UNIT-1

1.Explain the advantages of Databases over traditional file systems.

1) **No redundant data**: Redundancy removed by data normalization. No data duplication saves storage and improves access time.

2) **Data Consistency and Integrity**: As we discussed earlier the root cause of data inconsistency is data redundancy, since data normalization takes care of the data redundancy, data inconsistency also been taken care of as part of it

3) **Data Security**: It is easier to apply access constraints in database systems so that only authorized user is able to access the data. Each user has a different set of access thus data is secured from the issues such as identity theft, data leaks and misuse of data.

4) **Privacy**: Limited access means privacy of data.

5) **Easy access to data** – Database systems manages data in such a way so that the data is easily accessible with fast response times.

6) **Easy recovery**: Since database systems keeps the backup of data, it is easier to do a full recovery of data in case of a failure.

7) **Flexible**: Database systems are more flexible than file processing systems.

2.Explain the various constraints on relational model.

1). Schema based Constraints

1) Domain constraint
2) Key constraints
3) Constraints as null
4) Entity Integrity constraint
5) Referential Integrity constraint

1. Domain Constraints
   - This constraint specifies that within each tuple the value of each attribute A must be an atomic value from the domain dom(A).
   - The data types for domain include standard numeric data types for integer and real values, fixed length and variable length strings.

2. Key constraints
    - A relation is a set of tuples. By definition all the elements in a set are distinct.
    - No two tuples can have the same combination of values for all attributes.
    - Let SK be a sub set of attributes of R with the property that no two tuples have the same combination of values. Let t1,t2 be two distinct tuples. Then, t1[SK]≠t2[SK].
    - Such set of attributes is called a super key of the relation schema R.
    - A super key SK specifies uniqueness constraint that no two tuples in
    - the state r of R can have the same value for SK.
      Every relation has at least one default super key- the set of all its attributes.
    - A key K of a relation R is a super key of R with an additional property that removing any attribute A from K leaves a set of attributes K' that is not a super key of R.

      A key satisfies two constraints:
        1) Two distinct tuples in any state of a relation cannot have identical values for the attributes in the key.
        2) It is a minimal super key i.e. a super key from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 to hold.

      Candidate Key: A relation schema can have more than one key. Each of the key is called a candidate key.

      Primary key: A candidate key whose values are used to identify tuples in a relation is called a primary key.

3. Constraints as NULL

   This is a constraint on attributes that specifies whether null values are

   permitted or not.

   Example: If every student tuple must have a valid non-null value for

   name attribute, then the name attribute is constrained as not null.

4. Entity Integrity Constraints

It states that no primary key value can be null. This is because primary keys are used to identify the tuples in a relation.

5. Referential Integrity Constraints
    - Foreign Key: Let R1, R2 be two relation schemas. A set of attributes FK in a relation schema R1 is a foreign key of R1 that references a relation R2 if it satisfies the following rules
    - The attributes in FK have the same domains as primary key attributes PK of R2. The attributes FK are said to refer to the relation R2.
      A value of FK in a tuple t1 of current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is null. Then we have, t1[FK]=t2[PK]. We say that t1 refers to the tuple t2.
    - Here R1 is called the referencing relation and R2 is called the referenced relation.
    - If these conditions hold, a referential integrity constraint from R1 to R2 is said to hold.

3.Describe the various types of attributes in ER model with appropriate examples of your own.

1. Simple attributes
2. Composite attributes
3. Single valued attributes
4. Multi valued attributes
5. Derived attributes
6. Key attributes

1. Simple Attributes

- Simple Attributes are independent attributes that cannot be classified further. In other words, it is also known as atomic attributes.

- For example, a Student is an entity that consists of attributes Roll No, Age, Class. Here, we cannot divide the Roll no attribute into sub-attributes. Therefore, if we cannot divide the attribute further then it is a Simple Attribute.

2. Composite Attribute

- When it is possible to divide the attributes into different components then that attribute is called a Composite Attribute. We divide Composite Attribute into sub-parts that form simple attributes.
- For example, If Name is an attribute for Student entity. We can divide the Name attribute into first name, middle name, last name attributes. These sub-attributes that are classified from the composite Attribute works as Simple Attributes.

3. Single Valued Attributes

- Attributes stores values that are used to describe the entity. The attributes which are able to store only one value are known as Single Valued Attributes. These attributes cannot store more than one value.
- For example, The attributes of an Employee entity are Employee id, DOB, Gender. An employee has only one employee id which is unique and it also has a single date of birth. So these attributes can store only one value in it. Therefore, it is known as Single Valued Attributes.

4. Multi-Valued Attributes

- The attributes which are able to store more than one value are known as Multi-Valued Attributes.
- For example, let's assume Email id and Contact No are the attributes of the Employee entity. An employee can provide more than one email id and contact no. Therefore multiple values can be stored in Multi-Valued Attributes.

5. Derived Attributes

- The name itself describes the attribute. Derived attributes are those attributes that are derived from the value of another attribute.
- For example, We can calculate the age from the date of birth value. Therefore Age attribute can be derived from the DOB attribute.

6. Key Attributes

- Every entity has a special attribute that holds a unique value to identify the entity in the entity set. The value of key attributes must be unique and cannot be used again.
- For example, Employee id is the key attribute for employee entity, Roll No is the key attribute for Student entity, and Pincode is the key attribute for the place attribute.

4. Discuss the mechanism of attribute/relationship inheritance. Why is it useful?

● The connection between subclass and superclass is called inheritance.

A subclass inherits all attributes of its superclass. A subclass inherits all relationships of its superclass.

• In ER diagram, an entity type represents all entities underneath it. – Example, "Employee" represents all entities of employees.

• Sometimes, entities underneath an entity type can be classified to specific entities which have its own characteristics. – Example, SalariedEmployee, Hourly Employee, Manager, Secretary, Engineer, Technician. – All the previous entity types are considered employees, but each has its own unique attributes.

● Multiple Inheritance:

An entity can be a sub-class of multiple entity types; such entities are sub-class of multiple entities and have multiple super

classes. In multiple inheritances, attributes of sub-class are the union of attributes of all superclasses.

5. Discuss user-defined and attribute-defined specializations, and identify the differences between the two.

1. Attribute defined specialization:
   We can determine to which subclass an entity belongs by placing condition on some attribute in the superclass. We call this attribute "Defining Attribute".

2. User defined specialization:
   When we do not have a condition for determining membership in a subclass, the subclass is called user defined specialization.
   If a DB user has an entity "A" to add to the database, then he can use his understanding of the mini -world to determine to which subclass entity "A"
should be added.

6. Discuss the two main types of constraints on specializations and generalizations.

1. Based on Definition:

1) Predicate Defined Subclasses

2) User Defined Subclasses

2. Based on membership

1) Disjoint Subclasses

2) Overlapping Subclasses

3. Based on completeness of participation

1) Total Specialization

2) Partial Specialization

1. Based on Definition (Predicate Defined Subclasses) :
   • It is also called Condition Defined Subclasses.
   • We can determine to which subclass an entity belongs by placing a condition on some attribute in the superclass.
   • We call this attribute "Defining Attribute".
   • Example: – JobType is an attribute of superclass "Employee", based on the value of this attribute, we can determine if an employee belongs to "Secretary", "Technician", or "Engineer".
   If the membership in a subclass is not defined based on an attribute, then this subclass is called "User Defined Classes".

   • If a DB user has an entity "A" to add to the database, then he can use his understanding of the mini -world to determine to which subclass entity "A" should be added.

2. Based on Membership (Disjointness Constraint)
   • Specifies that all subclasses of a specialization must be disjoint.
   • In other words, if an entity belongs to one subclass of a specialization, then it cannot belong to another subclass of the same specialization.
   • Example: – {Secretary,Technicien,Engineer} is a disjoint specialization. – If an employee is a secretary, then she cannot be a technician
   • The disjointness is represented in EER using d symbol.

Sometimes an entity can belong to more than one subclass of the same specialization, this means that subclasses overlap with each other.
• In EER, this is represented by using notat o notation.

3. Based on Participation (Completeness Constraint)
• Completeness constraint states that a specialization can be:
– Total Specialization or
– Partial Specialization

Total Specialization: –

Every entity that belongs to a superclass must belong to at least one subclass of the specialization

• Example: If "every" employee must be either an HourlyEmployee or a SalaryEmployee then the specialization {HourlyEmployee,SalaryEmployee} is a total specialization of Employee.

• In EER, a total specialization is represented by using a double lines that is going out of the superclass.

Partial Specialization: –

Some entities might not belong to any of the subclasses of the specialization – In EER, it is represented as one line going out of the superclass.

• Example: – If an employee is not a secretary, technician, or engineer, then he does not belong to any of these subclasses of this specialization.

– So, the specialization {secretary, technician, engineer} is a partial specialization.

