COM106: Introduction to Databases

Introduction to SQL

Structured Query Language (SQL) – An Introduction

What is SQL?

SQL is a **Programming Language**, designed to manage data in a relational dB

Uses simple **declarative** statements

It is a defacto **Industry Standard**.

There is an ANSI standard - although there are many different versions and dialects.

SQL Server 2017 uses a version know as Transact-SQL (or T-SQL)

T-SQL commands may vary slightly from ANSI-SQL or SQL used in other systems (e.g. MySQL)

SQL consists of 3 main components:

DDL - Data Definition Language

(used in the creation/destruction of relational tables, indexes, views etc.)

e.g. **CREATE TABLE** EMP
(empnum smallint not null,
ename char(15),
salary smallmoney,
deptnum smallint**);**

As used in the Lab.

Indentation and layout are not required for execution
*They are **required for clarity**.*

SQL – An Introduction

DML - Data Manipulation Language

(used for retrieval, appending, deleting, updating tables, etc)

e.g.
```
INSERT INTO EMP
VALUES (1, 'Smith', 10000, 12);
```

```
DELETE FROM EMP
WHERE ename = 'Jones';
```

```
UPDATE EMP
SET salary = 20000
WHERE ename = 'Smith' ;
```

```
SELECT ename
FROM EMP
WHERE salary > 20000 ;
```

DCL - Data Control Language

(used for creating table permissions, valid users, etc)

e.g.
```
GRANT ALL [PRIVILEGES ] ON EMP TO John
[WITH GRANT OPTION];
```
(assuming John is a registered user)

Some SQL Reserved Words *(See Link in Lab Resources for Complete List)*

SELECT	HAVING	UPDATE	IN
FROM	GROUP BY	DELETE	ASC
WHERE	DISTINCT	INSERT	SET
AND	DESC	INTO	AS
OR	ORDER BY	VALUES	NOT

There are also a set of comparison operators:

- `=` Equal to
- `<>` Not Equal to
- `>` Greater than
- `<` Less than
- `>=` Greater than or Equal to
- `<=` Less than or Equal to

SQL – An Introduction

Some Common Data Types

Character Strings:

char(n) - data entries are consistent in size e.g. char (12)

varchar(n) - data entries vary considerably in size e.g. varchar (30)

Exact Numerics:

tinyint (0 to 255)

smallint (-32,768 to 32,767),

int (-2,147,483,648 to 2,147,483,648)

bigint (-2^63 to 2^63-1)

decimal [(p[,s])] – precision and scale e.g. decimal (7,2)

Dates and Times e.g. date = '1912-10-25';

(See the T-SQL Language Reference for the Complete List - link in the Lab Resources Folder)

SQL – An Introduction

Building a Database – Creating the Tables

Consider the following relational schema:

With sample data as shown

EMPLOYEE (enum , ename, salary, floor)
PROJECT (pnum , pname, leader)
WORKS_ON (enum* , pnum* , role)

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

PROJECT

pnum	pname	leader
121	IT	Gates
135	Design	Sinclair
147	Analysis	Einstein
216	Publicity	Saatchi
227	Theatre	Dench
251	Sport	Shearer

Notice the PKs and FKs, and how the tables will be linked

Decide on the Data Types

Can an attribute be **NULL**?

Now, write the SQL to create each table

WORKS_ON

enum	pnum	role
852341	121	Manager
852341	135	Designer
852358	147	Consultant
852358	135	Consultant
852407	216	Assistant
852455	121	Assistant
852455	227	Manager
852491	135	Designer
852491	216	Manager
852514	121	Assistant
852514	216	Consultant
852514	251	Manager
852530	147	Manager

SQL – An Introduction

1. Create the EMPLOYEE Table

EMPLOYEE (enum , ename, salary, floor)

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

SQL Style Guide:

Table names and SQL Reserved words – ALL CAPS, words separated by underscore (_)

Attribute names – meaningful, all lowercase, words separated by underscore (_)

