INDEX

INDEX

- Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval. Simply, **an index is a pointer to data in a table**.
- An index in a database is very similar to an index in the back of a book.
- For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.
- An index helps to speed up **SELECT** queries and **WHERE** clauses.
- Indexes can be created or dropped with no effect on the data.

WHY INDEX?

Database is composed of lots of tables and views. Each table will have lots of data in them. Main purpose of storing all these data in the database is to get them handy when it is required.

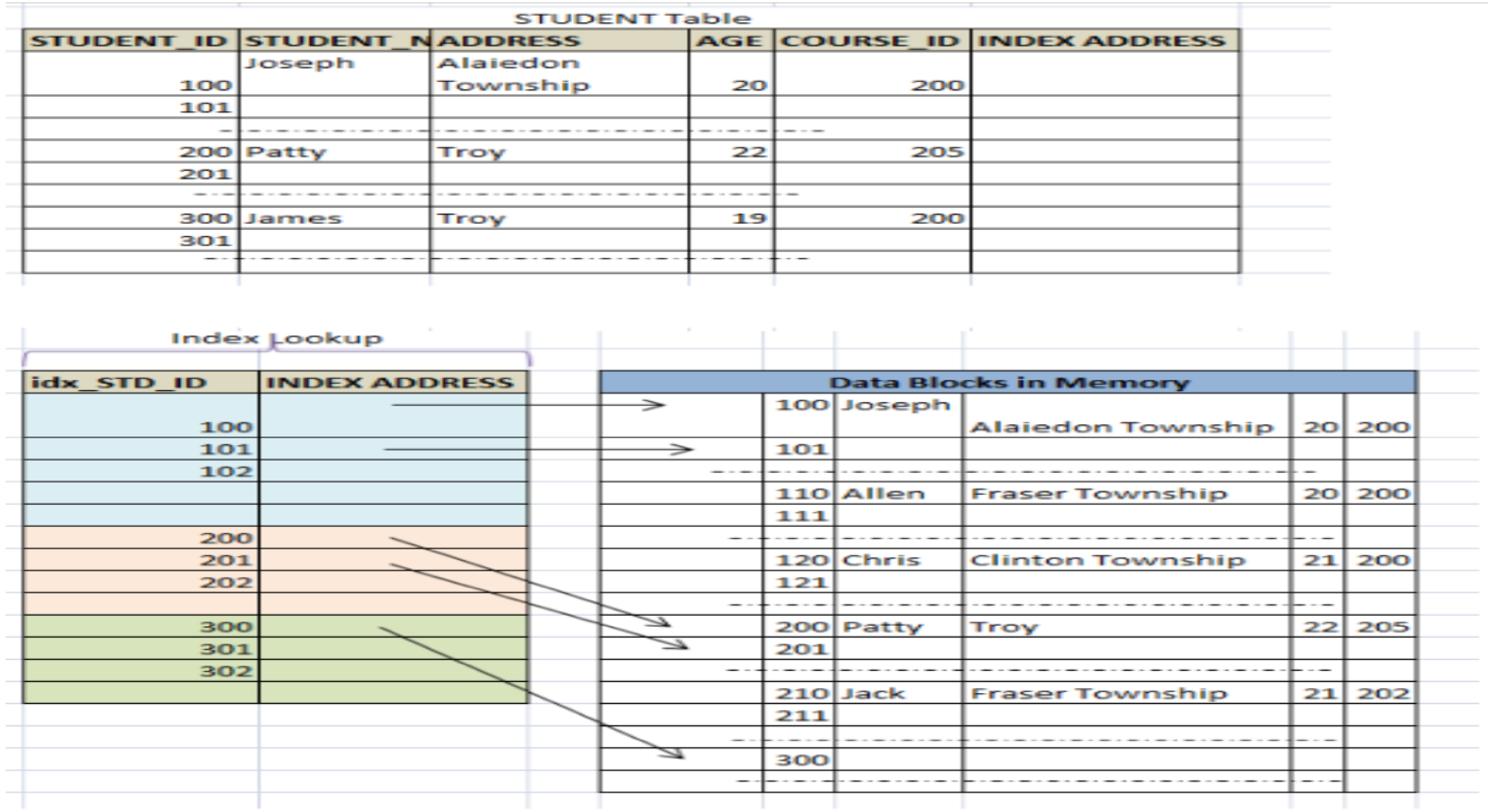
But when a table is very huge, searching for single record is really a difficult task. The usual method of searching the record in the table is to start from the beginning record by record, till we get the record. But it is not feasible when table is large. It will take lot of time.

In order to reduce the time to fetch the record, another table like structure is introduced where pointer to the memory address is stored for each record. This is called as index.

HOW INDEX WORKS?

