SQL

Repetition

- Creating Schemas
- Inserting
- Selection
- Constraints

Data Definition Language

SQL DDL

- Create a database with CREATE DATABASE

```
CREATE DATABASE IF NOT EXISTS USNavy;
```

SQL DDL

- Three type of tables in SQL
    - Stored Relations, called tables
    - Views: relations calculated by computation
    - Temporary tables: created during query execution

SQL DDL

- Data Types
  - Character strings of fixed or varying length
    - CHAR(n) - fixed length string of up to *n* characters
    - VARCHAR(n) - fixed length string of up to *n* characters
      - Uses and endmarker or string-length for storage efficiency
  - Bit strings
    - BIT(n) strings of length exactly *n*
    - BIT VARYING(n) - strings of length up to *n*

SQL DDL

- Data Types:
  - Boolean: BOOLEAN: TRUE, FALSE, UNKNOWN
  - Integers: INT = INTEGER, SHORTINT
  - Floats: FLOAT = REAL, DOUBLE, DECIMAL(n,m)
  - Dates: DATE
    - SQL Standard: ‘1948-05-14’)
  - Times: TIME
    - SQL Standard: 19:20:02.4

SQL DDL

- Data Types:
  - MySQL: ENUM('M', 'F')

SQL DDL

- CREATE TABLE creates a table

```
CREATE TABLE Movies(
   title          CHAR(100),
   year           INT,
   length         INT,
   genre          CHAR(10),
   studioName     CHAR(30),
   producerC#     INT
);
```

SQL DDL

```
CREATE TABLE MovieStar(
   name               CHAR(30),
   address            VARCHAR(255),
   gender             CHAR(1),
   birthday           DATE
);
```

SQL DDL

- Drop Table drops a table

```
DROP TABLE Movies;
```

SQL DDL

- Altering a table with ALTER TABLE
  - with ADD followed by attribute name and data type
  - with DROP followed by attribute name

```
ALTER TABLE MovieStar ADD phone CHAR(16);
```

```
ALTER TABLE MovieStar DROP Birthday;
```

SQL DDL

- Default Values
  - Conventions for unknown data
    - Usually, NULL
  - Can use other values for unknown data

```
CREATE TABLE MovieStar(
   name            CHAR(30),
   address         VARCHAR(255),
   gender          CHAR(1) DEFAULT '?',
   birthday        DATE DEFAULT '0000-00-00'
);
```

SQL DDL

- Declaring Keys
  1. Declare one attribute to be a key
  2. Add one additional declaration:
     - Particular set of attributes is a key
  - Can use
  1. PRIMARY KEY
  2. UNIQUE

SQL DDL

- UNIQUE for a set S:
  - Two tuples cannot agree on all attributes of S unless one of them is NULL
    - Any attempted update that violates this will be rejected
- PRIMARY KEY for a set S:
  - Attributes in S cannot be NULL

SQL DDL

```
CREATE TABLE MovieStar(
   name               CHAR(30) PRIMARY KEY,
   address            VARCHAR(255),
   gender             CHAR(1),
   birthday           DATE
);
```

SQL DDL

```
CREATE TABLE MovieStar(
   name            CHAR(30),
   address         VARCHAR(255),
   gender          CHAR(1) DEFAULT '?',
   birthday        DATE DEFAULT '0000-00-00',
   PRIMARY KEY (name)
);
```