Layout – generally, one SQL clause per line

Don't waste storage space – *choose data types that are just large enough.*

```
CREATE TABLE EMPLOYEE (
    enum  int  not null,
    ename char (15),
    salary decimal (7,2),
    floor tinyint,
    CONSTRAINT pk_emp PRIMARY KEY (enum)
);
```

Entity Integrity - no *attributes participating in the primary key of a relation can accept null values.*

Primary Key Constraint – *can be added when the table is created, or added later using an ALTER TABLE statement.*

In this example, pk_emp is the **name** given to the Primary Key Constraint

SQL – An Introduction

2. Create the WORK_ON Table

WORKS_ON (enum* , pnum* , role)

```
CREATE TABLE WORKS_ON (
   enum  int  not null,
   pnum smallint not null,
   role char(15),
   CONSTRAINT  pk_wo PRIMARY KEY (enum,
   pnum)
);
```

We will use an ALTER TABLE statement to make enum and pnum **FKs**

Data Types must be the same as PKs in corresponding tables

WORKS_ON

enum	pnum	role
852341	121	Manager
852341	135	Designer
852358	147	Consultant
852358	135	Consultant
852407	216	Assistant
852455	121	Assistant
852455	227	Manager
852491	135	Designer
852491	216	Manager
852514	121	Assistant
852514	216	Consultant
852514	251	Manager
852530	147	Manager

3. Create the PROJECT Table

PROJECT (pnum , pname, leader)

PROJECT

pnum	pname	leader
121	IT	Gates
135	Design	Sinclair
147	Analysis	Einstein
216	Publicity	Saatchi
227	Theatre	Dench
251	Sport	Shearer

```
CREATE TABLE PROJECT (
    pnum  smallint  not null,
    pname  char(15),
    leader  char(15),
    CONSTRAINT  pk_proj PRIMARY KEY (pnum)
);
```

SQL – An Introduction

Integrity Constraints can be used to apply **business rules** to the tables. They can be added in the **CREATE TABLE** statement or later using **ALTER TABLE**

We have already seen **NOT NULL** and **PRIMARY KEY**

Constraints can be named – as earlier with **PRIMARY KEY** in the **CREATE TABLE** statements – or applied directly to a column/attribute – as with **not null**.

Other useful constraints are :

DEFAULT – Provides a default value for a column when none is specified.
e.g., in the WORKS_ON table role char(15) **DEFAULT** 'To be set',

UNIQUE – Ensures that all values in a column are different.
e.g., in the PROJECT table leader char(15) **UNIQUE**,

CHECK – Ensures that all the values in a column satisfies certain conditions.
e.g., in the EMPLOYEE table floor tinyint **CHECK** (floor <= 7),

We'll look at the **FOREIGN KEY** constraint now and **INDEX** (used to create and retrieve data from the database very quickly) later in the module.

SQL – An Introduction

Building a Database – Create the Links (FKs) and establish Referential Integrity

Relational Schema: **EMPLOYEE** (enum , ename, salary, floor)
PROJECT (pnum , pname, leader)
WORKS_ON (enum*, pnum*, role)

Remember: If a relational table includes a foreign key (**FK**) matching the primary key (**PK**) of another relational table, then every value of the FK **must**:

- either be equal to a value of the PK in some tuple (row)
- or be wholly **NULL**

Foreign Keys can be created, and **Referential Integrity** established, using an **ALTER TABLE** statement

Requires referencing **from** the **Foreign Key towards** a corresponding **Primary Key** in another table

To put referential integrity between **enum** from **WORKS_ON** (**FK**) and **enum** from **EMPLOYEE** (**PK**)

Attributes don't have to have the same name, but **MUST** have the same Data Type

```
ALTER TABLE WORKS_ON
ADD CONSTRAINT fk_wo1 FOREIGN KEY (enum) REFERENCES EMPLOYEE (enum);
```

To put referential integrity between **pnum** from **WORKS_ON** (**FK**) & **pnum** from **PROJECT** (**PK**)

```
ALTER TABLE WORKS_ON
ADD CONSTRAINT fk_wo2 FOREIGN KEY (pnum) REFERENCES PROJECT (pnum);
```

SQL – An Introduction

Other useful ALTER TABLE Commands

In general, may not be possible. Depends on data types and if data has been entered already.

