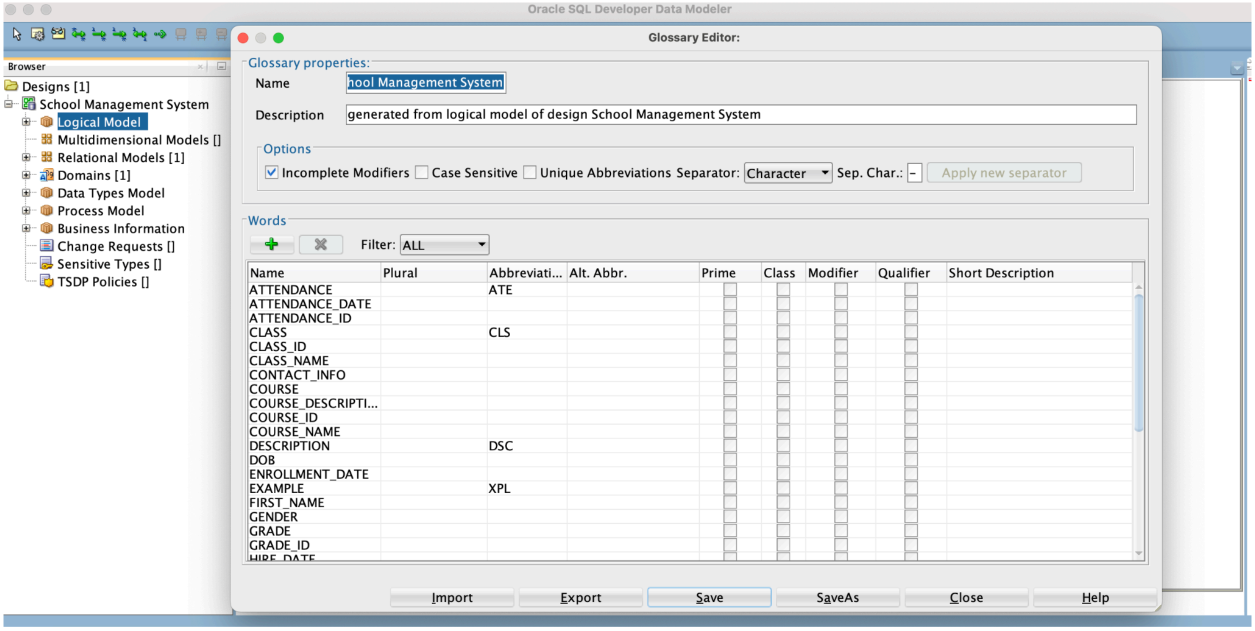
5.1 Glossary Creation



Glossary properties:

Name: hool Management System

Description: generated from logical model of design School Management System

Options

☑ Incomplete Modifiers □ Case Sensitive □ Unique Abbreviations Separator: Character Sep. Char.: - Apply new separator

Words

Filter: ALL

Name	Plural	Abbreviati...	Alt. Abbr.	Prime	Class	Modifier	Qualifier	Short Description
ATTENDANCE		ATE		□	□	□	□	
ATTENDANCE_DATE				□	□	□	□	
ATTENDANCE_ID				□	□	□	□	
CLASS		CLS		□	□	□	□	
CLASS_ID				□	□	□	□	
CLASS_NAME				□	□	□	□	
CONTACT_INFO				□	□	□	□	
COURSE				□	□	□	□	
COURSE_DESCRIPTI...				□	□	□	□	
COURSE_ID				□	□	□	□	
COURSE_NAME				□	□	□	□	
DESCRIPTION		DSC		□	□	□	□	
DOB				□	□	□	□	
ENROLLMENT_DATE				□	□	□	□	
EXAMPLE		XPL		□	□	□	□	
FIRST_NAME				□	□	□	□	
GENDER				□	□	□	□	
GRADE				□	□	□	□	
GRADE_ID				□	□	□	□	
HIRE_DATE				□	□	□	□	

5.2 Compare the Logical Model and the engineered Relational Model to verify:

a. The Unique Identifiers that have been mapped as Primary Keys

- Student_PK (Student ID)
- Class_PK (Class ID)
- Teacher_PK (Teacher ID)
- Subtype_Example_PK (Student ID)
- Subject_PK (Subject ID)
- Grade_PK (Grade ID)
- Course_PK (Course ID)
- Attendance_PK (Attendance ID)
- Relation_1_PK (Student_Student_ID, Class_Class_ID)
- Relation_9_PK (Student_Student_ID, Course_Course_ID)