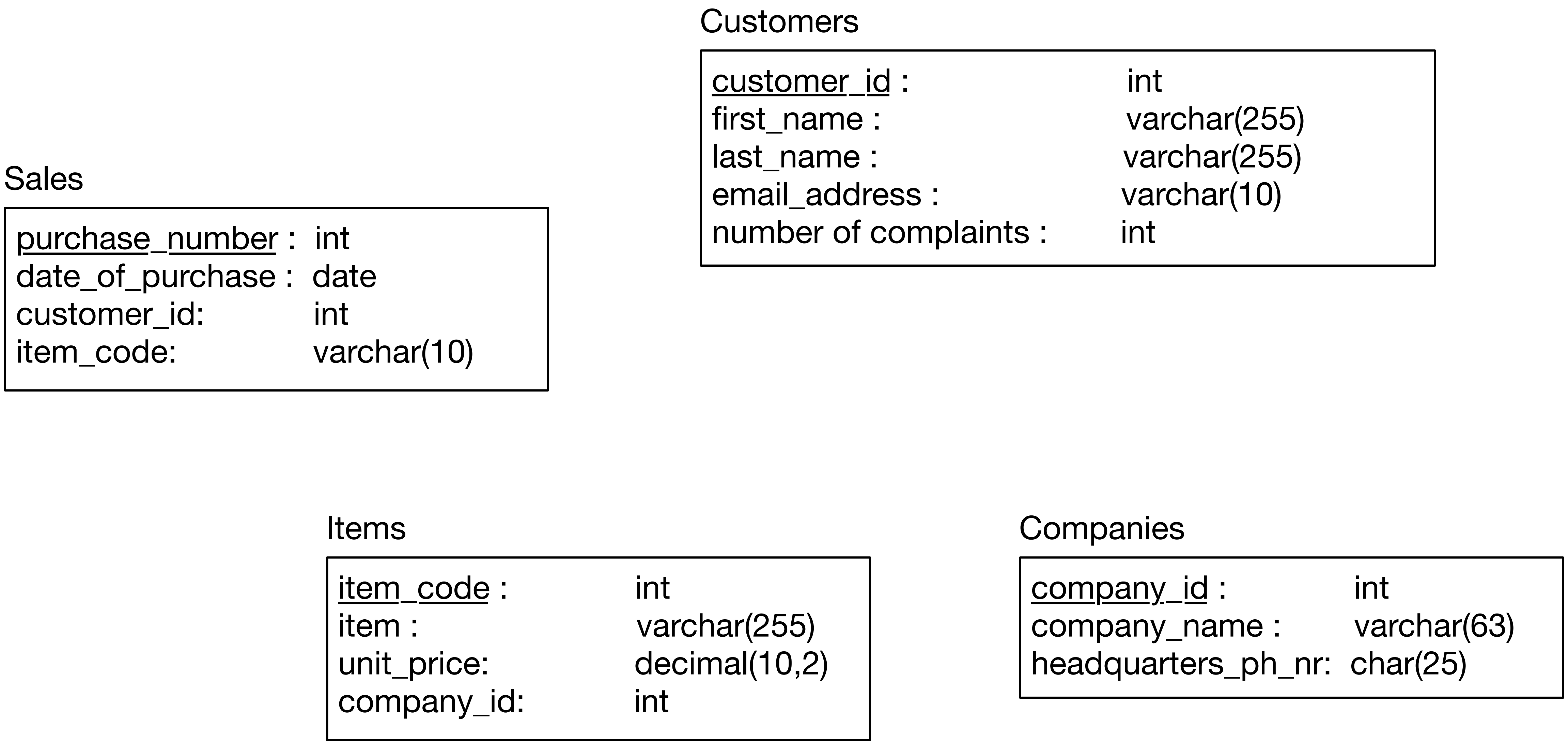
SQL DDL

```
CREATE TABLE Movies(
   title           CHAR(100),
   year            INT,
   length          INT,
   genre           CHAR(10),
   studioName      CHAR(30),
   producerC#      INT,
   PRIMARY KEY (title, year)
);
```

Simple Diagrams

- A schema is represented by a networked diagram
    - Nodes represent tables
        - Name of the table labels the node
        - Interior of the node are the name of the attributes
        - Underline the primary key
        - Optionally, add domain to each attribute

Simple Diagrams



Customers

customer_id :	int
first_name :	varchar(255)
last_name :	varchar(255)
email_address :	varchar(10)
number of complaints :	int

Sales

purchase_number :	int
date_of_purchase :	date
customer_id:	int
item_code:	varchar(10)

Items

item_code :	int
item :	varchar(255)
unit_price:	decimal(10,2)
company_id:	int

Companies

company_id :	int
company_name :	varchar(63)
headquarters_ph_nr:	char(25)