- The index table consists of columns which are frequently used for retrieval of records and the address of data block in the memory.
- Primary key itself acts as an index in a table.
- We can create any number of indexes, but we have to be cautious while creating index since it slows down data insertion and updating speed.
- **Index** can be created on **one or more columns**. If the index is created on two or more columns to form one index, then the index is called **composite index or concatenated index**.
- Index is usually created to enhance the performance of retrieval of records in huge tables. But sometimes, improper creation of index and usage will cause bad performance of the query. Hence proper management of the index is required.

EXAMPLE



STUDENT Table

STUDENT_ID	STUDENT_N	ADDRESS	AGE	COURSE_ID	INDEX ADDRESS
100	Joseph	Alaiedon Township	20	200	
101					
- - -	- - -	- - -	- - -	- - -	
200	Patty	Troy	22	205	
201					
- - -	- - -	- - -	- - -	- - -	
300	James	Troy	19	200	
301					
- - -	- - -	- - -	- - -	- - -	

Index Lookup

idx_STD_ID	INDEX ADDRESS
100	
101	
102	
200	
201	
202	
300	
301	
302	

Data Blocks in Memory

	100	Joseph	Alaiedon Township	20	200
	101				
	- - -	- - -	- - -	- - -	
	110	Allen	Fraser Township	20	200
	111				
	- - -	- - -	- - -	- - -	
	120	Chris	Clinton Township	21	200
	121				
	- - -	- - -	- - -	- - -	
	200	Patty	Troy	22	205
	201				
	- - -	- - -	- - -	- - -	
	210	Jack	Fraser Township	21	202
	211				
	- - -	- - -	- - -	- - -	
	300				
	- - -	- - -	- - -	- - -	

GUIDELINES FOR MANAGING INDEX:

- Create Indexes After Inserting Table Data.
- Limit the Number of Indexes for Each Table.
- Drop Indexes That Are No Longer Required.
- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

SAMPLE RELATION - COURSES:

COURSE_ID	COURSE_NAME	DURATION	FEES
JA154	JAVA	12	25300
OR215	ORACLE	15	28500
CP136	C	8	22300
RY258	RUBY	16	29000
PL256	PASCAL	6	15000

SINGLE-COLUMN INDEX:

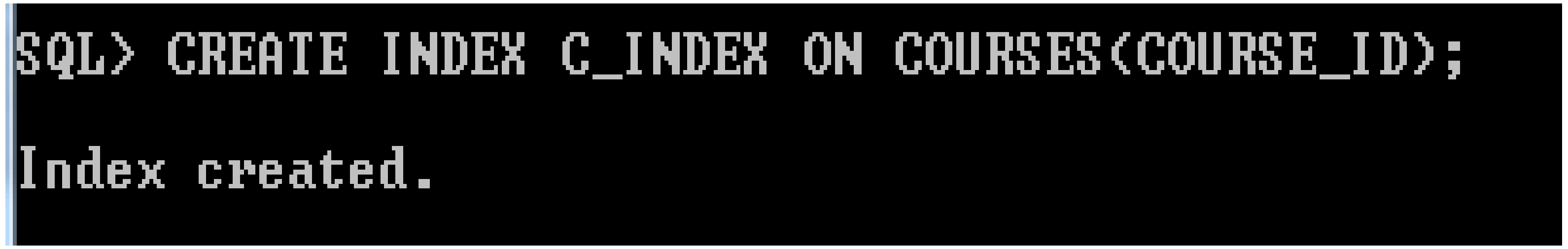
A single-column index is created based on only one table column. Duplicate values are allowed.

SYNTAX:

CREATE INDEX index_name

ON table_name (column_name);

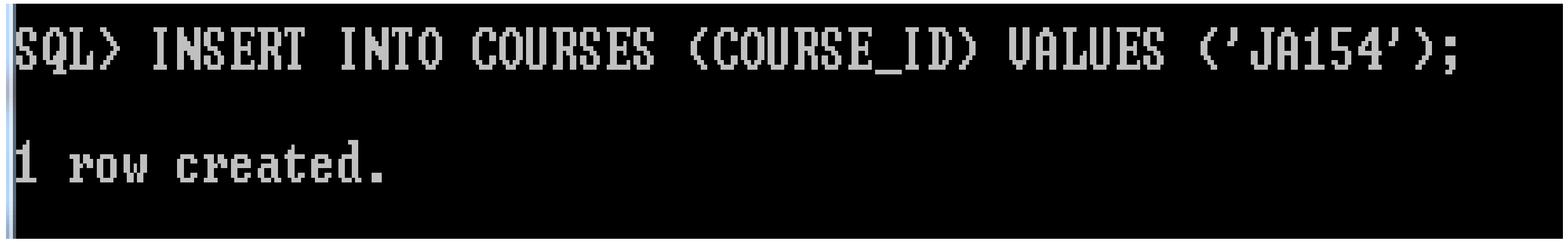
EXAMPLE:



```
SQL> CREATE INDEX C_INDEX ON COURSES(COURSE_ID);

Index created.
```

SINGLE-COLUMN INDEX:

Example to illustrate Single column index allows Duplicate values:



```
SQL> INSERT INTO COURSES (COURSE_ID) VALUES ('JA154');

1 row created.
```

The COURSE_ID column is created as an index column. It allows the insertion of the value ‘JA154’ which is being inserted already. Hence single-column index allows duplicate values which is not permitted in UNIQUE Index.