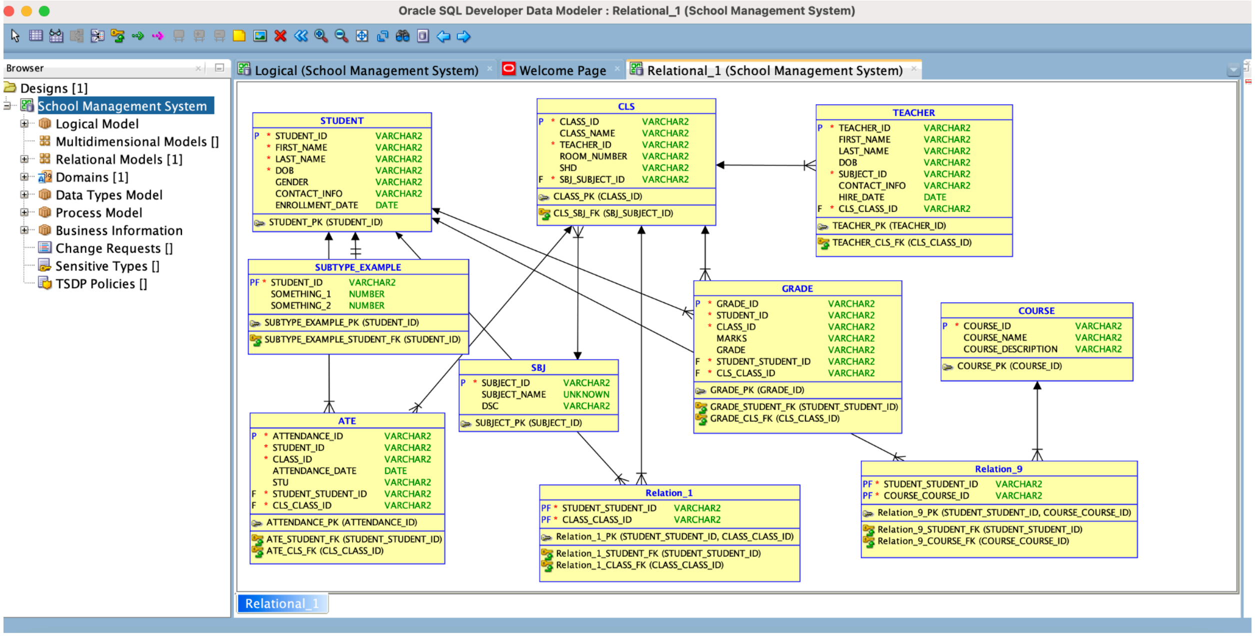
b. The Unique Identifiers that have been mapped as Unique Keys

- Student ID
- Class ID
- Teacher ID
- Subject ID
- Grade ID
- Course ID
- Attendance ID

c. The Relationships that have been mapped as Foreign Keys

- Class & Subject
- Teacher & Class
- Grades & Student, Grades & Class

- Attendance & Student, Attendance & Class



STUDENT

P	*	STUDENT_ID	VARCHAR2
	*	FIRST_NAME	VARCHAR2
	*	LAST_NAME	VARCHAR2
	*	DOB	VARCHAR2
		GENDER	VARCHAR2
		CONTACT_INFO	VARCHAR2
		ENROLLMENT_DATE	DATE

STUDENT_PK (STUDENT_ID)

CLS

P	*	CLASS_ID	VARCHAR2
		CLASS_NAME	VARCHAR2
	*	TEACHER_ID	VARCHAR2
		ROOM_NUMBER	VARCHAR2
		SHD	VARCHAR2
F	*	SBJ_SUBJECT_ID	VARCHAR2

CLASS_PK (CLASS_ID)

CLS_SBJ_FK (SBJ_SUBJECT_ID)

TEACHER

P	*	TEACHER_ID	VARCHAR2
		FIRST_NAME	VARCHAR2
		LAST_NAME	VARCHAR2
		DOB	VARCHAR2
	*	SUBJECT_ID	VARCHAR2
		CONTACT_INFO	VARCHAR2
		HIRE_DATE	DATE
F	*	CLS_CLASS_ID	VARCHAR2

TEACHER_PK (TEACHER_ID)

TEACHER_CLS_FK (CLS_CLASS_ID)

SUBTYPE_EXAMPLE

PF	*	STUDENT_ID	VARCHAR2
		SOMETHING_1	NUMBER
		SOMETHING_2	NUMBER

SUBTYPE_EXAMPLE_PK (STUDENT_ID)

SUBTYPE_EXAMPLE_STUDENT_FK (STUDENT_ID)