Constraints in MySQL

- Constraints in MySQL have names
    - Often automatically generated
    - Use the SHOW CREATE TABLE query

```
Table,"Create Table"
customers,"CREATE TABLE `customers` (
  `customer_id` int NOT NULL AUTO_INCREMENT,
  `first_name` varchar(255) DEFAULT NULL,
  `last_name` varchar(255) DEFAULT NULL,
  `email_address` varchar(255) DEFAULT NULL,
  `number_of_complaints` int DEFAULT (0),
  PRIMARY KEY (`customer_id`),
  UNIQUE KEY `email_address` (`email_address`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci"
```

Constraints in MySQL

- Missing values are usually a NULL
  - Can automatically assign INT with AUTO_INCREMENT
  - Used widely to assign artificial primary keys

Constraints in MySQL

- NOT NULL constraint
  - When inserting a tuple with NULL value in the constrained column, error will be thrown

```
CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE
);
```

  - Considered good practice to include in all columns where a NULL value is not expected

Constraints in MySQL

- ALTER TABLE allows to introduce new / remove old constraint
  - Need to check that the inserted values comply

```
ALTER TABLE tasks
CHANGE
    end_date
    end_date DATE NOT NULL;
```

```
ALTER TABLE tasks
MODIFY
    end_date
    end_date DATE NOT NULL;
```

Constraints in MySQL

- UNIQUE
  - Values in a single attribute are different
  - Value groups in a group of attributes are different
- Creating a constraint:
  - Specify in CREATE TABLE for a single attribute
  - Add a CONSTRAINT cstr_name UNIQUE(attr1, attr2, …)
    - Can leave out constraint name, will be replaced by an automatically created name
  - Use ALTER TABLE ADD CONSTRAINT

