

COMP67321

# Structured Query Language SQL

Goran Nenadic

School of Computer Science  
University of Manchester

1

---

---

---

---

---

---

---

---

## Aims

- Understand creating, modifying, querying and maintaining a relational DBMS using SQL
- Learn and understand basic concepts of SQL
- Inserting, deleting and modifying data
- Creating alternate views of data

2

---

---

---

---

---

---

---

---

## Plan

- Introduction to SQL
- Data definition (creating, updating tables)
- Data retrieval
- Views
- Integrity constraints in SQL
- Updating stored data

3

---

---

---

---

---

---

---

---

## A brief history of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced by many 'see-quel', though official pronunciation is 'S-Q-L'
- IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2.

4

---

---

---

---

---

---

---

---

continued

## A brief history of SQL

- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.
- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.

5

---

---

---

---

---

---

---

---

continued

## A brief history of SQL

- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released; includes XML-related features
- In 2006, SQL:2006, clarifies XML-data management, XML Query Language
- *SQL is an international standard*

6

---

---

---

---

---

---

---

---

## DBMS languages

- Data definition language (DDL)
  - Permits specification of data types, structures and any data constraints.
- Data manipulation language (DML)
  - General enquiry facility (query language) of the data.

7

---

---

---

---

---

---

---

---

## SQL as data definition language

- Define, create and maintain relations and views
  - e.g. CREATE table or view; ALTER table or view; DROP table or view
- Data control
  - allow the privilege for a user to see, change and use the data
  - e.g. GRANT operation on table to username
  - [will not be covered; for information only ]

8

---

---

---

---

---

---

---

---

## SQL as data manipulation language

- Insert, change and delete data
  - e.g. **INSERT** tuples into a table;  
**UPDATE** tuples from tables;  
**DELETE** tuples from tables
- Retrieve data
  - e.g. **SELECT** columns from tables

9

---

---

---

---

---

---

---

---

## General comments about SQL

- SQL statements consist of *reserved words* and *user-defined words*.
  - reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - user-defined words are made up by the user and represent names of various database objects such as relations, columns, views.

10

---

---

---

---

---

---

---

---

## Data definition using SQL

- Used for data definition (tables, views), constraints and schema changes
- CREATE, DROP and ALTER the descriptions of the tables (relations) of a database

11

---

---

---

---

---

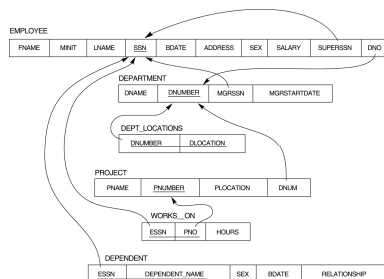
---

---

---

example

## Company schema



12

---

---

---

---

---

---

---

---

## CREATE TABLE

- Specifies a new relation by giving it a name, specifying each of its attributes and their data types
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT
(  DNAME    VARCHAR(10) NOT NULL,
   DNUMBER  INTEGER    NOT NULL,
   MGRSSN   CHAR(9),
   MGRSTARTDATE CHAR(9)
);
```

13

---

---

---

---

---

---

---

---

## Basic data types

- INTEGER
- FLOAT
- DECIMAL(i,j)
- CHAR(n)
- VARCHAR(n)

14

---

---

---

---

---

---

---

---

## Domain types in SQL

- **char(n)**: fixed length character string, with user-specified length  $n$ .
- **varchar(n)**: variable length character strings, with user-specified maximum length  $n$ .
- **int**: Integer (a finite subset of the integers that is machine-dependent).
- **smallint**: small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d)**: fixed point number, with user-specified precision of  $p$  digits, with  $n$  digits to the right of decimal point.

15

---

---

---

---

---

---

---

---

### Domain types in SQL

- **real, double precision:** floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n):** floating point number, with user-specified precision of at least *n* digits.
- **Null** values are allowed in all the domain types. But declaring an attribute to be **not null** prohibits null values for that attribute.

---

---

---

---

---

---

---

---

### Domain types in SQL

- **date:** dates, containing a (4 digit) year, month and date  
 e.g. **date** '2001-7-27'
- **time:** time of day, in hours, minutes and seconds.  
 e.g. **time** '09:00:30'     **time** '09:00:30.75'
- **timestamp:** date plus time of day  
 e.g. **timestamp** '2001-7-27 09:00:30.75'

---

---

---

---

---

---

---

---

### CREATE TABLE – keys

- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases
- Referential integrity via FOREIGN KEY