GRADE

P	*	GRADE_ID	VARCHAR2
	*	STUDENT_ID	VARCHAR2
	*	CLASS_ID	VARCHAR2
		MARKS	VARCHAR2
		GRADE	VARCHAR2
F	*	STUDENT_STUDENT_ID	VARCHAR2
F	*	CLS_CLASS_ID	VARCHAR2

GRADE_PK (GRADE_ID)

GRADE_STUDENT_FK (STUDENT_STUDENT_ID)
GRADE_CLS_FK (CLS_CLASS_ID)

COURSE

P	*	COURSE_ID	VARCHAR2
		COURSE_NAME	VARCHAR2
		COURSE_DESCRIPTION	VARCHAR2

COURSE_PK (COURSE_ID)

SBJ

P	*	SUBJECT_ID	VARCHAR2
		SUBJECT_NAME	UNKNOWN
		DSC	VARCHAR2

SUBJECT_PK (SUBJECT_ID)

ATE

P	*	ATTENDANCE_ID	VARCHAR2
	*	STUDENT_ID	VARCHAR2
	*	CLASS_ID	VARCHAR2
		ATTENDANCE_DATE	DATE
		STU	VARCHAR2
F	*	STUDENT_STUDENT_ID	VARCHAR2
F	*	CLS_CLASS_ID	VARCHAR2

ATTENDANCE_PK (ATTENDANCE_ID)

ATE_STUDENT_FK (STUDENT_STUDENT_ID)
ATE_CLS_FK (CLS_CLASS_ID)

Relation_1

PF	*	STUDENT_STUDENT_ID	VARCHAR2
PF	*	CLASS_CLASS_ID	VARCHAR2

Relation_1_PK (STUDENT_STUDENT_ID, CLASS_CLASS_ID)

Relation_1_STUDENT_FK (STUDENT_STUDENT_ID)
Relation_1_CLASS_FK (CLASS_CLASS_ID)