COMPOSITE INDEX:

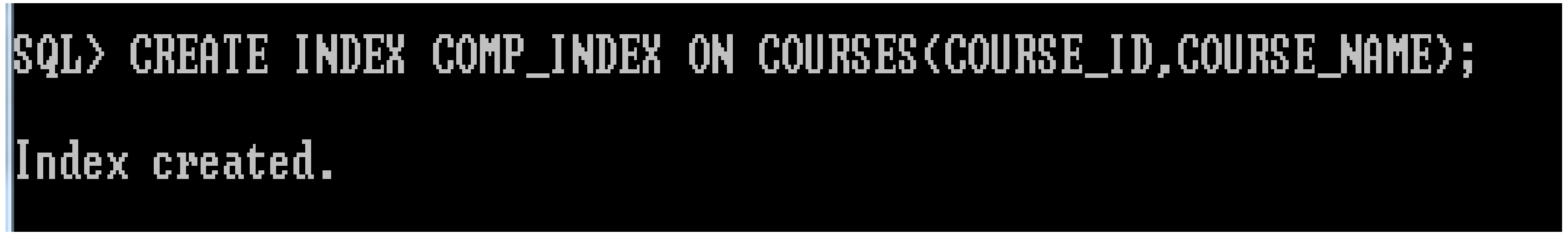
A composite index is an index on two or more columns of a table.

SYNTAX:

CREATE INDEX index_name

on table_name (column1, column2);

EXAMPLE:



```
SQL> CREATE INDEX COMP_INDEX ON COURSES(COURSE_ID,COURSE_NAME);

Index created.
```

NOTE:

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

IMPLICIT INDEX:

- Implicit indexes are indexes that are automatically created by the database server when an object/table is created.
- Indexes are automatically created for primary key constraints and unique constraints.

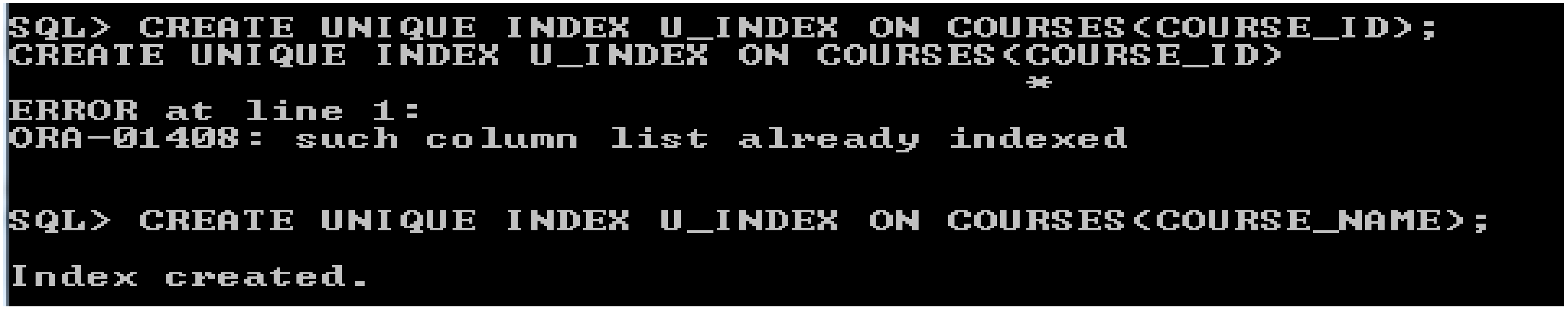
UNIQUE INDEX:

- A unique index does not allow any duplicate values to be inserted into the table. Unique indexes are used not only for performance, but also for data integrity.

SYNTAX:

CREATE UNIQUE INDEX index_name On table_name (column_name);

EXAMPLE:

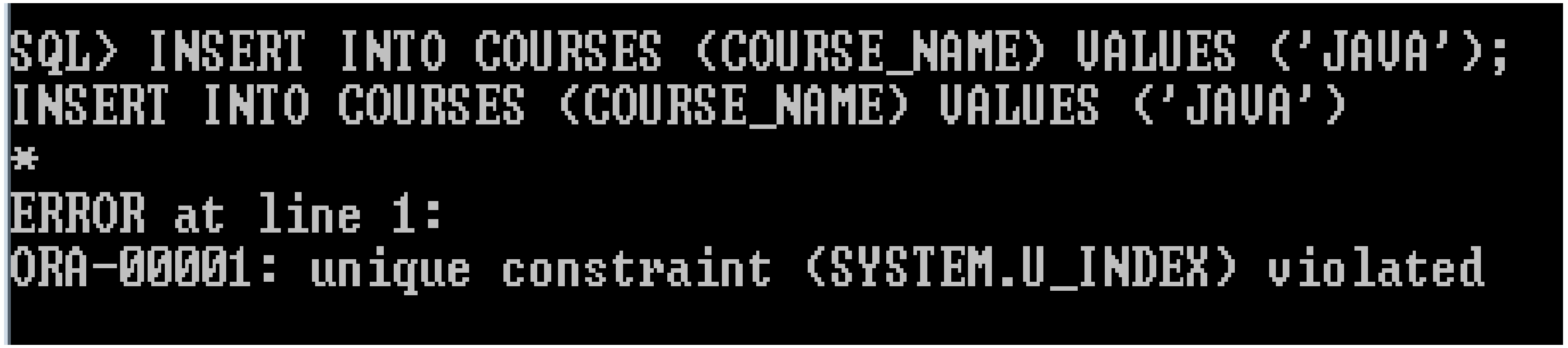


```
SQL> CREATE UNIQUE INDEX U_INDEX ON COURSES(COURSE_ID);
CREATE UNIQUE INDEX U_INDEX ON COURSES(COURSE_ID)
                                       *
ERROR at line 1:
ORA-01408: such column list already indexed


SQL> CREATE UNIQUE INDEX U_INDEX ON COURSES(COURSE_NAME);

Index created.
```

Note: The column which is already indexed cannot be indexed again.

UNIQUE INDEX:

Example to illustrate Unique index does not allows Duplicate values:



```
SQL> INSERT INTO COURSES (COURSE_NAME) VALUES ('JAVA');
INSERT INTO COURSES (COURSE_NAME) VALUES ('JAVA')
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.U_INDEX) violated
```

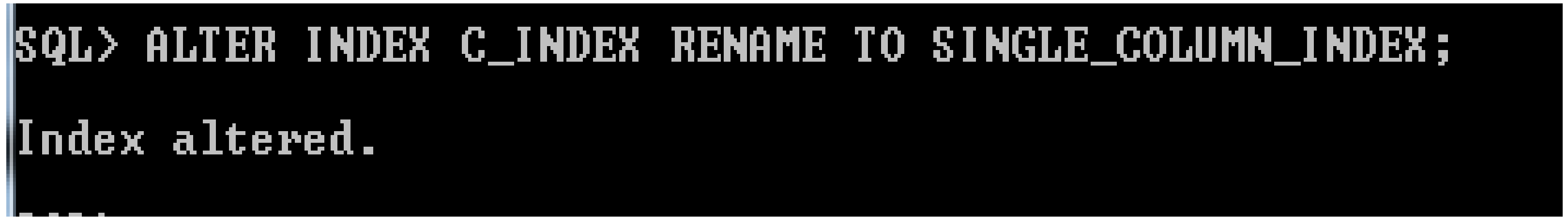
Unique Index behaves like a unique constraint and it avoids duplicate values.

RENAMING AN INDEX:

SYNTAX:

ALTER INDEX index_name RENAME TO new_index_name;

EXAMPLE:



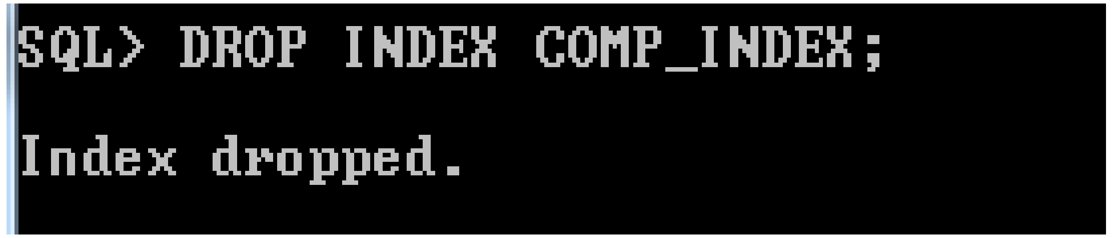
```
SQL> ALTER INDEX C_INDEX RENAME TO SINGLE_COLUMN_INDEX;

Index altered.
```

DROPPING AN INDEX:

SYNTAX:

DROP INDEX index_name;

EXAMPLE:



```
SQL> DROP INDEX COMP_INDEX;

Index dropped.
```