```
CREATE TABLE DEPT
( DNAME    VARCHAR(10)    NOT NULL,
  DNUMBER  INTEGER        NOT NULL,
  MGRSSN   CHAR(9),
  MGRSTARTDATE CHAR(9),
  PRIMARY KEY (DNUMBER),
  UNIQUE (DNAME),
  FOREIGN KEY (MGRSSN) REFERENCES EMP );
```

---

---

---

---

---

---

---

---

## DROP TABLE

- Used to remove a relation (table) *and its definition*
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

```
DROP TABLE DEPENDENT;
```

19

---

---

---

---

---

---

---

---

## ALTER TABLE

- Used to add an attribute to one of the base relations
  - The new attribute will have NULLs in all the existing tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Example:

```
ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);
```

  - the database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

20

---

---

---

---

---

---

---

---

21

---

---

---

---

---

---

---

---

## Data Retrieval using SQL

22

---

---

---

---

---

---

---

---

## Retrieval queries in SQL

- *SELECT-FROM-WHERE block*
  - SELECT** <attribute list>
  - FROM** <table list>
  - WHERE** <condition>
- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional expression that identifies the tuples to be retrieved by the query

23

---

---

---

---

---

---

---

---

## Retrieving data – SQL syntax

```

SELECT [ALL | DISTINCT] { * | {table.* | expr [alias]}
    [, {table.* | expr [alias]} ] ...}
FROM table [alias] [, table [alias]] ...
[WHERE condition]
[CONNECT BY condition [START WITH condition] ]
[GROUP BY expr [, expr] ....] [HAVING condition]
[ {UNION | INTERSECT | MINUS} SELECT .....]
[ORDER BY {expr | posn} [ASC | DESC] [, {expr |
    posn} [ASC | DESC] ..]
[FOR UPDATE OF column[, column] .... [NOWAIT] ]
    
```

24

---

---

---

---

---

---

---

---

## Explaining main retrieval parts

**SELECT** columns are to appear in output  
**FROM** table(s) to be used  
**WHERE** filters rows  
**GROUP BY** groups of rows with same column value  
**HAVING** groups subject to some condition  
**ORDER BY** the order of the output

- Order of the clauses cannot be changed
- Only SELECT and FROM are mandatory

25

---

---

---

---

---

---

---

---

example

## Example table DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

26

---

---

---

---

---

---

---

---

example

## Retrieving columns

```
SELECT *  
FROM DEPT
```



DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
SELECT DEPTNO, LOC  
FROM DEPT
```



DEPTNO	LOC
10	NEW YORK
20	DALLAS
30	CHICAGO
40	BOSTON

27

---

---

---

---

---

---

---

---

example

### Example table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7802	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	600	30
7600	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	08-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-NOV-81	3000		20
7839	KING	PRESIDENT		17-NOV-81	6000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-SEP-81	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7666	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

---

---

---

---

---

---

---

---

---

---

---

---

example

### Retrieving specific tuples

List all employee details that work in department with department no = 30

```
SELECT *
FROM EMP
WHERE DEPTNO = 30
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7900	JAMES	CLERK	7698	03-DEC-81	950		30

29

---

---

---

---

---

---

---

---

---

---

---

---

example

### Multiple retrieval conditions

List all employees that are managers and have a salary of more than 2,800

```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE JOB='MANAGER'
AND SAL > 2800
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850

30

---

---

---

---

---

---

---

---

---

---

---

---

example

### Alternate retrieval conditions

List all employees that are managers or earn more than 2,800

```
SELECT ENAME, JOB, SAL
FROM EMP
WHERE JOB='MANAGER'
OR SAL > 2800
```

ENAME	JOB	SAL
JONES	MANAGER	2975
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
SCOTT	ANALYST	3000
KING	PRESIDENT	5000
FORD	ANALYST	3000

31

---

---

---

---

---

---

---

---

---

---

example

### Negative retrieval conditions

List all managers that do not work in department 30

```
SELECT ENAME, JOB, DEPTNO
FROM EMP
WHERE JOB='MANAGER'
AND DEPTNO != 30
```