Constraints in MySQL

- UNIQUE

```
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

Constraints in MySQL

- UNIQUE constraint creates an *index*
    - Index is a data structure with quick look-up
- Access indices through the SHOW INDEX FROM table command

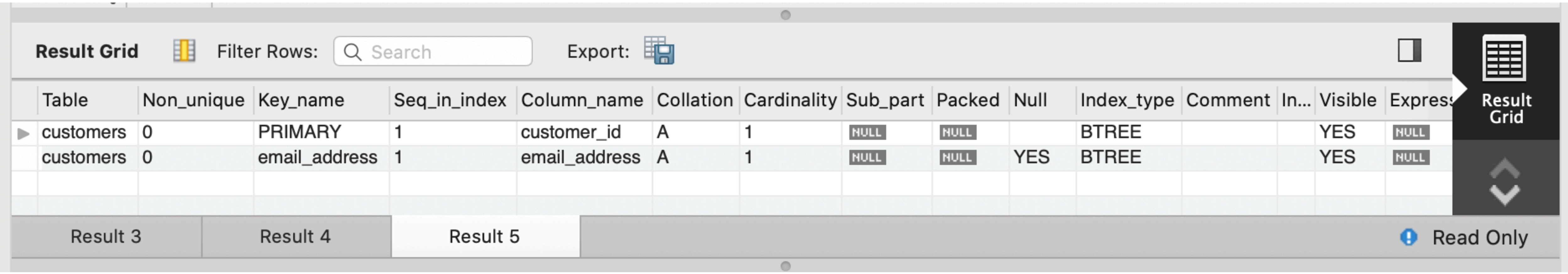
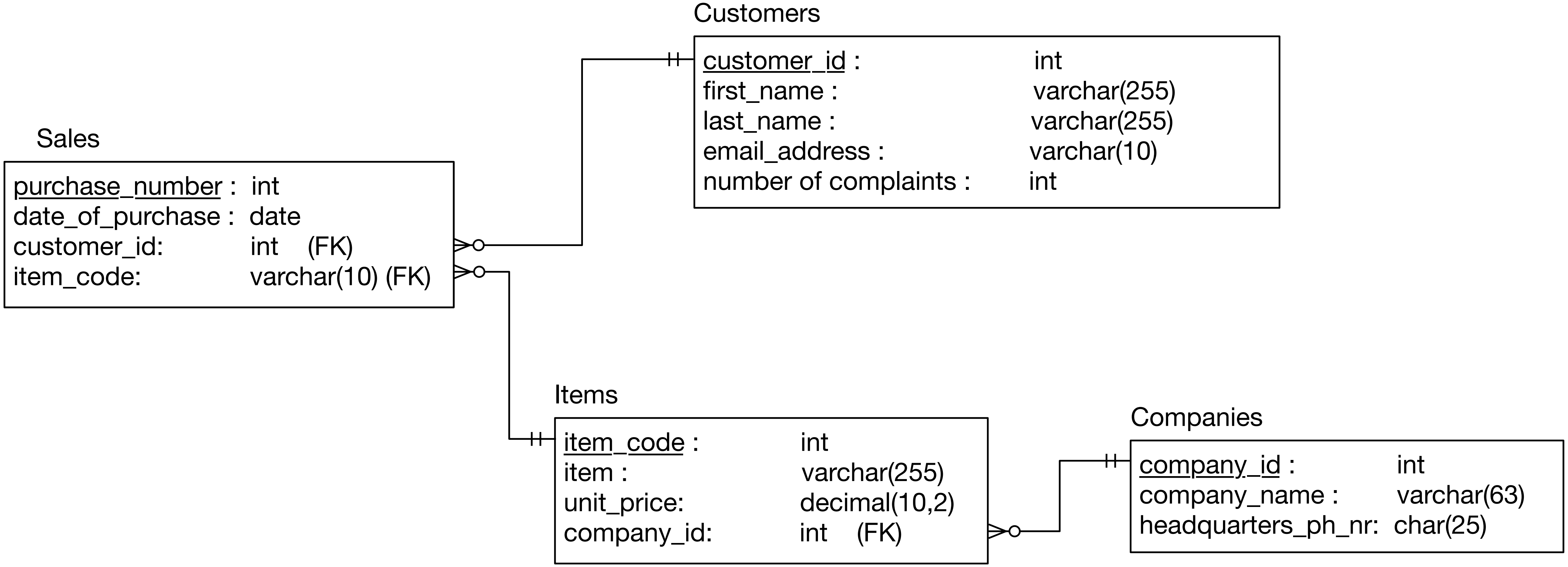


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	In...	Visible	Express
customers	0	PRIMARY	1	customer_id	A	1	NULL	NULL		BTREE			YES	NULL
customers	0	email_address	1	email_address	A	1	NULL	NULL	YES	BTREE			YES	NULL

Foreign Keys

- Relationships between tables are sometimes constructed with shared values
  - Sales has an attribute client_id
  - Customers has a primary key client_id
    - Need not be named the same
      - But it is usually convenient to do so

Constraints in MySQL



Customers

customer_id :	int
first_name :	varchar(255)
last_name :	varchar(255)
email_address :	varchar(10)
number of complaints :	int

Sales

purchase_number :	int
date_of_purchase :	date
customer_id:	int (FK)
item_code:	varchar(10) (FK)

Items

item_code :	int
item :	varchar(255)
unit_price:	decimal(10,2)
company_id:	int (FK)

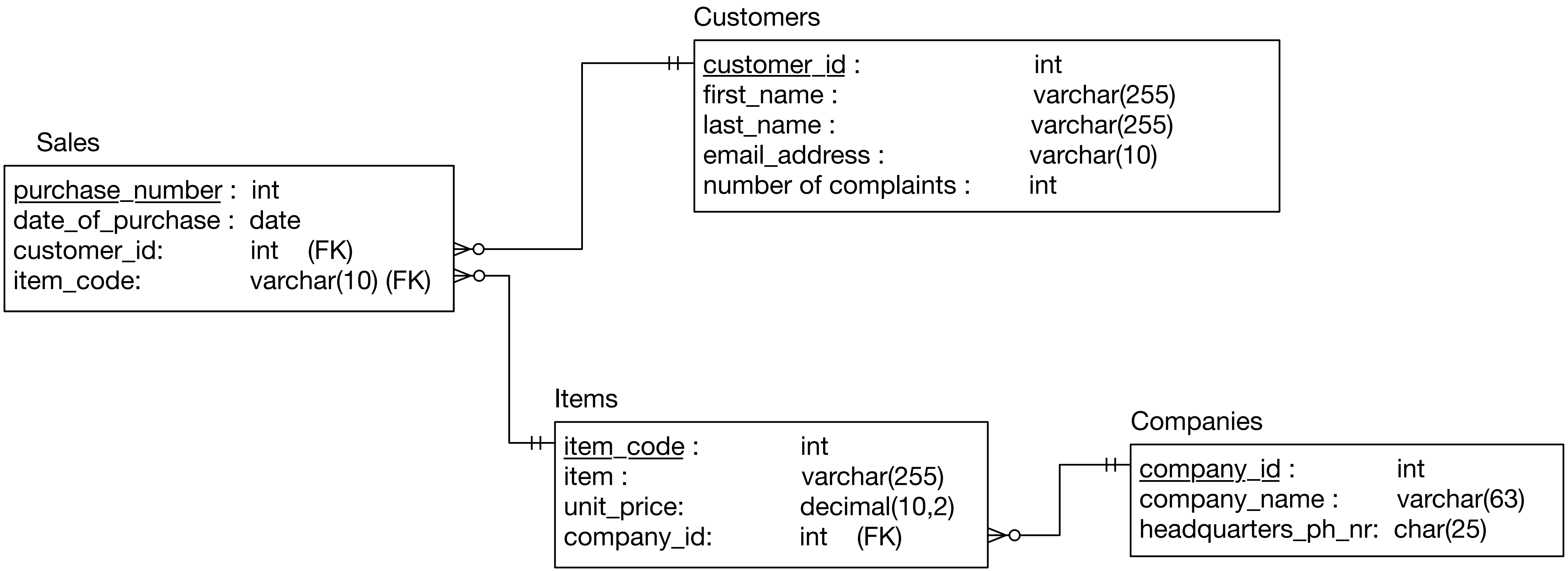
Companies

company_id :	int
company_name :	varchar(63)
headquarters_ph_nr:	char(25)

Constraints in MySQL

- Example:
  - A customer can have many sales
  - But each sale has only one customer
  - Relationship customers sales is a **one-to-many** relationship
  - customers is the *referenced* (or parent) table
  - sales is the *referencing* (or child) table