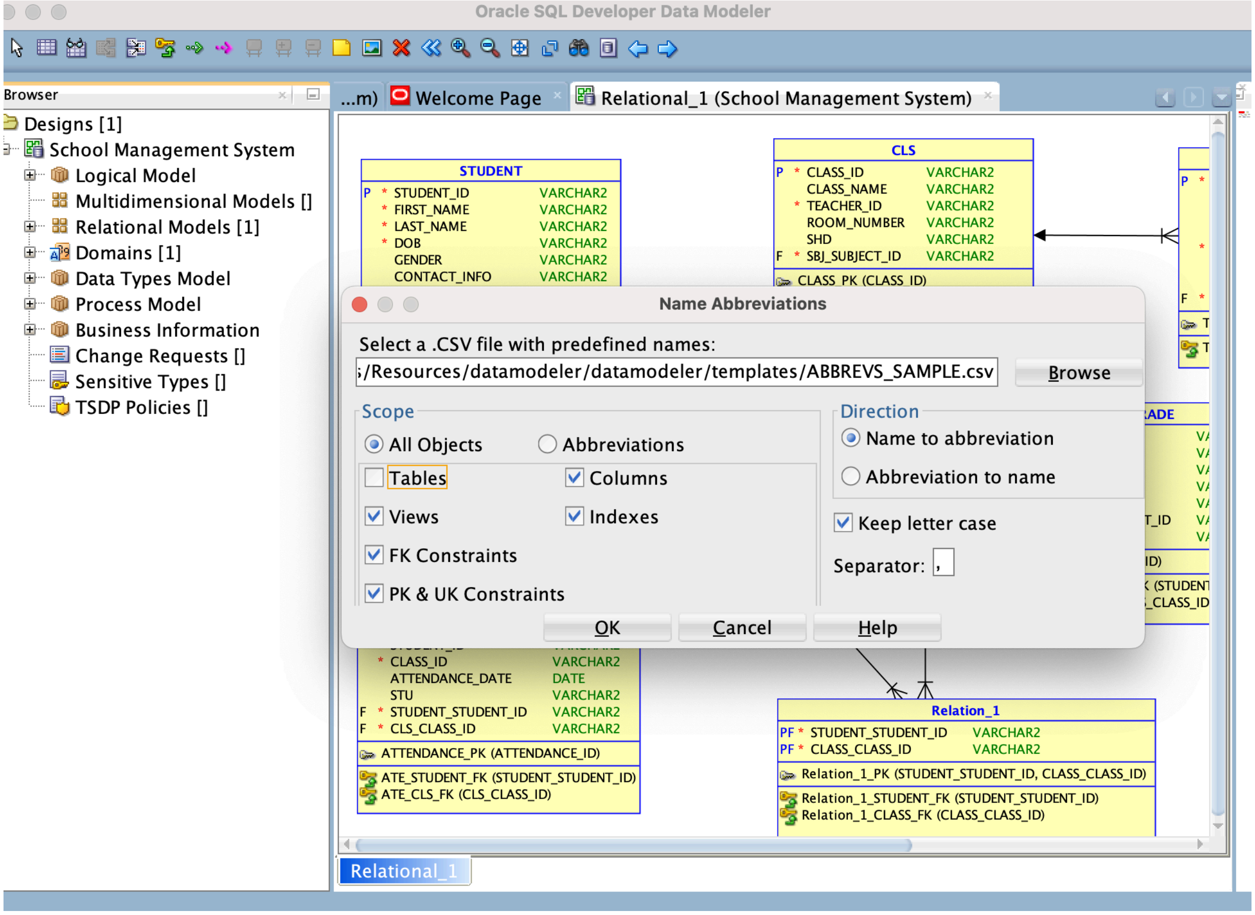
Relation_9

PF	*	STUDENT_STUDENT_ID	VARCHAR2
PF	*	COURSE_COURSE_ID	VARCHAR2

Relation_9_PK (STUDENT_STUDENT_ID, COURSE_COURSE_ID)

Relation_9_STUDENT_FK (STUDENT_STUDENT_ID)
Relation_9_COURSE_FK (COURSE_COURSE_ID)

Uploading CSV. File for Name Abbreviation:



STUDENT
P * STUDENT_ID VARCHAR2
* FIRST_NAME VARCHAR2
* LAST_NAME VARCHAR2
* DOB VARCHAR2
GENDER VARCHAR2
CONTACT_INFO VARCHAR2

CLS
P * CLASS_ID VARCHAR2
CLASS_NAME VARCHAR2
* TEACHER_ID VARCHAR2
ROOM_NUMBER VARCHAR2
SHD VARCHAR2
F * SBJ_SUBJECT_ID VARCHAR2
CLASS_PK (CLASS_ID)

* CLASS_ID VARCHAR2
ATTENDANCE_DATE DATE
STU VARCHAR2
F * STUDENT_STUDENT_ID VARCHAR2
F * CLS_CLASS_ID VARCHAR2
ATTENDANCE_PK (ATTENDANCE_ID)
ATE_STUDENT_FK (STUDENT_STUDENT_ID)
ATE_CLS_FK (CLS_CLASS_ID)

Relation_1
PF * STUDENT_STUDENT_ID VARCHAR2
PF * CLASS_CLASS_ID VARCHAR2
Relation_1_PK (STUDENT_STUDENT_ID, CLASS_CLASS_ID)
Relation_1_STUDENT_FK (STUDENT_STUDENT_ID)
Relation_1_CLASS_FK (CLASS_CLASS_ID)

Name Abbreviations

Select a .CSV file with predefined names:

;/Resources/datamodeler/datamodeler/templates/ABBREVS_SAMPLE.csv Browse

Scope

● All Objects ○ Abbreviations

☐ Tables ☑ Columns

☑ Views ☑ Indexes

☑ FK Constraints

☑ PK & UK Constraints

Direction

● Name to abbreviation

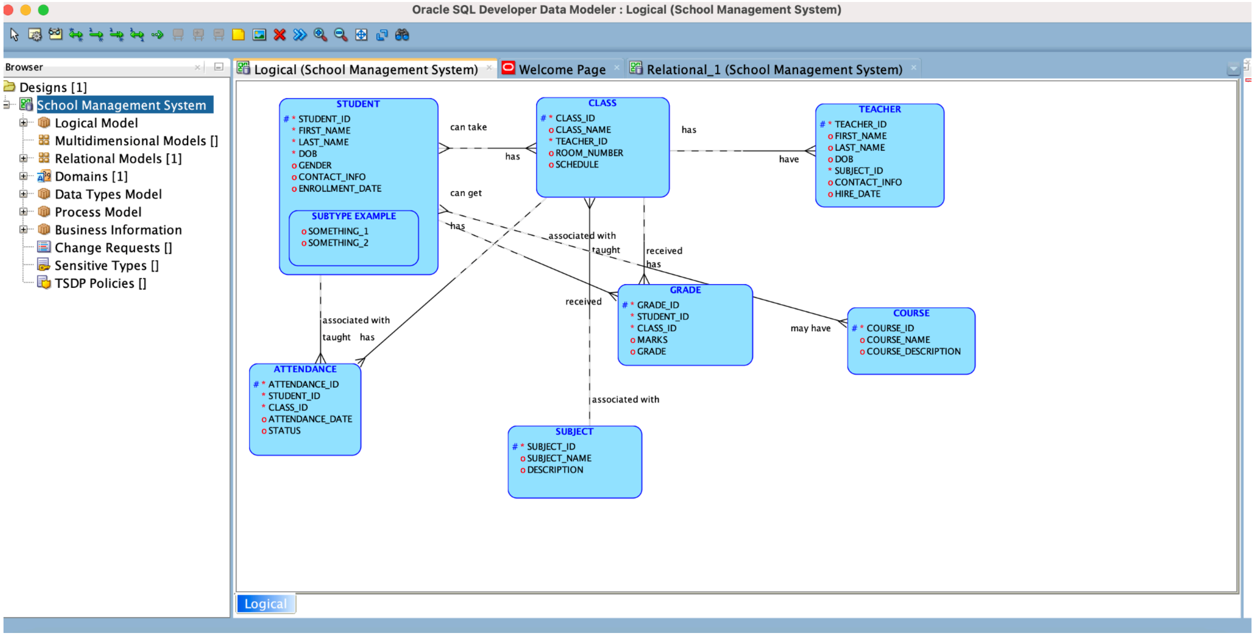
○ Abbreviation to name

☑ Keep letter case

Separator: ,

OK Cancel Help

Subtype Example:



6.3 Defining Data Definition Language (DDL) Practices

Exercise 1: Create the DDL Statements for creating the tables for the Academic Database listed above – include NOT NULL constraints where necessary. (Other constraints will be added later)

Course Name:

```
CREATE TABLE AD_COURSES (
   course_id NUMBER PRIMARY KEY,
   course_name VARCHAR2(100) NOT NULL,
   department_id NUMBER NOT NULL
);
```

Department Name:

```
CREATE TABLE AD_DEPARTMENTS (
   department_id NUMBER PRIMARY KEY,
   department_name VARCHAR2(100) NOT NULL
);
```

Student Email:

```
CREATE TABLE AD_STUDENTS (
    student_id NUMBER PRIMARY KEY,
    student_name VARCHAR2(100) NOT NULL,
    student_email VARCHAR2(100) UNIQUE NOT NULL
);
```

Faculty Email:

```
CREATE TABLE AD_FACULTY (
    faculty_id NUMBER PRIMARY KEY,
    faculty_name VARCHAR2(100) NOT NULL,
    faculty_email VARCHAR2(100) UNIQUE NOT NULL
);
```

Session Name:

```
CREATE TABLE AD_ACADEMIC_SESSIONS (
    session_id NUMBER PRIMARY KEY,
    session_name VARCHAR2(100) UNIQUE NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL
);
```

Altering The Tables:

```
ALTER TABLE AD_COURSES
ADD CONSTRAINT fk_department FOREIGN KEY (department_id)
REFERENCES AD_DEPARTMENTS (department_id);

ALTER TABLE AD_STUDENTS
ADD CONSTRAINT pk_student PRIMARY KEY (student_id);

ALTER TABLE AD_FACULTY
ADD CONSTRAINT pk_faculty PRIMARY KEY (faculty_id);
```

Exercise 2: Alter the AD_FACULTY_LOGIN_DETAILS table to set default value for LOGIN_DATE_TIME column:

```
ALTER TABLE AD_FACULTY_LOGIN_DETAILS
MODIFY (LOGIN_DATE_TIME DEFAULT SYSDATE);
```

Exercise 2: Set the AD_PARENT_INFORMATION table to read-only:

```
ALTER TABLE AD_PARENT_INFORMATION
SET READ ONLY;
```

Exercise 3: Creating Composite Primary, Foreign and Unique Keys

```
CREATE TABLE DEPT (
   dept_id NUMBER(8),
   dept_name VARCHAR2(30),
   loc_id NUMBER(4),
   PRIMARY KEY (dept_id, loc_id)
);

CREATE TABLE SUPPLIERS (
   sup_id NUMBER(15),
   sup_name VARCHAR2(30),
   contact_name NUMBER(4),
   PRIMARY KEY (sup_id, sup_name)
);

CREATE TABLE PRODUCTS (
   product_id NUMBER(10) PRIMARY KEY,
   sup_id NUMBER(15) NOT NULL,
   sup_name VARCHAR2(30) NOT NULL,
   FOREIGN KEY (sup_id, sup_name) REFERENCES SUPPLIERS (sup_id,
sup_name)
);
```

```
CREATE TABLE DEPT_SAMPLE (
   dept_id NUMBER(8),
   dept_name VARCHAR2(30),
   loc_id NUMBER(4),
   UNIQUE (dept_id, dept_name)
);
```

