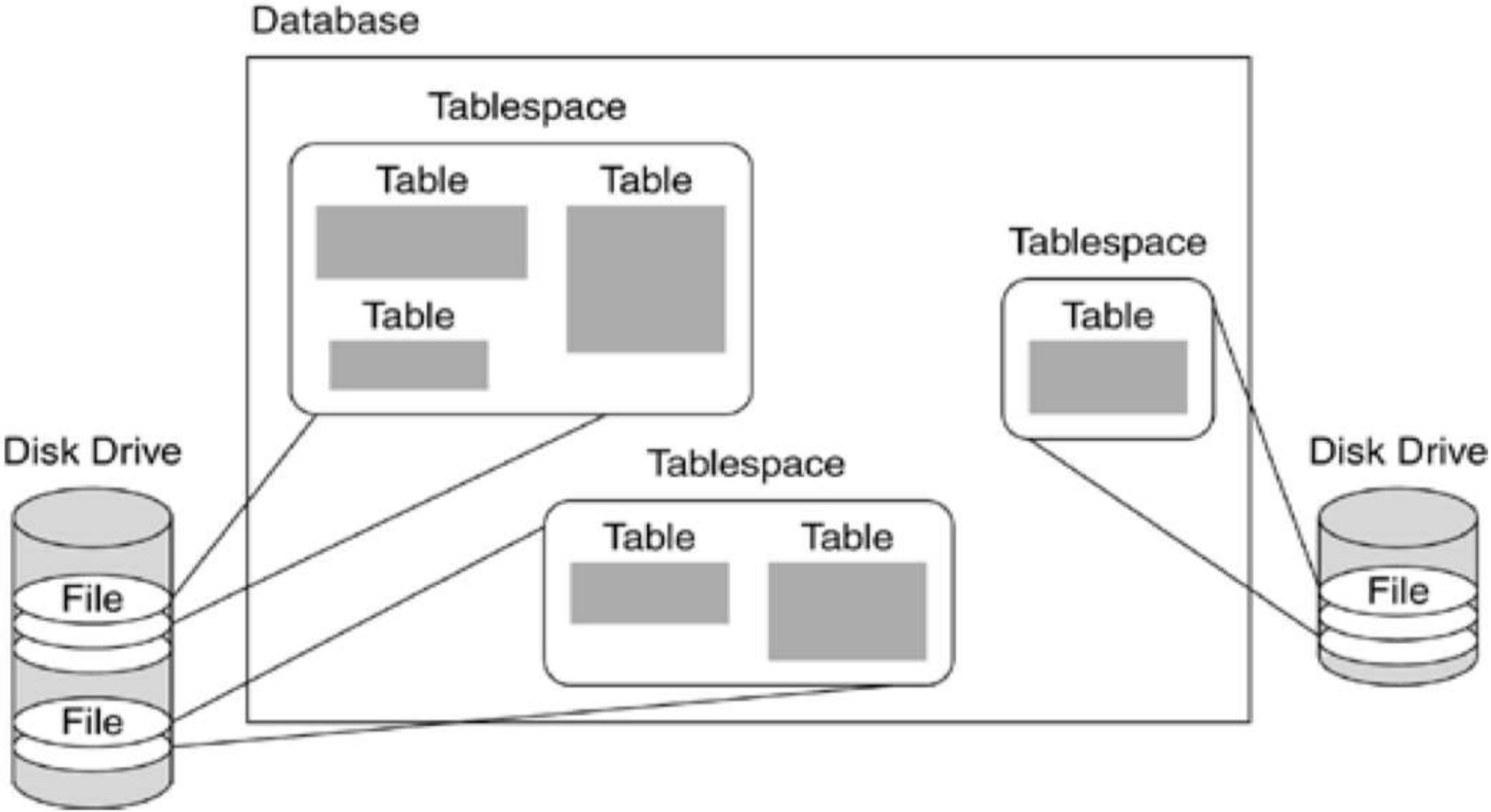
- **Referential Integrity**

  Referential integrity guarantees that an acceptable value is always in each foreign key column.

Build Physical Data Structures

- Designing and implementing a physical database from a logical data model is not just a simple matter of mapping entities to tables, attributes to columns, and relationships to referential constraints. Quite a few other database design issues must be addressed. One of these issues is preparing for the physical storage of data.

- Although relational data is expressed to the user by means of a table, underlying files or data sets must exist to store the actual data and those files are not necessarily stored as a simple grid of rows and columns. During the physical design process, the DBA must map each table to a physical structure to store the table's data. These physical structures are commonly called tablespaces (or data spaces).


Database
Tablespace
Table
Table
Table
Tablespace
Table
Tablespace
Table
Table
Disk Drive
File
File
Disk Drive
File

Database Performance Design

- When implementing a physical database from a logical data model, you must begin to consider how the database will perform when applications make requests to access and modify data. A basic fact of database processing is that disk access is slower than memory access lower by orders of magnitude. If the DBMS were required, in every instance, to scan through the database row-by-row, or block-by-block, looking for the requested data, no one could afford to use databases. Fortunately, several good techniques exist to allow data in the database to be accessed more rapidly.