Note :

• Disjointness and Completeness constraints are independent, so you might have the following combinations of specializations: – Disjoint, total – Disjoint, partial – Overlapping, total – Overlapping, partial

6. What is the difference between a specialization hierarchy and a specialization lattice?

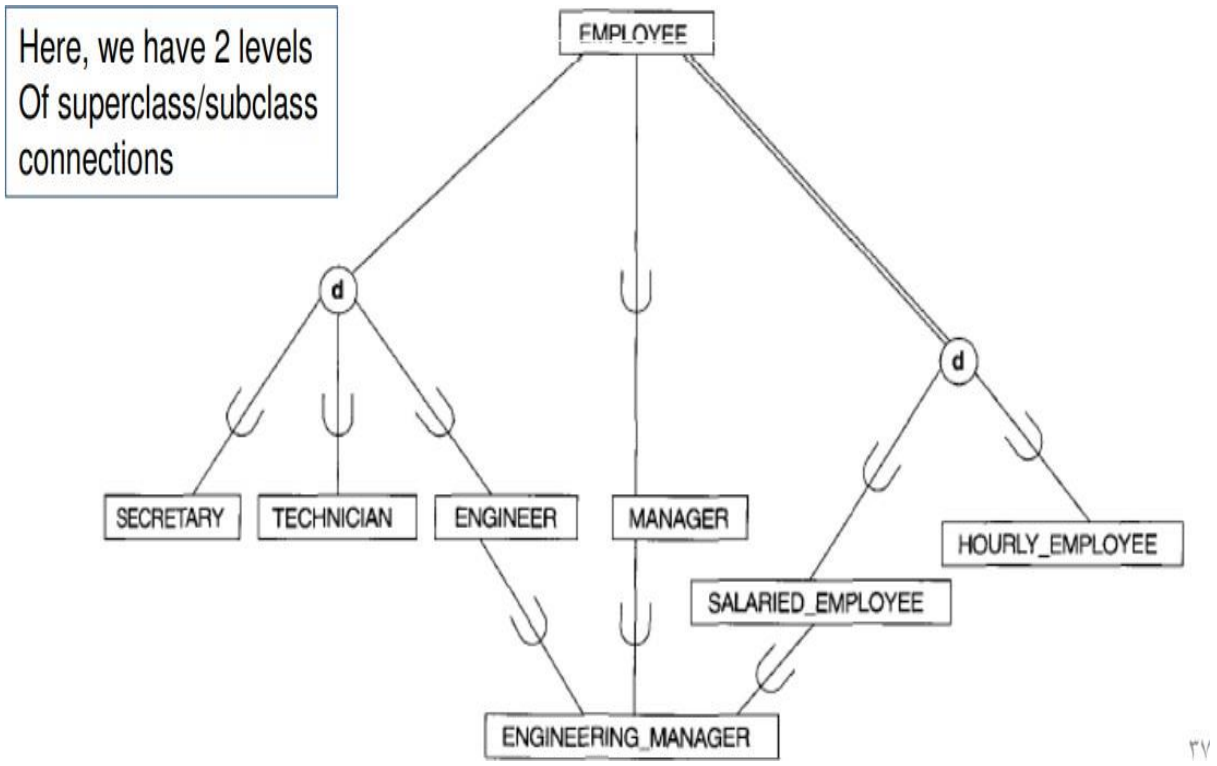
Specialization and Generalization Hierarchies and Lattices:
• In the specialization hierarchy that you have seen so far:
– There is one superclass for each subclass (One superclass/subclass connection).
– There is only one level of superclass/subclass connections

• If we have a subclass that has multiple parents (superclasses), then the hierarchy is called a "specialization lattice".

– Multiple superclass/subclass connections for the same subclass.

• Having more than one superclass for the same subclass is also called "Multiple Inheritance".

• Example: – Engineer_Manager is an engineer, a manager, and a salaried employee at the same time. – Engineer_Manager has 3 superclasses(Manager,Engineer,Salaried_Employee).

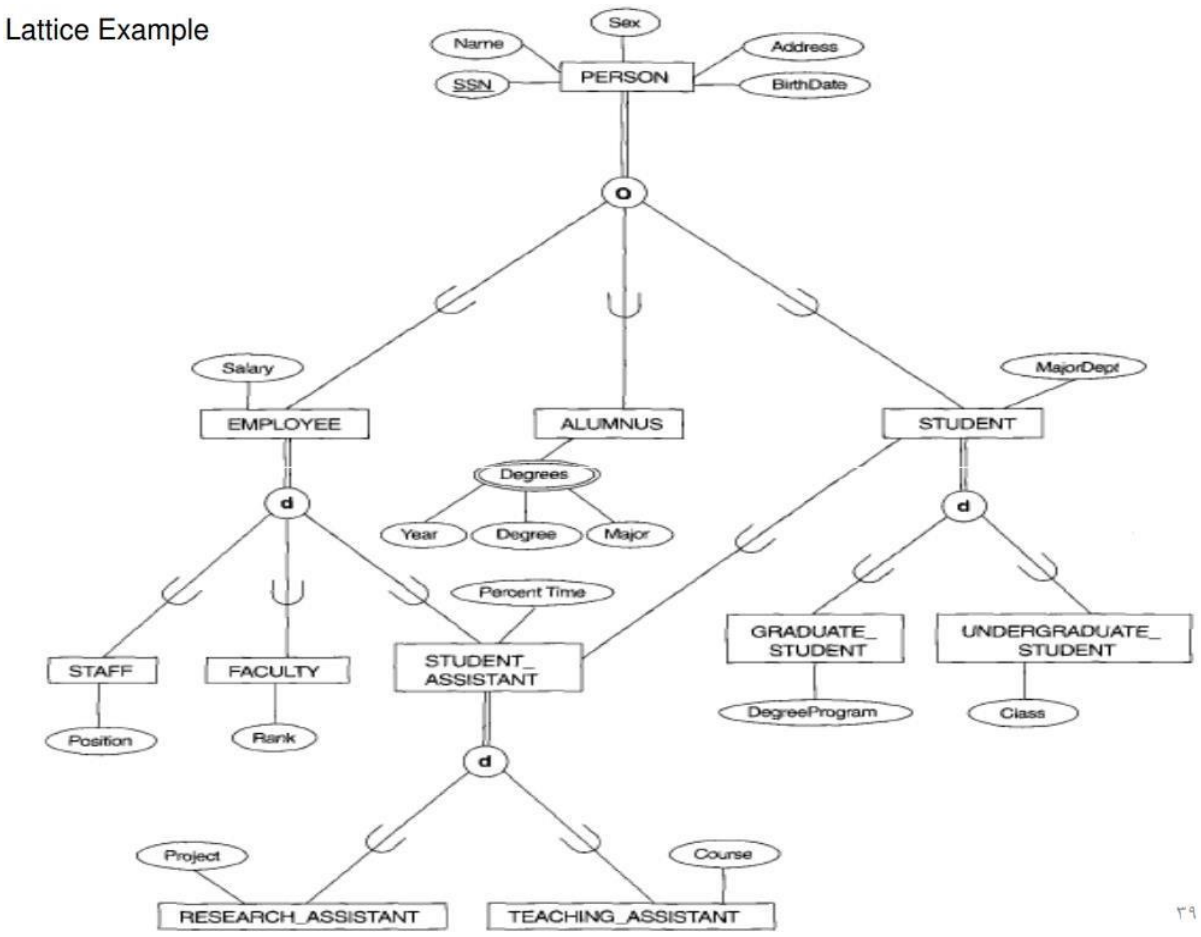


• Shared Subclass: A subclass with more than one superclass.

• Leaf Node: A class that has no subclasses of its own.

• In the previous example: – Engineer is one of the "direct" superclasses of subclass Engineer_Manager. – Employee is an "indirect" superclass of subclass Engineer_Manager.

Example For Specialization Lattice:



• Person entity type is specialized into the subclasses {Employee, Alumnus, Student}. • Question: Is this specialization overlapping?

• Answer: Yes, because an employee in a university can be an Alumni of the same university and at the same time studying a different degree while he is working in the university

7. What is the difference between specialization and generalization? Why do we not display this difference in schema diagrams?

The following are the major differences between generalization and specialization:

1. **Structure:**
   Generalization is a bottom-up approach in that it begins with lower-level, or narrower, entity types and broadens them to higher-level types. Specialization, in contrast, is a top-down approach since it creates lower-level entity types out of higher-level ones.

2. **Basis:**
   Common attributes between different entities form the basis for generalization. For specialization, it's uncommon attributes.

3. **Number:**
   Generalization involves multiple entities and combines them into a generalized entity. Specialization involves a single entity broken down into multiple sub-entities.

4. **Size:**
   Generalization reduces the schema of the data by unifying components. Specialization expands the schema by multiplying the components.