ENAME	JOB	DEPTNO
JONES	MANAGER	20
CLARK	MANAGER	10

32

---

---

---

---

---

---

---

---

---

---

example

### Retrieval on a range

List all employees whose salary is between 1,200 and 1,400

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL BETWEEN 1200 AND 1400
```

ENAME	SAL
WARD	1250
MARTIN	1250
MILLER	1300

33

---

---

---

---

---

---

---

---

---

---

example

## Retrieval from a list

List all departments whose number is either 10 or 30

```
SELECT *
FROM EMP
WHERE DEPTNO IN (10,30)
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
30	SALES	CHICAGO

34

---

---

---

---

---

---

---

---

example

## Ordering resulting tuples

List all employees in department 30 and display the result in salary order

```
SELECT SAL, JOB, ENAME
FROM EMP
WHERE DEPTNO = 30
ORDER BY SAL
```

SAL	JOB	ENAME
950	CLERK	JAMES
1250	SALESMAN	WARD
1250	SALESMAN	MARTIN
1500	SALESMAN	TURNER
1600	SALESMAN	ALLEN
2850	MANAGER	BLAKE

order:  
ASC, DSC

35

---

---

---

---

---

---

---

---

example

## Retrieving distinct values

List all the different kinds of job

```
SELECT DISTINCT JOB
FROM EMP
```

JOB
CLERK
SALESMAN
MANAGER
ANALYST
PRESIDENT

Note a difference between SQL and relational model

36

---

---

---

---

---

---

---

---

## Querying multiple relations

Find in which city employee Allen works

```
SELECT ENAME, LOC
FROM EMP, DEPT
WHERE ENAME = 'Allen'
AND EMP.DEPTNO = DEPT.DEPTNO
```

ENAME	LOC
ALLEN	CHICAGO

the join condition

37

---

---

---

---

---

---

---

---

## Joining entire relations

List department names and additional details about employees

```
SELECT DNAME, ENAME, JOB, SAL
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO
ORDER BY DNAME, SAL DESC
```

DNAME	ENAME	JOB	SAL
ACCOUNTING	KING	PRESIDENT	5000
ACCOUNTING	CLARK	MANAGER	2450
ACCOUNTING	MILLER	CLERK	1300
RESEARCH	SCOTT	ANALYST	3000
RESEARCH	FORD	ANALYST	3000
RESEARCH	JONES	MANAGER	2975
RESEARCH	ADAMS	CLERK	1100
RESEARCH	SMITH	CLERK	800
SALES	BLAKE	MANAGER	2850
SALES	ALLEN	SALESMAN	1600
SALES	TURNER	SALESMAN	1500
SALES	WARD	SALESMAN	1250
SALES	MARTIN	SALESMAN	1250
SALES	JAMES	CLERK	950

38

---

---

---

---

---

---

---

---

## Outer join

List all department names and employees' job and salary for all departments (even those without employees)

```
SELECT DNAME, ENAME, JOB, SAL
FROM EMP, DEPT
WHERE EMP.DEPTNO (+) =
DEPT.DEPTNO
```

DNAME	ENAME	JOB	SAL
ACCOUNTING	KING	PRESIDENT	5000
ACCOUNTING	CLARK	MANAGER	2450
ACCOUNTING	MILLER	CLERK	1300
RESEARCH	SCOTT	ANALYST	3000
RESEARCH	FORD	ANALYST	3000
RESEARCH	JONES	MANAGER	2975
RESEARCH	ADAMS	CLERK	1100
RESEARCH	SMITH	CLERK	800
SALES	BLAKE	MANAGER	2850
SALES	ALLEN	SALESMAN	1600
SALES	TURNER	SALESMAN	1500
SALES	WARD	SALESMAN	1250
SALES	MARTIN	SALESMAN	1250
SALES	JAMES	CLERK	950
OPERATIONS	<null>	<null>	<null>

query will return data even for departments with no employees

39

---

---

---

---

---

---

---

---

## Subqueries (nested queries)

List all employees with the same job as Jones

```
SELECT ENAME, JOB
FROM EMP
WHERE JOB =
(SELECT JOB
FROM EMP
WHERE ENAME = 'Jones')
```

ENAME	JOB
JONES	MANAGER
BLAKE	MANAGER
CLARK	MANAGER

A subquery is evaluated for the parent query

What happens if '=' is replaced with 'IN'?