6.4 Defining Data Manipulation Practices

Exercise 1: Inserting rows into tables

```
INSERT INTO AD_ACADEMIC_SESSIONS (ID, NAME)
VALUES (100, 'SPRING SESSION'),
      (200, 'FALL SESSION'),
      (300, 'SUMMER SESSION');
INSERT INTO AD_DEPARTMENTS (ID, NAME, HEAD)
VALUES (10, 'ACCOUNTING', 'MARK SMITH'),
      (20, 'BIOLOGY', 'DAVE GOLD'),
      (30, 'COMPUTER SCIENCE', 'LINDA BROWN'),
      (40, 'LITERATURE', 'ANITA TAYLOR');
ALTER TABLE AD_PARENT_INFORMATION READ WRITE;

INSERT INTO AD_PARENT_INFORMATION (ID, PARENT1_FN, PARENT1_LN,
PARENT2_FN, PARENT2_LN)
VALUES (600, 'NEIL', 'SMITH', 'DORIS', 'SMITH'),
      (610, 'WILLIAM', 'BEN', 'NITA', 'BEN'),
      (620, 'SEAN', 'TAYLOR', 'RHEA', 'TAYLOR'),
      (630, 'DAVE', 'CARMEN', 'CATHY', 'CARMEN'),
      (640, 'JOHN', 'AUDRY', 'JANE', 'AUDRY');

INSERT INTO AD_STUDENTS (ID, FIRST_NAME, LAST_NAME, REG_YEAR, EMAIL,
PARENT_ID)
VALUES (720, 'JACK', 'SMITH', '01-Jan-2012', 'JSMITH@SCHOOL.EDU',
600),
      (730, 'NOAH', 'AUDRY', '01-Jan-2012', 'NAUDRY@SCHOOL.EDU',
640),
      (740, 'RHONDA', 'TAYLOR', '01-Sep-2012', 'RTAYLOR@SCHOOL.EDU',
620),
      (750, 'ROBERT', 'BEN', '01-Mar-2012', 'RBEN@SCHOOL.EDU', 610),
      (760, 'JEANNE', 'BEN', '01-Mar-2012', 'JBEN@SCHOOL.EDU', 610),
      (770, 'MILLS', 'CARMEN', '01-Apr-2013', 'MCARMEN@SCHOOL.EDU',
630);

INSERT INTO AD_COURSES (ID, NAME, SESSION_ID, DEPT_ID, BUILDING,
ROOM, DATE_TIME)
VALUES (195, 'CELL BIOLOGY', 200, 20, 'BUILDING D', 401, 'MWF 9-10'),
      (190, 'PRINCIPLES OF ACCOUNTING', 100, 10, 'BUILDING A', 101,
'MWF 12-1'),
      (191, 'INTRODUCTION TO BUSINESS LAW', 100, 10, 'BUILDING B',
201, 'THUR 2-4'),
      (192, 'COST ACCOUNTING', 100, 10, 'BUILDING C', 301, 'TUES
5-7'),
      (193, 'STRATEGIC TAX PLANNING FOR BUSINESS', 100, 10, 'TAX123',
NULL, NULL, NULL),
      (194, 'GENERAL BIOLOGY', 200, 20, 'BIO123', NULL, NULL, NULL);


INSERT INTO AD_FACULTY (ID, FIRST_NAME, LAST_NAME, EMAIL, SALARY,
INSURANCE, HOURLY_RATE, DEPT_ID)
VALUES (800, 'JILL', 'MILLER', 'JMILL@SCHOOL.EDU', 10000, 'HEALTH',
NULL, 20),
      (810, 'JAMES', 'BORG', 'JBORG@SCHOOL.EDU', 30000,
'HEALTH,DENTAL', NULL, 10),
      (820, 'LYNN', 'BROWN', 'LBROWN@SCHOOL.EDU', NULL, NULL, 50,
30),
      (830, 'ARTHUR', 'SMITH', 'ASMITH@SCHOOL.EDU', NULL, NULL, 40,
10),
      (840, 'SALLY', 'JONES', 'SJONES@SCHOOL.EDU', 50000,
'HEALTH,DENTAL,VISION', NULL, 40);
```

Exercise 2: Updating rows in the tables

```
ALTER TABLE AD_FACULTY_LOGIN_DETAILS
ADD (DETAILS VARCHAR2(50));

UPDATE AD_FACULTY_LOGIN_DETAILS
SET DETAILS = 'First Login'
WHERE FACULTY_ID = 800 AND LOGIN_DATE_TIME = TO_TIMESTAMP('01-JUN-17
05.10.39 PM', 'DD-MON-RR HH.MI.SS PM');

UPDATE AD_FACULTY_LOGIN_DETAILS
SET DETAILS = 'Second Login'
WHERE FACULTY_ID = 800 AND LOGIN_DATE_TIME = TO_TIMESTAMP('01-JUN-17
05.13.15 PM', 'DD-MON-RR HH.MI.SS PM');
```

6.5 Defining Transaction Control Practices

Exercise 1: Would the new email field still be there?

- Yes the email will still be there because ALTER table is a DDL operation which are automatically within Oracle so they cannot be rolled back.

Exercise 2:

If an INSERT is done to add rows into the test table and a Savepoint is then created called INSERT_DONE.

Then an UPDATE to a row in the test table is done and a Savepoint is created called UPDATE_DONE.

Then a DELETE is executed to delete a row in the test table and a Savepoint is created called DELETE_DONE. At this point what records would be in the table?

- What would stay is everything but Student_id = 1.

Then a ROLLBACK to Savepoint UPDATE_DONE is issued. What changes would you notice with respect to the transactions and the records remaining in the table?

- If a ROLLBACK is performed to Savepoint then what was once deleted (Student_id = 1) would be restored and updated to First_Name = ‘Johnny’

6.6 Retrieving Data Practices

Exercise 1: Retrieving columns from tables

1. Write a simple query to view the data inserted in the tables created for the academic database

```
SELECT * FROM AD_FACULTY;
SELECT * FROM AD_COURSES;
SELECT * FROM AD_DEPARTMENTS;
SELECT * FROM AD_ACADEMIC_SESSIONS;
```

2. Write a query to retrieve the exam grade obtained by each student for every exam attempted.

```
SELECT STUDENT_ID, COURSE_ID, EXAM_ID, EXAM_GRADE
FROM AD_EXAM_RESULTS;
```

3. Write a query to check if a student is eligible to take exams based on the number of days he/she attended classes.

```
SELECT STUDENT_ID, SESSION_ID, NUM_WORK_DAYS, NUM_DAYS_OFF,
EXAM_ELIGIBILITY
FROM AD_STUDENT_ATTENDANCE
WHERE EXAM_ELIGIBILITY = 'Y';
```

4. Display the LOGIN_DATE_TIME for each faculty member.

```
SELECT FACULTY_ID, LOGIN_DATE_TIME
FROM AD_FACULTY_LOGIN_DETAILS;
```

5. Display the name of the Head of the Department for each of the Departments.

```
SELECT NAME AS DEPARTMENT_NAME, HEAD AS DEPARTMENT_HEAD
FROM AD_DEPARTMENTS;
```

6. Retrieve the student ID and first name for each student concatenated with literal text to look like this:
720: FIRST NAME IS JACK

```
SELECT STUDENT_ID || ': FIRST NAME IS ' || FIRST_NAME AS
STUDENT_INFO
```

```
FROM AD_STUDENTS;
```

7. Display all the distinct exam types from the AD_EXAMS table.

```
SELECT DISTINCT EXAM_TYPE
FROM AD_EXAMS;
```

6.7 Restricting Data Using WHERE Statement Practices

Exercise 1: Restricting Data Using SELECT

1. Display the course details for the Spring Session.

```
SELECT *
FROM AD_COURSES
WHERE SESSION_ID = 100;
```

2. Display the details of the students who have scored more than 95.

```
SELECT *
FROM AD_EXAM_RESULTS
WHERE EXAM_GRADE > 95;
```

3. Display the details of the students who have scored between 65 and 70.

```
SELECT *
FROM AD_EXAM_RESULTS
WHERE EXAM_GRADE BETWEEN 65 AND 70;
```

4. Display the students who registered after 01-Jun-2012.

```
SELECT *
FROM AD_STUDENTS
WHERE STUDENT_REG_YEAR > TO_DATE('01-Jun-2012', 'DD-MON-YYYY');
```

5. Display the course details for departments 10 and 30.

```
SELECT *
FROM AD_COURSES
WHERE DEPT_ID IN (10, 30);
```

6. Display the details of students whose first name begins with the letter "J".

```
SELECT *
FROM AD_STUDENTS
WHERE FIRST_NAME LIKE 'J%';
```

7. Display the details of students who have opted for courses 190 or 193.

```
SELECT *
FROM AD_STUDENT_COURSE_DETAILS
WHERE COURSE_ID IN (190, 193);
```

8. Display the course details offered by department 30 for the Fall Session (Session ID 200)

```
SELECT *
FROM AD_COURSES
WHERE DEPT_ID = 30 AND SESSION_ID = 200;
```

9. Display the course details of courses not being offered in the summer and fall session (Session ID 200 and 300).

```
SELECT *
FROM AD_COURSES
WHERE SESSION_ID NOT IN (200, 300);
```

10. Display the course details for department 20.

```
SELECT *
FROM AD_COURSES
WHERE DEPT_ID = 20;
```

6.8 Sorting Data Using ORDER BY Practices

Exercise 1: Sorting Data Using ORDER BY

1. Display all fields for each of the records in ascending order for the following tables:

a. AD_STUDENTS ordered by REG_YEAR

```
SELECT *
FROM AD_STUDENTS
ORDER BY STUDENT_REG_YEAR ASC;
```

b. AD_EXAM_RESULTS ordered by STUDENT_ID and COURSE_ID

```
SELECT *
FROM AD_EXAM_RESULTS
ORDER BY STUDENT_ID ASC, COURSE_ID ASC;
```

c. AD_STUDENT_ATTENDANCE ordered by STUDENT_ID

```
SELECT *
FROM AD_STUDENT_ATTENDANCE
ORDER BY STUDENT_ID ASC;
```

d. AD_DEPARTMENTS ordered by the department ID

```
SELECT *
FROM AD_DEPARTMENTS
ORDER BY DEPT_ID ASC;
```

2. Display the percentage of days students have taken days off and sort the records based on the percentage calculated.

```
SELECT STUDENT_ID, SESSION_ID,
      (NUM_DAYS_OFF / NUM_WORK_DAYS) * 100 AS PERCENT_DAYS_OFF
FROM AD_STUDENT_ATTENDANCE
ORDER BY PERCENT_DAYS_OFF DESC;
```

3. Display the top 5 students based on exam grade results.

```
SELECT STUDENT_ID, COURSE_ID, EXAM_GRADE
FROM AD_EXAM_RESULTS
ORDER BY EXAM_GRADE DESC
FETCH FIRST 5 ROWS ONLY;
```

4.Display the parent details ordered by the parent ID.

```sql
SELECT *
FROM AD_PARENT_INFORMATION
ORDER BY ID ASC;
```

6-9 : Joining Tables Using JOIN Practices

Exercise 1: Using JOINS in SQL Queries

1. Display the different courses offered by the departments in the school.

   ```sql
   SELECT C.ID AS COURSE_ID, C.NAME AS COURSE_NAME, D.NAME AS
   DEPARTMENT_NAME
   FROM AD_COURSES C
   JOIN AD_DEPARTMENTS D ON C.DEPT_ID = D.ID;
   ```

2. Display the courses offered in the Fall session.

   ```sql
   SELECT *
   FROM AD_COURSES
   WHERE SESSION_ID = 200;
   ```

3. Display the course details, the department that offers the courses and students who have enrolled for those courses.

   ```sql
   SELECT C.ID AS COURSE_ID, C.NAME AS COURSE_NAME, D.NAME AS
   DEPARTMENT_NAME, S.STUDENT_ID, S.GRADE
   FROM AD_COURSES C
   JOIN AD_DEPARTMENTS D ON C.DEPT_ID = D.ID
   JOIN AD_STUDENT_COURSE_DETAILS S ON C.ID = S.COURSE_ID;
   ```

4. Display the course details, the department that offers the courses and students who have enrolled for those courses for department 20.

   ```sql
   SELECT C.ID AS COURSE_ID, C.NAME AS COURSE_NAME, D.NAME AS
   DEPARTMENT_NAME, S.STUDENT_ID, S.GRADE
   FROM AD_COURSES C
   JOIN AD_DEPARTMENTS D ON C.DEPT_ID = D.ID
   JOIN AD_STUDENT_COURSE_DETAILS S ON C.ID = S.COURSE_ID
   WHERE C.DEPT_ID = 20;
   ```

5. Write a query to display the details of the exam grades obtained by students who have opted for the course with COURSE_ID in the range of 190 to 192.

   ```
   SELECT R.STUDENT_ID, R.COURSE_ID, R.EXAM_GRADE
   FROM AD_EXAM_RESULTS R
   JOIN AD_COURSES C ON R.COURSE_ID = C.ID
   WHERE C.ID BETWEEN 190 AND 192;
   ```

6. Retrieve the rows from the AD_EXAM_RESULTS table even if there are no matching records in the AD_COURSES table.

   ```
   SELECT R.STUDENT_ID, R.COURSE_ID, R.EXAM_GRADE, C.NAME AS
   COURSE_NAME
   FROM AD_EXAM_RESULTS R
   LEFT JOIN AD_COURSES C ON R.COURSE_ID = C.ID;
   ```

7. What output would be generated when the given statement is executed?

```
SELECT * FROM AD_EXAMS
CROSS JOIN AD_EXAM_TYPES;
```

- The output generated would be all possible combinations of rows from the AD_EXAMS table and the AD_EXAM_TYPES table.