Changing a datatype/size (to whole numbers only)

```
ALTER TABLE EMPLOYEE
ALTER COLUMN salary int;
```

Adding a new attribute (budget to **PROJECT**)

```
ALTER TABLE PROJECT
ADD budget decimal (9,2);
```

Delete budget attribute from **PROJECT**

```
ALTER TABLE PROJECT
DROP COLUMN budget;
```

Adding a PK (if not done in create table)

```
ALTER TABLE EMPLOYEE
ADD CONSTRIANT pk_emp PRIMARY KEY (enum);
```

Drop the PK from **PROJECT** (A two stage process:

1. Drop referential Integrity on PROJECT PK – refer by **name**
2. Drop the PK from PROJECT – refer by **name**)

```
ALTER TABLE WORKS_ON          (drop referential integrity)
DROP CONSTRAINT fk_wo2;

ALTER TABLE PROJECT           (drop PK)
DROP CONSTRAINT pk_proj;
```

SQL – An Introduction

Building a Database – Adding Records

General Syntax

INSERT INTO TABLENAME (attribute list)
VALUES (record values)

e.g. Add new employee record 852559 , 'Brown', 18000, 4

INSERT INTO EMPLOYEE (enum, ename, salary, floor)
VALUES (852559, 'Brown', 18000, 4) ;

Which can be shortened where a whole record is added to:

INSERT INTO EMPLOYEE
VALUES (852559, 'Brown', 18000, 4);

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

Building a Database – Deleting Records

General Syntax

DELETE FROM TABLENAME
WHERE selection clause;

e.g. Delete employees earning above 18000 on the 2nd floor

DELETE FROM EMPLOYEE
WHERE salary > 18000
AND floor = 2 ;

SQL – An Introduction

Building a Database – Editing Records

General Syntax

```
UPDATE TABLENAME
SET attribute = new value
WHERE selection clause;
```

e.g. Change the project leader of the Theatre project to Branagh

```
UPDATE PROJECT
SET leader = 'Branagh'
WHERE pname = ‘theatre';
```

NOTE – **Referential Integrity** must be considered when **adding**, **deleting** or **updating** records

Using the relational schema and sample data on slide 4, consider each of the examples above. Will the queries execute correctly?

1. *Add new employee record 852559 , 'Brown', 18000, 4*
2. *Delete employees earning above 18000 on the 2nd floor*
3. *Change the project leader of the Theatre project to Branagh*

```
INSERT INTO EMPLOYEE
VALUES (852559, 'Brown', 18000, 4);
```

```
DELETE FROM EMPLOYEE
WHERE salary > 18000
AND floor = 2 ;
```

```
UPDATE PROJECT
SET leader = 'Branagh‘
WHERE pname = ‘theatre';
```

SQL – An Introduction

Consider the following relational schema:

EMPLOYEE (enum , ename, salary, floor)
PROJECT (pnum , pname, leader)
WORKS_ON (enum* , pnum* , role)

With sample data as shown

Will the following queries execute correctly?

INSERT INTO EMPLOYEE
VALUES (852559, 'Brown', 18000, 4); ✓

DELETE FROM EMPLOYEE
WHERE salary > 18000
AND floor = 2 ; ✗

UPDATE PROJECT
SET leader = 'Branagh'
WHERE pname = 'theatre'; ✓

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Jones	19000	3
852407	Brown	16000	3
852455	White	25100	2
852491	Adams	30500	1
852514	Doyle	11650	2
852530	Evans	26980	4

PROJECT

pnum	pname	leader
121	IT	Gates
135	Design	Sinclair
147	Analysis	Einstein
216	Publicity	Saatchi
227	Theatre	Dench
251	Sport	Shearer

WORKS_ON

enum	pnum	role
852341	121	Manager
852341	135	Designer
852358	147	Consultant
852358	135	Consultant
852407	216	Assistant
852455	121	Assistant
852455	227	Manager
852491	135	Designer
852491	216	Manager
852514	121	Assistant
852514	216	Consultant
852514	251	Manager
852530	147	Manager