5. **Inheritance:**
   Inheritance is the capability of a lower-level entity to retain attributes from higher-level associations. Inheritance is possible with specialization but not generalization.

6. **Use:**
   Generalization allows users to visualize bigger-picture patterns. Specialization allows users to narrow their search.

Generalization

- It works using bottom-up approach.
- The size of schema is reduced.
- It is generally applied to a group of entities.
- Inheritance is not used in generalization.
- It can be defined as a process where grouping are created from multiple entity sets.

- It takes the union of two or more lower-level entity sets, and produces a higher-level entity set.
- Some of the common features are obtained in the resultant higher-level entity set.
- The differences and similarities between the entities that need to be in union operation are ignored

Specialization

- It uses a top-down approach.
- The size of schema is increased.
- It can be applied to a single entity.
- It can be defined as process of creation of subgroups within an entity set.
- It is the reverse of generalization.
- It takes a subset of higher level entity, and forms a lower-level entity set.
- A higher entity is split to form one or more low entity.
- Inheritance can be used in this approach.

UNIT-2

1,Discuss the various methods of storing and extracting XML Documents from databases.

Storing and Extracting XML documents from Databases

1. Using a file system or a DBMS to store the documents as text
2. Using a DBMS to store the document contents as data elements
3. Designing a specialized system for storing native XML data
4. Creating or publishing customized XML documents from preexisting relational databases

Several approaches to organizing the contents of XML documents to facilitate their subsequent querying and retrieval have been proposed. The following are the most common approaches:

1. **Using a DBMS to store the documents as text:**

   A relational or object DBMS can be used to store whole XML documents as text fields within the DBMS records or objects. This approach can be used if the DBMS has a special module for document processing, and would work for storing schemaless and documentcentric XML documents.

2. **Using a DBMS to store the document contents as data elements**

   This approach would work for storing a collection of documents that follow a specific XML DTD or XML schema. Because all the documents have the same structure, one can design a relational (or object) database to store the leaf-level data elements within the XML documents. This approach would require mapping algorithms to design a database schema that is compatible with the XML document structure as specified in the XML schema or DTD and to recreate the XML documents from the stored data. These algorithms can be implemented either as an internal DBMS module or as separate middleware that is not part of the DBMS.

3. **Designing a specialized system for storing native XML data:**

   A new type of database system based on the hierarchical (tree) model could be designed and implemented. Such systems are being called **Native XML DBMS**s. The system would include specialized indexing and querying techniques, and would work for all types of XML documents. It could also include data compression techniques to reduce the size of the documents for storage. Tamino by Software AG and the Dynamic Application Platform of eXcelon are two popular products that offer native XML DBMS capability. Oracle also offers a native XML storage option.

4. **Creating or publishing customized XML documents from preexisting relational databases:**

   Because there are enormous amounts of data already stored in relational databases, parts of this data may need to be formatted as documents for exchanging or displaying over the Web. This approach would use a separate middleware software layer to handle the conversions needed between the XML documents and the relational database. Section 12.6 dis-cusses this approach, in which datacentric XML documents are extracted from existing databases, in more detail. In particular, we show how tree structured documents can be

created from graph-structured databases. Section 12.6.2 discusses the problem of cycles and how to deal with it.

All of these approaches have received considerable attention. We focus on the fourth approach in Section 12.6, because it gives a good conceptual understanding of the differences between the XML tree data model and the traditional database models based on flat files (relational model) and graph representations (ER model). But first we give an overview of XML query languages

2.Explain the building blocks of XML.

•All XML documents are made up of the following building blocks:

1. Elements
2. Attributes
3. Entities
4. PCDATA
5. CDATA

1. Elements

• Elements are the basic building blocks of XML.
• Elements can contain text, other elements or be empty Example:
some text some text

2. Attributes

•Attributes provide extra information about elements.
• They provide characteristics of an element. Attributes are always placed inside the opening tag of an element.
•Attributes always come in name/value pairs. Example:

3. Entities

• Entities are expanded when a document is parsed by an XML parser.
• These characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Entity References	Character
&lt;	<
&gt;	>
&amp;	&
&quot;	"
&apos;	'

4. **PCDATA**

• PCDATA means parsed character data. •It is the text found between the start tag and the end tag of an XML element.

• PCDATA is text that WILL be parsed by a parser. The text will be examined by the parser for entities and markup.

• Tags inside the text will be treated as markup and entities will be expanded

5. **CDATA**

• CDATA means character data.
• CDATA is text that will NOT be parsed by a parser.
• Tags inside the text will NOT be treated as markup and entities will not be expanded.

3.Explain the structure of an XML document with an example. List the various rules to be followed while creating an XML document.

HTML is a forgiving language. It tolerates a host of sins, from imprecise markup to altogether missing elements, and can still generate a web page in the browser. XML, on the other hand, is basically a tyrant. Violate even the most trivial rule, and the browser or your application will crash. Some people find comfort in the uncompromising nature of XML, because it won't work unless you build it correctly. It's great to get instant feedback when you do something wrong!

There are nine basic rules for building good XML:

1. XML documents may begin with a prolog that appears before the root element. It has the metadata about the XML document, such as character encoding, document structure, and style sheets. For example,

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. All XML must have a root element.

```
<?xml version="1.0" encoding="UTF-8"?>
<website>
   <company category="geeksforgeeks">
      <title>Machine learning</title>
      <author>aarti majumdar</author>
      <year>2022</year>
   </company>
   <company category="geeksforgeeks">
      <title>Web Development</title>
      <author>aarti majumdar</author>
      <year>2022</year>
   </company>
   <company category="geeksforgeekse">
      <title>XML</title>
      <author>aarti majumdar</author>
      <year>2022</year>
   </company>
</website>
```

3. All tags must be closed.

```
<message>Welcome to GeeksforGeeks</message>
```

4. All tags must be properly nested.

```
<message>

   <company>GeeksforGeeks</company>

</message>
```

5. Tag names have strict limits.
6. Tag names are case sensitive.
7. Tag names cannot contain spaces.
8. Attribute values must appear within quotes ("").

```
<website category="open source">

   <company>Hello World</company>

</website>
```

9. White space is preserved.
10. HTML tags should be avoided (optional).
11. Comments can be defined in XML enclosed between <!– and –> tags.

```
<!-- XML Comments are defined like this -->
```

XML that follows these rules is said to be "well formed." But don't confuse well-formed XML with valid XML!

For Example:

```
<root>

   <section>

      <sub-section></sub-section>

      <sub-section></sub-section>

   </section>



   <section>

      <sub-section></sub-section>
```

<sub-section></sub-section>

</section>

<root>

4.Discuss the various methods of storing and extracting XML Documents from relational databases.

Extracting XML documents from Relational Databases

1. Creating Hierarchical XML Views over Flat or Graph-Based Data

2. Breaking Cycles to Convert Graphs into Trees

3. Other Steps for Extracting XML Documents from Databases

1. **Creating g Hierarchical XML Views over Flat or Graph-Based Data**

• XML uses a hierarchical (tree) model to represent documents.
• The database systems with the most widespread use follow the flat relational data model
•When we add referential integrity constraints, a relational schema can be considered to be a graph structure

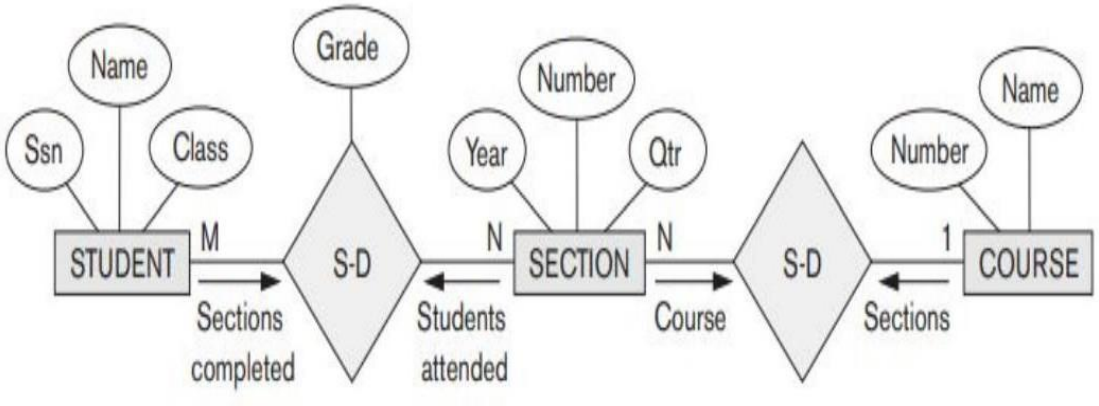


Figure 13.9
Subset of the UNIVERSITY database schema needed for XML document extraction.

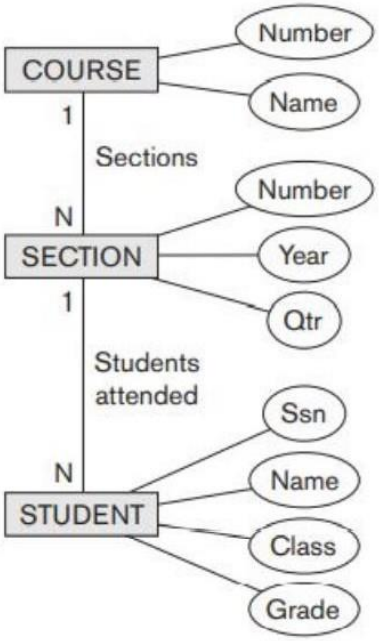


Figure 13.10
Hierarchical (tree) view with COURSE as the root.

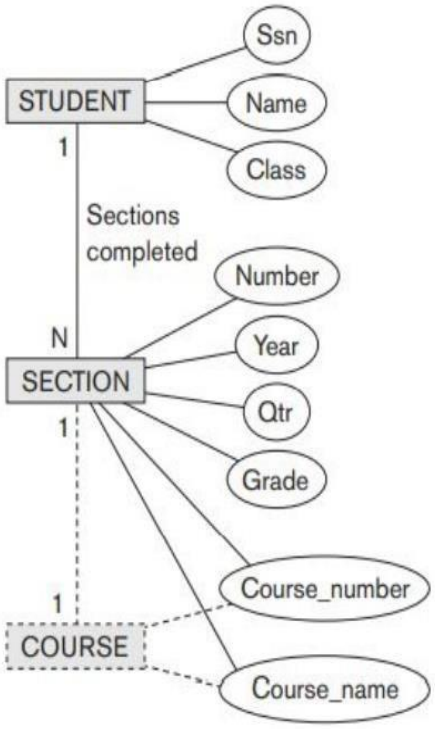


Figure 13.12
Hierarchical (tree) view with STUDENT as the root.

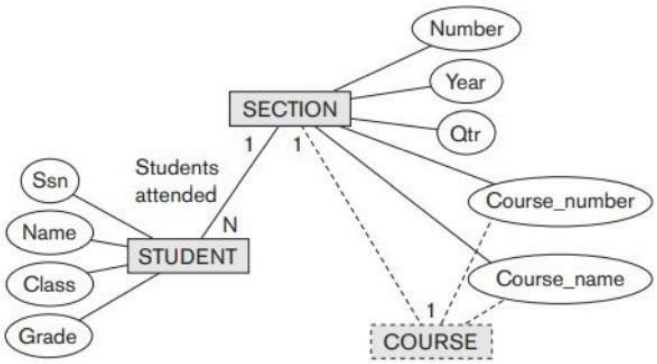


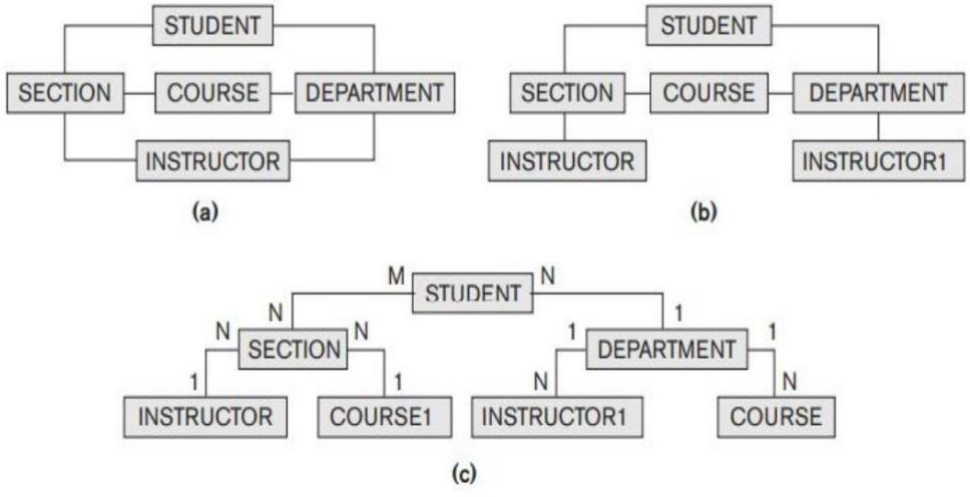
Figure 13.14
Hierarchical (tree) view with SECTION as the root.

2. **Breaking Cycles to Convert Graphs into Trees**

•It is possible to have a more complex subset with one or more cycles, indicating multiple relationships among the entities.

•It is more difficult to decide how to create the document hierarchies.

•Additional duplication of entities may be needed to represent the multiple relationships



3. **Other Steps for Extracting XML Documents from Databases**

1. It is necessary to create the correct query in SQL to extract the desired information for the XML document.

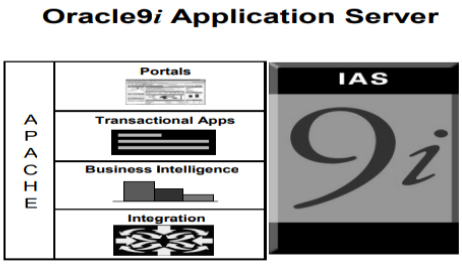
2. Once the query is executed, its result must be restructured from the flat relational form to the XML tree structure.

3. The query can be customized to select either a single object or multiple objects into the document

UNIT-3

– Explain the two products of Oracle 9i

There are two products, Oracle9i Application Server and Oracle9i Database that provide a complete and simple infrastructure for Internet applications



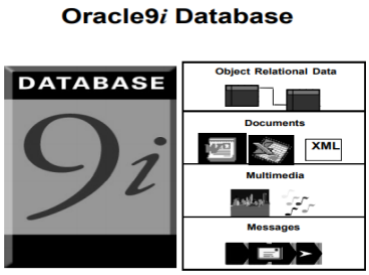
Oracle9i Application Server

* The Oracle9i Application Server (Oracle9iAS) runs all your applications.
* The Oracle9i Database stores all your data.

* Oracle9i Application Server is the only application server to include services for all the different server applications you will want to run. Oracle9iAS can run your:

• Portals or Web sites
• Java transactional applications

• Business intelligence applications



The roles of the two products are very straightforward. Oracle9i Database manages all your data. This is not just the object relational data that you expect an enterprise database to manage. It can also be unstructured data like:

• Spreadsheets

• Word documents

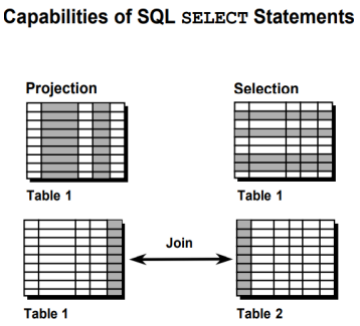
• PowerPoint presentations

• XML

• Multimedia data types like MP3, graphics, video, and more

The data does not even have to be in the database. Oracle9i Database has services through which you can store metadata about information stored in file systems. You can use the database server to manage and serve information wherever it is located

2.Explain the capabilities and syntax of SQL select statement with all clauses with examples



Capabilities of SQL SELECT Statements

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:

- **Projection**: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many columns of the table as you require.
- **Selection**: You can use the selection capability in SQL to choose the rows in a table that you want returned by a query. You can use various criteria to restrict the rows that you see
- **Joining**: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them. You learn more about joins in a later lesson.

SYNTAX:-

SELECT *|{[DISTINCT] column|expression [alias],...}
FROM table;

• SELECT identifies what columns

• FROM identifies which table

Basic SELECT Statement

In its simplest form, a SELECT statement must include the following:

• A SELECT clause, which specifies the columns to be displayed

• A FROM clause, which specifies the table containing the columns listed in the SELECT clause

In the syntax:

SELECT is a list of one or more columns

*	selects all columns
DISTINCT	suppresses duplicates
column|expression	selects the named column or the expression
alias	gives selected columns different headings

FROM table specifies the table containing the columns Note:

Throughout this course, the words keyword, clause, and statement are used as follows:

• A keyword refers to an individual SQL element.

For example, SELECT and FROM are keywords.

• A clause is a part of a SQL statement

. For example, SELECT employee_id, last_name, ... is a clause.

For example, SELECT * FROM employees is a SQL statement.

3. Explain the following comparison operator with example

a. BETWEEN... AND....

b. IN(set)

c. LIKE

d. IS NULL

Using the BETWEEN Condition

Use the BETWEEN condition to display rows based on a range of values.