  - As is typical, the referenced attribute is a primary key in the referenced table

Constraints in MySQL



Customers

customer_id : int
first_name : varchar(255)
last_name : varchar(255)
email_address : varchar(10)
number of complaints : int

Sales

purchase_number : int
date_of_purchase : date
customer_id: int (FK)
item_code: varchar(10) (FK)

Items

item_code : int
item : varchar(255)
unit_price: decimal(10,2)
company_id: int (FK)

Companies

company_id : int
company_name : varchar(63)
headquarters_ph_nr: char(25)

Constraints in MySQL

- In a diagram:
    - crow-feet with ball indicate many
    - double bar indicates one

Constraints in MySQL

- Foreign key constraint
  - Once established, insures that action is taken upon insertion or deletion of a record affecting the other table

Constraints in MySQL

- Possible Actions:
  - CASCADE: if a tuple from the referenced table is deleted or updated, the corresponding tuple in the referencing table is also deleted / updated
  - SET NULL: If a row from the referenced table is deleted or updated, the values of the foreign key in the referencing table are set to NULL

Constraints in MySQL

- Possible Actions:
  - RESTRICT: if a row from the referenced table has a matching row in the referencing table, then deletion and updates are rejected
  - SET DEFAULT: Accepted by MySQL parser but action not performed

Constraints in MySQL

- Foreign keys constraint actions
    - Are for
        - ON UPDATE
        - ON DELETE

Constraints in MySQL

- Creating foreign key constraints:

