Normalization and Indexing in Oracle

Normalization in Oracle

Normalization is a database design technique to reduce data redundancy and ensure data integrity by organizing data into multiple related tables. The goal is to break down large tables into smaller, well-structured tables that follow specific normal forms.

🔑 Key Normal Forms

1. **First Normal Form (1NF):**

- Ensures each column has atomic (indivisible) values.

- No repeating groups or arrays in columns.

2. **Second Normal Form (2NF)**

- Meets 1NF criteria.

- Removes partial dependencies (no non-prime attribute depends on part of a composite primary key).

3. **Third Normal Form (3NF)**

- Meets 2NF criteria.

- Removes transitive dependencies (no non-prime attribute depends on another non-prime attribute).

4. **Boyce-Codd Normal Form (BCNF):**

- A stronger version of 3NF where every determinant is a candidate key.

Example

Unnormalized Table

A `Student` table containing student details with multiple courses:

Student_ ID	Name	Course	Instructor
1	Ayub	Math	Belling Hum
1	Ayub	Physics	Brown
2	Miaze	Math	Kamal
2	Miaze	Chemistry	White

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

Step 1: Convert to 1NF

Separate repeating groups into rows, ensuring atomic values in each column:

Student_ ID	Name	Course	Instructor
1	Ayub	Math	Belling Hum
1	Ayub	Physics	Brown
2	Miaze	Math	Kamal
2	Miaze	Chemistry	White

Step 2: Convert to 2NF

Remove partial dependencies by separating courses into a different table:

Table 1: Student

Student_ ID	Name
1	Ayub
1	Ayub
2	Miaze
2	Miaze

Table 2: Course

Course	Instructor
Math	Belling Hum
Physics	Brown
Math	Kamal
Chemistry	White

Table 3: Student_Course

Student_ ID	Course
1	Math
1	Physics
2	Math
2	Chemistry

Step 3: Convert to 3NF

Remove transitive dependencies. For example, if `Instructor` depends on `Course`, separate it:

Table 1: Student

Student_ ID	Name
1	Ayub
2	Miaze

Table 2: Course

Course
Math
Physics
Chemistry

Table 2: Instructor

Instructor	
Belling Hum	Math
Brown	Physics
Kamal	Math
White	Chemistry

Table 4: Student_Course

Student_ ID	Course
1	Math
1	Physics
2	Math
2	Chemistry

Benefits of Normalization

1. **Eliminates Redundancy:** Prevents duplicate data.

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

2. **Improves Integrity:** Ensures consistent data through relationships.

3. **Facilitates Maintenanc**e: Easier to update and scale.

By following normalization principles, the database becomes more efficient, reducing storage costs and increasing data accuracy.

Indexing in Oracle:

In Oracle, indexing is a crucial feature for optimizing query performance by allowing faster retrieval of rows. Oracle provides several types of indexes, each suitable for specific use cases.

Types of Indexes in Oracle

1. **B-Tree Index** (Default Index)

- The most common type of index.

- Organizes data in a balanced tree structure.

- Suitable for queries that return a small subset of rows.

Example:

```
CREATE  INDEX  idx_employee_name ON employees (employee_name);
```

Use Case:

Querying employees by name:

```
SELECT * FROM employees WHERE employee_name = 'John';
```

2. **Bitmap Index**

- Uses bitmaps for storage.

- Efficient for columns with low cardinality (few distinct values, e.g., gender, status).

- Best for read-intensive operations like analytical queries.

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

Example:

```
CREATE BITMAP INDEX idx_emp_gender ON employees (gender);
```

Use Case:

Querying by gender:

SELECT * FROM employees WHERE gender = 'M';

3. Unique Index

- Ensures that all values in the indexed column(s) are unique.

- Automatically created when a `UNIQUE` or `PRIMARY KEY` constraint is defined.

Example:

```
CREATE UNIQUE INDEX idx_unique_email ON employees (email);
```

Use Case:

Ensuring unique email addresses:

```
SELECT * FROM employees WHERE email = 'john.doe@example.com';
```

4. Composite Index (Multi-Column Index)

- Combines multiple columns into a single index.

- Useful when queries filter on multiple columns.

Example:

```
CREATE INDEX idx_emp_name_dept ON employees (employee_name, department_id);
```

Use Case:

Querying by name and department:

```
SELECT * FROM employees WHERE employee_name = 'John' AND department_id = 10;
```

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

5. Function-Based Index

- Indexes the result of a function or expression.

- Useful for queries involving expressions or functions on columns.

Example:

```
CREATE INDEX idx_upper_name ON employees (UPPER(employee_name));
```

Use Case:

Querying with case-insensitive comparison:

```
SELECT * FROM employees WHERE UPPER(employee_name) = 'JOHN';
```

6. Reverse Key Index

- Reverses the bytes of the indexed column's values.

- Useful to avoid contention in inserting sequential values in high-concurrency environments.

Example:

```
CREATE INDEX idx_reverse_emp_id ON employees (employee_id) REVERSE;
```

Use Case:

Optimizing inserts for a high-transaction table:

```
 INSERT INTO employees (employee_id, employee_name) VALUES (101, 'John');
```

7. Domain Index

- Custom indexes defined for specific application requirements.

- Useful for indexing complex data types like spatial, text, or multimedia.

Example (Text Index):

```
CREATE INDEX idx_text_description ON products (description) INDEXTYPE IS
CTXSYS.CONTEXT;
```

Use Case:

Full-text search:

```
SELECT * FROM products WHERE CONTAINS(description, 'laptop') > 0;
```

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

8. Cluster Index

- Created automatically when creating a cluster.

- Organizes rows based on cluster key values.

Example:

```
CREATE CLUSTER emp_dept_cluster (department_id NUMBER);

CREATE INDEX idx_emp_dept_cluster ON CLUSTER emp_dept_cluster;
```

Use Case:

Efficiently query employees grouped by department:

```
SELECT * FROM employees WHERE department_id = 10;
```

9. Global and Local Partitioned Indexes

- **Global Partitioned Index**: Index spans all partitions in the table.

- **Local Partitioned Index:** Separate index for each table partition.

Example (Local Index):

```
CREATE INDEX idx_local_sales ON sales (sale_date) LOCAL;
```

Use Case:

Optimizing queries for partitioned tables:

```
SELECT * FROM sales WHERE sale_date = TO_DATE('2024-11-20', 'YYYY-MM-DD');
```

10. Invisible Index

- Hidden from the optimizer but available for testing and maintenance.

Example:

Use Case:

Test the performance impact of an index:

```
ALTER INDEX idx_invisible_salary VISIBLE;
```

11. Virtual Column-Based Index

- Index created on a virtual column derived from other columns.

Ayub miaze
20/11/2024

Normalization and Indexing in Oracle

Example:

```
ALTER TABLE employees ADD full_name AS (first_name || ' ' || last_name);

CREATE INDEX idx_full_name ON employees (full_name);
```

Use Case:

Query using the virtual column:

```
SELECT * FROM employees WHERE full_name = 'John Doe';
```

Choosing the Right Index

1. **B-Tree Index:** General-purpose queries.

2. **Bitmap Index:** Low-cardinality columns, analytical queries.

3. **Function-Based Index**: Expressions or function results.

4. **Composite Index**: Multi-column filtering.

5. **Partitioned Index**: Partitioned tables for scalability.

Indexes improve query performance but increase overhead for DML (INSERT, UPDATE, DELETE). Use them judiciously.

Ayub miaze
20/11/2024