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500;
```

Lower limit Upper limit

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

The BETWEEN Condition

You can display rows based on a range of values using the BETWEEN range condition. The range that you specify contains a lower limit and an upper limit. The SELECT statement on the slide returns rows from the EMPLOYEES table for any employee whose salary is between $2,500 and $3,500. Values specified with the BETWEEN condition are inclusive. You must specify the lower limit first.

Using the IN Condition

Use the IN membership condition to test for values in a list.

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

The IN Condition

To test for values in a specified set of values, use the IN condition. The IN condition is also known as the membership condition.

The slide example displays employee numbers, last names, salaries, and manager's employee numbers for all the employees whose manager's employee number is 100, 101, or 201.

The IN condition can be used with any data type. The following example returns a row from the EMPLOYEES table for any employee whose last name is included in the list of names in the WHERE clause:

SELECT employee_id, manager_id, department_id

FROM employees

WHERE last_name IN (’Hartstein’, ’Vargas’);

If characters or dates are used in the list, they must be enclosed in single quotation marks (’’)

Using the LIKE Condition

- **You can combine pattern-matching characters.**

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

- **You can use the ESCAPE identifier to search for the actual % and _ symbols.**

Combining Wildcard Characters

The % and _ symbols can be used in any combination with literal characters. The example on the slide displays the names of all employees whose last names have an o as the second character.

The ESCAPE Option

When you need to have an exact match for the actual % and _ characters, use the ESCAPE option. This option specifies what the escape character is. If you want to search for strings that contain ‘SA_’, you can use the following SQL statement:

```
SELECT employee_id, last_name, job_id
FROM   employees
WHERE  job_id LIKE '%SA\_%' ESCAPE '\';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID
149	Zlotkey	SA_MAN
174	Abel	SA_REP
176	Taylor	SA_REP
178	Grant	SA_REP

Using the NULL Conditions

Test for nulls with the IS NULL operator.

```
SELECT last_name, manager_id
FROM   employees
WHERE  manager_id IS NULL;
```

LAST_NAME	MANAGER_ID
King	

The NULL Conditions

The NULL conditions include the IS NULL condition and the IS NOT NULL condition. The IS NULL condition tests for nulls. A null value means the value is unavailable, unassigned, unknown, or inapplicable. Therefore, you cannot test with = because a null cannot be equal or unequal to any value.

For another example, to display last name, job ID, and commission for all employees who are NOT entitled to get a commission, use the following SQL statement:

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

LAST_NAME	JOB_ID	COMMISSION_PCT
King	AD_PRES	
Kochhar	AD_VP	
▪ ▪ ▪		
Higgins	AC_MGR	
Gietz	AC_ACCOUNT	

16 rows selected.

4.Explain different character functions with example

Character Functions Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-manipulation functions
- Character-manipulation functions

FUNCTION	PURPOSE
LOWER(column|expression)	Converts alpha character values to lowercase
UPPER(column|expression)	Converts alpha character values to uppercase
INITCAP(column|expression)	Converts alpha character values to uppercase for the first letter of each word, all other letters in lowercase

CONCAT(column1|expression1 , column2|expression2)	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
SUBSTR(column|expression,m [,n])	Returns specified characters from character value starting at character position m, n characters long (If m is negative, the count starts from the end of the character value. If n is omitted, all characters to the end of the string are returned.)

LENGTH(column|expression)	Returns the number of characters in the expression
INSTR(column|expression, 'string', [,m], [n])	Returns the numeric position of a named string. Optionally, you can provide a position m to start searching, and the occurrence n of the string. m and n default to 1, meaning start the search at the beginning of the search and report the function
LPAD(column|expression, n, 'string') RPAD(column|expression, n, `string	Pads the character value right-justified to a total width of n character positions Pads the character value left-justified to a total width of n character positions
TRIM(leading|trailing|both , strim_character FROM trim_source	Enables you to trim heading or trailing characters (or both) from a character string. If trim_character or trim_source is a character literal, you must enclose it in single quotes. This is a feature available from Oracle8i and later.
REPLACE(text, search_string, replacement_string)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

5. Explain different number and explicit data type conversion functions with example

Explicit Data Type Conversion

SQL provides three functions to convert a value from one data type to another:

Function	**Purpose**
TO_CHAR(*number*|*date*,[*fmt*], *[nlsparams*])	Converts a number or date value to a VARCHAR2 character string with format model *fmt*. **Number Conversion:** The nlsparams parameter specifies the following characters, which are returnedby number format elements: Decimal character Group separator Local currency symbol International currency symbol If nlsparamsor any other parameter is omitted, this function uses the default parameter values for the session.

TO_CHAR(number|date,[fmt], [nlsparams])	Date Conversion: The nlsparams parameter specifies the language in which month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session.
TO_NUMBER(char,[fmt], [nlsparams])	Converts a character string containing digits to a number in the format specified by the optional format model fmt. The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for number conversion.
TO_DATE(char,[fmt],[nlsparams])	Converts a character string representing a date to a date value according to the fmt specified. If fmt is omitted, the format is DD-MON-YY. The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for date conversion.

6. Explain different date functions with example

Function	**Description**
MONTHS_BETWEEN	**Number of monthsbetween two dates**
ADD_MONTHS	**Add calendar months todate**
NEXT_DAY	**Next day of the datespecified**
LAST_DAY	**Last day of the month**
ROUND	**Round date**
TRUNC	**Truncate date**

Date Functions

Date functions operate on Oracle dates. All date functions return a value of DATEdata type except MONTHS_BETWEEN, which returns a numeric value.

4. MONTHS_BETWEEN(*date1*, *date2*): Finds the number of months between *date1* and *date2*. The result can be positive or negative. If *date1* is later than *date2*, the result is positive; if *date1*is earlier than *date2*, the result is negative. The noninteger part of the resultrepresents a portion of the month.
5. ADD_MONTHS(*date*, *n*): Adds *n*number of calendar months to *date*. The value of *n*mustbe an integer and can be negative.
6. NEXT_DAY(*date*, '*char*'): Finds the date of the next specified day of the week ('*char*')following *date*. The value of *char*may be a number representing a day or acharacter string.
7. LAST_DAY(*date*): Finds the date of the last day of the month that contains *date*.

8. ROUND(*date*[,'*fmt*']): Returns *date*rounded to the unit specified by the format model *fmt.* If the format model *fmt* is omitted, *date* is rounded to the nearest day.
9. TRUNC(*date*[, '*fmt*']): Returns *date* with the time portion of the day truncated to theunit specified by the format model *fmt*. If the format model *fmt*is omitted, *date*is truncatedto the nearest day.

This list is a subset of the available date functions. The format models are covered later in this lesson.Examples of format models are month and year.

For example,

display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the hire month for all employees employed for fewer than 36 months.

```
SELECT employee_id, hire_date,
MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
ADD_MONTHS (hire_date, 6) REVIEW,
        NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)FROM      employees
WHERE   MONTHS_BETWEEN (SYSDATE, hire_date) < 36;
```

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(	LAST_DAY(
107	07-FEB-99	31.6982407	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	22.4079182	16-MAY-00	19-NOV-99	30-NOV-99
149	29-JAN-00	19.9885633	29-JUL-00	04-FEB-00	31-JAN-00
178	24-MAY-99	28.1498536	24-NOV-99	28-MAY-99	31-MAY-99

7. Explain different general functions and methods to build conditional expressions with example

GENERAL FUNCTION:-

These functions work with any data type and pertainto using nulls.

4. **NVL (expr1, expr2)**
5. **NVL2 (expr1, expr2, expr3)**
6. **NULLIF (expr1, expr2)**
7. **COALESCE (expr1, expr2, ..., expr*n*)**

The NVLFunction

To convert a null value to an actual value, use the NVLfunction.

Syntax

NVL (*expr1*, *expr2*)

In the syntax:

expr1 is the source value or expression that may contain a null

expr2 is the target value for converting the null

You can use the NVLfunction to convert any data type, but the return value is always the same asthe data type of *expr1*.

NVLConversions for Various Data Types

Data Type	**Conversion Example**
NUMBER	NVL(*number_column*,9)
DATE	NVL(*date_column*, '01-JAN-95')

CHAR or VARCHAR2	NVL(*character_column*, 'Unavailable')

The NVL2 Function

The NVL2function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned.

Syntax

NVL(*expr1*, *expr2*, *expr3*)

In the syntax:

expr1	is the source value or expression that may contain null
expr2	is the value returned if *expr1*is not null
expr3	is the value returned if *expr2*is null

In the example shown, the COMMISSION_PCTcolumn is examined. If a value is detected, the secondexpression of SAL+COMM is returned. If the COMMISSION_PCT column holds a null values, the third expression of SALis returned.

The argument *expr1*can have any data type. The arguments *expr2*and *expr3*can have any datatypes except LONG. If the data types of *expr2*and *expr3*are different, The Oracle server converts *expr3* to the data type of *expr2* before comparing them unless *expr3* is a null constant. In thatcase, a data type conversion is not necessary.

The data type of the return value is always the same as the data type of *expr2*, unless *expr2*is character data, in which case the return value's data type is VARCHAR

The NULLIFFunction

The NULLIF function compares two expressions. If they are equal, the function returns null. If theyare not equal, the function returns the first expression. You cannot specify the literal NULL for firstexpression.

Syntax

NULLIF (*expr1*, *expr2*)

In the syntax:

expr1	is the source value compared to *expr2*
expr2	is the source value compared with *expr1*. (If it is not equal to *expr1*, *expr1* is returned.)

In the example shown, the job ID in the EMPLOYEES table is compared to the job ID in the JOB_HISTORY table for any employee who is in both tables. The output shows the employee's current job. If the employee is listed more than once, that means the employee has held at least twojobs previously.

Note: The NULLIFfunction is logically equivalent to the following CASEexpression. The CASE
expression is discussed in a subsequent page:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

The COALESCE Function

The COALESCE function returns the first non-null expression in the list.

Syntax

COALESCE (*expr1*, *expr2*, ... *exprn*)

In the syntax:

expr1	returns this expression if it is not null
expr2	returns this expression if the first expression is null and this expression is notnull
exprn	returns this expression if the preceding expressions are null

1. Explain different types of joins with example and write the full syntax of joining tables using SQL 1999 syntax.

Joining More than Two Tables

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
2500	Oxford

Additional Search Conditions Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city

for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables. SELECT e.last_name, d.department_name, l.city FROM employees e, departments d, locations l WHERE e.department_id = d.department_id AND d.location_id = l.location_id;

LAST_NAME	DEPARTMENT_NAME	CITY
Hunold	IT	Southlake
Ernst	IT	Southlake
Lorentz	IT	Southlake
Mourgos	Shipping	South San Francisco
Rajs	Shipping	South San Francisco
Davies	Shipping	South San Francisco

Non-Equijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

JOB_GRADE

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Non-Equijoins

A non-equijoin is a join condition containing something other than an equality operator. The relationship between the EMPLOYEES table and the JOB_GRADES

table has an example of a non-equijoin. A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST_SALARY and HIGHEST_SALARY columns of the JOB_GRADES table. The relationship is obtained using an operator other than equals (=).

Non-Equijoins (continued) The slide example creates a non-equijoin to evaluate an employee's salary grade. The salary must be between any pair of the low and high salary ranges. It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

• None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.

• All of the employees' salaries lie within the limits provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Note: Other conditions, such as <= and >= can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

Returning Records with No Direct Match with Outer Joins If a row does not satisfy a join condition, the row will not appear in the query result. For example, in the equijoin condition of EMPLOYEES and DEPARTMENTS tables, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records. SELECT e.last_name, e.department_id, d.department_name FROM employees e, departments d WHERE e.department_id = d.department_id;

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping

Outer Joins Syntax

- **You use an outer join to also see rows that do not meet the join condition.**
- **The Outer join operator is the plus sign (+).**

SELECT *table1.column, table2.column* FROM *table1, table2* WHERE *table1.column(+) = table2.column;* SELECT *table1.column, table2.column* FROM *table1, table2* WHERE *table1.column = table2.column(+);*

Using Outer Joins to Return Records with No Direct Match The missing rows can be returned if an outer join operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the "side" of the join that is deficient in information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined. In the syntax: table1.column = is the condition that joins (or relates) the tables together. table2.column (+) is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides. (Place the outer join symbol

following the name of the column in the table without the matching rows.)

Self Joins

Joining a Table to Itself Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self join. For example, to find the name of Whalen's manager, you need to:

- Find Whalen in the EMPLOYEES table by looking at the LAST_NAME column.
- Find the manager number for Whalen by looking at the MANAGER_ID column. Whalen's manager number is 101.
- Find the name of the manager with EMPLOYEE_ID 101 by looking at the LAST_NAME

column. Kochhar's employee number is 101, so Kochhar is Whalen's manager. In this process, you look in the table twice. The first time you look in the table to find Whalen in the LAST_NAME column and MANAGER_ID value of 101. The second time you look in the EMPLOYEE_ID column to find 101 and the LAST_NAME column to find Kochhar.

Joining a Table to Itself SELECT worker.last_name || ' works for ' || manager.last_name FROM employees worker, employees manager WHERE worker.manager_id = manager.employee_id ;

Joining a Table to Itself (continued) The slide example joins the EMPLOYEEStable to itself. To simulate two tables in the FROM clause, there are two aliases, namely w and m, for the same table, EMPLOYEES. In this example, the WHEREclause contains the join that means "where a worker's manager number matches the employee number for the manager."

Joining Tables Using SQL: 1999 Syntax

Use a join to query data from more than one table. SELECT *table1.column, table2.column* FROM *table1* [CROSS JOIN *table2*] | [NATURAL JOIN *table2*] | [JOIN *table2* USING (*column_name*)] | [JOIN *table2* ON(*table1.column_name* = *table2.column_name*)] | [LEFT|RIGHT|FULL OUTER JOIN *table2* ON (*table1.column_name* = *table2.column_name*)];

Defining Joins Using the SQL: 1999 syntax, you can obtain the same results as were shown in the prior pages. In the syntax: *table1.column* Denotes the table and column from which data is retrieved CROSS JOIN Returns a Cartesian product from the two tables NATURAL JOIN Joins two tables based on the same column name JOIN *table* USING *column_name* Performs an equijoin based on the column name JOIN *table ON table1.column_name* Performs an equijoin based on the condition in the ON clause

= *table2.column_name LEFT/RIGHT/FULL OUTER* .

2. Explain different types of group functions with example.

Ans:- SQL GROUP Functions

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: **COUNT, MAX, MIN, AVG, SUM, DISTINCT**

SQL COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.

For Example: If you want the number of employees in a particular department, the query would be:

```
SELECT COUNT (*) FROM employee
WHERE dept = 'Electronics';
```

The output would be '2' rows.

If you want the total number of employees in all the department, the query would take the form:

```
SELECT COUNT (*) FROM employee;
```

The output would be '5' rows.

SQL DISTINCT(): This function is used to select the distinct rows.

For Example: If you want to select all distinct department names from employee table, the query would be:

```
SELECT DISTINCT dept FROM employee;
```

To get the count of employees with unique name, the query would be:

```
SELECT COUNT (DISTINCT name) FROM employee;
```

SQL MAX(): This function is used to get the maximum value from a column.

To get the maximum salary drawn by an employee, the query would be:

```
SELECT MAX (salary) FROM employee;
```

SQL MIN(): This function is used to get the minimum value from a column.

To get the minimum salary drawn by an employee, he query would be:

```
SELECT MIN (salary) FROM employee;
```

SQL AVG(): This function is used to get the average value of a numeric column.

To get the average salary, the query would be

```
SELECT AVG (salary) FROM employee;
```

SQL SUM(): This function is used to get the sum of a numeric column

To get the total salary given out to the employees,

```
SELECT SUM (salary) FROM employee;
```

3. Explain different types of subqueries with example and explain the comparison operators used in single row and multi row subqueries?

Ans:

4 .With example explain various DML statements?

Ans: DML stands for **Data Manipulation Language**. Tables and formulas are helpful when communicating with data stored up to a point in a database through SQL, but a time comes when we actually want to execute some fairly complicated data interactions. We will also need the Data Manipulation Language in that situation. DML is a way to inform a database precisely what we want it to do by conversing in a manner that it has been built to comprehend from the scratch. When it comes to interacting within existing data, whether adding, moving, or deleting data, it provides a convenient way to do so.

Characteristics :
It performs interpret-only data queries. It is used in a database schema to recall and

manipulate the information. DML It is a dialect which is used to select, insert, delete and update data in a database.

Data Manipulation Language (DML) commands are as follows:

1. **SELECT Command –**
   This command is used to get data out of the database. It helps users of the database to access from an operating system, the significant data they need. It sends a track result set from one tables or more.

Syntax :

SELECT *

FROM <table_name>;

Example :

SELECT *

FROM students;

OR

SELECT *

FROM students

where due_fees <=20000;

2. **INSERT Command –**
   This command is used to enter the information or values into a row. We can connect one or more records to a single table within a repository using this instruction. This is often used to connect an unused tag to the documents.

Syntax :

INSERT INTO <table_name> ('column_name1' <datatype>, 'column_name2' <datatype>)

VALUES ('value1', 'value2');

Example :

INSERT INTO students ('stu_id' int, 'stu_name' varchar(20), 'city' varchar(20))

VALUES ('1', 'Nirmit', 'Gorakhpur');

3. **UPDATE Command –**
   This command is used to alter existing table records. Within a table, it modifies data from one or more records. This command is used to alter the data which is already present in a table.

Syntax :

UPDATE <table_name>

SET <column_name = value>

WHERE condition;

Example :

UPDATE students

SET due_fees = 20000

WHERE stu_name = 'Mini';

4. **DELETE Command –**
   It deletes all archives from a table. This command is used to erase some or all of the previous table's records. If we do not specify the 'WHERE' condition then all the rows would be erased or deleted.

Syntax :

DELETE FROM <table_name>

WHERE <condition>;

Example :

DELETE FROM students

WHERE stu_id = '001';

5.With example explain various DDL statement.

DDL Commands :
In this section, We will cover the following DDL commands as follows.

1. Create
2. Alter
3. truncate
4. drop

Let's discuss it one by one.

Command-1 :
CREATE :
This command is used to create a new table in SQL. The user has to give information like table name, column names, and their datatypes.

Syntax –

CREATE TABLE table_name

(

column_1 datatype,

column_2 datatype,

column_3 datatype,

....

);

Example –
We need to create a table for storing Student information of a particular College. Create syntax would be as below.

CREATE TABLE Student_info

(

College_Id number(2),

College_name varchar(30),

Branch varchar(10)

);

Command-2 :

ALTER :

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can do add, delete or modify tasks easily.

Syntax –

Syntax to add a column to an existing table.

ALTER TABLE table_name

ADD column_name datatype;

Example –

In our Student_info table, we want to add a new column for CGPA. The syntax would be as below as follows.

ALTER TABLE Student_info

ADD CGPA number;

Command-3 :

TRUNCATE :

This command is used to remove all rows from the table, but the structure of the table still exists.

Syntax –

Syntax to remove an existing table.

TRUNCATE TABLE table_name;

Example –

The College Authority wants to remove the details of all students for new batches but wants to keep the table structure. The command they can use is as follows.

TRUNCATE TABLE Student_info;

Command-4 :

DROP :

This command is used to remove an existing table along with its structure from the Database.

Syntax –
Syntax to drop an existing table.

DROP TABLE table_name;

Example –
If the College Authority wants to change their Database by deleting the Student_info Table.

DROP TABLE Student_info;

6.With example explain various DCL and TCL statement.

Commands in DCL

The two most important **DCL commands** are:

- **GRANT**
- **REVOKE**

GRANT

This command is used to grant permission to the user to perform a particular operation on a particular object. If you are a database administrator and you want to restrict user accessibility such as one who only views the data or may only update the data. You can give the privilege permission to the users according to your wish.

Syntax:

GRANT privilege_list

ON Object_name

TO user_name;

REVOKE

This command is used to take permission/access back from the user. If you want to return permission from the database that you have granted to the users at that time you need to run REVOKE command.

Syntax:

REVOKE privilege_list

ON object_name

FROM user_name;

The TCL commands are:

1. COMMIT
2. ROLLBACK
3. SAVEPOINT

1. COMMIT :
This command is used to save the data permanently.
Whenever we perform any of the DDL command like -INSERT, DELETE or UPDATE, these can be rollback if the data is not stored permanently. So in order to be at the safer side COMMIT command is used.

Syntax:

commit;

2. ROLLBACK :
This command is used to get the data or restore the data to the last savepoint or last committed state. If due to some reasons the data inserted, deleted or updated is not correct, you can rollback the data to a particular savepoint or if savepoint is not done, then to the last committed state.

Syntax:

rollback;

3. SAVEPOINT :
This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

Syntax:

Savepoint A;

Consider the following Table Student:

Name	Marks
John	79
Jolly	65
Shuzan	70

UPDATE STUDENT

SET NAME = ‘Sherlock’

WHERE NAME = ‘Jolly’;

COMMIT;

ROLLBACK;

By using this command you can update the record and save it permanently by using **COMMIT** command.

Now after COMMIT :

Name	Marks
John	79
Sherlock	65
Shuzan	70

If commit was not performed then the changes made by the update command can be rollback.

Now if no **COMMIT** is performed.

UPDATE STUDENT

SET NAME = ‘Sherlock’

WHERE STUDENT_NAME = ‘Jolly’;

After update command the table will be:

Name	Marks
John	79
Sherlock	65
Shuzan	70

Now if **ROLLBACK** is performed on the above table:

rollback;

After Rollback:

Name	Marks
John	79
Jolly	65
Shuzan	70

If on the above table savepoint is performed:

INSERT into STUDENT

VALUES ('Jack', 95);

Commit;

UPDATE NAME

```
SET NAME= ‘Rossie’
WHERE marks= 70;

SAVEPOINT A;

INSERT INTO STUDENT
VALUES (‘Zack’, 76);

Savepoint B;

INSERT INTO STUDENT
VALUES (‘Bruno’, 85);

Savepoint C;

SELECT *
FROM STUDENT;
```

Name	Marks
John	79
Jolly	65
Rossie	70
Jack	95

Name	Marks
Zack	76
Bruno	85

Now if we Rollback to Savepoint B:

Rollback to B;

The resulting Table will be-

Name	Marks
John	79
Jolly	65
Rossie	70
Jack	95
Zack	76

Now if we Rollback to Savepoint A:

Rollback to A;

The resulting Table will be-

Name	Marks
John	79

Name	Marks
Jolly	65
Rossie	70
Jack	95

7.What is constraint? Explain various data integrity constraint that can be enforced on oracle database.

constraints are **the set of rules that ensures that when an authorized user modifies the database they do not disturb the data consistency** and the constraints are specified within the DDL commands like “alter” and “create” command.

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

How to specify constraints?
We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax:
Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

CREATE TABLE sample_table

(

column1 data_type(size) constraint_name,

column2 data_type(size) constraint_name,

column3 data_type(size) constraint_name,

....

);

sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

Let us see each of the constraint in detail.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

ADDRESS varchar(20)

);

2. UNIQUE –
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.
For example, the below query creates a table Student where the field ID is

specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

3. PRIMARY KEY –
Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.
A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

4. FOREIGN KEY –
Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.
Consider the two tables as shown below:

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.
Syntax:

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)
)
```

(i) CHECK –
Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

(ii) DEFAULT –
This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NsAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

8.With example explain different ways of creating defining and managing constraints.

Ans: Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK

- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

CREATE TABLE STUDENT(

ROLL_NO INT **NOT NULL**,

STU_NAME VARCHAR (35) **NOT NULL**,

STU_AGE INT **NOT NULL**,

STU_ADDRESS VARCHAR (235),

PRIMARY KEY (ROLL_NO)

);

Read more about this constraint here.

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

CREATE TABLE STUDENT(

ROLL_NO INT NOT NULL,

STU_NAME VARCHAR (35) NOT NULL **UNIQUE**,

STU_AGE INT NOT NULL,

STU_ADDRESS VARCHAR (35) **UNIQUE**,

PRIMARY KEY (ROLL_NO)

);

Read more about it here.

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

CREATE TABLE STUDENT(

ROLL_NO INT NOT NULL,

STU_NAME VARCHAR (35) NOT NULL,

STU_AGE INT NOT NULL,

EXAM_FEE INT **DEFAULT** 10000,

STU_ADDRESS VARCHAR (35) ,

PRIMARY KEY (ROLL_NO)

);

Read more: Default constraint

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

CREATE TABLE STUDENT(

ROLL_NO INT NOT NULL CHECK(ROLL_NO >1000) ,

STU_NAME VARCHAR (35) NOT NULL,

STU_AGE INT NOT NULL,

EXAM_FEE INT DEFAULT 10000,

STU_ADDRESS VARCHAR (35) ,

PRIMARY KEY (ROLL_NO)

);

In the above example we have set the check constraint on ROLL_NO column of STUDENT table. Now, the ROLL_NO field must have the value greater than 1000.

Key constraints:

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

CREATE TABLE STUDENT(

ROLL_NO INT NOT NULL,

STU_NAME VARCHAR (35) NOT NULL UNIQUE,

STU_AGE INT NOT NULL,

STU_ADDRESS VARCHAR (35) UNIQUE,

PRIMARY KEY (ROLL_NO)

);

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.
Read more about it here.

Domain constraints:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.

Domain constraints are **user defined data type** and we can define them like this:

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

DATABASE SYSTEMS

5th UNIT

1) What is database transaction? When does a transaction start and end?

➔ A database transaction consists of one of the following:
• DML statements which constitute one consistent change to the data
• One DDL statement
• One DCL statement

Database Transactions:

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure. Transactions consist of DML statements that make up one consistent change to the data. For example, a transfer of funds between two accounts should include the debit to one account and the credit to another account in the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

When Does a Transaction Start and End?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

• A COMMIT or ROLLBACK statement is issued
• A DDL statement, such as CREATE, is issued
• A DCL statement is issued • The user exits iSQL*Plus
• A machine fails or the system crashes

After one transaction ends, the next executable SQL statement automatically starts the next transaction. A DDL statement or a DCL statement is automatically committed and therefore implicitly ends a transaction.

2) What is privilege and role? What kind of privilege required for database users and explain the process of creating new user with required privilege with example?

➔ **Privileges**:

Privileges are the right to execute particular SQL statements. The database administrator (DBA) is a

high-level user with the ability to grant users access to the database and its objects. The users require system privileges to gain access to the database and object privileges to manipulate the content of the objects in the database. Users can also be given the privilege to grant additional privileges to other users or to roles, which are named groups of related privileges.

What is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges. A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

System Privileges

• More than 100 privileges are available.

• The database administrator has high-level system privileges for tasks such as:

– Creating new users

– Removing users

– Removing tables

– Backing up tables

System Privileges More than 100 distinct system privileges are available for users and roles. System privileges typically are provided by the database administrator.

CREATING USER

The DBA creates users by using the CREATE USER statement.

```
eate USER user
ENTIFIED BY password;
```

```
REATE USER scott
ENTIFIED BY tiger;
er created.
```

Creating a User:

The DBA creates the user by executing the CREATE USER statement. The user does not have any privileges at this point. The DBA can then grant privileges to that user. These privileges determine what the user can do at the database level.

The slide gives the abridged syntax for creating a user.

In the syntax:

User: is the name of the user to be created

Password: specifies that the user must log in with this password.

3)What is view and What are the advantages and rules for performing DML operations on view?

➔ **What Is a View?**

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called base tables. The view is stored as a SELECT statement in the data dictionary.

Advantages of Views:

• Views restrict access to the data because the view can display selective columns from the table.

• Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.

• Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.

• Views provide groups of users access to data according to their particular criteria.

Rules for Performing DML Operations on a View:

• You can perform DML operations on simple views.
• You cannot remove a row if the view contains the following:
– Group functions
– A GROUP BY clause
– The DISTINCT keyword
– The pseudocolumn ROWNUM keyword

You cannot modify data in a view if it contains:
• Group functions
• A GROUP BY clause
• The DISTINCT keyword
• The pseudocolumn ROWNUM keyword
• Columns defined by expression

Performing DML Operations on a View (continued):
You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions—for example, SALARY * 12**.**

You cannot add data through a view if the view includes:
• Group functions
• A GROUP BY clause
• The DISTINCT keyword
• The pseudocolumn ROWNUM keyword
• Columns defined by expressions
• NOT NULL columns in the base tables that are not selected by the view
Performing DML Operations on a View (continued):
You can add data through a view unless it contains any of the items listed in the slide or there are NOT NULL columns without default values in the base table that are not selected by the view. All required values must be present in the view. Remember that you are adding values directly into the underlying table through the view.

4)Explain with example the syntax of creating view with all options?

➔ **Creating a View**:
• You embed a subquery within the CREATE VIEW statement.

```
eate [OR REPLACE] [FORCE|NOFORCE] VIEW view
 [alias[, alias]. . .)]
As subquery
ITH CHECK OPTION [CONSRAINT constraint]]
ITH READ ONLY [CONSTRAINT constraint]];
```

• The subquery can contain complex SELECT syntax.

Creating a View:

You can create a view by embedding a subquery within the CREATE VIEW statement.

In the syntax:

OR REPLACE: re-creates the view if it already exists

FORCE: creates the view regardless of whether or not the base tables exist

NOFORCE: creates the view only if the base tables exist (This is the default.)

View: is the name of the view

Alias: specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)

Subquery: is a complete SELECT statement (You can use aliases for the columns in the SELECT list.)

WITH CHECK OPTION: specifies that only rows accessible to the view can be inserted or updated

Constraint: is the name assigned to the CHECK OPTION constraint

WITH READ ONLY: ensures that no DML operations can be performed on this view

Creating a View:

• Create a view, EMPVU80, that contains details of employees in department 80.

```
CREATE VIEW empvu80
AS SELECT employee_id,  last_name, salary
    FROM employees
    WHERE department_id = 80;
View created.
```

• Describe the structure of the view by using the iSQL*Plus DESCRIBE command.

```
DESCRIBE empvu80
```

Creating a View (continued) The example on the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the iSQL*Plus DESCRIBE command.

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR(25)
SALARY		NUMBER(0,2)

Guidelines for creating a view:

• The subquery that defines a view can contain complex SELECT syntax, including joins, groups, and subqueries.

• The subquery that defines the view cannot contain an ORDER BY clause. The ORDER BY clause is specified when you retrieve data from the view.

• If you do not specify a constraint name for a view created with the WITH CHECK OPTION, the system assigns a default name in the format SYS_Cn.

• You can use the OR REPLACE option to change the definition of the view without dropping and re-creating it or regranting object privileges previously granted on it.

Creating a View

• Create a view by using column aliases in the subquery.

```
CREATE VIEW salvu50
AS SELECT employee_id ID_NUMBER, last_name Name,
          Salary*12 ANN_SALARY
FROM      employees
WHERE     department_id = 50;
View created.
```

• Select the columns from this view by the given alias names

Creating a View (continued):

You can control the column names by including column aliases within the subquery. The example on the slide creates a view containing the employee number (EMPLOYEE_ID) with the alias ID_NUMBER, name (LAST_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN_SALARY for every employee in department 50. As an alternative, you can use an alias after the CREATE statement and prior to the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery. CREATE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY) AS SELECT employee_id, last_name, salary*12 FROM employees WHERE department_id = 50;

View created.

5)What is sequence? Explain with example the syntax of creating sequence and how to use it?

➔ **What Is a Sequence?**

A sequence is a user created database object that can be shared by multiple users to generate unique integers. A typical usage for sequences is to create a primary key value, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence-generating routine. Sequence numbers are stored and generated independently of tables. Therefore, the same sequence can be used for multiple tables.

The CREATE SEQUENCE Statement Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
      [INCREMENT BY n]
      [START WITH n]
      [ {MAXVALUE n  | NOMAXVALUE} ]
      [ {MINVALUE  n   | NOMINVALUE} ]
      [ {CYCLE  |  NOCYCLE} ]
      [ {CACHE n | NOCACHE} ] ;
```

Creating a Sequence

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

Sequence: is the name of the sequence generator

INCREMENT BY n: specifies the interval between sequence numbers where n is an integer (If this clause is omitted, the sequence increments by 1.)

START WITH n: specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)

MAXVALUE n: specifies the maximum value the sequence can generate

NOMAXVALUE: specifies a maximum value of 10^27 for an ascending sequence and –1 for a descending sequence (This is the default option.)

MINVALUE n: specifies the minimum sequence value

NOMINVALUE: specifies a minimum value of 1 for an ascending sequence and – (10^26) for a descending sequence (This is the default option.)

CYCLE | NOCYCLE: specifies whether the sequence continues to generate values after reaching its maximum or minimum value (NOCYCLE is the default option.)

CACHE n | NOCACHE: specifies how many values the Oracle server preallocates and keep in memory (By default, the Oracle server caches 20 values).

Creating a Sequence

• Create a sequence named DEPT_DEPTID_SEQ to be used for the primary key of the DEPARTMENTS table.

• Do not use the CYCLE option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
SEQUENCE CREATED.
```

Creating a Sequence (continued):

The example on the slide creates a sequence named DEPT_DEPTID_SEQ to be used for the DEPARTMENT_ID column of the DEPARTMENTS table. The sequence starts at 120, does not allow caching, and does not cycle.

Do not use the CYCLE option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

Using a Sequence:

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

NEXTVAL and CURRVAL Pseudocolumns

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference sequence.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When sequence.CURRVAL is referenced, the last value returned to that user's process is displayed.

6)What is index? Explain how to create index with example and what are the types of indexes?

➔ **Indexes**:

An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, then a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. The index is used and maintained automatically by the Oracle server. Once an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

Types of Indexes:

Two types of indexes can be created. One type is a unique index: the Oracle server automatically creates this index when you define a column in a table to have a PRIMARY KEY or a UNIQUE key constraint.

The name of the index is the name given to the constraint. The other type of index is a nonunique index, which a user can create. For example, you can create a FOREIGN KEY column index for a join in a query to improve retrieval speed.

Note: You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

How Are Indexes Created?

- **Automatically: A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a table definition.**
- **Manually: Users can create nonunique indexes on columns to speed up access to the rows**

Creating an Index

- **Create an index on one or more columns.**

```
REATE INDEX index
N table   (column[ ,  column]. . . );
```

- **Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table.**

```
REATE INDEX emp_last_name_idx
N                 employees(last_name) ;
DEX CREATED
```

Creating an Index:

Create an index on one or more columns by issuing the CREATE INDEX statement.

In the syntax:

Index :is the name of the index

Table: is the name of the table

Column: is the name of the column in the table to be indexed.