Designing Indexes

One of the best techniques for achieving acceptable query performance is the creation of appropriate indexes on your database tables. Of course, the trick is in determining how many indexes to create and how exactly to define each index. First, let's cover some index basics.

An index is an alternate path to data in the database. The structure of an index makes it easier to find data in the database, with fewer I/O operations. Therefore, queries can perform faster when using an index to look up data based on specific key values.

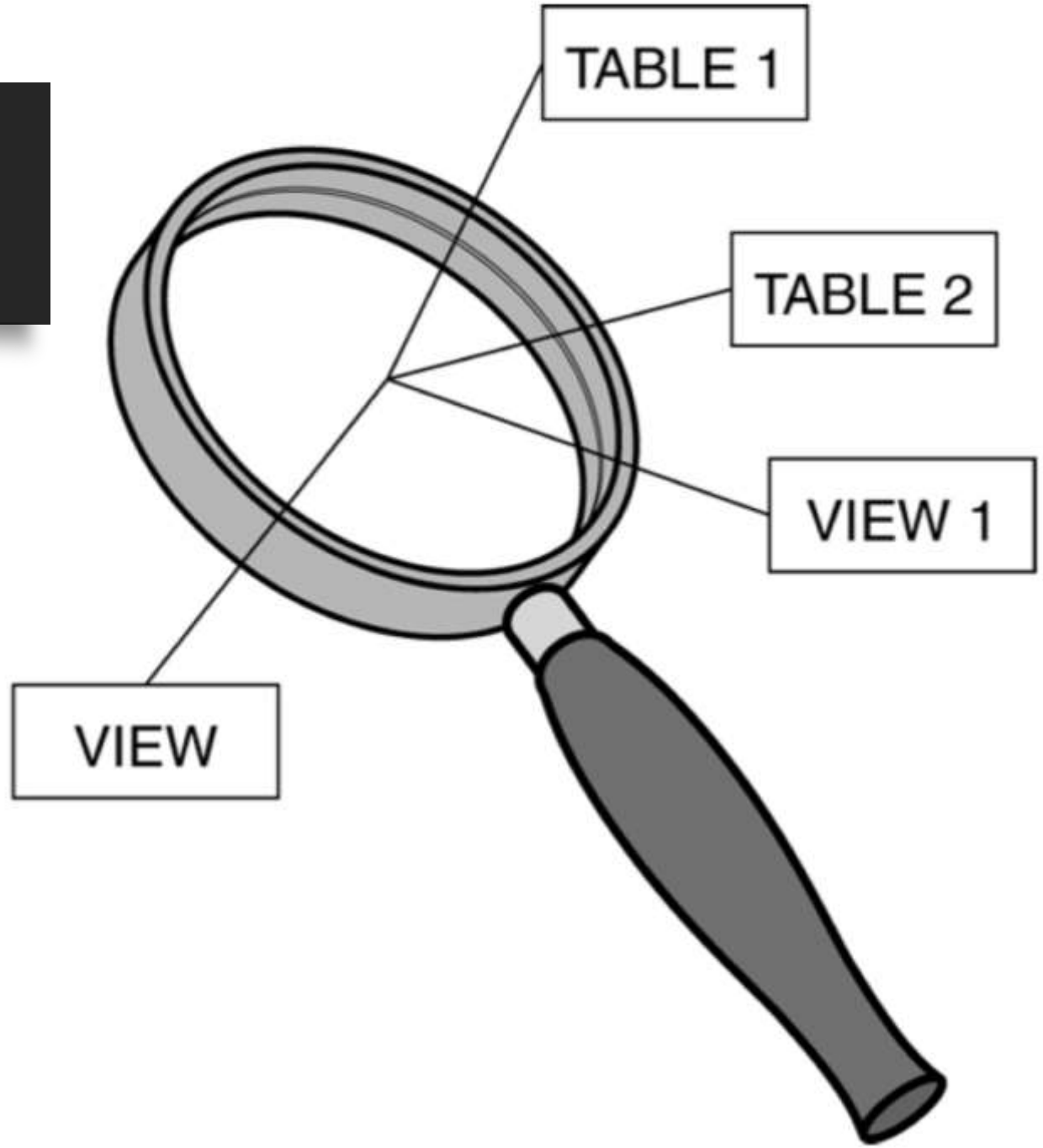
Views

Another aspect of physical database design is the creation of database views to support specific application data requirements. Views are not required to access a physical database, but they can be helpful to support specific application and user requirements. You can think of a view as a way of turning a SELECT statement into a "table" that is accessible using SQL. Therefore, a view can be considered a logical table. No physical structure is required of a view; it is a representation of data that is stored in other tables (or other views).

Views

Views are flexible and can consist of any combination of the following:

- Rows from tables. These can be a subset of rows from a single table, all rows from a single table, a subset of rows from multiple tables, or all rows from multiple tables.
- Rows from views. These can be the same combinations as listed for tables.
- Columns from tables. These can be a subset of columns from a single table, all columns from a single table, a subset of columns from multiple tables, or all columns from multiple tables.
- Columns from views. These can be the same combinations as listed for tables.



Views

Views can allow you to • Provide row and column level security

- Ensure efficient access paths
- Mask complexity from the user
- Ensure proper data derivation
- Rename tables
- Rename columns