```
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
);

CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Constraints in MySQL

- You can drop a foreign key restraint using the ALTER TABLE statement

```
ALTER TABLE table_name
DROP FOREIGN KEY constraint_name;
```

Constraints in MySQL

- When loading a database from (e.g.) .csv files
  - Can carefully create referenced tables before referencing tables
  - Temporarily disable foreign key checks

```
SET foreign_key_checks = 0;
```

```
SET foreign_key_checks = 1;
```

Select

Select

- SELECT * FROM table
- SELECT col1, col2 FROM table
- SELECT * FROM table WHERE conditions

SELECT

- =   equals (comparison operator)
- AND, OR
- IN, NOT IN
- LIKE, NOT LIKE
- BETWEEN … AND
- EXISTS, NOT EXISTS
- IS NULL, IS NOT NULL
- comparison operators

Comparisons with NULL

- NULL in any expression gives NULL
    - If you compare anything with NULL in MySQL, you get NULL
    - IF you order, NULL values appear last
- In other SQL dialects: UNKNOWN

SELECT

- LIKE
  - Pattern matching
    - Wild cards
      - % means zero or more characters
      - _ means a single letter
      - [ ] means any single character within the bracket
      - ^ means any character not in the bracket
      - - means a range of characters

SELECT

- BETWEEN … AND …
  - Selects records with a value in the range
    - endpoints included

```
SELECT
   *
FROM
   employees
WHERE
   hire_data between 1990-01-01 and 1999-12-31;
```

SELECT

- SELECT DISTINCT

```
SELECT DISTINCT
     gender
FROM
     employees
```

Like Examples

- WHERE name LIKE 't%'
  - any values that start with 't'
- WHERE name LIKE '%t'
  - any values that end with 't'
- WHERE name LIKE '%t%'
  - any value with a 't' in it
- WHERE name LIKE '_t%'
  - any value with a 't' in second position

SELECT

- LIMIT gives the maximum number of rows returned
  - Can be used for a sample
  - Can be used with ORDER BY ASC

Insert Operations

- Insert Syntax
  - No need to insert into automatic values
  - If only a few attributes are set,

```
INSERT INTO
table(attr1, attr2, …)
Values(v1, v2, …)
```

  - If all attributes are set, just list the values
  - Can set many tuples at once

```
INSERT INTO served
VALUES
('William Howe', 'Great Britain', '1746-1-1', '1778-4-1'),
('Benedict Arnold', 'Great Britain', '1757-1-1', '1775-1-1'),
('Benedict Arnold', 'United States', '1775-1-1', '1780-9-1'),
('Benedict Arnold', 'Great Britain', '1780-9-1', '1787-1-1')
```

Queries with more than one table

- SQL has explicit commands for the various joins and products
- Normally, combine tables by listing them in the FROM clause

```
SELECT name
FROM movies, moviesExec
WHERE title = ‘Star Wars’
        AND movies.producerC# = moviesExec.cert#
```

Queries with more than one table

- Find all movie execs that live with a star

-
  ```
  MovieStar(name, address, gender, birthdate)
  MovieExec(name, address, cert#, netWorth)
  ```

```
SELECT MovieStar.name, MovieExec.name)
FROM MovieStar, MovieExec
WHERE
   MovieStar.address = MovieExec.address
```

Queries with more than one table

- Tuple Variables
  - Sometimes need to combine two tuples in the same table
  - Can extend the FROM clause

```
SELECT Star1.name, Star2.name
FROM MovieStars Star1, MovieStars Star2
WHERE
  Star1.address = Star2.address
  AND  Star1.name < Star2.name
```

Queries with more than one table

- Unions, intersections, excepts
- To execute the corresponding set operations
-

```
(SELECT name, address
 FROM movieStars
 WHERE gender = 'F'
)
 INTERSECT
(SELECT name, address
 FROM movieExecs
 WHERE netWorth > 1000000
)
```

Updates

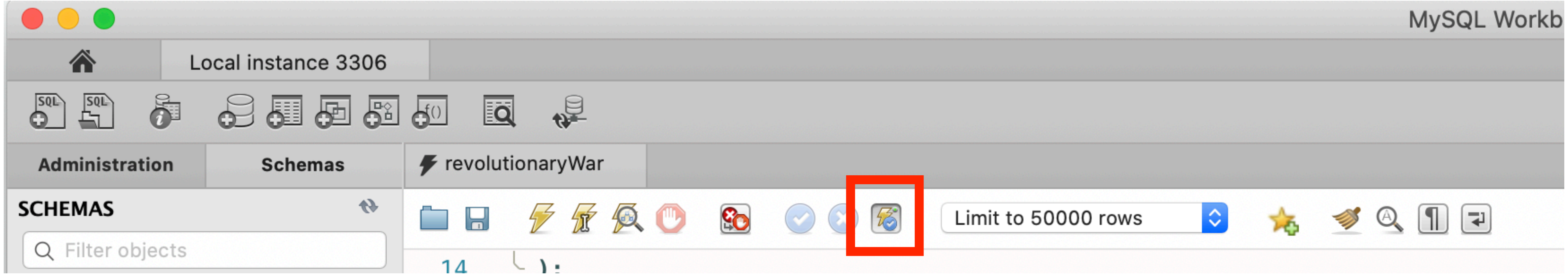
- Changes existing records
- Syntax:

```
UPDATE tablename
SET attr1=val1, attr2=val2, …
WHERE conditions;
```

