Inner / Outer Joins

Joins

Used to combine two or more tables, joins are a database technique used in SELECT statements. Joins are normally performed comparing primary keys to foreign keys, however, they can be performed with any type of column, as long as the types match. Joins are a way of *denormalizing* a set of tables in order to aggregate or process some query.

They can be performed in many ways:

- INNER JOIN.
  - The most commonly used type of join, returns rows only if the columns specified in the join clause match.
- OUTER JOIN.
  - The OUTER keyword can be used with LEFT, RIGHT or FULL keywords to obtain rows which some of the join columns are NULL.
  - However, in Oracle, this word is optional. LEFT, RIGHT or FULL will be automatically OUTER.
- LEFT [OUTER] JOIN.
  - Returns the matching rows plus the ones that where null in the first table.
- RIGHT [OUTER] JOIN.
  - Returns the matching rows plus the ones that where null on the second table.
- FULL [OUTER] JOIN.
  - Returns all rows from both tables specified including the ones which had null values on either side.
- CROSS JOIN.
  - Returns the Cartesian product two or more tables.
- SELF JOIN.
  - An INNER JOIN performed matching two columns existing in the same table.
  - They represent hierarchies.
- NATURAL JOIN
  - Used as a shortcut so that the join predicate is not needed to be specified
  - The tables are joined on matching column names

As you can see, all JOINS are INNER unless otherwise specified with keywords.

To write a join:

```
SELECT A.FIRSTNAME, C.NAME FROM STUDENT S INNER JOIN COURSE C  ON S.COURSEID = C.ID;

SELECT A.FIRSTNAME, C.NAME FROM STUDENT S, COURSE C  WHERE S.COURSEID = C.ID;

SELECT A.FIRSTNAME, C.NAME FROM STUDENT S, COURSE C  WHERE S.COURSEID = C.ID(+);
```

To optimize joins, put the tables to join from more data to less data, and then perform the joins in order.

Set Operators

Set operators are different from joins. Instead of combining columns of two tables, set operators combine the rows of different result sets. Essentially, set operators perform some kind of (set) operation on two different queries.

Some set operators are:

- UNION [ALL]
  - UNION does not keep duplicates, but UNION ALL will
- INTERSECT
  - Only returns records in common between the queries
- MINUS
  - Removes from the first result set any rows that appear in the second result set and returns what remains
- EXCEPT
  - Same as MINUS, but for SQLServer instead of Oracle

Views

Views are virtual tables - they are constructed from DQL queries and provide a window or "view" into the table. Views can be used to provide access to some portion of the data in a table but not all, which might be useful if the data is sensitive and needs to be kept private. Views are also used to abstract or hide complexity in the database - a view could be constructed with joins over multiple tables so that end users can query from denormalized tables easily. Users can query views just as if they were normal tables. Changes to the underlying table will be reflected in the view whenever it is queried next.

Materialized Views

Materialized views were introduced by Oracle and give a static snapshot of the data at a given point in time.

Creating Views

Create a view with the following syntax:

```
CREATE VIEW [viewname] AS [any DQL statement]
```

For example,

```
CREATE VIEW myview AS SELECT col1, col2 FROM mytable WHERE mycolumn > 12
```

Now, we can query from the view just as we would from a table:

```
SELECT * FROM myview WHERE mycolumn > 20
```

Indexes

To explain the concept of **indexes**, think of the purpose of indexes in a book - you use them to speed up the process of locating some specific piece of information. You wouldn't want to skim an entire book to find where one word was mentioned; similarly, indexes allow databases to speed up the process of retrieving data.

When an index is created on a database column, a separate data structure is created in the database (typically some sort of balanced tree), which stores references that point to the actual records in the table. This data structure can be searched much more quickly than searching through the entire table itself. Significant performance benefits can be achieved by implementing indexes on commonly queried columns. For this reason, primary keys are automatically indexed by the RDBMS system.

Types of Indexes

Indexes can be categorized into two types: clustered and non-clustered. Clustered indexes alter the order in which the records are physically stored on disk. Thus only one clustered index can be created on a given table. Non-clustered indexes specify a logical ordering of rows but do not affect the physical ordering, so there may be more than one non-clustered index in a table.

Additionally, there are further types of indexes:

- Bitmap
- Dense
- Sparse
- Reverse
- Primary
- Secondary

In-depth knowledge of indexes is not necessary for our training purposes, however.

Creating Indexes

The syntax to create an index on a table's column is:

```
CREATE INDEX index_name ON table_name (col1, col2, ...)
```

Deleting Indexes

```
ALTER TABLE table_name DROP INDEX index_name
```

Best Practices

You should consider adding an index to a column **when you anticipate or already know that the column will often be used** when searching the table for records. Often you will not know which columns should be indexed until you have gathered some data about queries run on the system in production.

Stored Procedures vs. functions

PL/SQL

PL/SQL stands for the Procedural Language extension to SQL and it is a complete programming language which also allows SQL statements.

Besides the programming itself, Oracle offers an OOP aspect to its databases, providing certain types of objects, some of which require PL/SQL code. CREATE [OR REPLACE] and DROP statements are allowed for these objects. ALTER is not allowed.

Sequence

A **sequence** is an object which holds a numeric number that starts from a certain point and it also contains a max. It increments by a specific amount every time NEXTVAL is called.

They can be combined with **Triggers** to auto increment primary key columns.

```
CREATE [OR REPLACE] SEQUENCE START WITH 1 INCREMENT BY 1
```

Trigger

A **trigger** is a block of code that executes when a specific event happens. These events can be INSERT, UPDATE or DELETE statements, and they can happen AFTER or BEFORE.

Syntax:

```
CREATE [OR REPLACE] TRIGGER
BEFORE INSERT ON TABLE_NAME
  FOR EACH ROW
  BEGIN
    (PL/SQL code)
  END;
```

Cursor

Pointers to a result set. They can be used to loop programmatically on the output of a SELECT statement (similar to iterators in Java).

Oracle provides `SYS_REFCURSOR`, its own type of `REFCURSOR`. This means you can create your own type of `REFCURSOR`.

Stored Procedure VS Functions

Stored Procedure

PL/SQL code that can be executed in certain ways and has some properties:

- They don't return anything.
- They may or may not contain IN (by value) and OUT (by reference) parameters.
- They allow any DML statements within.
    - These means transactions can be created in a stored procedure.
- Stored procedures can call other procedures and functions.
- Can NOT use stored procedures in DML statements.
    - `EXEC STORED_PROCEDURE`

Function

Also known as User Defined Functions, these are like stored procedures but have some other restrictions or abilities:

- They must return something.
- Cursors are allowed.
- It should be a single value.
- They may or may not contain IN parameters (by default).
- Only SELECT statements are allowed.
- Functions can only call other functions (no stored procedures).
- They can be used in any DML statement.
- To call a function, you have to use a DML statement (you can't `EXEC`).
- Use `FROM DUAL` if you are not selecting from a specific table.
- `DUAL` is a dummy table which returns anything your throw at it.

Types

Similar to classes, they can be used as your own data types for columns. We are not going to use this, because this is a high level technique which doesn't follow the normal forms.

VARRAY

Arrays of any Oracle data type or your own type, they can be used for columns.

Nested Table

Infinite arrays, they can be seen as tables in columns. They have to be of a specific data type.