40

---

---

---

---

---

---

---

---

## Using subqueries

Subqueries may be used in the following situations

- to define the set of rows to be inserted in the target table of an INSERT, CREATE TABLE, or COPY command
- to define one or more values to be assigned to existing rows in an UPDATE statement
- to provide values for comparison in WHERE, HAVING and START WITH clauses in SELECT, UPDATE and DELETE commands

41

---

---

---

---

---

---

---

---

## Correlated subqueries

List all employees located in Chicago with the same job as Allen

```
SELECT ENAME, LOC, SAL, JOB
FROM EMP, DEPT
WHERE LOC = 'CHICAGO'
AND JOB =
(SELECT JOB
FROM EMP
WHERE ENAME = 'Allen')
ORDER BY EMP.DEPTNO
```

ENAME	LOC	SAL	JOB
WARD	CHICAGO	1250	SALESMAN
MARTIN	CHICAGO	1250	SALESMAN
TURNER	CHICAGO	1500	SALESMAN

a correlated subquery is evaluated for every tuple evaluated by the parent query

42

---

---

---

---

---

---

---

---

## Arithmetic expressions in SQL

List the name, salary, commission and the sum of salary plus commission of all sales people

```
SELECT ENAME, SAL, COMM,
       SAL + COMM
FROM   EMP
WHERE  JOB = 'SALESMAN'
```

arithmetic function

ENAME	SAL	COMM	SAL+COMM
ALLEN	1,600.00	300.00	1900
WARD	1,250.00	500.00	1750
MARTIN	1,250.00	1,400.00	2650
TURNER	1,500.00	0.00	1500

43

---

---

---

---

---

---

---

---

## Min/Max/Avg calculation in SQL

Find the maximum salary

```
SELECT DEPTNO, MAX(SAL)
FROM   EMP
```

computing function

MIN  
MAX  
AVG

MAX(SAL)
5000

44

---

---

---

---

---

---

---

---

## GROUP BY

Find the maximum salary in each department

```
SELECT DEPTNO, MAX(SAL)
FROM   EMP
GROUP BY DEPTNO
```

GROUP BY =  
grouping rows  
with the same  
values

DEPTNO	MAX(SAL)
10	5000
20	3000
30	2850

45

---

---

---

---

---

---

---

---

## Count in SQL

Find the total number of employees working in each department and job, and show the total and average salaries in each group

```
SELECT DEPTNO, JOB, SUM(SAL)
       COUNT(*), AVG(SAL)
FROM   EMP
GROUP BY DEPTNO, JOB
```

DEPTNO	JOB	SUM(SAL)	COUNT(*)	AVG(SAL)
20	ANALYST	6000	2	3000
20	CLERK	1900	2	950
30	SALESMAN	5600	4	1400
20	MANAGER	2975	1	2975
30	MANAGER	2850	1	2850
10	MANAGER	2450	1	2450
20	PRESIDENT	5000	1	5000
30	CLERK	950	1	950
10	CLERK	1300	1	1300

---

---

---

---

---

---

---

---

## HAVING clause

Find the no of employees working at each job (at least 2 employees) and show the total and average salaries in each group

```
SELECT DEPTNO, JOB, SUM(SAL)
       COUNT(*), AVG(SAL)
FROM   EMP
GROUP BY DEPTNO, JOB
HAVING COUNT(*) >= 2
```

DNAME	JOB	SUM(SAL)	COUNT(*)	AVG(SAL)
20	ANALYST	6000	2	3000
20	CLERK	1900	2	950
20	SALESMAN	5600	4	1400

47

---

---

---

---

---

---

---

---

## ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by *all* values produced by the subquery.
- With ANY, condition will be true if it is satisfied by *any* values produced by the subquery.
- If the subquery result is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

48

---

---

---

---

---

---

---

---

## ANY and ALL: example

Find staff whose salary is larger than salary of at least one member of staff at department 20.

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE SAL > ANY
      (SELECT SAL
       FROM EMP
       WHERE DEPTNO = 20);
```

49

---

---

---

---

---

---

---

---

## EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by the subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.

50

---

---

---

---

---

---

---

---

continued

## EXISTS and NOT EXISTS

- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- Common for subqueries following (NOT) EXISTS to be of form:  
(SELECT \* ...)

51

---

---

---

---

---

---

---

---

## EXISTS: example

Find all staff who work in a department based in Dallas.

```
SELECT EMPNO, EMPNAME, JOB
FROM EMP e
WHERE EXISTS
  (SELECT *
   FROM DEPT d
   WHERE e.DEPTNO = d.DEPTNO AND LOC = 'Dallas');
```

52

---

---

---

---

---

---

---

---

## Set operations

- Union, intersection, and difference (might be found as EXCEPT) to combine results of two or more queries into a single result table.
  - union of two tables, A and B, is table containing all **rows** in either A or B or both
  - intersection is table containing all rows common to both A and B
  - difference is table containing all rows in A but not in B
- Two tables must be *union compatible*.

53

---

---

---

---

---

---

---

---

## Set operations: example

- Find all locations that have either an 'accounting' or a 'research' department.

```
(SELECT LOC
 FROM DEPT
 WHERE DNAME = 'Accounting') UNION
(SELECT LOC
 FROM DEPT
 WHERE DNAME = 'Research');
```

54

---

---

---

---

---

---

---

---

## Views in SQL

- *View*: a dynamic result of one or more relational operations operating on base relations to produce another relation.
- A view is a virtual relation that provides a window through which one can see data (stored in a base relation)
- Views contain no data of their own but can be operated on as real relations
- Can help simplify data access by isolating users from querying details

55

---

---

---

---

---

---

---

---

continued

## Views in SQL

- Used to
  - enforce data integrity
  - provide data independence
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

56

---

---

---

---

---

---

---

---

## CREATE VIEW

```
CREATE VIEW ViewName
    [ (newColumnName [...]) ]
AS subselect
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Can assign a name to each column in view.
- If list of column names is specified, it must have the same number of items as the number of columns produced by *subselect*.
- If omitted, each column takes name of corresponding column in *subselect*.

57

---

---

---

---

---

---

---

---

## CREATE VIEW

- The column list must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.

[Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.]

---

---

---

---

---

---

---

---

## Views: example

```
CREATE VIEW PERSONNEL AS
SELECT ENAME, JOB, PNAME
FROM EMP, PROJ
WHERE EMP.PROJNO = PROJ.PROJNO
```

query specifies data with which the view operates

- Once a view is created, the users need not know that data is really stored in two relations
- Instead of constructing a complex query one can use the PERSONNEL view to list the names of managers and their projects

```
SELECT ENAME, PNAME
FROM PERSONNEL
WHERE JOB='MANAGER'
```

---

---

---

---

---

---

---

---

## Types of views

- Horizontal views
  - Create view abc as select \* from table1 where <condition>
- Vertical views
  - Create view abc as select attr1, attr2 ... from table1 where <condition>
- Views involving more than one tables
  - Create view abc as select ... from table1, table2 where <condition>.
  - Condition here involves join operations

---

---

---

---

---

---

---

---

## Views and data independence

- Views insulate users from having to know anything about underlying relations by providing only the data with which a user (application program) is working
- Views can be used to keep existing queries (application programs) from becoming obsolete as the structure of the database changes - only queries defining the views may need to be changed

61

---

---

---

---

---

---

---

---

## Views and data integrity

```
CREATE VIEW EMP10 AS
SELECT EMPNO, ENAME, JOB
FROM EMP
WHERE DEPTNO = 10
WITH CHECK OPTION
```

EMPNO	ENAME	JOB
7782	CLARK	MANAGER
7839	KING	PRESIDENT
7934	MILLER	CLERK

enforces constraint that any INSERT or UPDATE on the view satisfy the WHERE clause condition

62

---

---

---

---

---

---

---

---

## DROP VIEW

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- Causes definition of view to be deleted from database.
- For example:

```
DROP VIEW Manager3Staff;
```

63

---

---

---

---

---

---

---

---

**More on  
Integrity Constraints**

64

---

---

---

---

---

---

---

---

Recall – relational model

**Integrity constraints**

- Five types of integrity constraints
  - required data  
position VARCHAR(10) NOT NULL
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints

65

---

---

---

---

---

---

---

---

**Domain constraints**

(a) Checking the values  
sex CHAR NOT NULL  
CHECK (sex IN ('M', 'F'))

(b) Creating a domain  
CREATE DOMAIN DomainName [AS] dataType  
[DEFAULT defaultOption]  
[CHECK (searchCondition)]

- Example  
CREATE DOMAIN SexType AS CHAR  
CHECK (VALUE IN ('M', 'F'));  
sex SexType NOT NULL

66

---

---

---

---

---

---

---

---

## Domain constraints

- *searchCondition* can involve a table lookup  

```
CREATE DOMAIN BranchNo AS CHAR(4)  
CHECK (VALUE IN (SELECT branchNo FROM Branch));
```
- Removing domains  

```
DROP DOMAIN DomainName  
[RESTRICT | CASCADE]
```

---

---

---

---

---

---

---

---

## Entity integrity

- Primary key of a table must contain a unique, non-null value for each row.
- ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:  

```
PRIMARY KEY(staffNo)  
PRIMARY KEY(clientNo, propertyNo)
```
- Only one PRIMARY KEY clause per table. Uniqueness can be ensured (for alternate keys) using UNIQUE  

```
UNIQUE(telNo)
```

---

---

---

---

---

---

---

---

## Referential integrity

- FK is a column or set of columns that links each row in a child table containing foreign FK to a row of parent table containing matching PK.
- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
- ISO standard supports definition of FKs (in CREATE and ALTER TABLE)  

```
FOREIGN KEY(deptno) REFERENCES Department
```

---

---

---

---

---

---

---

---

### Referential integrity

- Any INSERT/UPDATE attempting to create an FK value in the child table without matching CK value in the parent table is **rejected**.
- Action taken attempting to update/delete a CK value in the parent table with existing matching rows in the child table is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:
  - CASCADE                      - SET NULL
  - SET DEFAULT                - NO ACTION

---

---

---

---

---

---

---

---

### Referential integrity

- CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.
- SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.
- SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
- NO ACTION: Reject delete from parent. Default,,

---

---

---

---

---

---

---

---

### Referential integrity - examples

FOREIGN KEY (deptno) REFERENCES  
Department ON DELETE SET NULL

FOREIGN KEY (deptno) REFERENCES  
Department ON UPDATE CASCADE

---

---

---

---

---

---

---

---

## Referential integrity – example

```
CREATE TABLE DEPT
(  DNAME      VARCHAR(10)  NOT NULL,
  DNUMBER     INTEGER      NOT NULL,
  MGRSSN      CHAR(9),
  MGRSTARTDATE CHAR(9),
  PRIMARY KEY (DNUMBER),
  UNIQUE (DNAME),
  FOREIGN KEY (MGRSSN) REFERENCES EMP
  ON DELETE SET DEFAULT ON UPDATE CASCADE
);
```

73

---

---

---

---

---

---

---

---

continued

## Referential integrity – example

```
CREATE TABLE EMP
(  ENAME      VARCHAR(30)  NOT NULL,
  ESSN       CHAR(9),
  BDATE      DATE,
  DNO        INTEGER      DEFAULT 1,
  SUPERSSN   CHAR(9),
  PRIMARY KEY (ESSN),
  FOREIGN KEY (DNO) REFERENCES DEPT
  ON DELETE SET DEFAULT
  ON UPDATE CASCADE,
  FOREIGN KEY (SUPERSSN) REFERENCES EMP
  ON DELETE SET NULL
  ON UPDATE CASCADE
);
```

74

---

---

---

---

---

---

---

---

## General constraints

### ■ Assertions

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)
```

### ■ Example

```
CREATE ASSERTION
StaffNotWorkingInMoreThanOneDept
CHECK (NOT EXISTS (SELECT EMPNO
                   FROM EMP
                   GROUP BY EMPNO
                   HAVING COUNT(*) > 1))
```

75

---

---

---

---

---

---

---

---

### Example

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
  CHECK (VALUE IN (SELECT ownerNo FROM
    PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
  CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN PNumber AS VARCHAR(5);
CREATE DOMAIN PRooms AS SMALLINT;
  CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PRent AS DECIMAL(6,2)
  CHECK(VALUE BETWEEN 0 AND 9999.99);
```

76

---

---

---

---

---

---

---

---

continued

### Example

```
CREATE TABLE PropertyForRent (
  propertyNo PNumber NOT NULL,
  rooms PRooms NOT NULL, DEFAULT 4,
  rent PRent NOT NULL, DEFAULT 600,
  ownerNo OwnerNumber NOT NULL,
  staffNo StaffNumber
  Constraint StaffNotHandlingTooMuch ....
  branchNo BranchNumber NOT NULL,
  PRIMARY KEY (propertyNo),
  FOREIGN KEY (staffNo) REFERENCES Staff
  ON DELETE SET NULL ON UPDATE CASCADE ....);
```

77

---

---

---

---

---

---

---

---



78

---

---

---

---

---

---

---

---

**Data Management using SQL**

79

---

---

---

---

---

---

---

---

**Adding new data**

INSERT INTO Staff  
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',  
Date'1957-05-25', 8300, 'B003');

■ **Also**  
INSERT INTO *table*  
[WHERE *search-condition*]

80

---

---

---

---

---

---

---

---

continued

**Adding new data**

Insert into a new relation called PROMOTION the data of salespeople who had a commission greater than 25% of their salary

```
INSERT INTO PROMOTION (ENAME, JOB, SAL, COMM)
SELECT ENAME, JOB, SAL, COMM
FROM EMP
WHERE COMM > 0.25 * SAL
```

81

---

---

---

---

---

---

---

---

## Updating stored data

```
UPDATE table
SET column = expression
[WHERE search-condition]
```

Give all clerks an increase of 100 to their salary

```
UPDATE EMP
SET SAL = SAL + 100
WHERE JOB = 'CLERK'
```

82

---

---

---

---

---

---

---

---

## Deleting stored data

```
DELETE FROM table
[WHERE search-condition]
```

Delete department 40 from the department relation

```
DELETE FROM DEPT
WHERE DEPTNO = 40
```

83

---

---

---

---

---

---

---

---

## Altering an existing table

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.
- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.

84

---

---

---

---

---

---

---

---

## Adding new columns

- To add a new column PROJNO (designating the number of the project to which an employee is assigned) to table EMP the following command is used:

```
ALTER TABLE EMP ADD (PROJNO NUMBER (3))
```



85

---

---

---

---

---

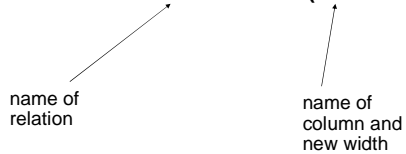
---

---

---

## Modifying existing columns

```
ALTER TABLE EMP MODIFY (SALARY (8,2))
```



86

---

---

---

---

---

---

---

---

## DROP TABLE

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

e.g. DROP TABLE PropertyForRent;

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

87

---

---

---

---

---

---

---

---

**SQL Summary**

88

---

---

---

---

---

---

---

---

**Data definition**

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- Main SQL DDL statements are:

CREATE SCHEMA	DROP SCHEMA
CREATE/ALTER DOMAIN	DROP DOMAIN
CREATE/ALTER TABLE	DROP TABLE
CREATE VIEW	DROP VIEW
- Also creating indexes

CREATE INDEX	DROP INDEX
--------------	------------

89

---

---

---

---

---

---

---

---

**Data retrieval using SQL**

- Select-From-Where
- Ordering of results (ORDER BY)
- Aggregate functions (count, avg, min, max, sum)
- Grouping (GROUP BY, HAVING)
- Sub-queries (nested queries)

90

---

---

---

---

---

---

---

---

## MySQL Workbench/DBDesigner

- Visual database design system that integrates database design, modeling, creation and maintenance
- Two modes:
  - Design Mode is used to create and maintain the visual databases model.
  - The Query Mode is used to work with table data and build complex SQL query statements

91

---

---

---

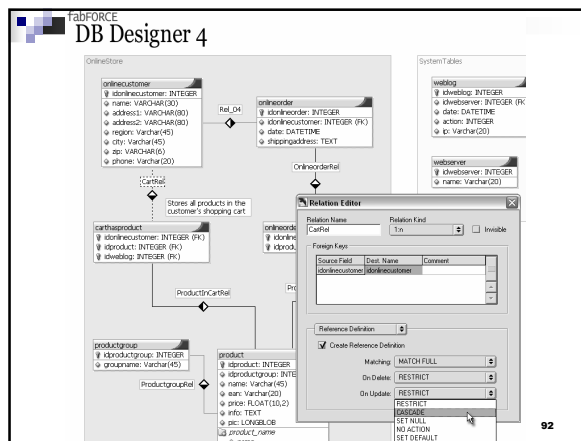
---

---

---

---

---



92

---

---

---

---

---

---

---

---

## Reading for this lecture

- Main text: Chapter 8 in [Elmasri & Navathe]  
Sections 9.1 and 9.2 in [Elmasri & Navathe]
- Also, Chapters 6 and 7 in [Connolly & Begg]
- See also additional documentation (on aliases, tuples as sets etc)
- Start using MySQL – create tables, run simple queries during the tutorial today.

93

---

---

---

---

---

---

---

---