- Does not need to change all attributes
- If there is no WHERE condition, all records are updated

Commit and Rollback

- A database allows us to rollback to a previous state unless we have committed
- MySQLWorkbench has an auto-commit button



- Rollback puts database into the state of the last commit

Delete

- Just like an update

```
DELETE FROM tablename
WHERE condition
```

- The Where clause is not necessary

Delete, Drop, Truncate

- Drop Table:
    - Definite action: cannot recover with rollback
- Truncate:
    - All records removed
    - Auto-increment values reset
    - Table description stays
- Delete:
    - Delete removes records row by row
    - Auto-increment values remain
    - Slower than truncate

Subqueries

- Subqueries are helper queries

Subqueries

- Subqueries producing a scalar value
  - Example: Producer of Star Wars

```
SELECT name
From movies, movieExec
WHERE title = 'Star Wars'
        AND
      producerC# = cert#;
```

  - Can achieve the same effect by first looking for the producerC#

Subqueries

- Example: Producer of Star Wars

```
SELECT name
FROM movieExec
WHERE cert# =
    (SELECT producerC#
     FROM movies
     WHERE title = 'star wars'
    )
```

- This might be implemented with the same query execution as before

Subqueries

- Subqueries with conditions involving relations
  - We obtain a relation R as a subquery
  - E.g. with subquery (SELECT * FROM foobar)
  - Queries are:
    - EXISTS R
    - *s* IN R   *s* NOT IN R
    - *s* > ALL R   NOT   *s* > ALL R
    - *s* > ANY R   NOT *s* > ANY R

Subqueries

- Subqueries involving tuples
  - Tuple is a list of scalar values
  - Can compare tuples with the same number of components
  - Example:
    - Finding the producers of 'Harrison Ford' movies

Subqueries

```
SELECT name
FROM movieExec
WHERE cert# IN
    (SELECT producerC#
     FROM movies
     WHERE (title, year) IN
        (SELECT movieTitle, movieYear
         FROM StarsIn
         WHERE starName = 'Harrison Ford'
        )
     );
```

Subqueries

- To analyze a query, start with the inmost query

```
SELECT name
FROM movieExec
WHERE cert# IN
    (SELECT producerC#
     FROM movies
     WHERE (title, year) IN
        (SELECT movieTitle, movieYear
         FROM StarsIn
         WHERE starName = 'Harrison Ford'
        )
     );
```

Subqueries

- This query can also be written without nested subqueries

```
SELECT name
FROM movieExec, movies, starsIn
WHERE  cert# = producerC#
     AND starsIn.title = movies.title
     AND starsIn.year = movie.year
     AND starName = 'Harrison Ford'
```

Subqueries

- Correlated subqueries
    - Subquery is evaluated many times
        - Once for each value given
- Example

```
SELECT title
FROM movies Old
WHERE year < ANY (
   SELECT year
   FROM movies
   WHERE title = Old.title
 );
```

Subqueries

- Scoping rules
  - First look for the subquery and tables in that subquery
  - Then go to the nesting subquery
  - etc.

Subqueries

- Subqueries in FROM clauses
    - Here we join on a subquery aliased Prod

```
SELECT name
FROM movieExecs, ( SELECT producerC#
                   FROM movies, starsIn
                   WHERE movies.title = starsIn.title
                      AND movies.year = starsIn.year
                      AND starName = 'Harrison Ford'
                 ) Prod
WHERE cert# = Prod.producerC#
```

Subqueries

- SQL JOIN expression
  - Explicit construction of various joins
    - CROSS JOIN (product)
    - NATURAL JOIN
    - FULL OUTER JOIN
    - NATURAL FULL OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN

Subqueries

- Examples

```
movies FULL OUTER JOIN starsIn ON
movies.title = stars.
```

Subqueries

- Examples

```
movieStar(name, address, gender, birthday)
movieExec(name, address, cert#, netWorth)
```

```
movieStar NATURAL FULL OUTER JOIN movieExec(
   name, address, gender, birthday, cert#, netWorth)
```

Eliminating Duplicates

- Use Distinct

```
SELECT DISTINCT name
FROM movies
```

- Warning: Invoking distinct is costly

Eliminating Duplicates

- Union, intersection, difference usually remove duplicates automatically
- If we do not want this, but bag semantics:
  - Use the keyword all

```
(SELECT title, year
FROM movies)
UNION ALL
(SELECT movieTitle AS title,
        movieYear AS year
 FROM
 starsIn);
```

Aggregate Functions

- COUNT
  - numeric and non-numeric data
  - null values excepted
- SUM, MIN, MAX, AVG - only numeric data

- Exercise: Find the number of different stars in the starsIn table

```
SELECT COUNT(DISTINCT name)
FROM starsIn
```

Aggregate Functions

- Find the combined net-worth of movieExecs

```
SELECT SUM(networth)
FROM movieExecs
```

- Find the average net-worth of movieExecs

```
SELECT ROUND(AVG(networth),2)
FROM movieExecs
```

Aggregate Functions

- Dealing if NULL values
  - IFNULL(EXPR1, EXPR2):
    - Gives EXPR1 if it is not NULL and EXPR2 if not

-
  ```
  SELECT
      name,
      IFNULL(studio, 'not president') AS studio
  FROM movieExecs;
  ```

Aggregate Functions

- COALESCE(EXPR1, EXPR2, EXPR3, … EXPRn)
  - Gives first nonNULL expression

Grouping

- Aggregation happens usually with grouping
    - To group, use GROUP BY followed by a WHERE clause

```
SELECT studioName, SUM(length) AS totalRunTime
FROM movies
GROUP BY studioName;
```

Grouping

- Example
    - Computing the total run time of movies produced by a producer

```
SELECT name, SUM(length) AS totalRunTime
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name;
```

Grouping

- Aggregation and Nulls
  - NULL does not contribute to a sum, average, or count
- Grouping and Nulls
  - NULL is an ordinary value for grouping purposes
- Aggregation except COUNT over an empty bag gives result NULL

Transactions

Transactions

- Databases have to process many operations in parallel
- This means some support for inter-process communication
  - Usually provided by logging
- DBMS differ in what they provide
  - Serializability:
    - All transactions appear to have been executed one after the other

Transactions

- Atomicity
    - A single query is never interrupted:
        - Example:
            - A transfer of money from one account to another is executed completely or not at all
            - Both accounts have changed or none

Transactions

- Transaction
    - A group of SQL statements that are all processed in the order given or not at all
- SQL:
    - START TRANSACTION
    - either
    - COMMIT
    - ROLLBACK

Transactions

- Read only transactions
  - By declaring a transaction as read-only, SQL can usually perform it quicker
  - SET TRANSACTION READ ONLY;
  - SET TRANSACTION READ WRITE;

Transactions

- Dirty Reads:
    - Reading a record from an update that will be rolled-back
- Are dirty reads bad?
    - Depends
        - Sometimes, it does not matter, and we do not want the DBMS spend time on making sure that there are no dirty reads
        - Sometimes, a dirty read can absolutely mess up things
            - Selling the same commodity to two customers, …

Transactions

- SQL Isolation Levels:
  - Allow dirty reads:
    - SET TRANSACTION READ WRITE
    - SET ISOLATION LEVEL READ UNCOMMITTED

Transactions

- SQL Isolation Levels:
    - Allow reads only of committed data:
        - SET TRANSACTION READ WRITE
        - SET ISOLATION LEVEL READ COMMITTED

Transactions

- SQL Isolation Levels:
    - Disallow dirty reads, but insure that the reads are consistent:
        - SET TRANSACTION READ WRITE
        - SET ISOLATION LEVEL READ REPEATABLE READ

Transactions

- SQL Isolation Levels:
    - Serializability (default):
        - SET TRANSACTION READ WRITE
        - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE